



Practice tests



Video Training



Flash Cards



Study Planner

Official Cert Guide

Advance your IT career with hands-on learning

Cisco Certified DevNet Associate

DEVASC 200-901

Chris Jackson, CCIE® x2 (R&S & SEC) No. 6256

Jason Gooley, CCIE® x2 (R&S & SP) No. 38759

Adrian Iliesiu, CCIE® R&S No. 43909

Ashutosh Malegaonkar

ciscopress.com

Contents

1. Cover Page
2. About This eBook
3. Title Page
4. Copyright Page
5. About the Authors
6. About the Technical Reviewers
7. Dedications
8. Acknowledgments
9. Contents at a Glance
10. Reader Services
11. Contents
12. Icons Used in This Book
13. Command Syntax Conventions
14. Introduction
 1. Goals and Methods
 2. Who Should Read This Book?
 3. Strategies for Exam Preparation
 4. The Companion Website for Online Content Review
 5. How This Book Is Organized
 6. Certification Exam Topics and This Book
15. Figure Credits
16. Chapter 1. Introduction to Cisco DevNet Associate Certification
 1. Do I Know This Already?
 2. Foundation Topics
 3. Why Get Certified
 4. Cisco Career Certification Overview
 5. Cisco DevNet Certifications
 6. Cisco DevNet Overview
 7. Summary
17. Chapter 2. Software Development and Design
 1. “Do I Know This Already?” Quiz
 2. Foundation Topics
 3. Software Development Lifecycle
 4. Common Design Patterns
 5. Linux BASH
 6. Software Version Control
 7. Git
 8. Conducting Code Review
 9. Exam Preparation Tasks
 10. Review All Key Topics
 11. Define Key Terms

18. Chapter 3. Introduction to Python

- 1. “Do I Know This Already?” Quiz**
- 2. Foundation Topics**
- 3. Getting Started with Python**
- 4. Understanding Python Syntax**
- 5. Data Types and Variables**
- 6. Input and Output**
- 7. Flow Control with Conditionals and Loops**
- 8. Exam Preparation Tasks**
- 9. Review All Key Topics**
- 10. Define Key Terms**
- 11. Additional Resources**

19. Chapter 4. Python Functions, Classes, and Modules

- 1. “Do I Know This Already?” Quiz**
- 2. Foundation Topics**
- 3. Python Functions**
- 4. Using Arguments and Parameters**
- 5. Object-Oriented Programming and Python**
- 6. Python Classes**
- 7. Working with Python Modules**
- 8. Exam Preparation Tasks**
- 9. Review All Key Topics**
- 10. Define Key Terms**

20. Chapter 5. Working with Data in Python

- 1. “Do I Know This Already?” Quiz**
- 2. Foundation Topics**
- 3. File Input and Output**
- 4. Parsing Data**
- 5. Error Handling in Python**
- 6. Test-Driven Development**
- 7. Unit Testing**
- 8. Exam Preparation Tasks**
- 9. Review All Key Topics**
- 10. Define Key Terms**
- 11. Additional Resources**

21. Chapter 6. Application Programming Interfaces (APIs)

- 1. “Do I Know This Already?” Quiz**
- 2. Foundation Topics**
- 3. Application Programming Interfaces (APIs)**
- 4. Exam Preparation Tasks**
- 5. Review All Key Topics**
- 6. Define Key Terms**

22. Chapter 7. RESTful API Requests and Responses

- 1. “Do I Know This Already?” Quiz**
- 2. Foundation Topics**
- 3. RESTful API Fundamentals**
- 4. REST Constraints**

5. REST Tools
6. Exam Preparation Tasks
7. Review All Key Topics
8. Define Key Terms

23. Chapter 8. Cisco Enterprise Networking Management Platforms and APIs

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. What Is an SDK?
4. Cisco Meraki
5. Cisco DNA Center
6. Cisco SD-WAN
7. Exam Preparation Tasks
8. Review All Key Topics
9. Define Key Terms

24. Chapter 9. Cisco Data Center and Compute Management Platforms and APIs

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. Cisco ACI
4. UCS Manager
5. Cisco UCS Director
6. Cisco Intersight
7. Exam Preparation Tasks
8. Review All Key Topics
9. Define Key Terms

25. Chapter 10. Cisco Collaboration Platforms and APIs

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. Introduction to the Cisco Collaboration Portfolio
4. Webex Teams API
5. Cisco Finesse
6. Webex Meetings APIs
7. Webex Devices
8. Cisco Unified Communications Manager
9. Exam Preparation Tasks
10. Review All Key Topics
11. Define Key Terms

26. Chapter 11. Cisco Security Platforms and APIs

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. Cisco’s Security Portfolio
4. Cisco Umbrella
5. Cisco Firepower
6. Cisco Advanced Malware Protection (AMP)
7. Cisco Identity Services Engine (ISE)
8. Cisco Threat Grid
9. Exam Preparation Tasks

10. Review All Key Topics

11. Define Key Terms

27. Chapter 12. Model-Driven Programmability

1. “Do I Know This Already?” Quiz

2. Foundation Topics

3. NETCONF

4. YANG

5. RESTCONF

6. Model-Driven Telemetry

7. Exam Preparation Tasks

8. Review All Key Topics

9. Define Key Terms

28. Chapter 13. Deploying Applications

1. “Do I Know This Already?” Quiz

2. Foundation Topics

3. Application Deployment Models

4. NIST Definition

5. Application Deployment Options

6. Application Deployment Methods

7. Bare-Metal Application Deployment

8. Virtualized Applications

9. Cloud-Native Applications

10. Containerized Applications

11. Serverless

12. DevOps

13. What Is DevOps?

14. Putting DevOps into Practice: The Three Ways

15. DevOps Implementation

16. Docker

17. Understanding Docker

18. Docker Architecture

19. Using Docker

20. Docker Hub

21. Exam Preparation Tasks

22. Review All Key Topics

23. Define Key Terms

24. Additional Resources

29. Chapter 14. Application Security

1. “Do I Know This Already?” Quiz

2. Foundation Topics

3. Identifying Potential Risks

4. Protecting Applications

5. Exam Preparation Tasks

6. Review All Key Topics

7. Define Key Terms

30. Chapter 15. Infrastructure Automation

1. “Do I Know This Already?” Quiz

2. Foundation Topics

3. Controller Versus Device-Level Management
4. Infrastructure as Code
5. Continuous Integration/Continuous Delivery Pipelines
6. Automation Tools
7. Cisco Network Services Orchestrator (NSO)
8. Exam Preparation Tasks
9. Review All Key Topics
10. Define Key Terms

31. Chapter 16. Network Fundamentals

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. Network Reference Models
4. Switching Concepts
5. Routing Concepts
6. Exam Preparation Tasks
7. Review All Key Topics
8. Define Key Terms

32. Chapter 17. Networking Components

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. What Are Networks?
4. Elements of Networks
5. Software-Defined Networking
6. Exam Preparation Tasks
7. Review All Key Topics
8. Define Key Terms

33. Chapter 18. IP Services

1. “Do I Know This Already?” Quiz
2. Foundation Topics
3. Common Networking Protocols
4. Network Address Translation (NAT)
5. Layer 2 Versus Layer 3 Network Diagrams
6. Troubleshooting Application Connectivity Issues
7. Exam Preparation Tasks
8. Review All Key Topics
9. Define Key Terms

34. Chapter 19. Final Preparation

1. Getting Ready
2. Tools for Final Preparation
3. Suggested Plan for Final Review/Study
4. Summary

35. Appendix A. Answers to the “Do I Know This Already?” Quiz Questions

36. Appendix B. DevNet Associate DEVASC 200-901 Official Cert Guide Exam Updates

1. Always Get the Latest at the Book’s Product Page

2. Technical Content

- 37. Glossary**
- 38. Index**
- 39. Appendix C. Study Planner**
- 40. Where are the companion content files? - Register**
- 41. Inside Front Cover**
- 42. Inside Back Cover**
- 43. Code Snippets**

- 1. i**
- 2. ii**
- 3. iii**
- 4. iv**
- 5. v**
- 6. vi**
- 7. vii**
- 8. viii**
- 9. ix**
- 10. x**
- 11. xi**
- 12. xii**
- 13. xiii**
- 14. xiv**
- 15. xv**
- 16. xvi**
- 17. xvii**
- 18. xviii**
- 19. xix**
- 20. xx**
- 21. xxi**
- 22. xxii**
- 23. xxiii**
- 24. xxiv**
- 25. xxv**
- 26. xxvi**
- 27. xxvii**
- 28. xxviii**
- 29. xxix**
- 30. xxx**
- 31. xxxi**
- 32. xxxii**
- 33. xxxiii**
- 34. xxxiv**
- 35. xxxv**
- 36. 2**
- 37. 3**
- 38. 4**
- 39. 5**
- 40. 6**
- 41. 7**
- 42. 8**
- 43. 9**
- 44. 10**

45. 11
46. 12
47. 13
48. 14
49. 15
50. 16
51. 17
52. 18
53. 19
54. 20
55. 21
56. 22
57. 23
58. 24
59. 25
60. 26
61. 27
62. 28
63. 29
64. 30
65. 31
66. 32
67. 33
68. 34
69. 35
70. 36
71. 37
72. 38
73. 39
74. 40
75. 41
76. 42
77. 43
78. 44
79. 45
80. 46
81. 47
82. 48
83. 49
84. 50
85. 51
86. 52
87. 53
88. 54
89. 55
90. 56
91. 57
92. 58
93. 59
94. 60
95. 61
96. 62
97. 63
98. 64

99. 65
100. 66
101. 67
102. 68
103. 69
104. 70
105. 71
106. 72
107. 73
108. 74
109. 75
110. 76
111. 77
112. 78
113. 79
114. 80
115. 81
116. 82
117. 83
118. 84
119. 85
120. 86
121. 87
122. 88
123. 89
124. 90
125. 91
126. 92
127. 93
128. 94
129. 95
130. 96
131. 97
132. 98
133. 99
134. 100
135. 101
136. 102
137. 103
138. 104
139. 105
140. 106
141. 107
142. 108
143. 109
144. 110
145. 111
146. 112
147. 113
148. 114
149. 115
150. 116
151. 117
152. 118

153. 119
154. 120
155. 121
156. 122
157. 123
158. 124
159. 125
160. 126
161. 127
162. 128
163. 129
164. 130
165. 131
166. 132
167. 133
168. 134
169. 135
170. 136
171. 137
172. 138
173. 139
174. 140
175. 141
176. 142
177. 143
178. 144
179. 145
180. 146
181. 147
182. 148
183. 149
184. 150
185. 151
186. 152
187. 153
188. 154
189. 155
190. 156
191. 157
192. 158
193. 159
194. 160
195. 161
196. 162
197. 163
198. 164
199. 165
200. 166
201. 167
202. 168
203. 169
204. 170
205. 171
206. 172

207. 173
208. 174
209. 175
210. 176
211. 177
212. 178
213. 179
214. 180
215. 181
216. 182
217. 183
218. 184
219. 185
220. 186
221. 187
222. 188
223. 189
224. 190
225. 191
226. 192
227. 193
228. 194
229. 195
230. 196
231. 197
232. 198
233. 199
234. 200
235. 201
236. 202
237. 203
238. 204
239. 205
240. 206
241. 207
242. 208
243. 209
244. 210
245. 211
246. 212
247. 213
248. 214
249. 215
250. 216
251. 217
252. 218
253. 219
254. 220
255. 221
256. 222
257. 223
258. 224
259. 225
260. 226

261. 227
262. 228
263. 229
264. 230
265. 231
266. 232
267. 233
268. 234
269. 235
270. 236
271. 237
272. 238
273. 239
274. 240
275. 241
276. 242
277. 243
278. 244
279. 245
280. 246
281. 247
282. 248
283. 249
284. 250
285. 251
286. 252
287. 253
288. 254
289. 255
290. 256
291. 257
292. 258
293. 259
294. 260
295. 261
296. 262
297. 263
298. 264
299. 265
300. 266
301. 267
302. 268
303. 269
304. 270
305. 271
306. 272
307. 273
308. 274
309. 275
310. 276
311. 277
312. 278
313. 279
314. 280

315. 281
316. 282
317. 283
318. 284
319. 285
320. 286
321. 287
322. 288
323. 289
324. 290
325. 291
326. 292
327. 293
328. 294
329. 295
330. 296
331. 297
332. 298
333. 299
334. 300
335. 301
336. 302
337. 303
338. 304
339. 305
340. 306
341. 307
342. 308
343. 309
344. 310
345. 311
346. 312
347. 313
348. 314
349. 315
350. 316
351. 317
352. 318
353. 319
354. 320
355. 321
356. 322
357. 323
358. 324
359. 325
360. 326
361. 327
362. 328
363. 329
364. 330
365. 331
366. 332
367. 333
368. 334

369. 335
370. 336
371. 337
372. 338
373. 339
374. 340
375. 341
376. 342
377. 343
378. 344
379. 345
380. 346
381. 347
382. 348
383. 349
384. 350
385. 351
386. 352
387. 353
388. 354
389. 355
390. 356
391. 357
392. 358
393. 359
394. 360
395. 361
396. 362
397. 363
398. 364
399. 365
400. 366
401. 367
402. 368
403. 369
404. 370
405. 371
406. 372
407. 373
408. 374
409. 375
410. 376
411. 377
412. 378
413. 379
414. 380
415. 381
416. 382
417. 383
418. 384
419. 385
420. 386
421. 387
422. 388

423. 389
424. 390
425. 391
426. 392
427. 393
428. 394
429. 395
430. 396
431. 397
432. 398
433. 399
434. 400
435. 401
436. 402
437. 403
438. 404
439. 405
440. 406
441. 407
442. 408
443. 409
444. 410
445. 411
446. 412
447. 413
448. 414
449. 415
450. 416
451. 417
452. 418
453. 419
454. 420
455. 421
456. 422
457. 423
458. 424
459. 425
460. 426
461. 427
462. 428
463. 429
464. 430
465. 431
466. 432
467. 433
468. 434
469. 435
470. 436
471. 437
472. 438
473. 439
474. 440
475. 441
476. 442

477. 443
478. 444
479. 445
480. 446
481. 447
482. 448
483. 449
484. 450
485. 451
486. 452
487. 453
488. 454
489. 455
490. 456
491. 457
492. 458
493. 459
494. 460
495. 461
496. 462
497. 463
498. 464
499. 465
500. 466
501. 467
502. 468
503. 469
504. 470
505. 471
506. 472
507. 473
508. 474
509. 475
510. 476
511. 477
512. 478
513. 479
514. 480
515. 481
516. 482
517. 483
518. 484
519. 485
520. 486
521. 487
522. 488
523. 489
524. 490
525. 491
526. 492
527. 493
528. 494
529. 495
530. 496

531. 497
532. 498
533. 499
534. 500
535. 501
536. 502
537. 503
538. 504
539. 505
540. 506
541. 507
542. 508
543. 509
544. 510
545. 511
546. 512
547. 513
548. 514
549. 515
550. 516
551. 517
552. 518
553. 519
554. 520
555. 521
556. 522
557. 523
558. 524
559. 525
560. 526
561. 527
562. 528
563. 529
564. 530
565. 531
566. 532
567. 533
568. 534
569. 535
570. 536
571. 537
572. 538
573. 539
574. 540
575. 541
576. 542
577. 543
578. 544
579. 545
580. 546
581. 547
582. 548
583. 549
584. 550

585. 551
586. 552
587. 553
588. 554
589. 555
590. 556
591. 557
592. 558
593. 559
594. 560
595. 561
596. 562
597. 563
598. 564
599. 565
600. 566
601. 567
602. 568
603. 569
604. 570
605. 571
606. 572
607. 573
608. 574
609. 575
610. 576
611. 577
612. 578
613. 579
614. 580
615. 581
616. 582
617. 583
618. 584
619. 585
620. 586
621. 587
622. 588
623. 589
624. 590
625. 591
626. 592
627. 593
628. 594
629. 595
630. 596
631. 597
632. 598
633. 599
634. 600
635. 601
636. 602
637. 603
638. 604

639. 605
640. 606
641. 607
642. 608
643. 609
644. 610
645. 611
646. 612
647. 613
648. 614
649. 615
650. 616
651. 617
652. 618
653. 619
654. 620
655. 621
656. 622
657. 623
658. 624
659. 625
660. 626
661. 627
662. 628
663. 629
664. 630
665. 631
666. 632
667. 633
668. 634
669. 635
670. 636
671. 637
672. 638

About This eBook

ePUB is an open, industry-standard format for eBooks. However, support of ePUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a "Click here to view code image" link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Cisco Certified DevNet Associate DEVASC 200-901

Official Cert Guide

**Chris Jackson, CCIEX2 (RS, SEC) [CCIE NO.
6256]**

**Jason Gooley, CCIEX2 (RS, SP) [CCIE NO.
38759]**

**Adrian Iliesiu, CCIE RS [CCIE NO. 43909]
Ashutosh Malegaonkar**

Cisco Press

Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide

Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar

Copyright© 2021 Cisco Systems, Inc.

Published by:
Cisco Press

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the publisher, except for the inclusion of brief quotations in a review.

ScoutAutomatedPrintCode

Library of Congress Control Number: 2020937218

ISBN-13: 978-01-3664296-1

ISBN-10: 01-3664296-9

Warning and Disclaimer

This book is designed to provide information about the Cisco DevNet Associate DEVASC 200-901 exam. Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied.

The information is provided on an “as is” basis. The authors, Cisco Press, and Cisco Systems, Inc. shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the discs or programs that may accompany it.

The opinions expressed in this book belong to the authors and are not necessarily those of Cisco Systems,

Inc.

Trademark Acknowledgments

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Cisco Press or Cisco Systems, Inc., cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Feedback Information

At Cisco Press, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers' feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us through email at feedback@ciscopress.com. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Editor-in-Chief: Mark Taub

Alliances Manager, Cisco Press: Arezou Gol

Director, ITP Project Management: Brett Bartow

Executive Editor: James Manly

Managing Editor: Sandra Schroeder

Development Editor: Ellie Bru

Technical Editors: Bryan Byrne, John McDonough

Project Editor: Lori Lyons

Copy Editor: Catherine D. Wilson

Editorial Assistant: Cindy Teeters

Cover Designer: Chuti Prasertsith

Production Manager: Vaishnavi Venkatesan,
codeMantra

Composition: codeMantra

Indexer: Ken Johnson

Proofreader: Donna Mulder



Americas Headquarters

Cisco Systems, Inc.
San Jose, CA

Asia Pacific Headquarters

Cisco Systems (USA) Pte. Ltd.
Singapore

Europe Headquarters

Cisco Systems International BV Amsterdam,
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at

www.cisco.com/go/offices.



Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

About the Authors

Chris Jackson, CCIE No. 6256 (R&S and SEC), is a Distinguished Architect and CTO for Global Sales Training at Cisco. Chris is focused on digital transformation and showing customers how to leverage the tremendous business value Cisco technologies can provide. He is the author of *Network Security Auditing* (Cisco Press, 2010), *CCNA Cloud CLDADM 210-455 Official Cert Guide* (Cisco Press, 2016), and various online video courses for Cisco Press. He holds dual CCIEs in security and routing and switching, CISA, CISSP, ITIL v3, seven SANS certifications, and a bachelor's degree in business administration. Residing in Franklin, Tennessee, Chris enjoys tinkering with electronics, robotics, and anything else that can be programmed to do his bidding. In addition, he is a 3rd Degree Black Belt in Taekwondo, rabid *Star Wars* fan, and has a ridiculous collection of Lego. His wife Piper and three children Caleb, Sydney, and Savannah are the true joy of his life and proof that not everything has to plug into a wall outlet to be fun.

Jason Gooley, CCIE No. 38759 (R&S and SP), is a very enthusiastic and spontaneous person who has more than 20 years of experience in the industry. Currently, Jason works as a Technical Evangelist for the Worldwide Enterprise Networking Sales team at Cisco Systems. Jason is very passionate about helping others in the industry succeed. In addition to being a Cisco Press author, Jason is a distinguished speaker at Cisco Live, contributes to the development of the Cisco CCIE and DevNet exams, provides training for Learning@Cisco, is an active CCIE mentor, is a committee member for the Cisco Continuing Education Program (CE), and is a program committee member of the Chicago Network Operators Group (CHI-NOG), www.chinog.org. Jason

also hosts a show called “MetalDevOps.” Jason can be found at www.MetalDevOps.com, @MetalDevOps, and @Jason_Gooley on all social media platforms.

Adrian Iliesiu, CCIE No. 43909 (R&S), is a network engineer at heart with more than 15 years of professional IT experience. Currently, Adrian works as a Technical Leader with the Cisco DevNet Co-Creations team. During his career, Adrian has worked in several roles, including team leader and network, systems, and QA engineer across multiple industries and international organizations. When not working on innovative projects with customers and partners, Adrian advocates the advantages of network programmability and automation with a focus on enterprise and data center infrastructure. He is an established blog author, distinguished speaker at Cisco Live, and a recipient of the coveted Cisco Pioneer award. Adrian also appeared on Cisco TechWiseTV, Cisco Champion podcasts, and DevNet webinars. He holds a bachelor’s degree in Electronics and Telecommunications from Technical University of Cluj-Napoca and a master’s degree in Telecommunication Networks from Politehnica University of Bucharest.

Ashutosh Malegaonkar is a Cisco Distinguished Engineer, a senior technical contributor, and an industry thought leader. His experience spans across different technology domains: ISR Platforms, Voice, Video, Search, Video Analytics, and Cloud. Over two decades at Cisco, he has done two startups and has won several accolades, including the Pioneer awards. He has delivered several keynotes and talks at Cisco Connect and Cisco Live. He has also been a Tech Field Day Speaker. With more than 25 years of professional experience, he currently leads the DevNet Co-Creations team whose mission is to co-create, innovate, and inspire alongside our strategic customers, partners, and developers. Ashutosh inspires those around him to

innovate, and he is continually developing creative new ways to use software and Cisco APIs to solve real problems for our customers. He has a deep understanding of the breadth of Cisco products and technologies and where they can best be applied to serve our customers. Ashutosh has 16 approved patents and two publications.

About the Technical Reviewers

Bryan Byrne, CCIE No. 25607 (R&S), is a Technical Solutions Architect in Cisco's Global Enterprise segment. With more than 20 years of data networking experience, his current focus is helping his customers transition from traditional LAN/WAN deployments toward Cisco's next-generation Software-Defined network solutions. Prior to joining Cisco, Bryan spent the first 13 years of his career in an operations role with a global service provider supporting large-scale IP DMVPN and MPLS networks. Bryan is multi-time Cisco Live Distinguished Speaker covering topics on NETCONF, RESTCONF, and YANG. He is a proud graduate of The Ohio State University and currently lives outside Columbus, Ohio, with his wife Lindsey and their two children Evan and Kaitlin.

John McDonough has more than 30 years of development experience and is currently a Developer Advocate for Cisco's DevNet. As a Developer Advocate, John writes code and creates DevNet Learning Labs about how to write code. He writes blogs about writing code and presents at Cisco Live, SXSW, AnsibleFest, and other industry events. John focuses on Cisco's Computing Systems Products, Cisco UCS, and Cisco Intersight. John's career at Cisco has varied from Product Engineer to Custom Application Developer, Technical Marketing Engineer, and now a Developer Advocate.

Dedications

Chris Jackson:

Writing a book is a solitary effort, but the real work is shared by everyone who loves and supports you. This book is just the latest project that my beautiful wife Piper has provided infinite patience, love, and understanding as I wrote late into the evening and on weekends. She is my rock, my light, and my greatest cheerleader. My life is amazing because of her and her love. My children Caleb, Sydney, and Savannah have been so forgiving of my time commitments and allowed me to focus on delivering something I could be proud of. Each of you are so wonderful and the time away from you has been a sacrifice that I do not make lightly. Now it's time to celebrate! Last, but certainly not least, are all my friends and co-workers who take up the slack when my inability to say no to new projects rears its head. They drive me to be better, and I am fortunate to work with some of the most professional and high-quality individuals in the industry.

Jason Gooley:

This book is dedicated to my wife, Jamie, and my children, Kaleigh and Jaxon. Without their support, these books would not be possible. I can't believe they let me write four books in one year! To my father and brother, thank you for always supporting me. In addition, I want to thank my extended family and friends for all the unwavering support over the years. It is because of all of you that I get to do these things! Huge thank-you to Thom Hazaert, Melody Myers, and David Ellefson for supporting and believing in MetalDevOps! Thank you for giving us a home at EMP Label Group, Combat Records, and Ellefson Coffee Co.! Can't wait to see what the future holds for us!

Adrian Iliesiu:

I dedicate this book to my family and especially my wife, Martina. This book wouldn't have been possible without her continuous support through so many challenges and sacrifices over the years. I am grateful she agreed to let me write this book, especially after the CCIE exam preparation "experience." I promise I'll be back at doing dishes for the foreseeable future. Special thank-you to my parents, Grigore and Ana, for all their support and encouragement through the years. Vă **mulțumesc** pentru tot, mamă **Și** tată! Big thank-you to my sister, and especially my grandmother, for shaping and instilling in me a set of values that has made me the person I am today. Thanks also to Susie Wee for her continuous support and leadership.

Ashutosh Malegaonkar:

I want to dedicate this book to my Guru, Shri. Gondavlekar Maharaj, for giving me this opportunity and letting me follow through with this opportunity.

I would also like to dedicate this book to my wife Medha. She has been my strength and biggest supporter. It is because of some of her sacrifices that I am where I am today. Our sons, Jai and Yash, and their constant positivity keep making me feel special in whatever I do.

I would like to thank my Mom (she would have been proud), Dad, my brother, and my sister, for shaping me during the years.

Last but not the least, I sincerely thank Susie Wee for believing in me and letting me be part of DevNet since the very early days of DevNet.

Acknowledgments

Chris Jackson:

This book would not have been written if it hadn't been for the team of amazing people at Cisco Press; you guys make us sound coherent, fix our silly mistakes, and encourage us to get the project done! James, Ellie, and Brett are the best in the industry. Thanks as well to our tech editors, John McDonough and Bryan Byrne, for making sure our code is tight and works.

I am so very thankful to my manager Jeff Cristee for being an incredible mentor and supporting me in so many ways. You are the best, and I feel blessed to work with you. Linda Masloske, you are an amazing friend and have been one of my biggest supporters during my 20-year career at Cisco. I could write an entire chapter on how much you have done for me over the years. Thank you for everything, but most importantly for giving a kid from Kentucky the chance to shine.

A big Thanks to my SNAP crew Jodi, Deonna, Angie, and Doug, for giving me time to work on all of my many projects. You guys are the best and I LOVE working with you. Virtual or live, you bring the magic.

Jason Gooley:

Big thank-you to Brett and Marianne Bartow, Ellie Bru, and everyone else involved at Cisco Press! You are all amazing to work with, and six books later, I'm not sure how you put up with me! Shout out to my brother in metal, Stuart Clark (@bigevilbeard), for letting me use his code examples! Thanks, brother!

Adrian Iliesiu:

Huge thank-you to Casey Tong, designer in chief, for all her help with images and graphics for this book. Big thank-you to Ashutosh for all his support. Thanks to Chris and Jason for allowing me to embark on this journey with them; Ellie Bru and James Manly from Cisco Press for editing and trying to keep the project on track; and to John and Bryan for their feedback and insight. I would also like to thank Mike Mackay for believing in me when it mattered and for giving *me* a chance to prove myself.

Ashutosh Malegaonkar:

Thanks to the entire Cisco DevNet team for being the soul of the program. Adrian—we did it! A special thanks to Susie Wee for the support and encouragement from day one. This being the first for me, thanks to Jason and Chris for the mentorship; Ellie Bru for keeping up with my novice questions; and finally John McDonough and Bryan Byrne for the excellent technical reviews.

Contents at a Glance

[Introduction](#)

[Chapter 1 Introduction to Cisco DevNet Associate Certification](#)

[Chapter 2 Software Development and Design](#)

[Chapter 3 Introduction to Python](#)

[Chapter 4 Python Functions, Classes, and Modules](#)

[Chapter 5 Working with Data in Python](#)

[Chapter 6 Application Programming Interfaces \(APIs\)](#)

[Chapter 7 RESTful API Requests and Responses](#)

[Chapter 8 Cisco Enterprise Networking Management Platforms and APIs](#)

[Chapter 9 Cisco Data Center and Compute Management Platforms and APIs](#)

[Chapter 10 Cisco Collaboration Platforms and APIs](#)

[Chapter 11 Cisco Security Platforms and APIs](#)

[Chapter 12 Model-Driven Programmability](#)

[Chapter 13 Deploying Applications](#)

[Chapter 14 Application Security](#)

[Chapter 15 Infrastructure Automation](#)

[Chapter 16 Network Fundamentals](#)

[Chapter 17 Networking Components](#)

[Chapter 18 IP Services](#)

[Chapter 19 Final Preparation](#)

[Appendix A Answers to the “Do I Know This Already?” Quiz Questions](#)

[Appendix B *DevNet Associate DEVASC 200-901 Official Cert Guide* Exam Updates](#)

[Glossary](#)

Index

Online Elements

Appendix C Study Planner

Glossary

Reader Services

Other Features

In addition to the features in each of the core chapters, this book has additional study resources on the companion website, including the following:

Practice exams: The companion website contains an exam engine that enables you to review practice exam questions. Use these to prepare with a sample exam and to pinpoint topics where you need more study.

Flash Cards: An online interactive application to help you drill on Key Terms by chapter.

Glossary quizzes: The companion website contains interactive quizzes that enable you to test yourself on every glossary term in the book.

Video training: The companion website contains unique test-prep videos.

To access this additional content, simply register your product. To start the registration process, go to www.ciscopress.com/register and log in or create an account.* Enter the product ISBN 9780136642961 and click Submit. After the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us to receive exclusive discounts on future editions of this product.

Contents

Introduction

Chapter 1 Introduction to Cisco DevNet Associate Certification

Do I Know This Already?

Foundation Topics

Why Get Certified

Cisco Career Certification Overview

Cisco DevNet Certifications

Cisco Certified DevNet Associate
Certification (DEVASC)

Cisco Certified DevNet Professional
Certification

Cisco DevNet Overview

Discover

Technologies

Community

Support

Events

DevNet Automation Exchange

Summary

Chapter 2 Software Development and Design

“Do I Know This Already?” Quiz

Foundation Topics

Software Development Lifecycle

Waterfall

Lean

Agile

Common Design Patterns

Model-View-Controller (MVC) Pattern

Observer Pattern

Linux BASH

Getting to Know BASH

Directory Navigation

cd

pwd

ls

mkdir

File Management

cp

mv

rm

touch

cat

Environment Variables

Software Version Control

Git

Understanding Git

Using Git

Cloning/Initiating Repositories

Adding and Removing Files

Committing Files

Pushing and Pulling Files

Working with Branches

Merging Branches

Handling Conflicts

Comparing Commits with diff

Conducting Code Review

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 3 Introduction to Python

“Do I Know This Already?” Quiz

Foundation Topics

Getting Started with Python

Understanding Python Syntax

Data Types and Variables

Variables

Data Types

*Integers, Floating Point, and Complex
Numbers*

Booleans

Strings

Lists

Tuples

Dictionaries

Sets

Input and Output

Getting Input from the User

The Mighty print() Function

Flow Control with Conditionals and Loops

If Statements

For Loops

While Loops

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Additional Resources

Chapter 4 Python Functions, Classes, and Modules

“Do I Know This Already?” Quiz

Foundation Topics

Python Functions

Using Arguments and Parameters

Object-Oriented Programming and Python

Python Classes

Creating a Class

Methods

Inheritance

Working with Python Modules

Importing a Module

The Python Standard Library

Importing Your Own Modules

Useful Python Modules for Cisco

Infrastructure

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 5 Working with Data in Python

“Do I Know This Already?” Quiz

Foundation Topics

File Input and Output

Parsing Data

Comma-Separated Values (CSV)

JavaScript Object Notation (JSON)

Extensible Markup Language (XML)

YAML Ain't Markup Language (YAML)

Error Handling in Python

Test-Driven Development

Unit Testing

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Additional Resources

Chapter 6 Application Programming Interfaces (APIs)

“Do I Know This Already?” Quiz

Foundation Topics

Application Programming Interfaces (APIs)

Northbound APIs

Southbound APIs

Synchronous Versus Asynchronous APIs

Representational State Transfer (REST)
APIs

RESTful API Authentication

Basic Authentication

API Keys

Custom Tokens

Simple Object Access Protocol (SOAP)

Remote-Procedure Calls (RPCs)

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 7 RESTful API Requests and Responses

“Do I Know This Already?” Quiz

Foundation Topics

RESTful API Fundamentals

API Types

API Access Types

HTTP Basics

Uniform Resource Locator (URL)

Method

REST Methods and CRUD

Deep Dive into GET and POST

HTTP Headers

Request Headers

Response Headers

Response Codes

XML

JSON

YAML

Webhooks

Tools Used When Developing with
Webhooks

Sequence Diagrams

REST Constraints

Client/Server

Stateless

Cache

Uniform Interface

Layered System

Code on Demand

REST API Versioning

Pagination

Rate Limiting and Monetization

Rate Limiting on the Client Side

REST Tools

Postman

curl

HTTPIe

Python Requests

REST API Debugging Tools for Developing
APIs

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 8 Cisco Enterprise Networking Management Platforms and APIs

“Do I Know This Already?” Quiz

Foundation Topics

What Is an SDK?

Cisco Meraki

Cisco DNA Center

Cisco SD-WAN

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 9 Cisco Data Center and Compute Management Platforms and APIs

“Do I Know This Already?” Quiz

Foundation Topics

Cisco ACI

Building Blocks of Cisco ACI Fabric Policies

APIC REST API

UCS Manager

Cisco UCS Director

Cisco Intersight

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 10 Cisco Collaboration Platforms and APIs

“Do I Know This Already?” Quiz

Foundation Topics

Introduction to the Cisco Collaboration

Portfolio

Unified Communications

Cisco Webex Teams

*Cisco Unified Communications Manager
(Unified CM)*

Unified Contact Center

Cisco Webex

Cisco Collaboration Endpoints

API Options in the Cisco Collaboration
Portfolio

Webex Teams API

API Authentication

Personal Access Tokens

Integrations

Bots

Guest Issuer

Webex Teams SDKs

Cisco Finesse

Cisco Finesse API

API Authentication

[Finesse User APIs](#)

[Finesse Team APIs](#)

[Dialog APIs](#)

[Finesse Gadgets](#)

[Webex Meetings APIs](#)

[Authentication](#)

[Integration API Keys](#)

[Webex XML APIs](#)

[*Creating a New Meeting*](#)

[*Listing All My Meetings Meeting*](#)

[*Setting or Modifying Meeting Attributes*](#)

[*Deleting a Meeting*](#)

[Webex Devices](#)

[xAPI](#)

[xAPI Authentication](#)

[xAPI Session Authentication](#)

[*Creating a Session*](#)

[*Getting the Current Device Status*](#)

[*Setting Device Attributes*](#)

[*Registering an Event Notification*](#)

[*Webhook*](#)

[Room Analytics People Presence Detector](#)

[Cisco Unified Communications Manager](#)

[Administrative XML](#)

[Cisco AXL Toolkit](#)

[Accessing the AXL SOAP API](#)

[*Using the Zeep Client Library*](#)

[*Using the CiscoAXL SDK*](#)

[Exam Preparation Tasks](#)

Review All Key Topics

Define Key Terms

Chapter 11 Cisco Security Platforms and APIs

“Do I Know This Already?” Quiz

Foundation Topics

Cisco’s Security Portfolio

Potential Threats and Vulnerabilities

Most Common Threats

Cisco Umbrella

Understanding Umbrella

Cisco Umbrella APIs

Authentication

Cisco Firepower

Firepower Management Center APIs

Cisco Advanced Malware Protection (AMP)

Listing All Computers

Listing All Vulnerabilities

Cisco Identity Services Engine (ISE)

ISE REST APIs

ERS API Authentication

Creating an Endpoint Group

Creating an Endpoint and Adding It to a Group

Other ISE APIs

Cisco Threat Grid

Threat Grid APIs

Threat Grid API Format

API Keys

Who Am I

The Data, Sample, and IOC APIs

Feeds

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 12 Model-Driven Programmability

“Do I Know This Already?” Quiz

Foundation Topics

NETCONF

YANG

RESTCONF

Model-Driven Telemetry

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 13 Deploying Applications

“Do I Know This Already?” Quiz

Foundation Topics

Application Deployment Models

NIST Definition

Essential Characteristics

Service Models

Application Deployment Options

Private Cloud

Public Cloud

Hybrid Cloud

Community Cloud

Edge and Fog Computing

Application Deployment Methods

Bare-Metal Application Deployment

Virtualized Applications

Cloud-Native Applications

Containerized Applications

Serverless

DevOps

What Is DevOps?

Putting DevOps into Practice: The Three
Ways

First Way: Systems and Flow

Second Way: Feedback Loop

Third Way: Continuous Experimentation
and Learning

DevOps Implementation

Docker

Understanding Docker

Namespaces

Cgroups

Union File System

Docker Architecture

Using Docker

Working with Containers

Dockerfiles

Docker Images

Docker Hub

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Additional Resources

Chapter 14 Application Security

“Do I Know This Already?” Quiz

Foundation Topics

Identifying Potential Risks

Common Threats and Mitigations

Open Web Application Security Project

Using Nmap for Vulnerability Scanning

Basic Nmap Scan Against an IP Address
or a Host

CVE Detection Using Nmap

Protecting Applications

Tiers of Securing and Protecting

Encryption Fundamentals

Public Key Encryption

Data Integrity (One-Way Hash)

Digital Signatures

Data Security

Secure Development Methods

Securing Network Devices

Firewalls

Intrusion Detection Systems (IDSs)

Intrusion Prevention Systems (IPSs)

Domain Name System (DNS)

Load Balancing

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 15 Infrastructure Automation

“Do I Know This Already?” Quiz

Foundation Topics

Controller Versus Device-Level Management

Infrastructure as Code

Continuous Integration/Continuous Delivery
Pipelines

Automation Tools

Ansible

Puppet

Chef

Cisco Network Services Orchestrator (NSO)

Cisco Modeling Labs/Cisco Virtual
Internet Routing Laboratory
(CML/VIRL)

Python Automated Test System (pyATS)

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 16 Network Fundamentals

“Do I Know This Already?” Quiz

Foundation Topics

Network Reference Models

The OSI Model

The TCP/IP Model

Switching Concepts

Ethernet

MAC Addresses

Virtual Local-Area Networks (VLANs)

Switching

Routing Concepts

IPv4 Addresses

IPv6 Addresses

Routing

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 17 Networking Components

“Do I Know This Already?” Quiz

Foundation Topics

What Are Networks?

Elements of Networks

Hubs

Bridges

Switches

Virtual Local Area Networks (VLANs)

Routers

Routing in Software

Functions of a Router

Network Diagrams: Bringing It All
Together

Software-Defined Networking

SDN Controllers

Cisco Software-Defined Networking
(SDN)

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 18 IP Services

“Do I Know This Already?” Quiz

Foundation Topics

Common Networking Protocols

Dynamic Host Configuration Protocol
(DHCP)

Server Discovery

Lease Offer

Lease Request

Lease Acknowledgment

Releasing

Domain Name System (DNS)

Network Address Translation (NAT)

Simple Network Management Protocol
(SNMP)

Network Time Protocol (NTP)

Layer 2 Versus Layer 3 Network Diagrams

Troubleshooting Application Connectivity
Issues

Exam Preparation Tasks

Review All Key Topics

Define Key Terms

Chapter 19 Final Preparation

Getting Ready

Tools for Final Preparation

Pearson Cert Practice Test Engine and
Questions on the Website

*Accessing the Pearson Test Prep Software
Online*

*Accessing the Pearson Test Prep Software
Offline*

Customizing Your Exams

Updating Your Exams

Premium Edition

Chapter-Ending Review Tools

Suggested Plan for Final Review/Study

Summary

Appendix A Answers to the “Do I Know This Already?”

Quiz Questions

Appendix B *DevNet Associate DEVASC 200-901*

Official Cert Guide Exam Updates

Glossary

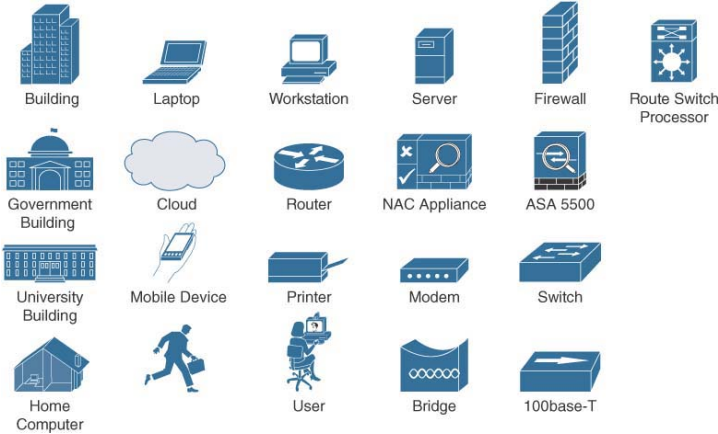
Index

Online Elements

Appendix C Study Planner

Glossary

Icons Used in This Book



Command Syntax Conventions

The conventions used to present command syntax in this book are the same conventions used in the IOS Command Reference. The Command Reference describes these conventions as follows:

- **Boldface** indicates commands and keywords that are entered literally as shown. In actual configuration examples and output (not general command syntax), boldface indicates commands that are manually input by the user (such as a **show** command).
- *Italic* indicates arguments for which you supply actual values.
- Vertical bars (|) separate alternative, mutually exclusive elements.
- Square brackets ([]) indicate an optional element.
- Braces ({}) indicate a required choice.
- Braces within brackets ([{}]) indicate a required choice within an optional element.

Introduction

This book was written to help candidates improve their network programmability and automation skills—not only for preparation of the DevNet Associate DEVASC 200-901 exam, but also for real-world skills for any production environment.

Readers of this book can expect that the blueprint for the DevNet Associate DEVASC 200-901 exam tightly aligns with the topics contained in this book. This was by design. Candidates can follow along with the examples in this book by utilizing the tools and resources found on the DevNet website and other free utilities such as Postman and Python.

This book is targeted for all learners who are learning these topics for the first time, as well as for those who wish to enhance their network programmability and automation skillset.

Be sure to visit www.cisco.com to find the latest information on DevNet Associate DEVASC 200-901 exam requirements and to keep up to date on any new exams that are announced.

GOALS AND METHODS

The most important and somewhat obvious goal of this book is to help you pass the DevNet Associate DEVASC (200-901) exam. Additionally, the methods used in this book to help you pass the DevNet Associate exam are designed to also make you much more knowledgeable about how to do your job. While this book and the companion website together have more than enough questions to help you prepare for the actual exam, the method in which they are used is not to simply make you memorize as many questions and answers as you possibly can.

One key methodology used in this book is to help you discover the exam topics that you need to review in more depth, to help you fully understand and remember those details, and to help you prove to yourself that you have retained your knowledge of those topics. So, this book does not try to help you pass by memorization but helps you truly learn and understand the topics. The DevNet Associate exam is just one of the foundation exams in the DevNet certification suite, and the knowledge contained within is vitally important to consider yourself a truly skilled network developer. This book would do you a disservice if it didn't attempt to help you learn the material. To that end, the book will help you pass the DevNet Associate exam by using the following methods:

- Helping you discover which test topics you have not mastered
- Providing explanations and information to fill in your knowledge gaps
- Supplying exercises and scenarios that enhance your ability to recall and deduce the answers to test questions

WHO SHOULD READ THIS BOOK?

This book is intended to help candidates prepare for the DevNet Associate DEVASC 200-901 exam. Not only can this book help you pass the exam, but also it can help you learn the necessary topics to provide value to your organization as a network developer.

Passing the DevNet Associate DEVASC 200-901 exam is a milestone toward becoming a better network developer. This in turn can help with becoming more confident with these technologies.

STRATEGIES FOR EXAM PREPARATION

The strategy you use for the DevNet Associate exam might be slightly different than strategies used by other readers, mainly based on the skills, knowledge, and experience you already have obtained.

Regardless of the strategy you use or the background you have, this book is designed to help you get to the point where you can pass the exam with the least amount of time required. However, many people like to make sure that they truly know a topic and thus read over material that they already know. Several book features will help you gain the confidence that you know some material already and to also help you know what topics you need to study more.

THE COMPANION WEBSITE FOR ONLINE CONTENT REVIEW

All the electronic review elements, as well as other electronic components of the book, are provided on this book's companion website.

How to Access the Companion Website

To access the companion website, start by establishing a login at www.ciscopress.com and registering your book. To do so, simply go to www.ciscopress.com/register and enter the ISBN of the print book: 9780136642961. After you have registered your book, go to your account page and click the Registered Products tab. From there, click the Access Bonus Content link to get access to the book's companion website.

Note that if you buy the Premium Edition eBook and Practice Test version of this book from Cisco Press, your book will automatically be registered on your account page. Simply go to your account page, click the Registered Products tab, and select Access Bonus Content to access the book's companion website.

How to Access the Pearson Test Prep (PTP) App

You have two options for installing and using the Pearson Test Prep application: a web app and a desktop app. To use the Pearson Test Prep application, start by

finding the registration code that comes with the book. You can find the code in these ways:

- **Print book:** Look in the cardboard sleeve in the back of the book for a piece of paper with your book's unique PTP code.
- **Premium Edition:** If you purchase the Premium Edition eBook and Practice Test directly from the Cisco Press website, the code will be populated on your account page after purchase. Just log in at www.ciscopress.com, click **Account** to see details of your account, and click the **Digital Purchases** tab.
- **Amazon Kindle:** For those who purchase a Kindle edition from Amazon, the access code will be supplied directly from Amazon.
- **Other Bookseller eBooks:** Note that if you purchase an eBook version from any other source, the practice test is not included because other vendors to date have not chosen to vend the required unique access code.

Note

Do not lose the activation code because it is the only means with which you can access the QA content with the book.

Once you have the access code, to find instructions about both the PTP web app and the desktop app, follow these steps:

- Step 1.** Open this book's companion website, as shown earlier in this Introduction under the heading "[How to Access the Companion Website.](#)"
- Step 2.** Click the Practice Exams button.
- Step 3.** Follow the instructions listed there both for installing the desktop app and for using the web app.

If you want to use the web app only at this point, just navigate to www.pearsonestprep.com, establish a free login if you do not already have one, and register this book's practice tests using the registration code you just found. The process should take only a couple of minutes.

Note

Amazon eBook (Kindle) customers: It is easy to miss Amazon's email that lists your PTP access code. Soon after you purchase the Kindle eBook, Amazon should send an email. However, the email uses very generic text and makes no specific mention of PTP or practice exams. To find your code, read every email from Amazon after you purchase the book. Also do the usual checks for ensuring your email arrives, like checking your spam folder.

Note

Other eBook customers: As of the time of publication, only the publisher and Amazon supply PTP access codes when you purchase their eBook editions of this book.

HOW THIS BOOK IS ORGANIZED

Although this book can be read cover-to-cover, it is designed to be flexible and allow you to easily move between chapters and sections of chapters to cover just the material that you need more work with. Chapter 1 provides an overview of the Cisco career certifications and offers some strategies for how to prepare for the exams. The dichotomy of that is in Chapters 2 through 18. These chapters are the core chapters and can be covered in any order. If you do intend to read them all, the order in the book is an excellent sequence to use.

The core chapters, Chapters 2 through 18, cover the following topics:

- **Chapter 2, “Software Development and Design”**: This chapter introduces key software development methods, like Waterfall and Agile, and includes the common design patterns MVC and Observer. Software version control systems, how to use Git, and how to conduct code reviews are covered as well.
- **Chapter 3, “Introduction to Python”**: This chapter provides an overview of Python syntax, working with various data types, getting input and producing output, and how to use conditionals and loops to control program flow.

- **Chapter 4, “Python Functions, Classes, and Modules”**: This chapter introduces Python functions and Object-Oriented Programming techniques. In addition, it also covers Python classes and how to work with modules to extend Python capabilities.
- **Chapter 5, “Working with Data in Python”**: This chapter covers the various ways you can input data into your Python program, parse data, and handle errors. Finally, test-driven development is introduced as well as how to perform unit tests.
- **Chapter 6, “Application Programming Interfaces (APIs)”**: This chapter covers a high-level overview of some common API types, REST API Authentication, Simple Object Access Protocol (SOAP), and Remote-Procedure Call (RPC) protocol as well as common examples of when and where each protocol is used.
- **Chapter 7, “RESTful API Requests and Responses”**: This chapter presents a detailed overview of REST APIs. It discusses several aspects of REST APIs including URL, methods, headers, return codes, data formats, architectural constraints, and various tools used for working with REST APIs.
- **Chapter 8, “Cisco Enterprise Networking Management Platforms and APIs”**: This chapter starts with what SDKs are and then covers Cisco Enterprise Networking Platforms and their APIs, including examples of how to interact with the APIs. The platforms covered in this chapter are Cisco Meraki, Cisco DNA Center, and Cisco SD-WAN.
- **Chapter 9, “Cisco Data Center and Compute Management Platforms and APIs”**: This chapter introduces key Cisco Data Center and Compute Management Platforms and their associated APIs. The following platforms are covered in this chapter: Cisco ACI, Cisco UCS Manager, Cisco UCS Director, and Cisco Intersight. Examples of API consumption for all these platforms are also included in this chapter.
- **Chapter 10, “Cisco Collaboration Platforms and APIs”**: This chapter discusses in detail Cisco’s Collaboration platforms and their associated APIs, along with examples. Platforms under consideration are Webex Teams, Cisco Finesse, Webex Meetings, Webex Devices, and Cisco Unified Call Manager.
- **Chapter 11, “Cisco Security Platforms and APIs”**: This chapter discusses in detail Cisco’s Security platforms, their associated APIs along with examples. Platforms under consideration are Cisco Firepower, Cisco Umbrella, Cisco Advanced Malware Protection—AMP, Cisco Identity Services Engine—ISE, and Cisco ThreatGrid.
- **Chapter 12, “Model-Driven Programmability”**: This chapter introduces key model-driven programmability concepts and protocols. An in-depth look at YANG, YANG data models, NETCONF, RESTCONF, and Model-Driven telemetry is covered in this chapter.
- **Chapter 13, “Deploying Applications”**: This chapter covers numerous application deployment models and methods. It also introduces the core concepts of DevOps as well as an introduction to Docker and how to use it.

- **Chapter 14, “Application Security”**: This chapter introduces application security issues, the methods of how to secure applications via modern networking components, and various tools used. Additionally, this chapter also discusses the Open Web Application Security Project (OWASP)’s top ten.
- **Chapter 15, “Infrastructure Automation”**: This chapter introduces several infrastructure automation concepts including controller versus device-level management, infrastructure as code, continuous integration/continuous delivery pipelines, and automation tools such as Ansible, Puppet, and Chef. An overview of Cisco-related products such as Cisco NSO, Cisco VIRT, and pyATS is also presented.
- **Chapter 16, “Network Fundamentals”**: This chapter presents several key networking concepts including networking reference models, switching, and routing concepts. OSI and TCP/IP reference models, Ethernet, MAC addresses, and VLANs—as well as IPv4 and IPv6 addressing concepts—are discussed in this chapter.
- **Chapter 17, “Networking Components”**: This chapter introduces some basic networking concepts, including network definitions, types, and elements such as hubs, switches, and routers. Further, it presents and differentiates between process, fast, and CEF switching. It also introduces software-defined networking discussing management, data, and control planes.
- **Chapter 18, “IP Services”**: This chapter starts by covering several protocols and technologies that are critical to networking: DHCP, DNS, NAT, SNMP, and NTP. The chapter continues with an overview of Layer 2 versus Layer 3 network diagrams and ends with a look at how to troubleshoot application connectivity issues.

CERTIFICATION EXAM TOPICS AND THIS BOOK

The questions for each certification exam are a closely guarded secret. However, we do know which topics you must know to *successfully* complete this exam. Cisco publishes them as an exam blueprint for DevNet Associate DEVASC 200-901 exam. [Table I-1](#) lists each exam topic listed in the blueprint along with a reference to the book chapter that covers the topic. These are the same topics you should be proficient in when working with network programmability and automation in the real world.

Table I-1 DEVASC Exam 200-901 Topics and Chapter References

--

DEVASC 200-901 Exam Topic	Chapter(s) In Which Topic Is Covered
1.0 Software Development and Design	2
1.1 Compare data formats (XML, JSON, YAML)	2
1.2 Describe parsing of common data formats (XML, JSON, YAML) to Python data structures	5
1.3 Describe the concepts of test-driven development	5
1.4 Compare software development methods (Agile, Lean, Waterfall)	2
1.5 Explain the benefits of organizing code into methods/ functions, classes, and modules	4
1.6 Identify the advantages of common design patterns (MVC and Observer)	2
1.7 Explain the advantages of version control	2
1.8 Utilize common version control operations with Git:	2
1.8.a Clone	2
1.8.b Add/remove	2
1.8.c Commit	2
1.8.d Push/pull	2
1.8.e Branch	2
1.8.f Merge and handling conflicts	2
1.8.g diff	2

2.0 Understanding and Using APIs	
2.1 Construct a REST API request to accomplish a task given API documentation	6
2.2 Describe common usage patterns related to webhooks	6, 7
2.3 Identify the constraints when consuming APIs	6
2.4 Explain common HTTP response codes associated with REST APIs	6
2.5 Troubleshoot a problem given the HTTP response code, request, and API documentation	6
2.6 Identify the parts of an HTTP response (response code, headers, body)	6
2.7 Utilize common API authentication mechanisms: basic, custom token, and API keys	6
2.8 Compare common API styles (REST, RPC, synchronous, and asynchronous)	6
2.9 Construct a Python script that calls a REST API using the requests library	7
3.0 Cisco Platforms and Development	
3.1 Construct a Python script that uses a Cisco SDK given SDK documentation	8, 9
3.2 Describe the capabilities of Cisco network management platforms and APIs (Meraki, Cisco DNA Center, ACI, Cisco SD-WAN, and NSO)	8, 9, 1, 5
3.3 Describe the capabilities of Cisco compute management platforms and APIs (UCS Manager, UCS Director, and Intersight)	9

3.4 Describe the capabilities of Cisco collaboration platforms and APIs (Webex Teams, Webex devices, Cisco Unified Communication Manager including AXL and UDS interfaces, and Finesse)	1 0
3.5 Describe the capabilities of Cisco security platforms and APIs (Firepower, Umbrella, AMP, ISE, and ThreatGrid)	11
3.6 Describe the device level APIs and dynamic interfaces for IOS XE and NX-OS	1 2
3.7 Identify the appropriate DevNet resource for a given scenario (Sandbox, Code Exchange, support, forums, Learning Labs, and API documentation)	7, 8, 9, 1 0, 11 , 1 2
3.8 Apply concepts of model driven programmability (YANG, RESTCONF, and NETCONF) in a Cisco environment	1 2
3.9 Construct code to perform a specific operation based on a set of requirements and given API reference documentation such as these:	8, 9, 1 5
3.9.a Obtain a list of network devices by using Meraki, Cisco DNA Center, ACI, Cisco SD-WAN, or NSO	8, 9, 1 5
3.9.b Manage spaces, participants, and messages in Webex Teams	1 0
3.9.c Obtain a list of clients/hosts seen on a network using Meraki or Cisco DNA Center	8
4.0 Application Deployment and Security	1 3

4.1 Describe benefits of edge computing	1 3
4.2 Identify attributes of different application deployment models (private cloud, public cloud, hybrid cloud, and edge)	1 3
4.3 Identify the attributes of these application deployment types:	1 3
4.3.a Virtual machines	1 3
4.3.b Bare metal	1 3
4.3.c Containers	1 3
4.4 Describe components for a CI/CD pipeline in application deployments	1 3, 1 5
4.5 Construct a Python unit test	5
4.6 Interpret contents of a Dockerfile	1 3
4.7 Utilize Docker images in local developer environment	1 3
4.8 Identify application security issues related to secret protection, encryption (storage and transport), and data handling	1 4
4.9 Explain how firewall, DNS, load balancers, and reverse proxy in application deployment	1 4
4.10 Describe top OWASP threats (such as XSS, SQL injections, and CSRF)	1 4
4.11 Utilize Bash commands (file management,	2

directory navigation, and environmental variables)	
4.12 Identify the principles of DevOps practices	1 3
5.0 Infrastructure and Automation	1 5
5.1 Describe the value of model driven programmability for infrastructure automation	1 2
5.2 Compare controller-level to device-level management	1 5
5.3 Describe the use and roles of network simulation and test tools (such as VIRL and pyATS)	1 5
5.4 Describe the components and benefits of CI/CD pipeline in infrastructure automation	1 3, 1 5
5.5 Describe principles of infrastructure as code	1 5
5.6 Describe the capabilities of automation tools such as Ansible, Puppet, Chef, and Cisco NSO	1 5
5.7 Identify the workflow being automated by a Python script that uses Cisco APIs including ACI, Meraki, Cisco DNA Center, or RESTCONF	8, 9
5.8 Identify the workflow being automated by an Ansible playbook (management packages, user management related to services, basic service configuration, and start/stop)	1 5
5.9 Identify the workflow being automated by a bash script (such as file management, app install, user management, directory navigation)	2
5.10 Interpret the results of a RESTCONF or NETCONF query	1 2

5.11 Interpret basic YANG models	1 2
5.12 Interpret a unified diff	2
5.13 Describe the principles and benefits of a code review process	2
5.14 Interpret sequence diagram that includes API calls	7
6.0 Network Fundamentals	
6.1 Describe the purpose and usage of MAC addresses and VLANs	1 6
6.2 Describe the purpose and usage of IP addresses, routes, subnet mask/prefix, and gateways	1 6
6.3 Describe the function of common networking components (such as switches, routers, firewalls, and load balancers)	1 6, 1 7
6.4 Interpret a basic network topology diagram with elements such as switches, routers, firewalls, load balancers, and port values	1 6, 1 7
6.5 Describe the function of management, data, and control planes in a network device	1 7
6.6 Describe the functionality of these IP services: DHCP, DNS, NAT, SNMP, NTP	1 8
6.7 Recognize common protocol port values (such as, SSH, Telnet, HTTP, HTTPS, and NETCONF)	1 2
6.8 Identify cause of application connectivity issues (NAT problem, Transport Port blocked, proxy, and VPN)	1 8

6.9 Explain the impacts of network constraints on applications	18
--	----

Each version of the exam can have topics that emphasize different functions or features, and some topics can be rather broad and generalized. The goal of this book is to provide the most comprehensive coverage to ensure that you are well prepared for the exam. Although some chapters might not address specific exam topics, they provide a foundation that is necessary for a clear understanding of important topics. Your short-term goal might be to pass this exam, but your long-term goal should be to become a qualified network developer.

It is also important to understand that this book is a “static” reference, whereas the exam topics are dynamic. Cisco can and does change the topics covered on certification exams often.

This exam guide should not be your only reference when preparing for the certification exam. You can find a wealth of information available at Cisco.com that covers each topic in great detail. If you think that you need more detailed information on a specific topic, read the Cisco documentation that focuses on that topic.

Note that as automation technologies continue to develop, Cisco reserves the right to change the exam topics without notice. Although you can refer to the list of exam topics in [Table I-1](#), always check Cisco.com to verify the actual list of topics to ensure that you are prepared before taking the exam. You can view the current exam topics on any current Cisco certification exam by visiting the Cisco.com website, choosing Menu, and Training & Events, then selecting from the Certifications list. Note also that, if needed, Cisco Press might post additional preparatory content on the web page associated with this book at

<http://www.ciscopress.com/title/9780136642961>. It's a good idea to check the website a couple of weeks before taking your exam to be sure that you have up-to-date content.

Figure Credits

Page No Selection Title Attribution		
	Cover image	Cisco Brand Exchange, Cisco Systems, Inc.
<u>1</u> <u>5</u> <u>7</u>	“YAML is a human-friendly data serialization standard for all programming languages.”	YAML Ain’t Markup Language, YAML
<u>1</u> <u>6</u> <u>5</u>	Figure 7-12: Postman: HTTP GET from the Postman Echo Server	Screenshot of Postman: HTTP GET from the Postman Echo Server ©2020 Postman, Inc.
<u>1</u> <u>6</u> <u>6</u>	Figure 7-13: Postman: HTTP POST to the Postman Echo Server	Screenshot of Postman: HTTP POST to the Postman Echo Server ©2020 Postman, Inc.
<u>1</u> <u>6</u> <u>6</u>	Figure 7-14: Postman Collection	Screenshot of Postman Collection ©2020 Postman, Inc.
<u>1</u> <u>6</u> <u>7</u>	Figure 7-15: Postman Automatic Code Generation	Screenshot of Postman Automatic Code Generation ©2020 Postman, Inc.
<u>1</u> <u>8</u> <u>9</u>	Figure 8-4: Output of the Python Script from Example 8-4	Screenshot of Output of the Python Script

		from Example 8-4 ©2020 Postman, Inc.
201	Figure 8-10 : Output of the Python Script from Example 8-7	Screenshot of Output of the Python Script from Example 8-7 ©2020 Postman, Inc.
211	Figure 8-15 : Output of the Python Script from Example 8-10	Screenshot of Output of the Python Script from Example 8-10 ©2020 Postman, Inc.
370	Figure 12-6 : Getting the REST API Root Resource	Screenshot of Getting the REST API root resource ©2020 Postman, Inc.
370	Figure 12-7 : Top-Level Resource Available in RESTCONF	Screenshot of Top level resource available in RESTCONF ©2020 Postman, Inc.
371	Figure 12-8 : Getting Interface Statistics with RESTCONF	Screenshot of Getting interface statistics with RESTCONF ©2020 Postman, Inc.
377	Figure 13-1 : NIST Cloud Computing Definition	Source: NIST Cloud Computing Definitions
378	Figure 13-2 : Cloud Service Models	Source: NIST Cloud Computing Service Models

3 7 9	Figure 13-3: Private Cloud	Source: NIST Special Publication 800-146 (May 2012)
3 8 0	Figure 13-4: Public Cloud	Source: NIST Special Publication 800-146 (May 2012)
3 8 0	Figure 13-5: Hybrid Cloud	Source: NIST Special Publication 800-146 (May 2012)
3 8 1	Figure 13-6: Community Cloud	Source: NIST Special Publication 800-146 (May 2012)
3 9 5	Figure 13-20: XebiaLabs Periodic Table of DevOps Tools	Source: https://xebialabs.com/periodic-table-of-devops-tools/
4 0 0	Figure 13-25: Docker Architecture	source: https://docs.docker.com/introduction/understanding-docker/
4 1 6	Figure 13-30: Kitematic	Screenshot of Kitematic © 2020 Docker Inc
4 2 2	Figure 14-1: NIST Cybersecurity Framework	NIST, CYBERSECURITY FRAMEWORK. U.S. Department of Commerce
5 1 2	“Information system(s) implemented with a collection of interconnected components. Such components may	NIST, COMPUTER SECURITY

	include routers, hubs, cabling, telecommunications controllers, key distribution centers, and technical control devices”	RESOURCE CENTER. NIST SP 800-53 Rev. 4 under Network (CNSSI 4009) CNSSI 4009-2015 (NIST SP 800-53 Rev. 4). U.S. Department of Commerce
5 2 9	“an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of applications. This architecture decouples the network control and forwarding functions, enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.”	Open Networking Foundation, Copyright © 2020

Chapter 1

Introduction to Cisco DevNet Associate Certification

This chapter covers the following topics:

- **Why Get Certified:** This section covers the benefits and advantages of becoming Cisco certified.
- **Cisco Career Certification Overview:** This section provides a high-level overview of the Cisco career certification portfolio.
- **Cisco DevNet Certifications:** This section covers various aspects of the Cisco Certified DevNet Associate, Professional, and Specialist certifications and how they fit into the overall Cisco career certification portfolio.
- **Cisco DevNet Overview:** This section provides an overview of DevNet, discusses the value DevNet provides to the industry, and covers the resources available and how to best leverage them.

DO I KNOW THIS ALREADY?

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 1-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 1-1 “Do I Know This Already?” Foundation Topics Section-to-Question Mapping

Foundations Topic	Section	Questions

Why Get Certified	2
Cisco Career Certification Overview	1, 4
Cisco DevNet Certifications	3
Cisco DevNet Overview	5

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following are levels of accreditation for Cisco certification? (Choose three.)

1. Associate
2. Entry
3. Authority
4. Expert
5. Maestro

2. What are some benefits certification provides for candidates? (Choose three.)

1. Highlights skills to employer
2. Increases confidence
3. Makes candidate appear smarter than peers
4. Reduces workload
5. Improves credibility

3. What type of exams are necessary to obtain DevNet Professional certification? (Choose two.)

1. Technology Core exam
2. Lab exam
3. CCT
4. Expert-level written exam
5. Concentration exam

4. True or false: In the new certification model, only a single exam is required to become CCNA certified.

1. True
2. False

5. Which of the following are part of DevNet? (Choose all that apply.)

1. Community
2. Technologies
3. Events
4. Cisco Automation Platform
5. Support

FOUNDATION TOPICS

WHY GET CERTIFIED

The IT industry is constantly changing and evolving. As time goes on, an ever-increasing number of technologies are putting strain on networks. New paradigms are formed as others fall out of favor. New advances are being developed and adopted in the networking realm. These advances provide faster innovation and the ability to adopt relevant technologies in a simplified way. We therefore need more intelligence and the capability to leverage the data from connected and distributed environments such as the campus, branch, data center, and WAN. Data is being used in interesting and more powerful ways than ever in the past. The following are some of the advances driving these outcomes:

- Artificial intelligence (AI)
- Machine learning (ML)
- Cloud services
- Virtualization
- Internet of Things (IoT)

The influx of these technologies is putting strain on IT operations staff, who are required to do more robust planning, find relevant use cases, and provide detailed adoption journey materials for easy consumption. All

these requirements are becoming critical to success. Another area of importance is the deployment and day-to-day operations of these technologies as well as how they fit within the network environment. Some of these technologies tend to disrupt typical operations and present challenges in terms of how they will be consumed by the business. Some advances in technology are being adopted to reduce cost of operations as well as complexity. It can be said that every network, to some degree, has inherent complexity. Having tools to help manage this burden is becoming a necessity.

Many in the industry are striving for automation to handle networks as they become more and more complicated. Businesses are often forced to operate with lean IT staffs and flat or shrinking budgets; they must struggle to find ways to increase the output of what the network can do for the business. Another driver for the adoption of these technologies is improving the overall user experience within the environment. Users often need the flexibility and capability to access any business-critical application from anywhere in the network and want to have an exceptional experience. In addition to trying to improve user experience, operations staff seek ways to simplify the operations of the network. There are many inherent risks associated with manually configuring networks. One risk is not being able to move fast enough when deploying new applications or services to the network. In addition, misconfigurations can cause outages or suboptimal network performance, which can impact business operations and potentially cause financial repercussions. Finally, there is risk in that a business relies on its network for business-critical services but those services might not be available due to the IT operations staff not being able to scale to keep up with the demands of the business.

According to a 2016 Cisco Technical Assistance Center (TAC) survey, 95% of Cisco customers are performing

configuration and deployment tasks manually in their networks. The survey also found that 70% of TAC cases created are related to misconfigurations. This means that typos or improperly used commands are the culprits in a majority of issues in the network environment. Dealing with such issues is where automation shines. Automation makes it possible to signify the intent of a change that needs to be made, such as deploying quality of service across the network, and then having the network configure it properly and automatically. Automation can configure services or features with great speed and is a tremendous value to a business. Simplifying operations while reducing human error can ultimately reduce risk and potentially lower complexity.

As a simple analogy, think of an automobile. The reason most people use an automobile is to meet a specific desired outcome: to get from point A to point B. An automobile is operated as a holistic system, not as a collection of parts that make up that system. For example, the dashboard provides the user all the necessary information about how the vehicle is operating and the current state of the vehicle. To use an automobile, a driver must take certain operational steps, such as putting it in gear and then using the system to get from point A to point B. [Figure 1-1](#) illustrates this analogy.



Figure 1-1 *Automobile as a System*

We can think of networks as systems much as we think of automobiles as systems. For over 30 years, the industry has thought of a network as a collection of devices such as routers, switches, and wireless components. The shift in mindset to look at a network as a holistic system is a

more recent concept that stems from the advent of network controllers, which split role and functionality from one another. This is often referred to as separating the control plane from the data plane. At a high level, the control plane is where all the instructions on a device live (for example, the routing protocols that exchange routing updates). The data plane is where all the user or data traffic flows (for example, the traffic between users on a network). Having a controller that sits on top of the rest of the devices makes it possible to operate the network as a whole from a centralized management point—much like operating an automobile from the driver’s seat rather than trying to manage the automobile from all the pieces and components of which it is composed. To put this in more familiar terms, think of the command-line interface (CLI). The CLI was not designed to make massive-scale configuration changes to multiple devices at the same time. Traditional methods of managing and maintaining the network aren’t sufficient to keep up with the pace and demands of the networks of today. Operations staff need to be able to move faster and simplify all the operations and configurations that have traditionally gone into networking. Software-defined networking (SDN) and controller capabilities are becoming areas of focus in the industry, and they are evolving to a point where they can address the challenges faced by IT operations teams. Controllers offer the ability to manage a network as a system, which means the policy management can be automated and abstracted. They provide the capability of supporting dynamic policy changes rather than requiring manual policy changes and device-by-device configurations when something requires a change within the environment.

It is important from a career and skills perspective to adapt to the changes within the industry. Keeping on top of new skillsets is critical to maintaining relevance in the industry or job market. Becoming Cisco certified helps with this for multiple reasons, including the following:

- Highlighting skills to employers
- Highlighting skills to industry peers
- Providing value to employers
- Providing credibility
- Providing a baseline of understanding
- Building confidence
- Enabling career advancement
- Increasing salary

When pursuing certification, it is imperative to understand why getting certified is beneficial in the first place. Many people pursue certifications as a way to break into the job market. Having certifications can be a great differentiator in terms of skillset and discipline. Pursuing a certification may also be valuable from a continuing education perspective and to bolster the candidate's current job role. Pursuing certifications can also help candidates evolve their skillsets to keep up with the ever-changing advancements in the technology industry. As mentioned earlier in this chapter, new network operations techniques are rapidly becoming the norm. Automation and programmability are at the center of this paradigm shift. Becoming certified not only helps embrace this type of momentum but also prepares the candidate for the world of tomorrow.

CISCO CAREER CERTIFICATION OVERVIEW

Cisco has evolved the way it offers certifications. In the past, there were many separate tracks for each discipline, such as Routing and Switching, Collaboration, Service Provider, Data Center, and Security. Although there are still separate disciplines, the number of disciplines has been greatly reduced, and the process candidates go through to achieve those certifications has changed significantly. The traditional path for Routing and Switching was Cisco Certified Network Associate (CCNA), Cisco Certified Network Professional (CCNP),

and then Cisco Certified Internetwork Expert (CCIE). In the past, in order to become CCNP certified, a candidate would have to have previously completed the CCNA certification and be in current certified status prior to completing the CCNP. In addition, for the CCIE, a written qualification exam had to be completed prior to attempting the CCIE lab exam. However, having CCNA or CCNP certification was not necessary in order to pursue the CCIE certification. Today, the certification process has been greatly simplified. As mentioned previously, Cisco has evolved the certification structure and the prerequisites for each track.

There are five levels of accreditation for Cisco career certifications. The following sections cover how the process has evolved and ultimately created the ability for candidates to “choose their own destiny” when it comes to what certifications and skillsets they want to pursue.

Figure 1-2 shows the pyramid hierarchy used to describe the five levels of accreditation in Cisco certifications:

- Architect
- Expert
- Professional
- Associate
- Entry



Figure 1-2 *Cisco Career Certification Levels of Accreditation*

Each level of accreditation has multiple certifications and exams that pertain to that specific level. The higher the level, the more skill and rigorous hands-on experience are required. For example, the CCIE lab exam is experience based, and completing it requires hands-on expertise. In addition to the five tiers of the pyramid, there are also specialist certifications that candidates can achieve in specific technologies to showcase their knowledge and base level of understanding. (These specialist certifications are covered later in this chapter.) Simplifying the certification portfolio reduced the number of certifications available in general and also improved the process of achieving these certifications. [Table 1-2](#) lists some of the certifications and tracks that were available in the past, as they relate to the five-level pyramid. You can see that there were a tremendous number of options available for each track.

Table 1-2 *Cisco Career Certifications Tracks Prior to Restructuring*

Entry Associate Professional Expert Architect

Cisco Certified Entry Networking Technician (CCENT)	Cisco Certified Design Associate (CCDA)	Cisco Certified Design Professional (CCDP)	Cisco Certified Design Expert (CCDE)	Cisco Certified Architect (CCAr)
Cisco Certified Technician (CCT)	CCNA Cloud	CCNP Cloud		
	CCNA Collaboration	CCNP Collaboration	CCIE Collaboration	
	CCNA Cyber Ops			
	CCNA Data Center	CCNP Data Center	CCIE Data Center	
	CCNA Industrial			
	CCNA Routing and Switching	CCNP Routing and Switching	CCIE Routing and Switching	
	CCNA Security	CCNP Security	CCIE Security	
	CCNA Service Provider	CCNP Service Provider	CCIE Service	

			Provide r	
	CCNA Wireless	CCNP Wireless	CCIE Wireless	

Table 1-3 shows the new and simplified certification portfolio and how the certifications now fit into each level of the pyramid. You can see that there has been a significant reduction in the number of available certifications, and there is now a succinct path to follow through these certifications.

Table 1-3 Cisco Career Certifications Tracks After Restructuring

Entry Associate Professional Expert Architect				
			Cisco Certified Design Expert (CCDE)	Cisco Certified Architect (CCAr)
	Dev Net Ass ocia te	DevNe t Profes sional	DevNet Expert (TBA)	
Cisco Certified Technicia n (CCT)	CC NA	CCNP Enter prise	CCIE Enterprise Infrastructu re CCIE Enterprise Wireless	
		CCNP Collab	CCIE Collaboratio	

		oratio n	n	
		CCNP Data Center	CCIE Data Center	
		CCNP Securi ty	CCIE Security	
		CCNP Servic e Provid er	CCIE Service Provider	

As changes were being made in the certification portfolio, some certifications were completely removed. Table 1-3 shows that there is now only a single CCNA certification. Prior to this change, there were nine CCNA certifications, and multiple exams had to be completed in order to become a CCNA in any of the tracks. Now with the new CCNA, a candidate need pass only a single exam to become CCNA certified. An additional certification that was removed was the CCENT. Now that the CCNA is a broader exam and covers many introductory-level topics, the CCENT topics have been absorbed into the new CCNA. Furthermore, the CCDA and CCDP certifications were retired as that design information has been incorporated into other certifications within each track, and separate certifications are no longer required for the Associate and Professional levels of design knowledge.

The CCNP has changed significantly as well. Previously, for example, the CCNP Routing and Switching exam consisted of three exams:

- 300-101 ROUTE
- 300-115 SWITCH

- 300-135 TSHOOT

A candidate would have to successfully pass all three of these exams as well as the CCNA in the same track in order to become CCNP Routing and Switching certified. Today, only two exams are required in order to become CCNP Enterprise certified. Candidates can now start wherever they want; there are no prerequisites, and a candidate can start earning any level of certification—even Associate, Specialist, Professional, or Expert level certification. For the CCNP Enterprise certification, the first exam is the 300-401 ENCOR exam, which covers core technologies in enterprise networks, including the following:

- Dual-stack (IPv4 and IPv6) architecture
- Virtualization
- Infrastructure
- Network assurance
- Security
- Automation

Once the ENCOR exam is completed, a concentration exam must be taken. This is perhaps the most important and fundamental change made to the CCNP. The available concentration exams include a variety of different technology specialties and allow candidates to build their own certification (or “choose their own destiny”). Each CCNP track has its own core exam and concentrations.

Cisco made a number of changes to the specialist certifications, which allow candidates to get certified in specific areas of expertise. For example, a candidate who is proficient at Cisco Firepower can pursue a specialist certification for Firepower. The specialist certifications are important because candidates, especially consultants, often have to use many different technologies in many different customer environments. Specialist certifications can help show a candidate’s ability to work

on a variety of projects. They also help build credibility on a plethora of different platforms and technologies. For example, a candidate looking to focus on routing and Cisco SD-WAN could take the CCNP 300-401 ENCOR exam and then take the 300-415 ENSDWI concentration exam to become a CCNP with an SD-WAN specialty. In essence, the concentration exams are the new specialist exams, and a candidate can simply take a single specialist exam and become certified in that technology (for example, the 300-710 SNCF exam for certification in network security and Firepower).

Table 1-4 lists and describes the different type of CCNP concentration and specialist exams currently available.

Table 1-4 CCNP Core and Concentration Exams

Track	Description	Concentration Exam
Enterprise	Cisco Certified Specialist–Enterprise Core	300-401 ENCOR
Enterprise	Cisco Certified Specialist–Enterprise Advanced Infrastructure Implementation	300-410 ENARSI
Enterprise	Cisco Certified Specialist–Enterprise SD-WAN Implementation	300-415 ENSDWI
Enterprise	Cisco Certified Specialist–Enterprise Design	300-420 ENSLID
Enterprise	Cisco Certified Specialist–Enterprise Wireless Design	300-425

		ENWL SD
Enterprise	Cisco Certified Specialist–Enterprise Wireless Implementation	300-430 ENWLSI
Data Center	Cisco Certified Specialist–Data Center Core	300-601 DCCOR
Data Center	Cisco Certified Specialist–Data Center Design	300-610 DCID
Data Center	Cisco Certified Specialist–Data Center Operations	300-615 DCIT
Data Center	Cisco Certified Specialist–Data Center ACI Implementation	300-620 DCACI
Data Center	Cisco Certified Specialist–Data Center SAN Implementation	300-625 DCSAN
Security	Cisco Certified Specialist–Security Core	300-701 SCOR
Security	Cisco Certified Specialist–Network Security Firepower	300-710 SNCF
Security	Cisco Certified Specialist–Network Security VPN Implementation	300-730 SVPN
Security	Cisco Certified Specialist–Email Content Security	300-720 SESA

Security	Cisco Certified Specialist–Web Content Security	300-725 SWSA
Security	Cisco Certified Specialist–Security Identity Management Implementation	300-715 SISE
Service Provider	Cisco Certified Specialist–Service Provider Core	300-501 SPCOR
Service Provider	Cisco Certified Specialist–Service Provider Advanced Routing Implementation	300-510 SPRI
Service Provider	Cisco Certified Specialist–Service Provider VPN Services Implementation	300-515 SPVI
Collaboration	Cisco Certified Specialist–Collaboration Core	300-801 CLCOR
Collaboration	Cisco Certified Specialist–Collaboration Applications Implementation	300-810 CLICA
Collaboration	Cisco Certified Specialist–Collaboration Call Control & Mobility Implementation	300-815 CLACCM
Collaboration	Cisco Certified Specialist–Collaboration Cloud & Edge Implementation	300-820 CLCEI
DevNet, Enterprise	Cisco Certified DevNet Specialist–Enterprise Automation and Programmability	300-435 ENAUTO

DevNet, Data Center	Cisco Certified DevNet Specialist– Data Center Automation and Programmability	300- 635 DCAU TO
DevNet, Security	Cisco Certified DevNet Specialist– Security Automation and Programmability	300- 735 SAUT O
DevNet, Service Provider	Cisco Certified DevNet Specialist– Service Provider Automation and Programmability	300- 535 SPAU TO
DevNet, Collaboration	Cisco Certified DevNet Specialist– Collaboration Automation and Programmability	300- 835 CLAU TO
DevNet	Cisco Certified DevNet Specialist– Core	300- 901 DEVC OR
DevNet	Cisco Certified DevNet Specialist– DevOps	300- 910 DEVO PS
DevNet	Cisco Certified DevNet Specialist–IoT	300- 915 DEVI OT
DevNet	Cisco Certified DevNet Specialist– Webex	300- 920 DEVW BX

Note

The exams listed in Table 1-4 were available at the time of publication. Please visit

<http://www.cisco.com/go/certifications> to keep up on all the latest available certifications and associated tracks.

In addition to the Associate- and Professional-level certifications, the Cisco certified specialist certifications have changed as well. Previously, in some cases multiple exams had to be completed to become certified as a specialist in a specific topic or discipline. With the new changes, however, candidates can take and complete any one of the specialist exams mentioned in [Table 1-4](#) to become certified in that technology area. For example, a candidate who is proficient with Cisco Identity Services Engine (ISE) could pursue a specialist certification for security identity management implementation by taking the 300-715 SISE exam.

Another major change to the certification program is that changes have been made to the flagship CCIE program. The CCIE Routing and Switching certification and the CCIE Wireless certification have both been rebranded as CCIE Enterprise certifications: CCIE Routing and Switching became CCIE Enterprise Infrastructure, and CCIE Wireless became CCIE Enterprise Wireless. The goal of this change was to align the certifications with the current technologies that candidates are seeing in their work environments as well as the industry trends that are changing the way networking is being consumed and managed. As mentioned earlier in this chapter, software-defined networking, automation, programmability, IoT, and other trends are drastically changing the approach network operations teams are taking to networking in general. The business outcomes and use case-driven adoption of these new technologies are shaping the industry, as are the approaches vendors are taking to building and designing their products. User experience as well as adoption are now critical and are top-of-mind priority topics for many customers. Cisco therefore wanted to align its career certification portfolio with

what candidates and the industry are seeing in their networking environments. For all other Expert-level certifications, there are now currently only the following specialties:

- Cisco Certified Design Expert (CCDE)
- CCIE Enterprise Infrastructure
- CCIE Enterprise Wireless
- CCIE Collaboration
- CCIE Data Center
- CCIE Security
- CCIE Service Provider

CISCO DEVNET CERTIFICATIONS

The following sections provide an overview of the new Cisco DevNet certifications. It also explains the required skillsets necessary to achieve these new certifications. As you will see, there are many different options available for candidates to pursue.

Note

The DevNet Expert certification will be announced in the future. Please visit <http://www.cisco.com/go/certifications> to keep up on all the latest available certifications and associated tracks.

Cisco Certified DevNet Associate Certification (DEVASC)

Considering everything covered up to this point in the chapter and the main focus of this book, this section covers the Cisco DevNet Associate certification at a high level. Although there was previously a very broad and robust Cisco career certification portfolio that was long established and well known, it had a gap—and that gap was becoming more and more noticeable with the changes that were happening in the industry, such as the

need for automation in the network environment across all tracks and areas of the network, ranging from enterprise and data center networks to large-scale service provider networks. Today, all areas of the business must work together, and it is important to remove the silos that once compartmentalized different departments. Applications are being instantiated at speeds that have never been seen before. Furthermore, with user experience becoming the benchmark for how a business measures success, it is paramount to deploy applications, network services, and security in an agile, consistent, and repeatable manner. Much like the CCNA, the DevNet Associate certification requires only a single exam. The DevNet Associate certification covers multiple knowledge domains, as shown in [Figure 1-3](#).

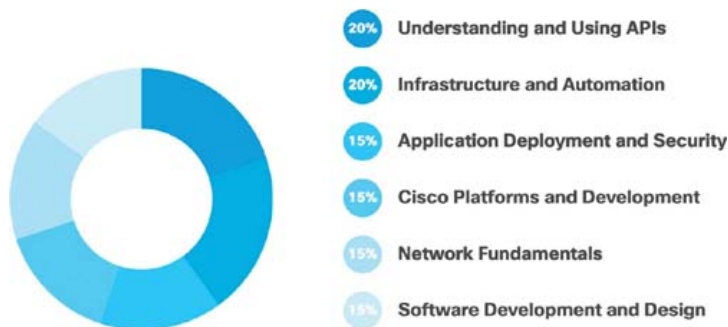


Figure 1-3 *Cisco DevNet Associate Knowledge Domains*

It is recommended that candidates attempting the Cisco DevNet Associate exam have at least one year of experience developing and maintaining applications built on top of Cisco platforms. In addition, they must have hands-on experience with programming languages such as Python. This certification was designed for early-in-career developers and for experienced network engineers looking to expand their skillsets to include software and automation practices. It is important to note that the line is blurring between network engineers and developers. The two skillsets are slowly merging, and candidates are becoming “network developers.” Having a

certification like the DevNet Associate can open doors for candidates to approach new job roles that didn't necessarily exist in the past including the following:

- Junior or entry-level DevOps engineer
- Cloud developer
- Automation engineer

When pursuing any certification, a candidate should remember the reason the certification is important in the first place. Certifications can help expand a candidate's current skillset as well as ensure a baseline level of knowledge around specific topics. The Cisco DevNet Associate certification can help businesses find individuals who possess a certain level of programmability or automation skills. It gives businesses a clear way to determine the base level of skills when looking at hiring a candidate and ensure that new hires have the necessary relevant skills. The upcoming chapters of this book align directly with the DevNet Associate blueprint. This book covers the topics necessary to build a foundational level of understanding for candidates to feel comfortable in pursuing the Cisco DevNet Associate certification.

Cisco Certified DevNet Professional Certification

The next certification in the path after Cisco DevNet Associate would be the Cisco DevNet Professional. This more robust certification requires a more advanced skillset. [Figure 1-4](#) illustrates some of the high-level requirements for this certification and their associated topic domains.

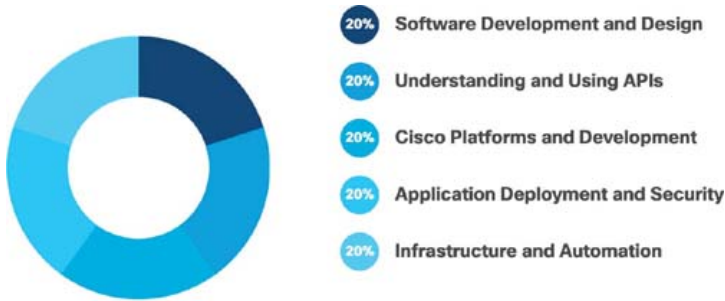


Figure 1-4 Cisco DevNet Professional Knowledge Domains

It is recommended that candidates attempting the Cisco DevNet Professional exam have a minimum of three to five years of experience designing and implementing applications built on top of Cisco platforms. It is also critical that they have hands-on experience with programming languages such as Python. This certification was designed for experienced network engineers looking to expand their capabilities and include software and automation on their resume. It is also designed for developers moving into automation and DevOps roles as well as for solution architects who leverage the Cisco ecosystem. Infrastructure developers designing hardened production environments will also benefit from the Cisco DevNet Professional certification. Because the DevNet Professional provides many avenues for a candidate to create a unique journey, it is one of the most eagerly anticipated certifications and will be integral to aligning candidates' skillsets with their daily job tasks.

Table 1-5 lists the CCNP concentration and specialist exams currently available.

Table 1-5 DevNet Concentration and Specialist Exams

Track	Description	Specialist Exam

DevNet, Enterprise	Cisco Certified DevNet Specialist– Enterprise Automation and Programmability	300- 435 ENAU TO
DevNet, Data Center	Cisco Certified DevNet Specialist– Data Center Automation and Programmability	300- 635 DCAU TO
DevNet, Security	Cisco Certified DevNet Specialist– Security Automation and Programmability	300- 735 SAUT O
DevNet, Service Provider	Cisco Certified DevNet Specialist– Service Provider Automation and Programmability	300- 535 SPAU TO
DevNet, Collaboration	Cisco Certified DevNet Specialist– Collaboration Automation and Programmability	300- 835 CLAU TO
DevNet	Cisco Certified DevNet Specialist– Core	300- 901 DEV OR
DevNet	Cisco Certified DevNet Specialist– DevOps	300- 910 DEVO PS
DevNet	Cisco Certified DevNet Specialist–IoT	300- 915 DEVI OT
DevNet	Cisco Certified DevNet Specialist– Webex	300- 920 DEVW BX

You might notice that some of the specializations listed in [Table 1-5](#) were listed earlier in this chapter for the CCNP exams as well. This is because they can be used for both the CCNP exams and the DevNet Professional exams. In addition, the DevNet Specialist exams can be taken independently for Cisco Certified DevNet Specialist certification. This is similar to the CCNP concentration exams covered earlier. [Figure 1-5](#) illustrates the entire Cisco career certification structure. As you can see, regardless of what track a candidate decides to pursue—whether it’s a Specialist or a Professional level—it is possible to choose a variety of DevNet skills as part of the journey.

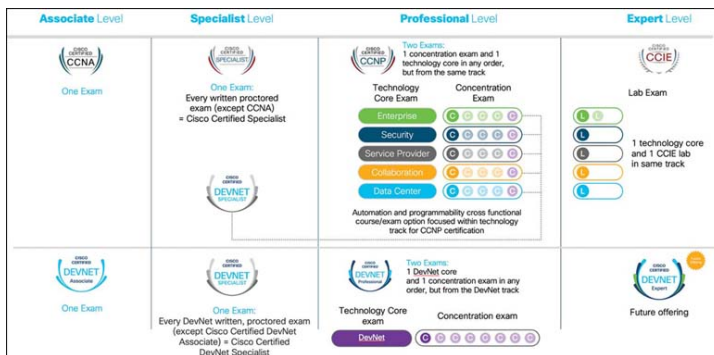


Figure 1-5 Cisco Career Certification Overview

Note

The DevNet Expert certification is a planned offering that was not available as this book went to press.

CISCO DEVNET OVERVIEW

This section looks at the tools and resources available for Cisco DevNet certification candidates. These tools help candidates to learn, practice, and share ideas as well as experience.

The examples and tools discussed in this chapter are all available to use and practice at <http://developer.cisco.com>, which is the home for Cisco

DevNet. This site provides a single place for network operators to go when looking to enhance or increase their skills with APIs, coding, Python, or even controller concepts. DevNet makes it easy to find learning labs and content to help build or solidify current knowledge in network programmability. Whether a candidate is just getting started or a seasoned programmatic professional, DevNet is the place to be! This section provides a high-level overview of DevNet. It describes the different sections of DevNet, some of the labs available, and other content that is available. [Figure 1-6](#) shows the DevNet main page.

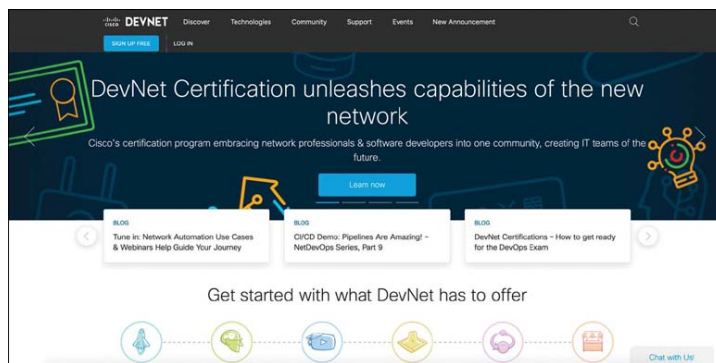


Figure 1-6 *DevNet Main Page*

Across the top of the main DevNet page, you can see that the following menu options:

- Discover
- Technologies
- Community
- Support
- Events

The following sections cover these menu options individually.

Discover

The Discover page shows the different offerings that DevNet has available. This page includes the subsection

Learning Tracks; the learning tracks on this page guide you through various different technologies and associated API labs. Some of the available labs are Programming the Cisco Digital Network Architecture (DNA), ACI Programmability, Getting Started with Cisco WebEx Teams APIs, and Introduction to DevNet.

When you choose a learning lab and start a module, DevNet tracks all your progress and allows you to go away and then come back and continue where you left off. This is an excellent feature if you are continuing your education over the course of multiple days or weeks. Being able to keep track of your progress means you can easily see what you have already learned and also determine what might be the next logical step in your learning journey.

Technologies

The Technologies page allows you to pick relevant content based on the technology you want to study and dive directly into the associated labs and training for that technology. [Figure 1-7](#) shows some of the networking content that is currently available in DevNet.

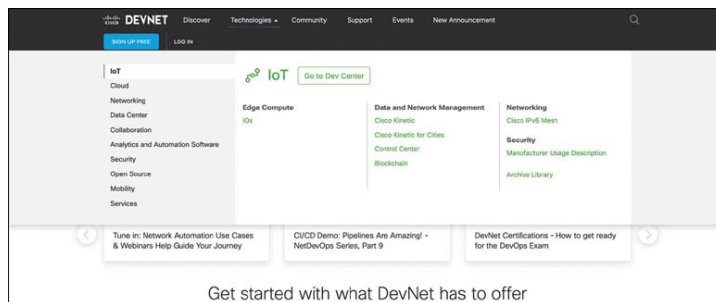


Figure 1-7 DevNet Technologies Page

Note

Available labs may differ from those shown in this chapter's figures. Please visit

<http://developer.cisco.com> to see the latest content available and to interact with the current learning labs.

Community

Perhaps one of the most important section of DevNet is the Community page, where you have access to many different people at various stages of learning. You can find DevNet ambassadors and evangelists to help at various stages of your learning journey. The Community page puts the latest events and news at your fingertips. This is also the place to read blogs, sign up for developer forums, and follow DevNet on all major social media platforms. This is the safe zone for asking any questions, regardless of how simple or complex they might seem. Everyone has to start somewhere. The DevNet Community page is the place to start for all things Cisco and network programmability. [Figure 1-8](#) shows some of the options currently available on the Community page.

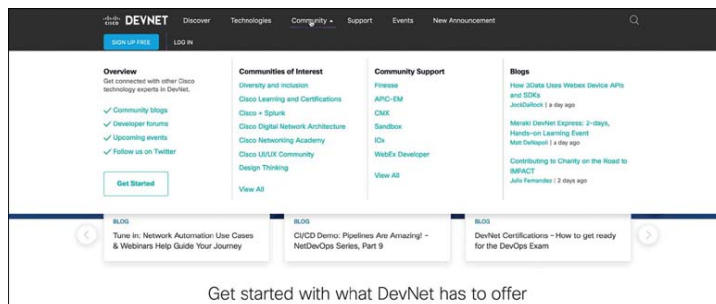


Figure 1-8 DevNet Community Page

Support

On the DevNet Support page you can post questions and get answers from some of the best in the industry. Technology-focused professionals are available to answer questions from both technical and theoretical perspectives. You can ask questions about specific labs or overarching technologies, such as Python or YANG models. You can also open a case with the DevNet Support team, and your questions will be tracked and

answered in a minimal amount of time. This is a great place to ask the Support team questions and to tap into the expertise of the Support team engineers. **Figure 1-9** shows the DevNet Support page, where you can open a case. Being familiar with the options available from a support perspective is key to understanding the types of information the engineers can help provide.

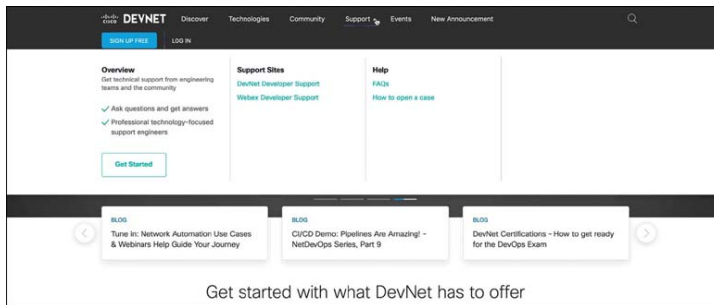


Figure 1-9 DevNet Support Page Events

Events

The Events page, shown in **Figure 1-10**, provides a list of all events that have happened in the past and will be happening in the future. This is where you can find the upcoming DevNet Express events as well as any conferences where DevNet will be present or participating. Be sure to bookmark this page if you plan on attending any live events. DevNet Express is a one- to three-day event led by Cisco developers for both customers and partners. Attending one of these events can help you with peer learning and confidence as well as with honing your development skills.

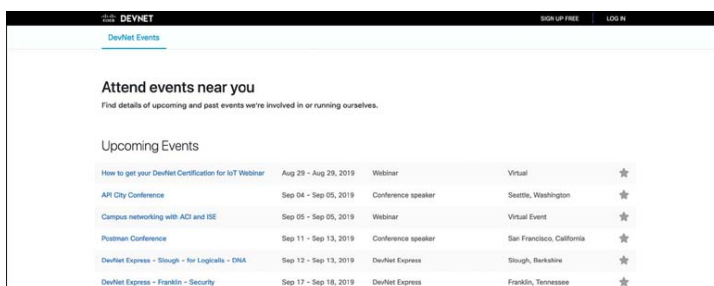


Figure 1-10 DevNet Events Page

Note

Keep in mind that the schedule shown in [Figure 1-10](#) will differ from the schedule you see when you read this chapter.

DevNet gives customers the opportunity to learn modern development and engineering skills and also get hands-on experience with them in a safe environment. DevNet Express offers foundational development skills training to expose attendees to the latest languages and tools available. Once the foundational skills have been covered, specific learning tracks or technology-specific modules are then covered so attendees can apply their newly learned skills to working with APIs on Cisco products. These events are guided, which helps ensure that attendees have the support they need to get started in the world of APIs and programmability.

DevNet Automation Exchange

DevNet Automation Exchange makes code available for consumption. This code is based on consumable use cases; that is, use case–specific solutions have been uploaded by various developers and are designed to accomplish particular business outcomes. For example, whereas one solution may contain the steps to fully automate the provisioning of devices in Cisco DNA Center, and another may make it possible to deploy a fabric, the specific use case for both solutions might be to increase the speed of onboarding new site locations or improve the user experience for mobile users moving from one area of a campus network to another, regardless of whether they are connected via wire or wirelessly. The use cases in the DevNet Automation Exchange are divided by three different categories:

- Walk
- Run
- Fly

Figure 1-11 shows the landing page for the DevNet Automation Exchange. You can see that you can view the use case library as well as share any use cases that you have created.

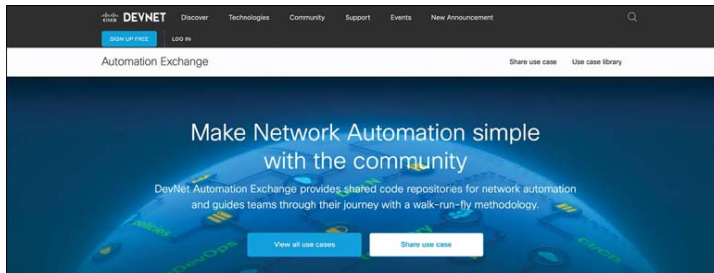


Figure 1-11 DevNet Automation Exchange

When searching the use case library, you can search using the Walk, Run, or Fly categories as well as by type of use case. In addition, you can find use cases based on the automation lifecycle stage or the place in the network, such as data center, campus, or collaboration. Finally, you can simply choose the product for which you want to find use cases, such as IOS XE, Cisco DNA Center, or ACI (see Figure 1-12).

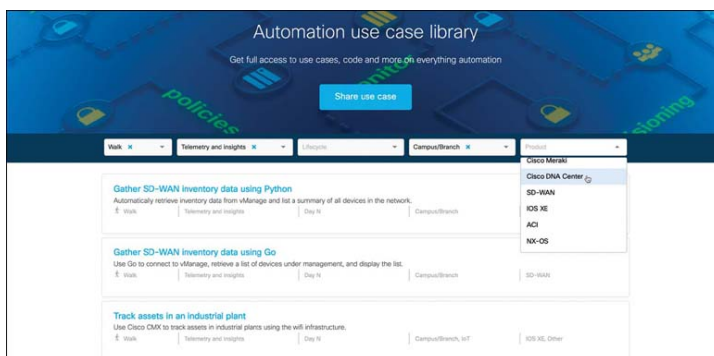


Figure 1-12 DevNet Automation Exchange Use Case Library

The Walk category allows you to gain visibility and insights into your network. You can find various projects involving gathering telemetry and insight data in a read-only fashion. These projects can provide auditing

capabilities to ensure the network's security and compliance. Because the projects are read-only, gathering the information has minimal risk of impacting a network negatively. You could, for example, use programmability to do a side-by-side configuration comparison to see what has changed in the configuration on a device. Using tools like this would be the next step past using the DevNet sandboxes to write code in a production environment.

The Run in Automation Exchange is where read/write actions start taking place in the network environment, such as when a network operations team begins to activate policies and signify intent across different network domains. These types of projects can also allow for self-service network operations and ensure compliance with security policies and operational standards. Automation tools are key to ensuring consistency and simplicity in day-to-day operations.

Finally, the Fly category is for proactively managing applications, users, and devices by leveraging a DevOps workflow. With such projects, you can deploy applications using continuous integration and delivery (CI/CD) pipelines while at the same time configuring the network and keeping consistent application policies. By combining machine learning capabilities with automation, a business can shift from a reactive application development approach to a more holistic proactive approach—which lowers risk and increases agility. Each of the Automation Exchange use cases adheres to the automation lifecycle, which consists of Day 0–2 operations. [Table 1-6](#) lists the functions of the automation lifecycle.

Table 1-6 Automation Lifecycle

Day	Function	Description
.	.	.

0	In sta ll	Bringing devices into an initial operational state
1	Co nfi gu re	Applying configurations to devices
2	Op ti mi ze	Implementing dynamic services, optimizing network behavior, and troubleshooting issues
N	M an ag e	Ensuring consistent and continuous operation of the network, with reduced risk and human error

SUMMARY

This chapter provides a high-level overview of Cisco's career certifications and how candidates can choose their own destiny by picking the areas where they want to build experience and become certified. This chapter describes Cisco's new specialist exams, which focus on many different technologies, such as Firepower, SD-WAN, and IoT. This chapter also discusses some of the benefits of becoming certified, from career advancement to building confidence to commanding a higher salary in the workplace. This chapter also details at a high level the components of Cisco DevNet, the value of the DevNet community, and DevNet events such as Cisco DevNet Express. Finally, this chapter introduces DevNet tools such as DevNet Automation Exchange and DevNet learning labs.

Chapter 2

Software Development and Design

This chapter covers the following topics:

- **Software Development Lifecycle:** This section covers the Software Development Lifecycle (SDLC) and some of the most popular SDLC models, including Waterfall, Agile, and Lean.
- **Common Design Patterns:** This section covers common software design patterns, including the Model-View-Controller (MVC) and Observer design patterns.
- **Linux BASH:** This section covers key aspects of the Linux BASH shell and how to use it.
- **Software Version Control:** This section includes the use of version control systems in software development.
- **Git:** This section discusses the use of the Git version control system.
- **Conducting Code Review:** This section discusses using peer review to check the quality of software.

Are you a software developer? This has become an existential question for traditional network engineers, as programming and automation have become more pervasive in the industry. Professional programmers have picked up software integration with infrastructure gear as a new service or capability they can add to their applications. Traditional infrastructure engineers are being expected to know how to use APIs and automation tool sets to achieve more agility and speed in IT operations. The bottom line is that we are all being asked to pick up new skills to accomplish the business goals that keep us relevant and employed. This chapter discusses a few of the fundamental principles and tools that modern software development requires. You will learn about Agile, Lean, and common software design

patterns that are used to enable a whole new operational model. In addition, you will see the importance of version control and how to use Git to collaborate with others and share your work with the world. These core concepts are essential to understanding the influence of software development methodologies as they pertain to infrastructure automation.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 2-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 2-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Software Development Lifecycle	1, 2
Common Design Patterns	3, 4
Linux BASH	5, 6
Software Version Control	7
Git	8–10
Conducting Code Review	11

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. What is Waterfall?

1. A description of how blame flows from management on failed software projects
2. A type of SDLC
3. A serial approach to software development that relies on a fixed scope
4. All of the above

2. What is Agile?

1. A form of project management for Lean
2. An implementation of Lean for software development
3. A strategy for passing the CCNA DevNet exam
4. A key benefit of automation in infrastructure

3. The Model-View-Controller pattern is often used in which of the following applications? (Choose three.)

1. Web applications with graphical interfaces
2. Client/server applications with multiple client types
3. PowerShell scripts
4. Django

4. Which of the following are true of the Observer pattern? (Choose two.)

1. It is a publisher/subscriber pattern.
2. It is a multicast pattern.
3. It is used for configuration management applications and event handling.
4. It is not commonly used in infrastructure application design.

5. What does BASH stand for?

1. Born Again Shell
2. Basic Shell
3. Bourne Again Shell
4. None of the above

6. Which of the following is the best option for displaying your current environment variables?

1. `env | cat >env.txt`
2. `env | more`
3. `export $ENV | cat`
4. `echo env`

7. Which of the following is true of software version control?

1. It is a naming convention for software releases.
2. It is also known as source code management.
3. It is the same thing as BitKeeper.
4. None of the above are true.

8. Who created Git?

1. Junio Hamano
2. Marc Andreessen
3. John Chambers
4. Linus Torvalds

9. What are the three main structures tracked by Git?

1. Index, head, and local repo
2. Local workspace, index, and local repository
3. Remote repository, head, and local index
4. None of the above

10. What command do you use to add the specific filename `file.py` to the Git index?

1. `git add .`
2. `git index file.py`
3. `git index add .`
4. `git add file.py`

11. Which is one of the key benefits of conducting a code review?

1. It helps you create higher-quality software
2. You can find weak programmers who need more training
3. It can be used to identify defects in software that are obvious
4. None of the above

FOUNDATION TOPICS

SOFTWARE DEVELOPMENT LIFECYCLE

Anyone can program. Once you learn a programming language's syntax, it's just a matter of slapping it all together to make your application do what you want it to do, right? The reality is, software needs to be built using a structure to give it sustainability, manageability, and coherency. You may have heard the phrase "cowboy coding" to refer to an unstructured software project, where there is little formal design work, and the programmer just sort of "shoots from the hip" and slaps code in with little or no forethought. This is a path that leads straight to late-night support calls and constant bug scrubbing. Heaven forbid if you inherit a ball of spaghetti like this and you are asked to try to fix, extend, or modernize it. You will more than likely be updating your resume or packing your parachute for a quick escape.

To prevent problems from slapdash approaches such as cowboy coding, disciplines such as architecture and construction establish rules and standards that govern the process of building. In the world of software, the Software Development Lifecycle (SDLC) provides sanity by providing guidance on building sustainable software packages. SDLC lays out a plan for building, fixing, replacing, and making alterations to software.

As shown in [Figure 2-1](#), these are the stages of the SDLC:

- **Stage 1—Planning:** Identify the current use case or problem the software is intended to solve. Get input from stakeholders, end users, and experts to determine what success looks like. This stage is also known as *requirements analysis*.
- **Stage 2—Defining:** This stage involves analyzing the functional specifications of the software—basically defining what the software is supposed to do.
- **Stage 3—Designing:** In this phase, you turn the software specifications into a design specification. This is a critical stage as stakeholders need to be in agreement in order to build the software appropriately; if they aren't, users won't be happy, and the project will not be successful.
- **Stage 4—Building:** Once the software design specification is complete, the programmers get to work on making it a reality. If the

previous stages are completed successfully, this stage is often considered the easy part.

- **Stage 5—Testing:** Does the software work as expected? In this stage, the programmers check for bugs and defects. The software is continually examined and tested until it successfully meets the original software specifications.
- **Stage 6—Deployment:** During this stage, the software is put into production for the end users to put it through its paces. Deployment is often initially done in a limited way to do any final tweaking or detect any missed bugs. Once the user has accepted the software and it is in full production, this stage morphs into maintenance, where bug fixes and software tweaks or smaller changes are made at the request of the business user.



Figure 2-1 *Software Development Lifecycle*

Note

ISO/IEC 12207 is the international standard for software lifecycle processes, and there are numerous organizations around the globe that use it for certification of their software development efforts. It is compatible with any SDLC models and augments them

from a quality and process assurance standpoint. It does not, however, replace your chosen SDLC model.

There are quite a few SDLC models that further refine the generic process just described. They all use the same core concepts but vary in terms of implementation and utility for different projects and teams. The following are some of the most popular SDLC models:

- Waterfall
- Lean
- Agile
- Iterative model
- Spiral model
- V model
- Big Bang model
- Prototyping models

Luckily, you don't need to know all of these for the 200-901 DevNet Associate DEVASC exam. The following sections cover the ones you should know most about: Waterfall, Lean, and Agile.

Waterfall



Back in the 1950s, when large companies started to purchase large mainframe computers to crunch numbers, no one really knew how to run an IT organization. It really wasn't anything that had been done before, and for the most part, computers were only really understood by an elite group of scientists. Programming a mainframe required structure and a process. This caused a problem for businesses looking to tap into the capabilities of these new systems since there wasn't a well-known method to create business applications. So they looked around at other industries for guidance.

The construction industry was booming at the time. The construction industry followed a rigid process in which every step along the way was dependent on the completion of the previous step in the process. If you want to end up with a building that stays standing and meets the original design, you can't start construction until you have a plan and analyze the requirements for the building. This thought process mapped nicely to software development, and the complexity of designing and constructing a building was similar to that of creating software applications. Waterfall, which is based on the construction industry approach, became one of the most popular SDLC approaches.

As illustrated in [Figure 2-2](#), Waterfall is a serial approach to software development that is divided into phases:

- **Requirements/analysis:** Software features and functionality needs are cataloged and assessed to determine the necessary capabilities of the software.
- **Design:** The software architecture is defined and documented.
- **Coding:** Software coding begins, based on the previously determined design.
- **Testing:** The completed code is tested for quality and customer acceptance.
- **Maintenance:** Bug fixes and patches are applied.

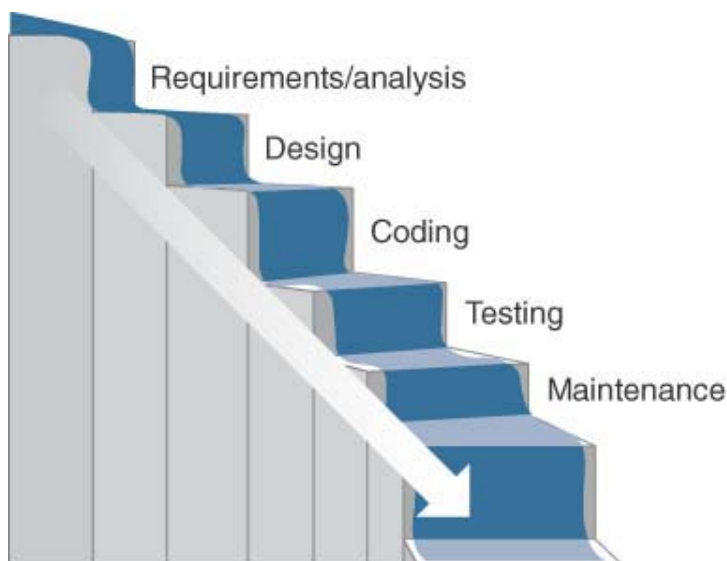


Figure 2-2 *Waterfall*

While this approach has worked successfully over the years, a number of shortcomings have become weaknesses in this approach. First, the serial nature of Waterfall, while easy to understand, means that the scope of a software project is fixed at the design phase. In construction, making changes to the first floor of a building after you have begun the fifth floor is extremely difficult—and may even be impossible unless you knock down the building and start from scratch. In essence, the Waterfall approach does not handle change well at all. When you finally get to the coding phase of the application development process, you might learn that the feature you are building isn't needed anymore or discover a new way of accomplishing a design goal; however, you cannot deviate from the predetermined architecture without redoing the analysis and design. Unfortunately, it is often more painful to start over than to keep building. It is similar to being stuck building a bridge over a river that no one needs anymore.

The second aspect of Waterfall that is challenging is that value is not achieved until the end of the whole process. We write software to automate some business function or capability—and value is only realized when the software is in production and producing results. With the Waterfall approach, even if you are halfway done with a project, you still have no usable code or value to show to the business. Figure 2-3 shows this concept.

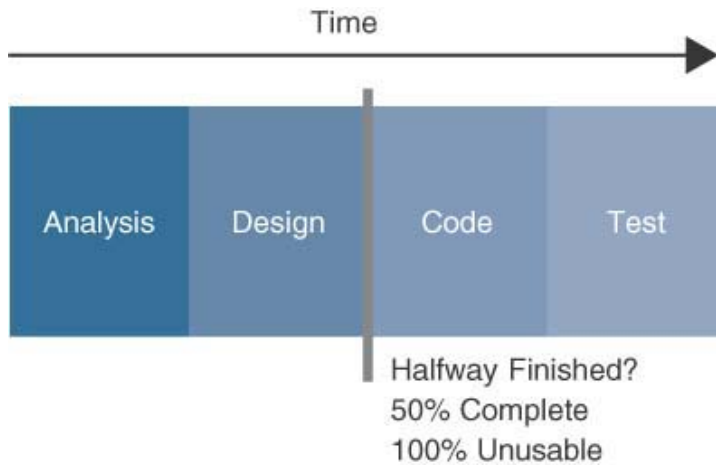


Figure 2-3 *The Value Problem of Waterfall*

The third aspect of Waterfall that is challenging is quality. As mentioned earlier, time is the enemy when it comes to delivering value. If we had unlimited time, we could create perfect software every time, but we simply don't live in that world. When software developers run out of time, testing often suffers or is sacrificed in the name of getting the project out the door.

The three challenges for Waterfall led to the development of a new way of creating software that was faster, better, and more adaptive to a rapidly changing environment.

Lean



After World War II, Japan was in desperate need of rebuilding. Most of Japan's production capabilities had been destroyed, including those in the auto industry. When Japan tackled this rebuilding, it didn't concentrate on only the buildings and infrastructure; it looked at ways to do things differently. Out of this effort, the Toyota Production System (TPS) was born. Created by Taiichi Ohno and Sakichi Toyoda (founder of Toyota),

this management and manufacturing process focuses on the following important concepts:

- **Elimination of waste:** If something doesn't add value to the final product, get rid of it. There is no room for wasted work.
- **Just-in-time:** Don't build something until the customer is ready to buy it. Excess inventory wastes resources.
- **Continuous improvement (Kizan):** Always improve your processes with lessons learned and communication.

While these concepts seem glaringly obvious and practical, TPS was the first implementation of these principles as a management philosophy. TPS was the start of the more generalized Lean manufacturing approach that was introduced to the Western world in 1991 through a book written by Womack, Jones, and Roos, *The Machine That Changed the World*. This book was based on a five-year study MIT conducted on TPS, and it has been credited with bringing Lean concepts and processes beyond the auto industry.

Why spend this time talking about moldy old management books? Lean led to Agile software development, which has served as a lightning rod of change for IT operations.

Agile



Agile is an application of Lean principles to software development. With Agile, all the lessons learned in optimizing manufacturing processes have been applied to the discipline of creating software. In 2001, 17 software developers converged on the Snowbird resort in Utah to discuss new lightweight development methods. Tired of missing deadlines, endless documentation, and the inflexibility of existing software development practices, these Agile pioneers created the “Manifesto for Agile Software Development,” which codifies the guiding

principles for Agile development practices. The following 12 principles are the core of the Agile Manifesto:

- Customer satisfaction is provided through early and continuous delivery of valuable software.
- Changing requirements, even in late development, are welcome.
- Working software is delivered frequently (in weeks rather than months).
- The process depends on close, daily cooperation between business stakeholders and developers.
- Projects are built around motivated individuals, who should be trusted.
- Face-to-face conversation is the best form of communication (co-location).
- Working software is the principal measure of progress.
- Sustainable development requires being able to maintain a constant pace.
- Continuous attention is paid to technical excellence and good design.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- A team regularly reflects on how to become more effective and adjusts accordingly.

These core tenets were the main spark of the Agile movement. Mary Poppendieck and Tom Poppendieck wrote *Lean Software Development: An Agile Toolkit* in 2003, based on the principles of the Agile Manifesto and their many years of experience developing software. This book is still considered one of the best on the practical uses of Agile.

Developing software through Agile results in very different output than the slow serial manner used with Waterfall. With Waterfall, a project is not “finished” and deployable until the very end. With Agile, the time frame is changed: Agile uses smaller time increments (often 2 weeks), or “sprints,” that encompass the full process of analysis, design, code, and test but on a much smaller aspect of an application. The goal is to finish a feature or capability for each sprint, resulting in a potentially

shippable incremental piece of software. Therefore, with Agile, if you are 40% finished with a project, you have 100% usable code. Figure 2-4 shows how this process looks on a timeline.

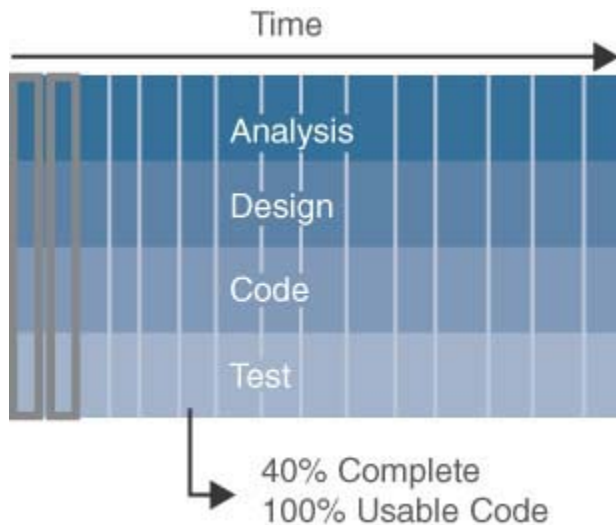


Figure 2-4 *Agile Development Practices*

By leveraging Agile, you can keep adding value immediately and nimbly adapt to change. If a new capability is needed in the software, or if a feature that was planned is determined to no longer be necessary, the project can pivot quickly and make those adjustments.

COMMON DESIGN PATTERNS

When creating software, you will often run into the same problem over and over again. You don't want to reinvent the wheel each time you need a rolling thing to make something move. In software engineering, many common design paradigms have already been created, and you can reuse them in your software project. These design patterns make you faster and provide tried-and-true solutions that have been tested and refined. The following sections introduce a couple of design patterns that are really useful for network automation projects: the Model-View-Controller (MVC) and Observer patterns. While there are many more that you may be

interested in learning about, these are the ones you will most likely see on the 200-901 DevNet Associate DEVASC exam.

Model-View-Controller (MVC) Pattern



The Model-View-Controller (MVC) pattern was one of the first design patterns to leverage the separation of concerns (SoC) principle. The SoC principle is used to decouple an application's interdependencies and functions from its other parts. The goal is to make the various layers of the application—such as data access, business logic, and presentation (to the end user)—modular. This modularity makes the application easier to construct and maintain while also allowing the flexibility to make changes or additions to business logic. It also provides a natural organization structure for a program that anyone can follow for collaborative development. If you have used a web-based application, more than likely the app was constructed using an MVC pattern.

Note

Numerous web frameworks use MVC concepts across many programming languages. Angular, Express, and Backbone are all written in JavaScript. Django and Flask are two very popular examples written in Python.

The classical MVC pattern has three main parts:

- **Model:** The model is responsible for retrieving and manipulating data. It is often tied to some type of database but could be data from a simple file. It conducts all data operations, such as select, insert, update, and delete operations. The model receives instructions from the controller.
- **View:** The view is what the end users see on the devices they are using to interact with the program. It could be a web page or text from the command line. The power of the view is that it can be tailored to any device and any representation without changing any of the business logic of the model. The view communicates with the controller by

sending data or receiving output from the model through the controller. The view's primary function is to render data.

- **Controller:** The controller is the intermediary between what the user sees and the backend logic that manipulates the data. The role of the controller is to receive requests from the user via the view and pass those requests on to the model and its underlying data store.

Figure 2-5 shows the interactions between components of the MVC pattern.

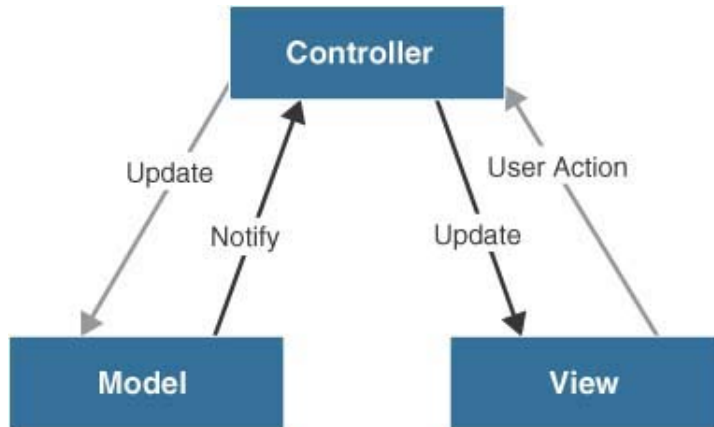


Figure 2-5 MVC Pattern Interactions

Observer Pattern



The Observer pattern was created to address the problem of sharing information between one object to many other objects. This type of pattern describes a very useful behavior for distributed systems that need to share configuration information or details on changes as they happen. The Observer pattern is actually very simple and consists of only two logical components (see [Figure 2-6](#)):

- **Subject:** The subject refers to the object state being observed—in other words, the data that is to be synchronized. The subject has a registration process that allows other components of an application or even remote systems to subscribe to the process. Once registered, a subscriber is sent an update notification whenever there is a change in the subject's data so that the remote systems can synchronize.
- **Observer:** The observer is the component that registers with the subject to allow the subject to be aware of the observer and how to

communicate to it. The only function of the observer is to synchronize its data with the subject when called. The key thing to understand about the observer is that it does not use a polling process, which can be very inefficient with a larger number of observers registered to a subject. Updates are push only.

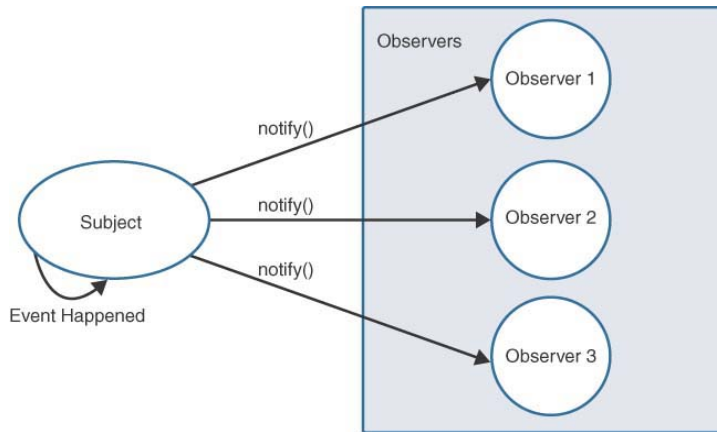


Figure 2-6 *Observer Pattern*

The Observer pattern is often used to handle communications between the model and the view in the MVC pattern. Say, for example, that you have two different views available to an end user. One view provides a bar graph, and the other provides a scatter plot. Both use the same data source from the model. When that data changes or is updated, the two views need to be updated. This is a perfect job for the Observer pattern.

LINUX BASH

Knowing how to use Linux BASH is a necessary skill for working with open-source technologies as well as many of the tools you need to be proficient with to be successful in the development world. Linux has taken over the development world, and even Microsoft has jumped into the game by providing the Windows Subsystem for Linux for Windows 10 pro. For the DEVASC exam, you need to know how to use BASH and be familiar with some of the key commands.

Getting to Know BASH



BASH is a shell, and a shell is simply a layer between a user and the internal workings of an operating system. A user can use the shell to input commands that the operating system will interpret and perform. Before graphical user interfaces (GUI) became common, the shell reigned supreme, and those who knew its intricacies were revered as some tech wizards. Today that skill is still in high demand, and without it you will find yourself struggling as the GUI simply doesn't make possible many of the powerful operations available through the shell.

While there are many shells you can use, BASH, which stands for Bourne Again Shell, is one of the most popular. It has been around since 1989 and is the default shell on most Linux operating systems. Until recently, it was also the default for the Mac operating system, but Apple has replaced BASH with Z shell. The commands and syntax you will learn with BASH are transferable to Z shell, as it was built to maintain compatibility with BASH.

BASH is not only a shell for command processing using standard Linux operating system commands. It can also read and interpret scripts for automation. These capabilities are beyond the scope of what you need to know for the DEVASC exam but would be worth looking into as you continue your journey as these automation scripts are where BASH really shines. Like all other UNIX shells, BASH supports features such as piping (which involves feeding output from one command as the input of another command), variables, evaluation of conditions, and iteration (repeated processing of a command with **if** statements). You also have a command

history as part of the shell, where you can use the arrow keys to cycle through and edit previous commands.

UNIX platforms such as Linux and OSX have built-in documentation for each command the operating system uses. To access help for any command, type **man** (short for manual) and then the command you are curious about. The output gives you a synopsis of the command, any optional flags, and required attributes. [Example 2-1](#) shows the man page for the **man** command.

Example 2-1 *Example of the Manual Page for the man Command*

[Click here to view code image](#)

```
$ man man

man(1)
man(1)

NAME

    man - format and display the on-line
    manual pages

SYNOPSIS

    man [-acdfFhkKtW] [--path] [-m
    system] [-p string] [-C config_file]
    [-M pathlist] [-P pager] [-B browser] [-
    H htmlpager] [-S section_list]
    [section] name ...

DESCRIPTION

    man formats and displays the on-line
    manual pages.  If you specify sec-
    tion, man only looks in that section of
    the manual.  name is normally
    the name of the manual page, which is
    typically the name of a command,
    function, or file.  However, if name
    contains a slash (/) then man
    interprets it as a file specification,
    so that you can do man ./foo.5
    or even man /cd/foo/bar.1.gz.

    See below for a description of where man
    looks for the manual page
    files.
```

```
<output cut for brevity>
```

Not every command is intended to be run with user-level privileges. You can temporarily upgrade your privileges by prepending the **sudo** command before you execute a command that needs higher-level access. You will often be prompted for a password to verify that you have the right to use **sudo**. You need to be careful when using **sudo**, as the whole idea of reduced privileges is to increase security and prevent average users from running commands that are dangerous. Use **sudo** only when required, such as when you need to kick off an update for your Linux distribution, which you do by using the **apt-get update** command:

```
$ sudo apt-get update
```

As mentioned earlier, one of the most powerful features of BASH is something called piping. This feature allows you to string together commands. For example, the **cat** command displays the contents of a file to the screen. What if a file contains too much to fit on a single screen? The **cat** command will happily spew every character of the file at the screen until it reaches the end, regardless of whether you could keep up with it. To address this, you can pipe the output of **cat** to the **more** command to stream the content from **cat** to **more**, which gives you a prompt to continue one page at a time. To use the piping functionality, you use the pipe character (|) between commands, as shown in [Example 2-2](#).

Example 2-2 *Output of the **cat** Command Piped to the **more** Command*

[Click here to view code image](#)

```
$cat weather.py | more  
  
import json
```

```

import urllib.request
from pprint import pprint

def get_local_weather():

    weather_base_url =
'http://forecast.weather.gov/MapClick.php?
FcstType=json&'

    places = {
        'Austin': ['30.3074624',
'-98.0335911'],
        'Portland': ['45.542094',
'-122.9346037'],
        'NYC': ['40.7053111', '-74.258188']
    }

    for place in places:
        latitude, longitude = places[place][0],
places[place][1]
        weather_url = weather_base_url + "lat="
+ latitude + "&lon=" + longitude
        # Show the URL we use to get the
weather data. (Paste this URL into your
browser!)
        # print("Getting the current weather
for", place, "at", weather_url, ":")

        page_response =
urllib.request.urlopen(weather_url).read()
<output cut for brevity>

```

Directory Navigation

A UNIX-based file system has a directory tree structure. The top of the tree is called the *root* (as it's an upside-down tree), and you use the forward slash (/) to refer to root. From root you have numerous directories, and under each directory you can have more directories or files. [Figure 2-7](#) shows a UNIX directory structure.

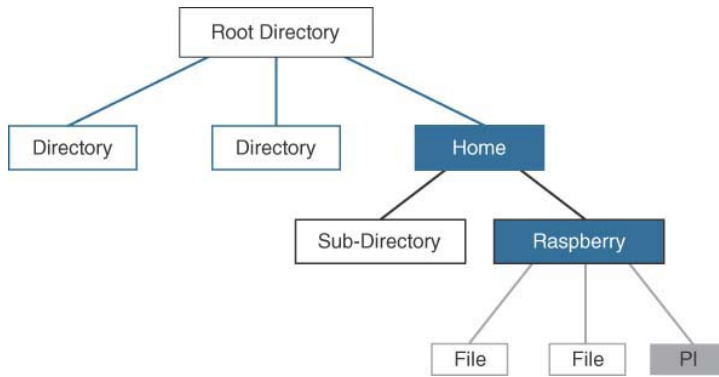


Figure 2-7 UNIX Directory Structure

Whenever you call a file, you have to supply its path. Everything you execute in UNIX is in relationship to root. To execute a file in the directory you are in, you can use **./filename.sh**, where the leading **.** is simply an alias for the current directory.

In addition to the root file system, each user has a home directory that the user controls and that stores the user's individual files and applications. The full path to the home directory often looks something like `/home/username` on Linux and `/Users/username` on Mac OS X, but you can also use the tilde shortcut (`~/`) to reference the home directory.

The following sections describe some of the commands most commonly used to interact with the BASH shell and provide examples of their options and use.

cd

The **cd** command is used to change directories and move around the file system. You can use it as follows:

<code>\$ cd /</code>	Changes directory to the root directory
<code>\$ cd /home/username</code>	Changes directory to the /home/username directory
<code>e</code>	

\$ cd test	Changes directory to the test folder
-------------------	--------------------------------------

\$ cd ..	Moves up one directory
-----------------	------------------------

pwd

If you ever get lost while navigating around the file system, you can use the **pwd** command to print out your current working directory path. You can use it as follows:

\$ pwd	Print your current working directory
---------------	--------------------------------------

ls

Once you have navigated to a directory, you probably want to know what is in it. The **ls** command gives you a list of the current directory. If you execute it without any parameters, it just displays whatever is in the directory. It doesn't show any hidden files (such as configuration files). Anything that starts with a **.** does not show up in a standard directory listing, and you need to use the **-a** flag to see all files, including hidden files. By using the **-l** flag, you can see permissions and the user and group that own the file or directory. You can also use the wildcard ***** to list specific filename values; for example, to find any files with test as a part of the name, you can use **ls *test***, which would match both **1test** and **test1**. You can use the **ls** command as follows:

\$ ls	Lists files and directories in the current working directory
--------------	--

\$ ls -a	Lists everything in the current directory, including hidden files
-----------------	---

\$ ls /home/username	Lists everything in the /home/username directory
---------------------------------------	--

\$ ls -l	Lists permissions and user and group ownership
-----------------	--

\$ ls -F	Displays files and directories and denotes which are which
-----------------	--

mkdir

To create a directory, you use the **mkdir** command. If you are in your home directory or in another directory where you have the appropriate permissions, you can use this command without **sudo**. You can use the **mkdir** command as follows:

\$ mkdir test	Makes a new directory called test in the current working directory if you have permission
----------------------	---

\$ mkdir /home/username test	Makes a new directory called test at /home/username/test
---	--

File Management

Working with files is easy with BASH. There are just a few commands that you will use often, and they are described in the following sections.

cp

The purpose of the **cp** command is to copy a file or folder someplace. It does not delete the source file but instead makes an identical duplicate. When editing configuration files or making changes that you may want to roll back, you can use the **cp** command to create a copy as a sort of

backup. The command requires several parameters: the name of the file you want to copy and where you want to copy it to and the name of the copy. When copying the contents of a folder, you need to use the **-r**, or recursive, flag. You can use the **cp** command as follows:

\$ cp sydney.txt sydney2.txt	Copies a file called sydney.txt from the current directory and names the copy sydney2.txt
\$ cp /home/username/ sydney.txt ~/sydney2.txt	Copies a file as described above but using the full path and the home directory path
\$ cp -r folder folder.old	Copies a folder

mv

The **mv** command allows you to move a file or folder from one directory to another, and it is also used to rename files or folders from the command line, as BASH does not have a dedicated renaming function. The **mv** command takes a source and destination, just as **cp** does. You can use the **-i** flag to create an interactive option prompt when moving files that exist at the destination. The **-f** flag forces the move and overwrites any files at the destination. Wildcards also work to select multiple source files or directories. You can use the **mv** command as follows:

\$ mv caleb.txt calebfinal.txt	Renames a file called caleb.txt to calebfinal.txt
\$ mv /home/username/cal	Renames a file as described above but using full paths

```
eb.txt
~/calebfinal.txt
```

```
$ mv -i *           Moves all files and directories in
/home/username/new the current folder to a directory
w/                 called new
```

rm

To delete a file or directory, you use the **rm** command. If the item you are deleting is a file or an empty directory, you just need to supply the name and press Enter. On the other hand, if you try to delete a directory that has files in it, **rm** tells you that the directory is not empty. In that case, you can use the **-rf** flag to force the deletion. You can use the **rm** command as follows:

```
$ rm test.txt      Deletes the file test.txt in the current working
                  directory
```

```
$ rm -rf test     Forces the deletion of the folder test and
                  everything in it
```

touch

The **touch** command is used to create a file and/or change the timestamps on a file's access without opening it. This command is often used when a developer wants to create a file but doesn't want to put any content in it. You can use the **touch** command as follows:

```
$ touch emptyfile.txt  Creates an empty file named
                       emptyfile.txt
```

```
$ touch file{1..20}.txt Bulk creates files from file1.txt to
                       file20.txt
```

cat

The **cat** (which stands for *concatenate*) command allows you to view or create files and also pipe to other commands. It's one of the most useful commands in UNIX when it comes to working with files. You can use the **cat** command as follows:

\$cat file1.txt	Displays the contents of file1.txt
\$cat file1.txt more	Displays the contents of file1.txt and pipes the output to more to add page breaks
\$cat >file2.txt	Sends a user's typed or copied content from the command line to file2.txt

Environment Variables

BASH environment variables contain information about the current session. Environment variables are available in all operating systems and are typically set when you open your terminal from a configuration file associated with your login. You set these variables with similar syntax to how you set them when programming. You do not often use these variables directly, but the programs and applications you launch do. You can view all of your currently set environment variables by entering the **env** command. Since there can be more entries than you have room to display on a single terminal page, you can pipe the results to the **more** command to pause between pages:

\$env more	Shows all environment variables with page breaks
-------------------------	--

If you execute this command, you are likely to notice a lot of keywords with the = sign tied to values. One environment variable that you use every time you execute a command is the PATH variable. This is where your shell looks for executable files. If you add a new command and can't execute it, more than likely the place where the command was copied is not listed in your PATH. To view any variable value, you can use the **echo** command and the variable you want to view. You also need to tell BASH that it's a variable by using the \$ in front of it. Here's an example:

[Click here to view code image](#)

```
$ echo $PATH

/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/VMware
Fusion.app/Contents/Public:/opt/X11/bin
```

To add a new value to the PATH variable, you can't just type **\$PATH=/new_directory** because the operating system reads the environment variables only when the terminal session starts. To inform Linux that an environment variable needs to be updated, you use the **export** command. This command allows you to append your additional path to BASH and exists for the duration of the session. Make sure you add the : or , in front of your new value, depending on your operating system. The following example is for a Linux-style OS:

[Click here to view code image](#)

```
$ export PATH=$PATH:/Home/chrijack/bin
```

When you end your terminal session, the changes you made are not saved. To retain the changes, you need to write the path statement to your **.bashrc** (or **.zshrc** if using Z shell) profile settings. Anything written here will be available anytime you launch a terminal. You can simply copy, add the previous command to the end of the

.bashrc with your favorite text editor, or use the following command:

[Click here to view code image](#)

```
$ echo "export PATH=$PATH:/Home/chrijack/bin" >>
.bashrc
```

This addition becomes active only after you close your current session or force it to reload the variable. The **source** command can be used to reload the variables from the hidden configuration file **.bashrc**:

```
$ source ~/.bashrc
```

The following command does the same thing as the previous one because **.** is also an alias for the **source** command:

```
$ . ~/.bashrc
```

There are many more tricks you can uncover with BASH. You will get plenty of chances to use it as you study for the DEVASC exam.

SOFTWARE VERSION CONTROL



The term *version control* is used to describe the process of saving various copies of a file or set of files in order to track changes made to those files. This description highlights how incredibly useful it is to the world of programming. Software version control (SVC) typically involves a database that stores current and historical versions of source code to allow multiple people or teams to work on it at the same time. If a mistake is made and you want to go back a revision (or to any previous version), SVC is the answer. You may also hear it called

revision control or source control, but it all falls under the topic of software configuration management. Once you have used an SVC, you will quickly become a believer in its power and won't want to go back to using manual processes again.

If you have a team of developers working on a project, all writing new code or changing previously written code, losing those files could set you back weeks or months. Version control can protect your code by allowing changes to be checked in (through a process known as a *code commit*) to a hierarchical tree structure of folders with files in them. You often don't know what a developer might need to change or has changed, but the version control system does. Each check-in is tagged with who made the change and what the person changed within the code. Instead of using inefficient techniques such as file locking, a version control system handles concurrent check-ins, allowing two programmers to commit code at the same time.

Another aspect of a version control system is the ability to branch and merge code built independently. This is very useful if you are writing code on a part of an application that could conflict with another part written by another team. By creating a branch, you effectively create a separate work stream that has its own history and does not impact the main "trunk" of the code base. Once the code is written and any conflicts are resolved, the code from the branch can be merged back into the main trunk. Many application developers use this technique for new features or application revisions.

Can you write software without a version control system? Sure, but why would you? A lot of version control software options are available, many of them free, and it is good practice to always use a version control system to store your code. Git is one of the most commonly used version control systems today, and the 200-901 DevNet

Associate DEVASC exam will test your knowledge of it, so the next section covers how to use Git.

GIT

If you are working with version control software, chances are it is Git. A staggering number of companies use Git, which is free and open source. In 2005, Linus Torvalds (the father of Linux) created Git as an alternative to the SCM system BitKeeper, when the original owner of BitKeeper decided to stop allowing free use of the system for Linux kernel development. With no existing open-source options that would meet his needs, Torvalds created a distributed version control system and named it Git. Git was created to be fast and scalable, with a distributed workflow that could support the huge number of contributors to the Linux kernel. His creation was turned over to Junio Hamano in 2006, and it has become the most widely used source management system in the world.

Note

GitHub is not Git. GitHub is a cloud-based social networking platform for programmers that allows anyone to share and contribute to software projects (open source or private). While GitHub uses Git as its version control system underneath the graphical front end, it is not directly tied to the Git open-source project (much as a Linux distribution, such as Ubuntu or Fedora, uses the Linux kernel but is independently developed from it).

Understanding Git

Git is a distributed version control system built with scalability in mind. It uses a multi-tree structure, and if you look closely at the design, you see that it looks a lot like a file system. (Linus Torvalds is an operating system

creator after all.) Git keeps track of three main structures, or trees (see [Figure 2-8](#)):

- **Local workspace:** This is where you store source code files, binaries, images, documentation, and whatever else you need.
- **Staging area:** This is an intermediary storage area for items to be synchronized (changes and new items).
- **Head, or local repository:** This is where you store all committed items.



Figure 2-8 *Git Tree Structure*

Another very important concept with Git is the file lifecycle. Each file that you add to your working directory has a status attributed to it. This status determines how Git handles the file. [Figure 2-9](#) shows the Git file status lifecycle, which includes the following statuses:

- **Untracked:** When you first create a file in a directory that Git is managing, it is given an untracked status. Git sees this file but does not perform any type of version control operations on it. For all intents and purposes, the file is invisible to the rest of the world. Some files, such as those containing settings or passwords or temporary files, may be stored in the working directory, but you may not want to include them in version control. If you want Git to start tracking a file, you have to explicitly tell it to do so with the **git add** command; once you do this, the status of the file changes to tracked.
- **Unmodified:** A tracked file in Git is included as part of the repository, and changes are watched. This status means Git is watching for any file changes that are made, but it doesn't see any yet.
- **Modified:** Whenever you add some code or make a change to the file, Git changes the status of the file to modified. Modified status is where Git sees that you are working on the file but you are not finished. You have to tell Git that you are ready to add a changed (modified) file to the index or staging area by issuing the **git add** command again.
- **Staged:** Once a changed file is added to the index, Git needs to be able to bundle up your changes and update the local repository. This process is called staging and is accomplished through **git commit**. At this point, your file status is moved back to the tracked status, and it stays there until you make changes to the file in the future and kick off the whole process once again.

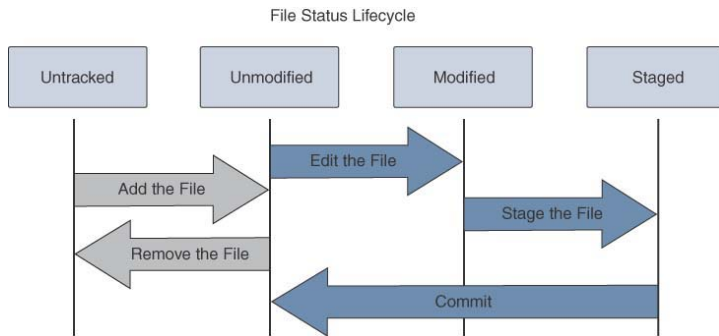


Figure 2-9 *Git File Status Lifecycle*

If at any point you want to see the status of a file from your repository, you can use the extremely useful command **git status** to learn the status of each file in your local directory.

You can pull files and populate your working directory for a project that already exists by making a clone. Once you have done this, your working directory will be an exact match of what is stored in the repository. When you make changes to any source code or files, you can add your changes to the index, where they will sit in staging, waiting for you to finish all your changes or additions. The next step is to perform a commit and package up the changes for submission (or pushing) to the remote repository (usually a server somewhere local or on the Internet). This high-level process uses numerous commands that are covered in the next section. If you understand Git's tree structure, figuring out what command you need is simple. [Figure 2-10](#) shows the basic Git workflow.

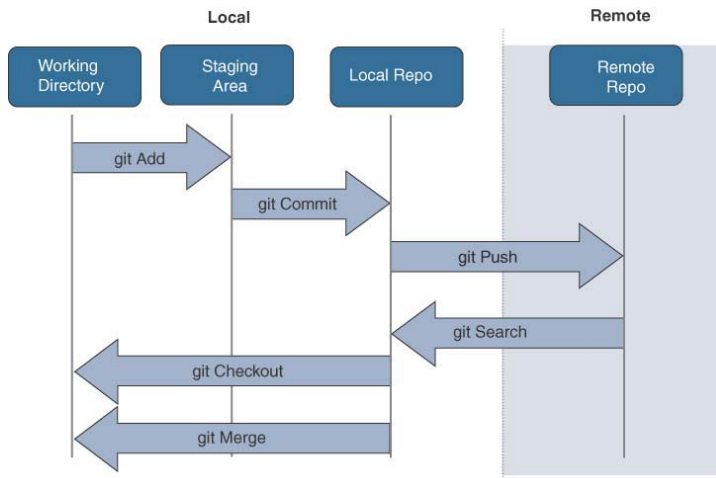


Figure 2-10 *Git Workflow*

Using Git



Git may not come natively with your operating system. If you are running a Linux variation, you probably already have it. For Mac and Windows you need to install it. You can go to the main distribution website (<https://git-scm.com>) and download builds for your operating system directly. You can install the command-line version of Git and start using and practicing the commands discussed in this section. There are also GUI-based Git clients, but for the purposes of the DEVASC exam, you should focus your efforts on the command line. Git commands come in two different flavors. The standard user-friendly commands are called “porcelain,” and the more complicated inner workings of Git manipulating commands are called “plumbing.” At its core, Git is a content-addressable file system. The version control system part was layered on top to make it easier to use. For the DEVASC exam, you need to know your way around Git at a functional level (by using the porcelain). There is a significant amount of manipulation you can do with Git at the plumbing level. Most of the

plumbing commands and tools are not ones you will be using on a regular basis and are not covered on the exam.

Cloning/Initiating Repositories

Git operates on a number of processes that enable it to do its magic. The first of these processes involves defining a local repository by using either **git clone** or **git init**. The **git clone** command has the following syntax:

[Click here to view code image](#)

```
git clone (url to repository) (directory to clone to)
```

If there is an existing repository you are planning to start working on, like one from GitHub that you like, you use **git clone**. This command duplicates an existing Git project from the URL provided into your current directory with the name of the repository as the directory name. You can also specify a different name with a command-line option. [Example 2-3](#) shows an example of cloning a repository and listing the files in the newly created local repository.

Example 2-3 *Cloning a GitHub Repository*

[Click here to view code image](#)

```
#git clone
https://github.com/CiscoDevNet/pyats-coding-
101.git
Cloning into 'pyats-coding-101'...
remote: Enumerating objects: 71, done.
remote: Total 71 (delta 0), reused 0 (delta 0),
pack-reused 71
Unpacking objects: 100% (71/71), done.
#cd pyats-coding-101
#pyats-coding-101 git:(master) ls
COPYRIGHT          coding-102-parsers
LICENSE            coding-103-
yaml
README.md         coding-201-
advanced-parsers
```

```
coding-101-python
git init (directory name)
```

To create a completely new repository, you need to create a directory. Luckily, **git init** can be supplied with a directory name as an option to do this all in one command:

[Click here to view code image](#)

```
#git init newrepo
Initialized empty Git repository in
/Users/chrijack/Documents/
GitHub/newrepo/.git/

#newrepo git:(master)
```

What you just created is an empty repository, and you need to add some files to it. By using the **touch** command, you can create an empty file. The following example shows how to view the new file in the repository with the directory (**ls**) command:

[Click here to view code image](#)

```
#newrepo git:(master) touch newfile
#newrepo git:(master) ls
newfile
```

Once the file is added, Git sees that there is something new, but it doesn't do anything with it at this point. If you type **git status**, you can see that Git identified the new file, but you have to issue another command to add it to index for Git to perform version control on it. Here's an example:

[Click here to view code image](#)

```
# git status

On branch master
```

```
No commits yet

Untracked files:
  (use "git add <file>..." to include in what
  will be committed)

    newfile

nothing added to commit but untracked files
present (use "git add"
to track)
```

Git is helpful and tells you that it sees the new file, but you need to do something else to enable version control and let Git know to start tracking it.

Adding and Removing Files

When you are finished making changes to files, you can add them to the index. Git knows to then start tracking changes for the files you identified. You can use the following commands to add files to an index:

- **git add . or -A:** Adds everything in the entire local workspace.
- **git add (filename):** Adds a single file.

The **git add** command adds all new or deleted files and directories to the index. Why select an individual file instead of everything with the **.** or **-A** option? It comes down to being specific about what you are changing and adding to the index. If you accidentally make a change to another file and commit everything, you might unintentionally make a change to your code and then have to do a rollback. Being specific is always safest. You can use the following commands to add the file `newfile` to the Git index (in a process known as *staging*):

[Click here to view code image](#)

```
# git add newfile
# git status
On branch master
```

```
No commits yet

Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

    new file:   newfile
```

Removing files and directories from Git is not as simple as just deleting them from the directory itself. If you just use file system commands to remove files, you may create headaches as the index can become confused. You can remove files and directories from Git, but it requires an extra step of adding the file deletion to the index (which sounds counterintuitive, right?). The best way is to use the **git rm** command, which has the following syntax:

[Click here to view code image](#)

```
git rm (-r) (-f) (folder/file.py)
```

This command removes a file or directory and syncs it with the index in one step. If you want to remove a directory that is not empty or has subdirectories, you can use the **-r** option to remove recursively. In addition, if you add a file to Git and then decide that you want to remove it, you need to use the **-f** option to force removal from the index. This is required only if you haven't committed the changes to the local repository. Here is an example:

```
# touch removeme.py
# git add .
# ls
newfile    removeme.py
# git rm -f removeme.py
rm 'removeme.py'
```

git mv is the command you use to move or rename a file, directory, or symbolic link. It has the following syntax:

[Click here to view code image](#)

```
git mv (-f) (source) (destination)
```

For this command you supply a source argument and a destination argument to indicate which file or directory you want to change and where you want to move it. (Moving in this case is considered the same as renaming.) Keep in mind that when you use this command, it also updates the index at the same time, so there is no need to issue **git add** to add the change to Git. You can use the **-f** argument if you are trying to overwrite an existing file or directory where the same target exists. The following example shows how to change a filename in the same directory:

[Click here to view code image](#)

```
# ls
oldfile.py
# git mv oldfile.py newfile.py
# ls
newfile.py
```

Committing Files

When you commit a file, you move it from the index or staging area to the local copy of the repository. Git doesn't send entire updates; it sends just changes. The **commit** command is used to bundle up those changes to be synchronized with the local repository. The command is simple, but you can specify a lot of options and tweaks. In its simplest form, you just need to type **git commit**. This command has the following syntax:

[Click here to view code image](#)

```
git commit [-a] [-m] <"your commit message">
```

The **-a** option tells Git to add any changes you make to your files to the index. It's a quick shortcut instead of

using **git add -A**, but it works only for files that have been added at some point before in their history; new files need to be explicitly added to Git tracking. For every commit, you will need to enter some text about what changed. If you omit the **-m** option, Git automatically launches a text editor (such as **vi**, which is the default on Linux and Mac) to allow you to type in the text for your commit message. This is an opportunity to describe the changes you made so others know what you did. It's tempting to type in something silly like "update" or "new change for a quick commit," but don't fall into that trap. Think about the rest of your team. Here is an example of the **commit** command in action:

[Click here to view code image](#)

```
# git commit -a -m "bug fix 21324 and 23421"
[master elfec3d] bug fix 21324 and 23421
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 newfile
```

Note

As a good practice, use the first 50 characters of the commit message as a title for the commit followed by a blank line and a more detailed explanation of the commit. This title can be used throughout Git to automate notifications such as sending an email update on a new commit with the title as the subject line and the detailed message as the body.

Pushing and Pulling Files

Up until this point in the chapter, you have seen how Git operates on your local computer. Many people use Git in just this way, as a local version control system to track documents and files. Its real power, however, is in its distributed architecture, which enables teams from around the globe to come together and collaborate on projects.

In order to allow Git to use a remote repository, you have to configure Git with some information so that it can find it. When you use the command **git clone** on a repository, Git automatically adds the remote repository connection information via the URL entered with the **clone** command.

When using the **git init** command, however, you need to make sure that you enter the information to find the remote location for the server with the **git remote add** command, which has the following syntax:

```
git remote add (name) (url)
```

git remote -v can be used to show which remote repository is configured. The following example shows how to add a remote repository and then display what is configured:

[Click here to view code image](#)

```
# git remote add origin
https://github.com/chrijack/devnetccna.git
# git remote -v
origin
https://github.com/chrijack/devnetccna.git (fetch)
origin
https://github.com/chrijack/devnetccna.git (push)
```

What if you make a mistake or want to remove remote tracking of your repository? This can easily be done with the **git remote rm** command, which has the following syntax:

```
git remote rm (name)
```

Here is an example of this command in action:

```
# git remote rm origin
# git remote -v
```

In order for your code to be shared with the rest of your team or with the rest of the world, you have to tell Git to sync your local repository to the remote repository (on a shared server or service like GitHub). The command **git push**, which has the following syntax, is useful in this case:

[Click here to view code image](#)

```
git push (remotename) (branchname)
```

This command needs a remote name, which is an alias used to identify the remote repository. It is common to use the name **origin**, which is the default if a different name is not supplied. In addition, you can reference a branch name with **git push** in order to store your files in a separately tracked branch from the main repository. (You can think of this as a repository within a repository.) The sole purpose of the **git push** command is to transfer your files and any updates to your Git server. The following is an example of the **git push** command in use:

[Click here to view code image](#)

```
# git push origin master

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 210 bytes | 210.00
KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/chrijack/devnetccna.git
 * [new branch]      master -> master

Branch 'master' set up to track remote branch
'master' from 'ori-
gin'.
```

The command **git pull** syncs any changes that are on the remote repository and brings your local repository up to the same level as the remote one. It has the following syntax:

[Click here to view code image](#)

```
git pull (remotename) (branchname)
```

Whenever you begin to work with Git, one of the first commands you want to issue is **pull** so you can get the latest code from the remote repository and work with the latest version of code from the master repository. **git pull** does two things: fetches the latest version of the remote master repository and merges it into the local repository. If there are conflicts, they are handled just as they would be if you issued the **git merge** command, which is covered shortly. [Example 2-4](#) shows an example of using the **git pull** command.

Example 2-4 **git pull** Command

[Click here to view code image](#)

```
# git pull origin master
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 0 (delta 0),
pack-reused 0
Unpacking objects: 100% (8/8), done.
From https://github.com/chrijack/devnetccna
 * branch          master      -> FETCH_HEAD
    8eb16e3..40aaf1a  master    ->
origin/master
Updating 8eb16e3..40aaf1a
Fast-forward
 2README.md          | 3 +++
 Picture1.png        | Bin 0 -
 > 83650 bytes
 Picture2.jpg         | Bin 0 -
 > 25895 bytes
 Picture3.png         | Bin 0 -
 > 44064 bytes
 4 files changed, 3 insertions(+)
 create mode 100644 2README.md
 create mode 100644 Picture1.png
```

```
create mode 100644 Picture2.jpg
create mode 100644 Picture3.png
```

Working with Branches

Branches are an important workflow in software development. Say you want to add a new feature to your software or want to fix a bug. You can create a branch in order to add a separate development workspace for your project and prevent changes from destabilizing the main project (the master branch in Git). Remember that Git keeps a running history of every commit you make. This history (called a *snapshot* in Git terminology) details all the changes to the software over time and ensures the integrity of this record by applying an SHA-1 hash. This hash is a 40-character string that is tied to each and every commit. [Example 2-5](#) shows an example of three commits with a hash, displayed using the **git log** command.

Example 2-5 git log Command Output

[Click here to view code image](#)

```
#git log

commit 40aaf1af65ae7226311a01209b62ddf7f4ef88c2
(HEAD -> master, origin/master)
Author: Chris Jackson <chrijack@cisco.com>
Date:   Sat Oct 19 00:00:34 2019 -0500

    Add files via upload

commit 1a9db03479a69209bf722b21d8ec50f94d727e7d
Author: Chris Jackson <chrijack@cisco.com>
Date:   Fri Oct 18 23:59:55 2019 -0500

    Rename README.md to 2README.md

commit 8eb16e3b9122182592815fa1cc029493967c3bca
Author: Chris Jackson <chrijack@me.com>
Date:   Fri Oct 18 20:03:32 2019 -0500

    first commit
```

Notice that the first entry is the current commit state, as it is referenced by HEAD. The other entries show the chronological history of the commits. [Figure 2-11](#) shows a visual representation of this simple three-step commit history; for brevity, only the first four values of the hash are used.

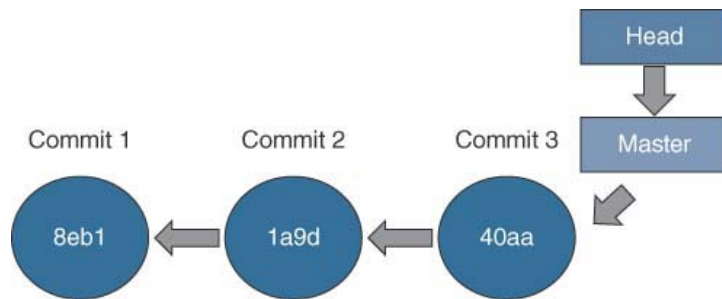


Figure 2-11 *Git Commit History*

To add a Git branch, you simply issue the **git branch** command and supply the new branch with a name, using the following syntax:

[Click here to view code image](#)

```
git branch (-d) <branchname> [commit]
```

You can alternatively specify a commit identified by a tag or commit hash if you want to access a previous commit from the branch history. By default, Git selects the latest commit. In addition, you can delete a branch when you no longer need it but using the **-d** argument. The following example shows how to create a branch and display the current branches with the **git branch** command with no argument:

```
# git branch newfeature  
# git branch  
* master  
  newfeature
```

The * next to **master** shows that the branch you are currently in is still master, but you now have a new branch named **newfeature**. Git simply creates a pointer to the latest commit and uses that commit as the start for the new branch. [Figure 2-12](#) shows a visual representation of this change.

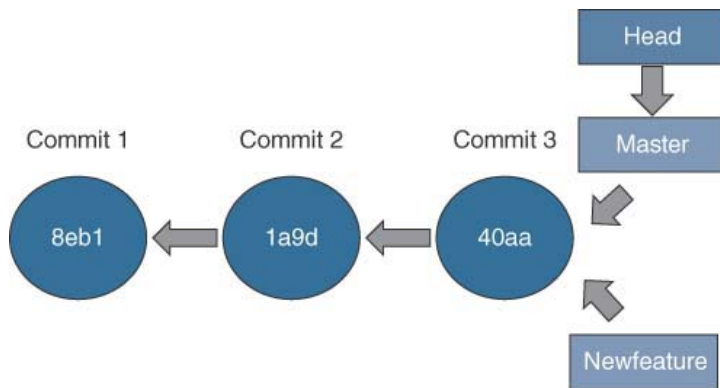


Figure 2-12 *Adding a Branch*

In order to move to the new branch and change your working directory, you have to use the **git checkout** command, which has the following syntax:

[Click here to view code image](#)

```
git checkout [-b] (branchname or commit)
```

The **-b** argument is useful for combining the **git branch** command with the checkout function and saves a bit of typing by creating the branch and checking it out (switching to it) all at the same time. This example moves Head on your local machine to the new branch, as shown in [Figure 2-13](#):

[Click here to view code image](#)

```
#git checkout newfeature  
  
Switched to branch 'newfeature'
```

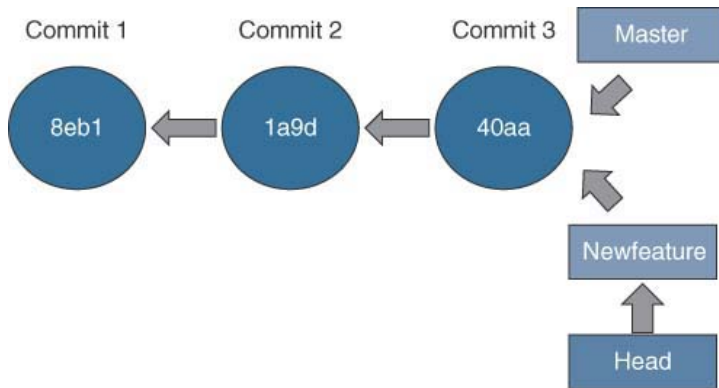


Figure 2-13 *Changing Head to a New Branch*

Now you have a separate workspace where you can build your feature. At this point, you will want to perform a **git push** to sync your changes to the remote repository. When the work is finished on the branch, you can merge it back into the main code base and then delete the branch by using the command **git branch -d (branchname)**.

Merging Branches

The merge process in Git is used to handle the combining of multiple branches into one. The **git merge** command is used to make this easier on the user and provide a simple way to manage the changes. It has the following syntax:

[Click here to view code image](#)

```
git merge (branch to merge with current)
```

To understand the merge process, it helps to take a look at your two branches. Figure 12-14 shows all of the commits that have taken place as part of the feature build. In addition, you can see other commits that have also occurred on the master branch.

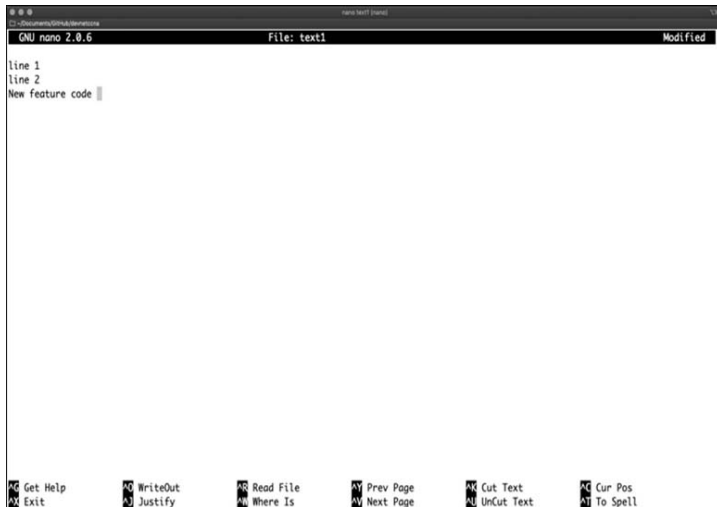


Figure 2-16 *Changes to the Text File in the newfeature Branch*

On the newfeature branch, you can issue the following commands to add the changes to the index and then commit the change:

[Click here to view code image](#)

```
#git add .  
#git commit -a -m "new feature"
```

Now the branch is synced with the new changes, and you can switch back to the master branch with the following command:

[Click here to view code image](#)

```
#git checkout master  
Switched to branch 'master'
```

From the master branch, you can then issue the **git merge** command and identify the branch to merge with (in this case, the newfeature branch):

[Click here to view code image](#)

```
# git merge newfeature
Updating 77f786a..dd6bce5
Fast-forward
 text1 | 1 +
 1 file changed, 1 insertion(+)
```

In this very simple example, no changes were made to the master branch, and Git was able to automatically merge the two branches and create a new combined commit that had the new content from the newfeature branch. Notice that the output above says “Fast-forward”; this refers to updating past the changes in the branch, which is much like fast-forwarding through the boring parts of a movie. At this point, you can delete the branch newfeature, as the code in it has been moved to master. [Figure 2-17](#) illustrates how this is done: Git creates a new commit that has two sources.

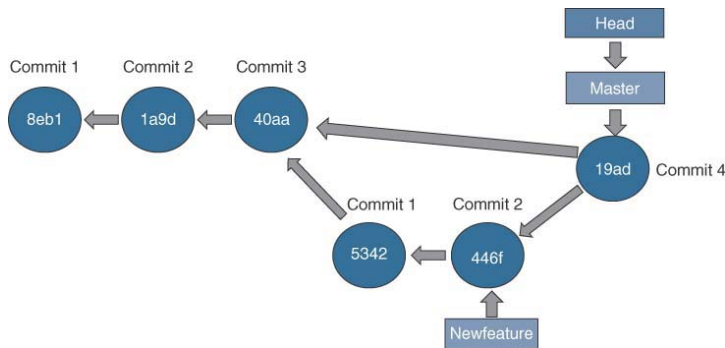


Figure 2-17 *Git Merge Between Two Branches*

Handling Conflicts

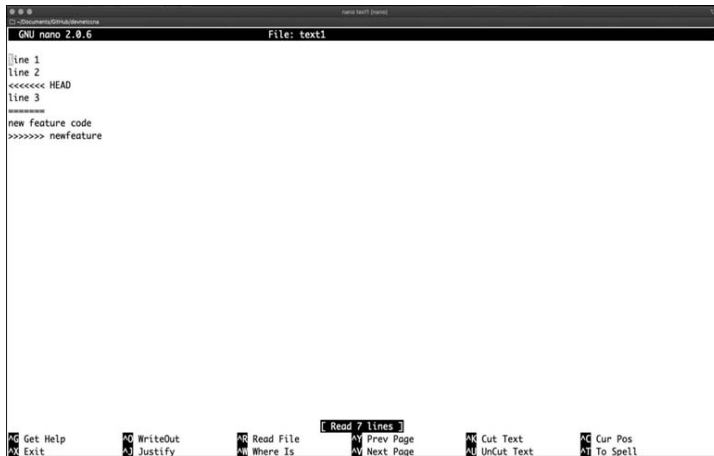
Merging branches is a very useful capability, but what happens if the same file is edited by two different developers? You can have a conflict in terms of which change takes precedence. Git attempts to handle merging automatically, but where there is conflict, Git relies on human intervention to decide what to keep. In the previous example, if there had been changes made to text1 in both the master branch and the newfeature

branch, you would have seen the following message after using the command **git merge**:

[Click here to view code image](#)

```
#git merge newfeature
Auto-merging text1
CONFLICT (content): Merge conflict in text1
Automatic merge failed; fix conflicts and then
commit the result.
```

In addition, text1 would look as shown in [Figure 2-18](#) (which shows the conflicting merge).



```
GNU nano 2.0.6 File: text1
line 1
line 2
<<<<<< HEAD
line 3
~~~~~
new feature code
>>>>> newfeature

Get Help  WriteOut  Read File  Read 7 Lines  Cut Text  Cur Pos
Exit      Justify   Where Is  Prev Page Next Page UnCut Text At To Spell
```

Figure 2-18 *Git Conflicting Merge*

Git shows you that “line 3” was added to text1 on the master branch and “new feature code” was added to text1 on the newfeature branch. Git is letting you delete one or keep both. You can simply edit the file, remove the parts that Git added to highlight the differences, and save the file. Then you can use **git add** to index your changes and **git commit** to save to the local repository, as in the following example:

[Click here to view code image](#)

```
#git add .  
#git commit -m "merge conflict fixed"  
[master fe8f42d] merge conflict fixed
```

Comparing Commits with diff

The **diff** command is one of the most powerful Git tools. It allows you to compare files and text to see which you want to use if there are multiple options. The ability to compare specific commits in Git makes it easier to know what to keep and what to discard between two similar versions of code.

The **diff** command takes two sets of inputs and outputs the differences or changes between them. This is its syntax:

[Click here to view code image](#)

```
git diff [--stat] [branchname or commit]
```

git diff looks at the history of your commits, individual files, branches, and other Git resources. It's a very useful tool for troubleshooting issues as well as comparing code between commits. It has a lot of options and command-line parameters, which makes it a bit of a Swiss Army knife in terms of functionality. One of the most useful functions of **diff** is to be able to see the differences between the three Git tree structures. The following are variations of the **git diff** command that you can use:

- **git diff:** This command highlights the differences between your working directory and the index (that is, what isn't yet staged).
- **git diff --cached:** This command shows any changes between the index and your last commit.
- **git diff HEAD:** This command shows the differences between your most recent commit and your current working directory. It is very useful for seeing what will happen with your next commit.

The following is an example of executing **git diff --cached** after **text2** is added to the index:

[Click here to view code image](#)

```
#git diff --cached
diff --git a/text2 b/text2
new file mode 100644
index 0000000..b9997e5
--- /dev/null
+++ b/text2
@@ -0,0 +1 @@
+new bit of code
```

git diff identified the new file addition and shows the a/b comparison. Since this is a new file, there is nothing to compare it with, so you see **--- /dev/null** as the *a* comparison. In the *b* comparison, you see **+++ b/text2**, which shows the addition of the new file, followed by stacks on what was different. Since there was no file before, you see **-0,0** and **+1**. (The + and - simply denote which of the two versions you are comparing. It is not actually a **-0**, which would be impossible.) The last line shows the text that was added to the new file. This is a very simple example with one line of code. In a big file, you might see a significant amount of text.

Another very useful capability of **git diff** is to compare branches. By using **git diff (branchname)**, you can see the differences between a file in the branch you are currently in and one that you supply as an argument. The following compares text1 between the branches master and newfeature, where you can see that line 3 is present on newfeature branch's text1 file:

[Click here to view code image](#)

```
#git diff newfeature text1
diff --git a/text1 b/text1
index 45c2489..ba0a07d 100644
--- a/text1
+++ b/text1
```

```
@@ -1,3 +1,4 @@  
line 1  
line 2  
+line 3  
new feature code
```

This section has covered quite a bit of the syntax and commands that you will see on a regular basis when working with Git. You need to spend some time working with these commands on your local machine and become familiar with them so you will be ready for the 200-901 DevNet Associate DEVASC exam. Make sure you are using resources such as Git documentation on any command you don't understand or for which you want to get deeper insight. All of these commands have a tremendous amount of depth for you to explore.

CONDUCTING CODE REVIEW



Every good author needs an editor. This book wouldn't have been even half as understandable if it hadn't been for the fact that we had other people check our work for comprehension and technical accuracy. Why should code you write be treated any differently? The intent behind a code review process is to take good code and make it better by showing it to others and having them critique it and look for potential errors. When you develop software, the vast majority of your time is spent by yourself—just you and the keyboard. Sometimes when you are this close to a software project, you miss errors or use ineffective coding techniques; a simple code review can quickly uncover such issues.

Beyond the aspects mentioned above, why should you conduct code reviews? The following are a few common benefits of code review:

- It helps you create higher-quality software.
- It enables your team to be more cohesive and deliver software projects on time.
- It can help you find more defects and inefficient code that unit tests and functional tests might miss, making your software more reliable.

There are many ways to conduct code reviews. Some organizations use specialized applications such as Gerrit, and others conduct reviews as if they were professors grading college papers. Whatever process you use, the following are some good practices to help make your code review effective:

- Use a code review checklist that includes organization-specific practices (naming conventions, security, class structures, and so on) and any areas that need special consideration. The goal is to have a repeatable process that is followed by everyone.
- Review the code, not the person who wrote it. Avoid being robotic and harsh so you don't hurt people's feeling and discourage them. The goal is better code, not disgruntled employees.
- Keep in mind that code review is a gift. No one is calling your baby ugly. Check your ego at the door and listen; the feedback you receive will make you a better coder in the long run.
- Make sure the changes recommended are committed back into the code base. You should also share findings back to the organization so that everyone can learn from mistakes and improve their techniques.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 2-2](#) lists these key topics and the page number on which each is found.



Table 2-2 Key Topics

Key Topic Element	Description	Page Number
Paragraph	Waterfall	27
Paragraph	Lean	28
Paragraph	Agile	29
Paragraph	Model-View-Controller (MVC) pattern	30
Section	Observer Pattern	31
Paragraph	Getting to Know BASH	32
Paragraph	Software version control	38
Section	Using Git	42
Section	Conducting Code Review	55

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Software Development Lifecycle (SDLC)

Waterfall

Lean

Agile

Model-View-Controller (MVC)

Observer

software version control

Git

GitHub

repository

staging

index

local workspace

Chapter 3

Introduction to Python

This chapter covers the following topics:

- **Getting Started with Python:** This section covers what you need to know when using Python on your local machine.
- **Understanding Python Syntax:** This section describes the basic Python syntax and command structure.
- **Data Types and Variables:** This section describes the various types of data you need to interact with when coding.
- **Input and Output:** This section describes how to get input from a user and print out results to the terminal.
- **Flow Control with Conditionals and Loops:** This section discusses adding logic to your code with conditionals and loops.

Python is an easy language that anyone can learn quickly. It has become the de facto language for custom infrastructure automation. Thanks to its English-like command structure, readability, and simple programming syntax, you will find that you can accomplish your goals more quickly with Python than with languages such as C or Java. While you are not expected to be an expert in Python for the 200-901 DevNet Associate DEVASC exam, you do need to be fluent enough to understand what is going on in a sample of Python code. You also need to be able to construct Python code by using samples from DevNet and GitHub to interact with Cisco products. While the next few chapters are not intended to replace a deep dive into Python programming, they serve as a starting point for success on the exam.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 3-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 3-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Getting Started with Python	1, 2
Understanding Python Syntax	3, 4
Data Types and Variables	5, 6
Input and Output	7, 8
Flow Control with Conditionals and Loops	9, 10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. What is the appropriate way to create a virtual environment for Python 3?

1. `python3 -virtual myvenv`
2. `python3 virtual myvenv`
3. `python3 -m vrt myvenv`
4. `python3 -m venv myvenv`

2. What command is used to install Python modules from PyPI?

1. `pip load packagename`
2. `pip install packagename`
3. `python3 -m pip install packagename`
4. `python3 -t pip install packagename`

3. What is the standard for indention in Python?

1. One space for each block of code
2. Four spaces for each block of code
3. One tab for each block of code
4. One tab and one space per block of code

4. How are comments in Python denoted?

1. `//` on each line you want to make a comment
2. `#` or `'''` quotation marks encompassing multiline comments
3. `/*` comment `*/`
4. `@$` comment `%@`

5. Which of the following are mutable data types?
(Choose two.)

1. Lists
2. Dictionary
3. Integers
4. Tuples

6. Which of the following would create a dictionary?

1. `a= (" name","chris","age",45)`
2. `a= dict()`
3. `a= [name, chris, age, 45]`
4. `a= {" name":"chris", "age": 45}`

7. What data type does the `input()` function create when assigned to a variable?

1. List
2. Raw
3. String
4. An auto typed one

8. Which print statement is valid for Python 3?

1. `print 'hello world'`
2. `print('hello world')`

3. `print(hello, world)`
4. `print("hello world")`

9. How do `if` statements operate?

1. `If` evaluates a variable against a condition to determine whether the condition is true.
2. `If` uses Boolean operators.
3. An `if` statement needs to end with `:`.
4. All of the above are correct.

10. Which statements are true about the `range()` function? (Choose two.)

1. The `range()` function iterates by one, starting at 0, up to but not including the number specified.
2. The `range()` function iterates by one, starting at 1, up to the number specified.
3. A `range()` function cannot count down, only up.
4. A `range()` function can count up or down, based on a positive or negative step value.

FOUNDATION TOPICS

GETTING STARTED WITH PYTHON

Those from many engineering backgrounds are looking to integrate programming into their infrastructure. Maybe you are a hardcore computer science major and have been programming in multiple languages for years. You might be an infrastructure engineer who is strong in the ways of Cisco IOS and looking for new ways to operate in a diverse environment. You might even be a server expert who is familiar with Ansible or Terraform automation and are being asked to bring that automation knowledge to the networking team. Regardless of your background or experience level, the DevNet certifications are designed to help you build the competency needed to be successful and give you a chance to prove what you have learned. If you haven't coded in years, or if the language that you currently program in isn't one that is very popular for infrastructure automation, where do you start? Python.

The Python language has become the most popular language in infrastructure automation because it is super easy to pick up and doesn't have all of the crazy syntax and structure that you see in languages like Java or C. It's based on the English language and is not only readable but extendable and powerful enough to be the one language you can count on to be able to get things accomplished in your day-to-day life. Those repetitive tasks that suck your productivity dry can be automated with just a few lines of code. Plus, due to the popularity of Python, millions of sample scripts provided by users like you as well as engineers at Cisco are free on GitHub for you to use and modify.

The software company TIOBE has published a list of the most popular programming languages each year for the past 10 years, and Python has consistently made the list. As of December 2019, it was ranked number 3. In addition, a significant number of job postings reference Python programming skills as a requirement for successful candidates. Python is used in AI, machine learning, big data, robotics, security, penetration testing, and many other disciplines that are being transformed by automation. Needless to say, learning Python has become a differentiator for skilled engineers, and it is part of the core tool set for DevOps and cloud operational models.

The 200-901 DevNet Associate DEVASC exam is not a Python test per se. You will not be asked to answer esoteric questions about techniques and syntax that only a Python wizard would know. The exam ensures that you are competent enough with Python to know how to interact with Cisco hardware through APIs and to use Cisco software development kits and frameworks such as pyATS and Genie. It is a very good idea to continue learning Python and spend some time in either online courses or self-study via books focused on the Python language itself; you should also spend lots of time

working through examples on DevNet at developer.cisco.com. This chapter and the several that follow provide a crash course in functional Python to get you going with the basics you need for success on the exam.

Many UNIX-based operating systems, such as Mac and Linux, already have Python installed, but with Windows, you need to install it yourself. This used to be a hassle, but now you can even install Python from the Windows Store. On a Mac, the default version of Python is 2.7, and you should update it to the more current 3.8 version. One of the easiest ways is to head over to python.org and download the latest variant from the source. The installation is fast, and there are many tutorials on the Internet that walk you through the process.

Note

Why would a Mac have such an old version of Python? Well, that's a question for Apple to answer, but from a community standpoint, the move to version 3 historically was slow to happen because many of the Python extensions (modules) were not updated to the newer version. If you run across code for a 2.x version, you will find differences in syntax and commands (also known as the Python standard library) that will prevent that code from running under 3.x. Python is not backward compatible without modifications. In addition, many Python programs require additional modules that are installed to add functionality to Python that aren't available in the standard library. If you have a program that was written for a specific module version, but you have the latest version of Python installed on your machine, the program might not work properly. You will learn more about common Python modules and how to use them in [Chapter 4, "Python Functions, Classes, and Modules."](#)

The use of Python 3 has changed dramatically as support for the 2.x version ended in January 2020. The 3.x version came out in 2008 and is the one that you should be using today. Of course, this version issue is still a problem even within the 3.x train of Python and the corresponding modules you may want to use. To address this compatibility conundrum across different versions and modules, Python virtual environments have been created. Such an environment allows you to install a specific version of Python and packages to a separate directory structure. This way, you can ensure that the right modules are loaded, and your applications don't break when you upgrade your base Python installation. As of Python 3.3, there is native support for these virtual environments built into the Python distribution. You can code in Python without using virtual environments, but the minute you update modules or Python itself, you run the risk of breaking your apps. A virtual environment allows you to lock in the components and modules you use for your app into a single "package," which is a good practice for building Python apps.

To use virtual environments, you launch Python 3 with the **-m** argument to run the **venv** module. You need to supply a name for your virtual environment, which will also become the directory name that will include all the parts of your virtual environment. Next, you need to activate the virtual environment by using the **source** command in Linux or Mac, as shown in this example. On Windows, you will need to run the activate batch file.

[Click here to view code image](#)

```
# MacOS or Linux
python3 -m venv myvenv
source myvenv/bin/activate

# Windows
```

```
C:\py -3 -m venv myvenv
C:\myvenv\Scripts\activate.bat
```

At this point, you will see your virtual environment name in parentheses at the beginning of your command prompt:

```
(myvenv) $
```

This indicates that you are running Python in your virtual environment. If you close the terminal, you have to reactivate the virtual environment if you want to run the code or add modules via **pip** (the Python module package manager).

To turn off the virtual environment, just type **deactivate** at the command prompt, and your normal command prompt returns, indicating that you are using your local system Python setup and not the virtual environment.

To install new modules for Python, you use **pip**, which pulls modules down from the PyPI repository. The command to load new modules is as follows:

```
pip install packagename
```

where *packagename* is the name of the package or module you want to install. You can go to pypi.org and search for interesting modules and check to see what others are using. There are more than 200,000 projects in the PyPI repository, so you are sure to find quite a few useful modules to experiment with. These modules extend Python functionality and contribute to its flexibility and heavy use today.

The **pip** command also offers a search function that allows you to query the package index:

```
pip search "search value"
```

The output includes the name, version, and a brief description of what the package does.

To install a specific version of a package, you can specify a version number or a minimum version so that you can get recent bug fixes:

[Click here to view code image](#)

```
pip install package==1.1.1. To install a specific
version

pip install package>=1.0    To install a version
greater than or             equal to 1.0
```

When you download sample code, if there are package dependencies, there is usually a readme file that lists these requirements.

Using a requirements.txt file included with your code is another essential good practice. Such a file makes it simpler to get your Python environment ready to go as quickly as possible. If you have a requirements.txt file included with your code, it will give **pip** a set of packages that need to be installed, and you can issue this one command to get them loaded:

[Click here to view code image](#)

```
pip install -r requirements.txt
```

The requirements.txt file is just a list that maps Python package names to versions. [Example 3-1](#) shows what it looks like.

Example 3-1 *Contents of requirements.txt*

```
ansible==2.6.3
black==19.3b0
flake8==3.7.7
genie==19.0.1
ipython==6.5.0
```

```
napalm==2.4.0
ncclient==0.6.3
netmiko==2.3.3
pyang==1.7.5
pyats==19.0
PyYAML==5.1
requests==2.21.0
urllib3==1.24.1
virlutils==0.8.4
xmldict==0.12.0
```

If you are building your own code and want to save the current modules configured in your virtual environment, you can use the **freeze** command and have it automatically populate the requirements.txt file:

[Click here to view code image](#)

```
pip freeze > requirements.txt
```

UNDERSTANDING PYTHON SYNTAX

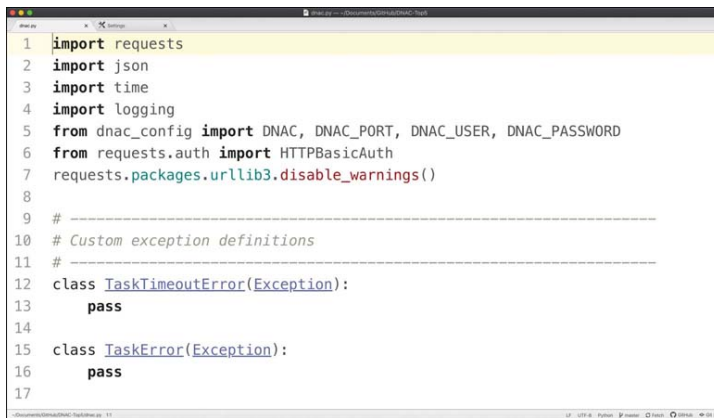
The word *syntax* is often used to describe structure in a language, and in the case of programming syntax, is used in much the same way. Some programming languages are very strict about how you code, which can make it challenging to get something written. While Python is a looser language than some, it does have rules that should be followed to keep your code not only readable but functional. Keep in mind that Python was built as a language to enhance code readability and named after Monty Python (the British comedy troop) because the original architects of Python wanted to keep it fun and uncluttered. Python is best understood through its core philosophy (The Zen of Python):

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Python is a scripted language, which means that any text editor can become a Python programming environment. Your choice of editor is completely up to your own preferences. The easiest way to get started with some Python code is to just enter **python3** at the command prompt (assuming that Python 3 is installed, of course) and use the interactive interpreter:

```
$ python3
>>> print("Savannah rules!")
Savannah rules!
```

For doing simple repetitive tasks, the interpreter is a quick interface to Python. For any program that you want to do a lot of editing, an editor is your best bet. Atom and Visual Studio Code are two editors that are very popular among Python programmers. Any modern editor will do, but a strong ecosystem of plug-ins can certainly make your life easier when interacting with GitHub and creating more complex applications. [Figure 3-1](#) shows the Atom editor in action.

A screenshot of the Atom text editor interface. The editor window displays a Python script with the following code:

```
1 import requests
2 import json
3 import time
4 import logging
5 from dnac_config import DNAC, DNAC_PORT, DNAC_USER, DNAC_PASSWORD
6 from requests.auth import HTTPBasicAuth
7 requests.packages.urllib3.disable_warnings()
8
9 # -----
10 # Custom exception definitions
11 # -----
12 class TaskTimeoutError(Exception):
13     pass
14
15 class TaskError(Exception):
16     pass
17
```

The code is syntax-highlighted, with imports in blue, class names in green, and comments in grey. The editor's status bar at the bottom shows the file path and various icons.

Figure 3-1 Atom Text Editor



One aspect in which Python is different from other languages is that within Python code, whitespace matters. This can seem really weird and frustrating if you are coming from another language, such as Java or C, that uses curly braces or start/stop keywords; instead, Python uses indentation to separate blocks of code. This whitespace is not used just to make your code readable; rather, Python will not work without it. Here is an example of a simple loop that highlights why whitespace is so important:

[Click here to view code image](#)

```
>>> for kids in ["Caleb", "Sydney", "Savannah"]:
...     print("Clean your room,", kids, "!")
File "<stdin>", line 2
    print("Clean your room,", kids, "!")
    ^
IndentationError: expected an indented block
```

This code will generate a syntax error the minute you try to run it. Python is expecting to see indentation on the line after the `:`. If you insert four spaces before the `print()` statement, the code works:

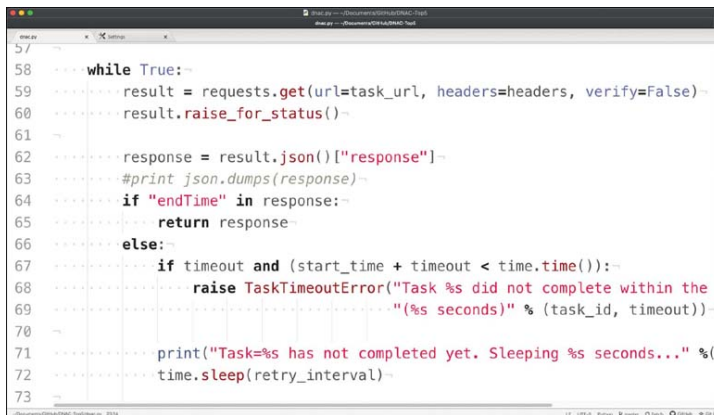
[Click here to view code image](#)

```
>>> for kids in ["Caleb", "Sydney", "Savannah"]:
...     print("Clean your room,", kids, "!")
...
Clean your room, Caleb !
Clean your room, Sydney !
Clean your room, Savannah !
```

Python allows you to use spaces or tabs. You can use both spaces and tabs in Python 2, but Python 3 will return a syntax error; however, if you use both tabs and spaces, you might end up with really weird issues that you need to troubleshoot. The standard for Python from

the PEP 8 style guide is to use four spaces of indentation before each block of code. Why four spaces? Won't one space work? Yes, it will, but your code blocks will be hard to align, and you will end up having to do extra work.

The alignment issue becomes especially important when you nest loops and conditional statements, as each loop needs to correspond to another block of code, indented using spaces. Many text editors allow you to view whitespace, and some even give you a visual indication of what is in a code block. [Figure 3-2](#) shows this in Atom.

A screenshot of the Atom text editor interface. The editor window displays Python code with visual indentation guides. The code is as follows:

```
57  
58 .....while True:-  
59 .....    result = requests.get(url=task_url, headers=headers, verify=False)-  
60 .....    result.raise_for_status()-  
61 .....  
62 .....    response = result.json()["response"]-  
63 .....    #print json.dumps(response)-  
64 .....    if "endTime" in response:-  
65 .....        return response-  
66 .....    else:-  
67 .....        if timeout and (start_time + timeout < time.time):-  
68 .....            raise TaskTimeoutError("Task %s did not complete within the  
69 .....                "%s seconds)" % (task_id, timeout))-  
70 .....  
71 .....        print("Task=%s has not completed yet. Sleeping %s seconds.." %(  
72 .....            time.sleep(retry_interval)-  
73 .....
```

The code is indented with four spaces per level. The editor shows a dark background with light-colored text and vertical lines indicating the indentation levels.

Figure 3-2 Spaces and Code Blocks in Atom

Comments in Python are created by entering # or a string of three quotation marks (either single or double quotation marks). One very important good practice when coding is to write a description of what is happening in code that is not obvious. You probably will not want to write a comment for a simple **print** statement, but describing the output of a nested function would be useful for anyone who needs to make additions to your code in the future or to remind yourself why you did what you did during that late night caffeine-fueled coding session. The # is used to comment out a single line so the Python interpreter ignores it. Here is an example:

[Click here to view code image](#)

```
#get input from user in numeric format
```

The triple quote method is used to comment multiple lines and can be helpful when you want to provide a bit more context than what can fit on a single line of text. Here is an example:

```
''' This is  
    line 2  
    and line 3'''
```

DATA TYPES AND VARIABLES

Data and variables are like the fuel and the fuel tank for a program. You can insert various types of data into a variable, and Python supports many types natively. Python can also be expanded with modules to support even more variables. A variable is really just a label that maps to a Python object stored somewhere in memory. Without variables, your programs would not be able to easily identify these objects, and your code would be a mess of memory locations.

Variables

Assigning a variable in Python is very straightforward. Python auto types a variable, and you can reassign that same variable to another value of a different type in the future. (Try doing that in C!) You just need to remember the rules for variable names:

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number.
- A variable name can only consist of alphanumeric characters and underscores (A-Z, 0-9, and _).
- A variable name is case sensitive (so Value and value are two different variable names).

To assign a variable you just set the variable name equal to the value you want, as shown in these examples:

Pip = "cat"	Variable assigned to a string
Age = 9	Variable assigned to an integer
Chill = True	Variable assigned to a Boolean
Variable1 = Variable2	Variable assigned to another variable

Data Types

Everything in Python is an object, and depending on the type of object, there are certain characteristics that you must be aware of when trying to determine the correct action you can perform on them. In Python, whenever you create an object, you are assigning that object an ID that Python uses to recall what is being stored. This mechanism is used to point to the memory location of the object in question and allows you to perform actions on it, such as printing its value. When the object is created, it is assigned a type that does not change. This type is tied to the object and determines whether it is a string, an integer, or another class.

Within these types, you are allowed to either change the object (mutable) or are not allowed to change the object (immutable) after it has been created. This doesn't mean that variables are not able to be changed; it means that most of the basic data types are not able to be modified but need to be replaced (or *assigned*, in Python speak) with another value. You can't just add a character at the end of a string value, for example. You have to instead reassign the whole string if you want to change it. This mutable/immutable concept will make more sense as you interact with various data types in programs. Python treats these two types of objects differently, and each has nuances that you must work around as you build your Python programs. To make it simple, think of immutable

objects as ones that you want to stay the same, such as constants. Mutable objects, on the other hand, are objects that you will be adding elements to and subtracting from on a regular basis.

Table 3-2 lists the most commonly used Python data types. The rest of this section covers them in more detail.



Table 3-2 Python Data Types

Name	Type	Mutable	Description
Integer	int	No	Whole numbers, such as 6, 600, and 1589
Boolean	bool	No	Comparison value, either True or False
String	str	No	Sequence of characters delimited by quotes, such as "Cisco", 'Piper', and "2000"
List	list	Yes	Ordered sequence of objects, such as [10, "DNA", 19.8]
Tuple	tuple	No	Ordered sequence of immutable objects, such as (10, "DNA", 19.8)
Dictionary	dict	Yes	Unordered key:value pairs, such as {"key1": "value1", "name": "Pip"}

Set	s e t	Y e s	Unordered collection of unique objects, such as {"a","b"}
-----	-------------	-------------	--

Integers, Floating Point, and Complex Numbers

The integers and floating point numbers are the simplest of data types:

- **Integers:** Whole numbers without decimal points
- **Floating point:** Numbers with decimal points or exponents (such as $10e5$, which indicates 10 to the fifth power)

Python can perform advanced calculations, so it is used heavily in data science, and many features are built into the language and can be easily added with modules. For our purposes, though, the basic building blocks will suffice. Python has a simple set of built-in operators for working with numbers, much like the operators on a regular calculator. [Table 3-3](#) lists Python's numeric operators.

Table 3-3 Python's Numeric Operators

Operator	Description	Example	Evaluates to
+	Adds two expressions together	5 + 5	1 0
-	Subtracts one expression from another	35 - 15	2 0
*	Multiplies two expressions	10 * 10	1 0 0
/	Divides one expression by another	20 / 5	4
/ /	Performs integer division (leaving off the remainder)	30 // 7	4

%	Performs modulus division (printing the remainder only)	30 % 7	2
*	Indicates an exponent	2 **	2
*		8	5 6

When working with numbers in Python, a defined order of precedence must be observed in calculations. Python uses the following order (also known as PEMDAS):

- 1. Parentheses:** Parentheses are always evaluated first.
- 2. Power:** The exponent is evaluated.
- 3. Multiplication:** Any multiplication is performed.
- 4. Division:** Division is evaluated.
- 5. Addition:** Addition is performed.
- 6. Subtraction:** Subtraction is performed.
- 7. Left to right:** After PEMDAS, anything else (such as `sqrt()` or other math functions) is evaluated from left to right.

In most languages, the parentheses are preferred over anything else. Most Python programmers use them liberally in their math formulas to make them simpler to construct without being so strict with the rules. Take the following example:

```
>>> 5 * 6 - 1
29
```

Python evaluates the multiplication first and then subtracts 1 from the result. If you wanted the subtraction to happen first, you could simply add parentheses around the parts you want evaluated:

```
>>> 5 * (6 - 1)
25
```

A floating point number is just a whole number with a decimal point. When you divide in Python, you often get a remainder that is displayed as a floating point number, as in this example:

```
>>> 10 / 7
1.4285714285714286
```

If you just want to see whole numbers, you can use integer division and lop off the remainder, as shown here:

```
>>> 10 // 7
1
```

Likewise, if you are only interested in the remainder, you can have modulus division show you what is left over:

```
>>> 10 % 7
3
```

You have the option to use other base systems instead of just the default base 10. You have three choices in addition to base 10: binary (base 2), octal (base 8), and hex (base 16). You need to use prefixes before integers in order for Python to understand that you are using a different base:

- `0b` or `0B` for binary
- `0o` or `0O` for octal
- `0x` or `0X` for hex

From Python's perspective, these are still just integers, and if you type any of them into the interpreter, it will return the decimal value by default, as shown in this example:

```
>>> 0xbadbeef
195935983
```

You can also convert back and forth by using the `base` keyword in front of the value you want to exchange, as in these examples:

```
>>> hex(195935983)
'0xbadbeef'

>>> bin(195935983)
'0b1011101011011011111011101111'
```

Booleans

A Boolean has only two possible values, **True** and **False**. You use comparison operators to evaluate between two Boolean objects in Python. This data type is the foundation for constructing conditional steps and decisions within programs. [Table 3-4](#) shows the various Boolean comparison operators and some examples of how to use them.

Table 3-4 Boolean Comparisons

Operator	What It Does	Example	Evaluates to
<	Less than	5 < 10	True
>	Greater than	6.5 > 3.5	True
<=	Less than or equal to	0 <= -5	False
>=	Greater than or equal to	6 >= 6	True
==	Equal to	5 = "5"	False

!=	Not equal to	5 != "5"	True
----	--------------	----------	------

Strings

The string data type is a sequence of characters and uses quotes to determine which characters are included. The string **'Hello'** is just a set of characters that Python stores in order from left to right. Even if a string contains a series of numbers, it can still be a string data type. If you try to add a 1 to a string value, Python gives you an error, as shown in this example:

[Click here to view code image](#)

```
>>> '10' + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to
str
```

This error tells you that you have to convert the string to an integer or another data type to be able to use it as a number (in a math formula, for example). The **int()** function can convert a string value into an integer for you, as shown in this example:

```
>>> int('10') + 1
11
```



A string is just a list of characters in a certain order that Python keeps track of. In fact, this aspect of strings makes them easy to manipulate. If you use the string **'DevNet'**, you can pull out any individual characters of the string by knowing where it sits in the string index. One thing to keep in mind is that indexes in Python

always start with 0, so if you want the very first character in a string, your index value would be 0 and not 1. [Figure 3-3](#) shows the string DevNet with its corresponding index values.

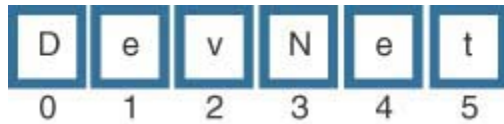


Figure 3-3 *DevNet String Index*

If you assign the string DevNet to a variable, you can separate and manipulate the component values of the string by using the index value. You can use brackets to specify the index number. The following example prints a capitol **D** from **DevNet**:

```
>>> a='DevNet '  
>>> a[0]  
'D'
```

You can also specify ranges to print. The colon operator gives you control over whole sections of a string. The first number is the beginning of the slice, and the second number determines the end. The second number may be confusing at first because it is intended to identify “up to but not including” the last character. Consider this example:

```
>>> a[0:3]  
'Dev'
```

This example shows a **3** at the end, but this is technically four characters since the index starts at 0, but Python doesn't print the last character and instead stops right before it. For new Python programmers, this can be confusing, but remember that Python is literal. If you think of an index value as a box, in [Figure 3-3](#), you want

to stop at box 3 (but don't want to open the box). If you want to print the whole string, just pick a number beyond the index value, and Python will print everything, as in this example:

```
>>> a[0:6]
'DevNet'
```

If you omit a value for the first number, Python starts at 0, as in this example:

```
>>> a[:2]
'De'
```

If you omit the second value, Python prints to the end of the string, as in this example:

```
>>> a[2:]
'vNet'
```

You can also reverse direction by using negative numbers. If you put a negative first number, you start from the end of the string, as in this example:

```
>>> a[-2:]
'et'
```

A negative value on the other side of the colon causes Python to print using the end as a reference point, as in this example:

```
>>> a[:-2]
'DevN'
```

You can perform math operations on strings as well. The + is used to add or concatenate two strings together, as

in this example:

```
>>> 'DevNet' + 'Rocks'  
'DevNetRocks'
```

Multiplication works as well, as in this example:

[Click here to view code image](#)

```
>>> 'DevNet Rocks ' * 5  
'DevNet Rocks DevNet Rocks DevNet Rocks DevNet  
Rocks DevNet Rocks '
```

There is a tremendous amount of string manipulation you can do with Python, and there are a number of built-in methods in the standard string library. These methods are called with a dot after the variable name for a string. [Table 3-5](#) lists some commonly used string manipulation methods, including the syntax used for each and what each method does.

Table 3-5 String Methods

Method	What It Does
str.capitalize()	Capitalize the string
str.center(width [, fillchar])	Center justify the string
str.endswith(suffix[, start[, end]])	Add an ending string to the string
str.find(sub[, start[, end]])	Find the index position of the characters in a string
str.lstrip([chars])	Remove whitespace characters from the end of the string

str.replace(old, new[, count])	Replace characters in the string
str.lower()	Make the string all lowercase
str.rstrip([chars])	Strip whitespace characters from the front of the string
str.strip([chars])	Remove whitespace characters from the beginning and end of the string
str.upper()	Make the string all uppercase

Lists

Python, unlike other programming languages, such as C++ and Java, doesn't have arrays. If you want to store a bunch of values, you can use a list. You can use a variable to store a collection of items in a list. To create a list, you assign the contents of the list to a variable with the = and [] and separate the items with commas, as in this example:

[Click here to view code image](#)

```
>>> kids = ['Caleb', 'Sydney', 'Savannah']
>>> kids
['Caleb', 'Sydney', 'Savannah']
```

A list can contain any Python object, such as integers, strings, and even other lists. A list can also be empty and is often initialized in an empty state for programs that pull data from other sources. To initialize a list in an empty state, you just assign two brackets with nothing in them or you can use the built-in **list()** function:

```
emptylist = []
emptylist2 = list()
```

Lists are similar to strings in that each is a set of items indexed by Python that you can interact with and slice and dice. To pull out values, you just use the variable name with brackets and the index number, which starts at 0, as in this example:

```
>>> print(kids[1])
Sydney
```

Figure 3-4 shows a list from the perspective of the index.

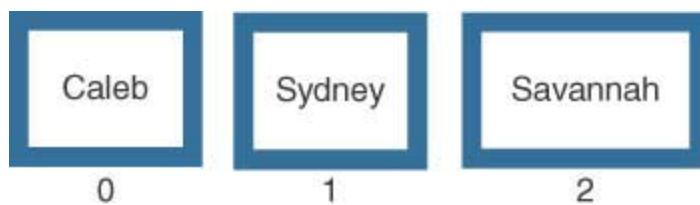


Figure 3-4 List Index

Unlike strings, lists are mutable objects, which means you can change parts of the list at will. With a string, you can't change parts of the string without creating a new string. This is not the case with lists, where you have a number of ways to make changes. If you have a misspelling, for example, you can change just one element of the list, leaving the rest untouched, as in this example:

[Click here to view code image](#)

```
>>> kids
['Caleb', 'Sidney', 'Savannah']
>>> kids[1]="Sydney"
>>> kids
['Caleb', 'Sydney', 'Savannah']
>>>
```

You can concatenate lists as well by using the + operator to join two lists together. The list items do not need to be

unique. Python just connects the two together into a new list, as shown in the following example:

```
>>> a = [1, 2, 4]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 4, 4, 5, 6]
```

Remember all of the slicing you saw with strings? The same principles apply here, but instead of having a single string with each letter being in a bucket, the elements in the list are the items in the bucket. Don't forget the rule about the second number after the colon, which means "up to but not including." Here is an example:

[Click here to view code image](#)

```
>>> c = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> c[1:4]
[2, 3, 4]
>>> c[: -4]
[1, 2, 3, 4, 5, 6]
>>> c[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Table 3-6 describes some of the most common list methods.

Table 3-6 List Methods

Method	What It Does
list.append(element)	Adds an element to the end of the list
list.clear()	Removes everything from the list

list.copy(alist)	Returns a copy of the list
list.count(element)	Shows the number of elements with the specified value
list.extend(alist)	Adds the elements of a list to the end of the current list
list.index(x)	Returns the index number of the first element with a specified value
list.insert(index, element)	Adds an element at a specified index value
list.pop(index)	Removes an element at a specific index position, or if no index position is provided, removes the last item from the list
list.remove(value)	Removes a list item with a specified value
list.reverse()	Reverses the list order
list.sort()	Sorts the list alphabetically and/or numerically

Tuples

Tuples and lists are very similar. The biggest difference between the two comes down to mutability. As discussed earlier, Python data types are either mutable or immutable. Lists are mutable, and tuples are immutable. So why would you need these two types if they are so similar? It all comes down to how Python accesses objects and data in memory. When you have a lot of changes occurring, a mutable data structure is preferred

because you don't have to create a new object every time you need to store different values. When you have a value that is constant and referenced in multiple parts of a program, an immutable data type (such as a tuple), is more memory efficient and easier to debug. You don't want some other part of your program to make changes to a crucial piece of data stored in a mutable data type.

To create a tuple, you use parentheses instead of brackets. You can use the **type()** function to identify a Python data type you have created. Here is an example:

[Click here to view code image](#)

```
>>> person = (2012, 'Mike', 'CCNA')
>>> person
(2012, 'Mike', 'CCNA')
>>> type(person)
<class 'tuple'>
```

You access data in a tuple the same way as in a list—by using brackets and the index value of the item in the tuple that you want to return:

```
>>> person[0]
2012
```

What you can't do with a tuple is make an assignment to one of the values:

[Click here to view code image](#)

```
>>> person[0]=15
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment
```

Tuples can, however, be used to assign a set of variables quickly:

[Click here to view code image](#)

```
>>> (a, b, c) = (12, 'Fred',18)
>>> c
18
```

Dictionaries

A dictionary provides another way of creating a collection of items. Why do you need another way of storing data when you already have lists and tuples? A list is an ordered set of items tracked by an index. What if you need to access data that is tied to a certain value, such as a person's name? This capability is exactly why you need dictionaries in Python. A dictionary saves a ton of effort by giving you a built-in system for storing data in a key:value pair. As when you use labels on files in a filing cabinet, you can assign data to a key and retrieve it by calling that key as if you are pulling a file from the cabinet. Dictionaries don't have any defined order; all you need is the key—and not some index number—to get access to your data. There are some rules regarding dictionaries.

- **Keys:** A dictionary's keys are limited to only using immutable values (int, float, bool, str, tuple, and so on). No, you can't use a list as a key, but you can use a tuple, which means you could use a tuple as a key (immutable) but you can't use a list as a key (mutable).
- **Values:** A value can be any Python object or any combination of objects.

To create a dictionary, you use braces and your key and value separated by a colon. You separate multiple items with commas. Here's an example:

[Click here to view code image](#)

```
>>> cabinet = { "scores": (98,76,95),
                "name": "Chris",
                "company": "Cisco" }
```



```
>>> type(cabinet)
<class 'dict'>
```

Instead of using an index, you use the key, as shown in this example:

```
>>> cabinet["scores"]
(98, 76, 95)
>>> cabinet["company"]
'Cisco'
```

To add more items to a dictionary, you can assign them with a new key. You can even add another dictionary to your existing dictionary, as shown in this example:

[Click here to view code image](#)

```
>>> cabinet["address"] = {"street": "123 Anywhere Dr",
"city": "Franklin", "state": "TN"}
>>> cabinet["address"]
{'street': '123 Anywhere Dr', 'city': 'Franklin', 'state': 'TN'}
```

Sets

A set in Python consists of an unordered grouping of data and is defined by using the curly braces of a dictionary, without the key:value pairs. Sets are mutable, and you can add and remove items from the set. You can create a special case of sets called a frozen set that makes the set immutable. A frozen set is often used as the source of keys in a dictionary (which have to be immutable); it basically creates a template for the dictionary structure. If you are familiar with how sets work in mathematics, the various operations you can perform on mutable sets in Python will make logical sense. To define a set, do the following:

[Click here to view code image](#)

```
>>> nums = {1, 2, 4, 5, 6, 8, 10}
>>> odds = {1, 3, 5, 7, 9}
```

To check that these are indeed sets, use the `type()` function:

```
>>> type(odds)
<class 'set'>
```

To join two sets (just as in a mathematical join), you can use the `|` operator to show a combined set with no duplicates:

[Click here to view code image](#)

```
>>> nums | odds
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

You can get an intersection of two sets and show what numbers are in both by using the `&` operator:

```
>>> nums & odds
{1, 5}
```

There are many ways to evaluate sets, and the Python documentation can be used to explore them all. In addition, Python has library collections that give you even more options for ways to store data. Search the Python documentation for “collections” and take a look at ordered dictionaries, named tuples, and other variations that may suit your data collecting needs. For the purposes of the DEVASC exam, though, you do not need to know data structures beyond the basic types discussed here.

INPUT AND OUTPUT

Input and output pretty much define what a computer does for you. In Python, the **input()** function and the **print()** function are two essential components that allow you to create interactive applications. In this section you will learn how to leverage these powerful functions in your applications.

Getting Input from the User

Python has the **input()** function to get information from a user running your Python code. The user is asked a question, and the program waits until the user types a response. It really is as simple as that. The **input()** function takes the characters that are entered and automatically stores them as a string data type, regardless of what the user enters. Here is an example:

[Click here to view code image](#)

```
>>> inpt = input('Type your name: ')
Type your name: Chris Jackson
>>> inpt
'Chris Jackson'
```

You assign a variable (in this case, **inpt**) to the **input()** function with a text prompt so that the user knows what is expected. That variable now holds the string the user typed. What if you need to get an integer or a floating point number? Since Python stores every input as a string, you need to do a conversion on the data supplied. Here is an example:

[Click here to view code image](#)

```
>>> inpt = float(input('What is the Temperature in
F: '))
What is the Temperature in F: 83.5
>>> inpt
83.5
```

Here you asked the user for the temperature in Fahrenheit, which can be expressed as a floating point number. To make sure the variable holds the correct type, you used the **input()** function inside the **float()** function to convert the string into a floating point number.

The Mighty print() Function

The **print()** function provides output that can be displayed in the user's terminal. Just like every other function in Python, it is called with parentheses. This function is usually the gateway to writing your first code in Python, as in this example:

```
>>> print('Hello World')
Hello World
```

Every line printed with the **print()** function includes a newline character (**\n**) at the end, which is a special character that tells the terminal to advance one line. If you use the **\n** sequence within the **print()** string, it is interpreted as a new line, as shown in this example:

```
>>> print('Hello\nWorld')
Hello
World
```

There are numerous codes like this that can control how text is displayed in a string and how Python interprets the output. Without them, you would not be able to use, for example, a backslash in your text output. Here are a few of the most common ones:

- ****: Backslash
- **\b**: Backspace
- **\'**: Single quote
- **\"**: Double quote
- **\t**: Tab

- `\r`: Carriage return

You can add multiple arguments to the `print()` function by using commas between elements. This is very useful in creating meaningful text, and the `print()` function also handles concatenation of the different data types automatically. Consider this example:

[Click here to view code image](#)

```
>>> print('Numbers in set', 1, ':', nums )
Numbers in set 1 : {1, 2, 4, 5, 6, 8, 10}
```

By default, the `print()` function uses a separator between elements. This is normally not an issue if you want spaces to appear between words or elements. In the previous example, you can see a space between the `1` and the `:` that just doesn't look good. You can fix it by changing the separator that the `print()` functions uses with the `sep=""` attribute (using single quotes with nothing in between). Since you will be removing all automatic spacing, you have to compensate for this by adding spaces in your actual text if you need them. Remember that separators come between elements and don't add anything to the start or end of the `print()` function. Consider this example:

[Click here to view code image](#)

```
>>> print('Numbers in set ',1, ': ', nums,
sep=' ' )
Numbers in set 1: {1, 2, 4, 5, 6, 8, 10}
```

One capability added to Python 3.6 and up is the addition of f-string formatting. Not only are these strings easier to read and less prone to syntax errors but they allow you to write formatting code a lot faster. To create an f-string, you put an `f` at the beginning of a string, within the `print()` function, to let Python know what you

are doing, and then you can use `{}` within your string to insert values or other functions. Here is an example:

[Click here to view code image](#)

```
>>> name = 'Piper'
>>> name2 = 'Chris'
>>> print(f'{name2} says Hi to {name}!')
Chris says Hi to Piper!
```

For more on formatting strings and beautifying your output, see the Python documentation.

FLOW CONTROL WITH CONDITIONALS AND LOOPS

So far you have been exposed to many of the building blocks of the Python language. The real power of a programming language is in the mechanisms you can use to embed logic and respond to different conditions by changing the flow of operation. Python has three primary control statements:

- **if:** An **if** statement is a conditional statement that can compare values and make branching decisions.
- **for:** A **for** loop is a counting loop that can iterate through data a specific number of times.
- **while:** The **while** loop can iterate forever when certain conditions are met.

You can use these three statements in various combinations to create very sophisticated programs. In this section you will see how each of these statements work.

If Statements

An **if** statement starts with an **if** and then sets up a comparison to determine the truth of the statement it is evaluating and ending with a **:** to tell Python to expect the clause (the action if the condition is true) block of code next. As mentioned earlier in this chapter,

whitespace indenting matters very much in Python. The clause of an **if** statement must be indented (four spaces is the standard) from the beginning of the **if** statement. The following example looks for a condition where the variable **n** is equal to **5** and prints a message to the console indicating that the number is indeed a 5:

[Click here to view code image](#)

```
>>> n = 20
>>> if n == 20:
...     print('The number is 20')
...
The number is 20
```

The Python interpreter uses three dots to let you continue the clause for the **if** statement. Notice that there is space between the start of the dots and the **print()** statement. Without these four spaces, Python would spit back a syntax error like this:

[Click here to view code image](#)

```
>>> if n == 20:
... print('oops')
    File "<stdin>", line 2
      print('oops')
      ^
IndentationError: expected an indented block
```

The goal of an **if** statement is to determine the “truth” of the elements under evaluation. This is Boolean logic, meaning that the operators evaluate True or False (refer to [Table 3-4](#)). The previous example is determining whether a variable is equal to a specific integer. What if the number is different? You might want to apply other logic by asking more questions. This is where else if (**elif** in Python) comes into play.

An **if** statement can have as many **elif** conditions as you want to add to the conditional check. Good coding practices recommend simplification, but there is no real limit to how many you add. Here is an example that uses two **elif** conditionals:

[Click here to view code image](#)

```
>>> n = 3
>>> if n == 17:
...     print('Number is 17')
... elif n < 10:
...     print('Number is less than 10')
... elif n > 10:
...     print('Number is greater than 10')
...
Number is less than 10
```

Since each **if** and **elif** statement does something only if the condition identified is true, it may be helpful to have a default condition that handles situations where none of the **if** or **elif** statements are true. For this purpose, you can assign a single **else** statement at the end, as shown in [Example 3-2](#).

Example 3-2 *Adding a Final else Statement*

[Click here to view code image](#)

```
score = int(input('What was your test
score?:'))

if score >= 90:
    print('Grade is A')
elif score >= 80:
    print('Grade is B')
elif score >= 70:
    print('Grade is C')
elif score >= 60:
    print('Grade is D')
else:
    print('Grade is F')

What was your test score?:53
```



```
Grade is F
>>>
```

For Loops

The **for** statement allows you to create a loop that continues to iterate through the code a specific number of times. It is also referred to as a counting loop and can work through a sequence of items, such as a list or other data objects. The **for** loop is heavily used to parse through data and is likely to be your go-to tool for working with data sets. A **for** loop starts with the **for** statement, followed by a variable name (which is a placeholder used to hold each sequence of data), the **in** keyword, some data set to iterate through, and then finally a closing colon, as shown in this example:

```
>>> dataset=(1,2,3,4,5)
>>> for variable in dataset:
...     print(variable)
...
1
2
3
4
5
```

The **for** loop continues through each item in the data set, and in this example, it prints each item. You can also use the **range()** function to iterate a specific number of times. The **range()** function can take arguments that let you choose what number it starts with or stops on and how it steps through each one. Here is an example:

```
>>> for x in range(3):
...     print(x)
...
0
```

```
1
2
```

By default, if you just give **range()** a number, it starts at 0 and goes by 1s until it reaches the number you provided. Zero is a valid iteration, but if you don't want it, you can start at 1. Consider this example:

```
>>> for x in range(1,3):
...     print(x)
...
1
2
```

To change the increment from the default of 1, you can add a third attribute to the **range()** statement. In the following example, you start at 1 and increment by 3 until you reach 10:

```
>>> for x in range(1,11,3):
...     print(x)
...
1
4
7
10
```

Remember that these ranges are up to and not including the final number you specify. If you want to go all the way to 10 in this example, you need to set your range to 11.

While Loops

Whereas the **for** loop counts through data, the **while** loop is a conditional loop, and the evaluation of the condition (as in **if** statements) being true is what determines how many times the loop executes. This

difference is huge in that it means you can specify a loop that could conceivably go on forever, as long as the loop condition is still true. You can use **else** with a **while** loop. An **else** statement after a **while** loop executes when the condition for the **while** loop to continue is no longer met. [Example 3-3](#) shows a **count** and an **else** statement.

Example 3-3 *else Statement with a while Loop*

[Click here to view code image](#)

```
>>> count = 1
>>> while (count < 5):
...     print("Loop count is:", count)
...     count = count + 1
... else:
...     print("loop is finished")
...
Loop count is: 1
Loop count is: 2
Loop count is: 3
Loop count is: 4
loop is finished
```

You can probably see similarities between the loop in [Example 3-3](#) and a **for** loop. The difference is that the **for** loop was built to count, whereas this example uses an external variable to determine how many times the loop iterates. In order to build some logic into the **while** loop, you can use the **break** statement to exit the loop. [Example 3-4](#) shows a **break** with **if** statements in an infinite **while** loop.

Example 3-4 *Using the break Statement to Exit a Loop*

[Click here to view code image](#)

```
while True:
    string = input('Enter some text to print.
\nType "done" to quit> ')
    if string == 'done' :
        break
    print(string)
```

```
print('Done!')

Enter some text to print.
Type "done" to quit> Good luck on the test!
Good luck on the test!
Enter some text to print
Type "done" to quit> done
Done!
```

Notice the condition this example is checking with the **while** statement. Because **True** will always be **True** from an evaluation perspective, the **while** condition is automatically met. Without that **if** statement looking for the string **'done'**, your loop would keep asking for input and printing what you typed forever.

This chapter provides an overview of some of the key concepts and capabilities in Python. The goal is to prepare you to be able to read and understand code snippets that you might see on the DEVASC exam. The next two chapters dive into other aspects of working with Python and Cisco APIs that are considered essential skills. Make sure you have followed along with the code examples here and that you are familiar with how to construct these basic examples. The following chapters build on these skills.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page.

Table 3-7 lists these key topics and the page number on which each is found.



Table 3-7 Key Topics for Chapter 3

Key Topic Element	Description	Page Number
Paragraph	Whitespace in Python code blocks	64
Table 3-2	Python Data Types	67
Paragraph	Strings	70

DEFINE KEY TERMS

There are no key terms for this chapter.

ADDITIONAL RESOURCES

Python Syntax:

https://www.w3schools.com/python/python_syntax.asp

A Quick Tour of Python Language Syntax:

<https://jakevdp.github.io/WhirlwindTourOfPython/02-basic-python-syntax.html>

PEP 8—Style Guide for Python Code:

<https://www.python.org/dev/peps/pep-0008/>

Coding & APIs:

<https://developer.cisco.com/startnow/#coding-apis-vo>

Mutable vs Immutable Objects in Python:

<https://medium.com/@meghamohan/mutable-and->

[immutable-side-of-python-c2145cf72747](#)

Your Guide to the Python print() Function:

<https://realpython.com/python-print/>

Chapter 4

Python Functions, Classes, and Modules

This chapter covers the following topics:

- **Python Functions:** This section provides an overview of working with and building Python functions.
- **Object-Oriented Programming and Python:** This section describes key aspects of using object-oriented programming techniques.
- **Python Classes:** This section provides an overview of creating and using Python classes.
- **Working with Python Modules:** This section provides an overview of creating and using Python modules.

This chapter moves away from the basics introduced in [Chapter 3, “Introduction to Python,”](#) and introduces Python functions, classes, and modules. Building Python functions allows for the creation of reusable code and is the first step toward writing object-oriented code. Classes are the Python tools used to construct Python objects and make it easier to produce scalable applications that are easy to maintain and readable. Finally, this chapter introduces the wide world of Python modules and how they can extend the capabilities of Python and make your job of coding much easier.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics,

read the entire chapter. [Table 4-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 4-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Python Functions	1–3
Object-Oriented Programming and Python	4–5
Python Classes	6–8
Working with Python Modules	9–10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following is the correct syntax for a Python function?
 1. define function (arg):
 2. function function(arg);
 3. def function(arg):
 4. func function(arg):

- 2.** Which of the following is a valid Python function name?
1. `ifunction`
 2. `__init__`
 3. `True`
 4. `Function`
- 3.** When three single quotation marks are used on the next line directly after defining a function, what does this indicate?
1. Multi-line text
 2. A docstring
 3. A string value including double or single quotation marks
 4. None of the above
- 4.** What are key components of object-oriented programming in Python? (Choose two.)
1. Functions that can be performed on a data structure
 2. Attributes that are stored in an object
 3. Configuration templates
 4. YAML files
- 5.** Which of the following are benefits of OOP? (Choose all that apply.)
1. Reusable code
 2. Easy to follow
 3. Low coupling/high cohesion
 4. Complex integration
- 6.** Which of the following are used to define a class in Python? (Choose two.)
1. `class classname(parent):`
 2. `class classname:`
 3. `def class classname(arg):`
 4. None of the above
- 7.** What is a method?
1. A variable applied to a class
 2. Syntax notation
 3. A function within a class or an object
 4. Something that is not used in a class
- 8.** Which of the following describes inheritance?
1. A hierarchy for functions in Python
 2. Class attributes and methods used as the starting point for another class
 3. A function only applied to methods being used in another class
 4. None of the above

9. Which module provides access to the file system and directory structure?

1. `filesystem`
2. `open`
3. `system`
4. `os`

10. Which module is a testing framework for Cisco infrastructure?

1. `pyATS`
2. `pyang`
3. `devnetats`
4. `ncclient`

FOUNDATION TOPICS

PYTHON FUNCTIONS

In Python, a *function* is a named block of code that can take a wide variety of input parameters (or none at all) and return some form of output back to the code that called the function to begin with. It represents a key concept in programming sometimes referred to as DRY, which stands for Don't Repeat Yourself. The idea behind DRY is that if you perform some particular operations in your code multiple times, you can simply create a function to reuse that block of code anywhere you need it instead of duplicating effort by typing it each time.

Python offers two types of functions: built-in functions that are part of the standard library and functions you create yourself. The standard library includes a huge number of functions you can use in your program, like `print()`, many of which you have already been introduced to in [Chapter 3](#). Building your own functions is how you construct capabilities that are not already present within the Python language.



To define a function in Python, you use the keyword **def**, a name for the function, a set of parentheses enclosing any arguments you want to pass to the function, and a colon at the end. The name of a function must follow these rules:

- Must not start with a number
- Must not be a reserved Python word, a built-in function (for example, **print()**, **input()**, **type()**), or a name that has already been used as a function or variable
- Can be any combination of the A–Z, a–z, 0–9 and the underscore (`_`) and dash (`-`)

The following is an example of an incredibly simple function that could be entered into the interactive Python interpreter:

[Click here to view code image](#)

```
Python 3.8.1 (v3.8.1:1b293b6006, Dec 18 2019,
14:08:53)

[Clang 6.0 (clang-600.0.57)] on darwin

Type "help", "copyright", "credits" or "license"
for more
information.

>>> def devnet():

    '''prints simple function'''

    print('Simple function')

>>> devnet()

Simple function
```

This function prints out the string “Simple function” any time you call it with **devnet()**. Notice the indented portion that begins on the next line after the colon. Python expects this indented portion to contain all the code that makes up the function. Keep in mind that whitespace matters in Python. The three single quotation marks that appear on the first line of the indented text of the function are called a *docstring* and can be used to describe what the function does.

As shown in the following example, you can use the built-in Python function **help()** to learn what a function does and any methods that can be used:

[Click here to view code image](#)

```
>>> help(devnet)
Help on function devnet in module __main__:

devnet()
    prints simple function
```

USING ARGUMENTS AND PARAMETERS

An argument is some value (or multiple values) that you pass to a function when you call the function within code. Arguments allow a function to produce different results and make code reuse possible. You simply place arguments within the parentheses after a function name. For example, this example shows how you can pass multiple numeric arguments to the **max()** function to have it return the largest number in the list:

```
>>> max(50, 5, 8, 10, 1)
50
```

Each function must define how it will use arguments, using parameters to identify what gets passed in and how it gets used. A parameter is simply a variable that is used in a function definition that allows code within the function to use the data passed to it. To get results back from the function, you use the keyword **return** and the object you want to pass back. The following example shows how to create a function that subtracts two numbers and stores the result in a local variable called **result** that gets returned when the function is called:

```
>>> def sub(arg1, arg2):
    result = arg1 - arg2
    return result

>>> sub(10, 15)
-5
```

The variable **result** is local, meaning that it is not accessible to the main Python script, and it is used only within the function itself. If you tried to call **result** directly, Python would produce an error saying that **result** is not defined. You can, however, access global variables from within the function; you might do this, for example, to set certain constants or key variables that any function can use (for example, IP addresses). The difference in accessibility between a local variable and global variable is important, because they allow your code to maintain separation and can keep your functions self-contained.

The previous example uses positional arguments, which must be passed to a function in a particular order. Positional arguments work with a simple set of consistently applied arguments, but if a function needs more flexible alignment to parameters within a function, you can use keyword arguments instead. A *keyword argument* is a name/value pair that you pass to a function. Instead of using position, you specify the argument the function uses. It is a lot like assigning a variable to an argument. In the previous example, *arg1* is subtracted from *arg2*, and if the positions of these arguments were switched, you would get a different result when subtracting the values. With keyword arguments, it doesn't matter in what order they are passed to the function. Here is an example:

```
>>> sub(arg2=15, arg1=10)
-5
```

What happens if you don't know the total number of arguments that are being passed to a function? When you read in data, you might not know how many arguments to expect. Python allows you to use `*` and `**` (often referred to as ***args** and ****kwargs**) to define any number of arguments or keyword arguments. `*` and `**` allow you to iterate through a list or other collection of data, as shown in this example:

[Click here to view code image](#)

```
>>> def hello(*args):
    for arg in args:
        print("Hello", arg, "!")

>>> hello('Caleb', 'Sydney', 'Savannah')
Hello Caleb !
Hello Sydney !
Hello Savannah !
```

By using keyword arguments, you can send a list of key/value pairs to a function, as in the following example:

[Click here to view code image](#)

```
>>> def hello(**kwargs):
    for key, value in kwargs.items():
        print("Hello", value, "!")

>>> hello(kwarg1='Caleb', kwarg2='Sydney',
kwarg3='Savannah')

Hello Caleb !
Hello Sydney !
Hello Savannah !
```

Note the use of the **items()** function in the **for** statement to unpack and iterate through the values.

You can also supply a default value argument in case you have an empty value to send to a function. By defining a function with an assigned key value, you can prevent an error. If the value in the function definition is not supplied, Python uses the default, and if it is supplied, Python uses what is supplied when the function is called and then ignores the default value. Consider this example:

[Click here to view code image](#)

```
>>> def greeting(name, message="Good morning!") :
        print("Hello", name + ', ' + message)

>>> greeting('Caleb')
Hello Caleb, Good morning!

>>> greeting('Sydney', "How are you?")
Hello Sydney, How are you?
```

OBJECT-ORIENTED PROGRAMMING AND PYTHON

Python was developed as a modern object-oriented programming (OOP) language. Object-oriented programming is a computer programming paradigm that makes it possible to describe real-world things and their relationships to each other. If you wanted to describe a router in the physical world, for example, you would list all its properties, such as ports, software versions, names, and IP addresses. In addition, you might list different capabilities or functions of the router that you would want to interact with. OOP was intended to model these types of relationships programmatically, allowing you to create an object that you can use anywhere in your code by just assigning it to a variable in order to instantiate it.



Objects are central to Python; in fact, Python really is just a collection of objects interacting with each other. An object is self-contained code or data, and the idea of OOP is to break up a program into smaller, easier-to-understand components. Up until now, you have mainly seen procedural programming techniques, which take a top-down approach and follow predefined sets of instructions. While this approach works well for simple programs, to write more sophisticated applications with better scalability, OOP is often the preferred method used by professional programmers. However, Python is very flexible in that you can mix and match these two approaches as you build applications.

Functions are an important part of the OOP principles of reusability and object-oriented structure. For the 200-901 DevNet Associate DEVASC exam, you need to be able to describe the benefits and techniques used in Python to build modular programs. Therefore, you need to know how to use Python classes and methods, which are covered next.

PYTHON CLASSES

In Python, you use classes to describe objects. Think of a class as a tool you use to create your own data structures that contain information about something; you can then use functions (methods) to perform operations on the data you describe. A class models how something should be defined and represents an idea or a blueprint for creating objects in Python.

Creating a Class



Say that you want to create a class to describe a router. The first thing you have to do is define it. In Python, you define a class by using the **class** keyword, giving the

class a name, and then closing with a colon. Pep8 (introduced in [Chapter 3](#)) recommends capitalizing a class name to differentiate it from a variable. Here is a simple example of creating a class in Python:

```
>>> class Router:
    pass
```

This example uses **pass** as a sort of placeholder that allows the class to be defined and set up to be used as an object. To make the class more useful, you can add some attributes to it. In the case of a router, you typically have some values that you want to have when you instantiate the class. Every router has a model name, a software version, and an IP address for management. You also need to pass some values to get started. The first value is always **self**. The reason for this will become obvious when you instantiate the class: The **self** value passes the object name that you select to instantiate the class. In the following example, the object you will create is **rtr1**:

[Click here to view code image](#)

```
class Router:
    '''Router Class'''
    def __init__(self, model, swversion, ip_add):
        '''initialize values'''
        self.model = model
        self.swversion = swversion
        self.ip_add = ip_add

rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
```

After defining the class, you add a docstring to document what the class is for and then you create a function that calls **__init__**, which is a special case that is used for the setup of the class. (In **__init__**, the double underscores are called *dunder* or *magic methods*.) Functions that are within the class are called *methods*

and become actions that you can perform on the object you are creating. To store attributes, you map the name **self** and the values you pass to it become variables inside the object, which then store those values as attributes. The last bit of code instantiates the object itself. Up until now, you have been creating a template, and by assigning data to the variables within the class, you have been telling Python to build the object. Now you can access any of the stored attributes of the class by using dot notation, as shown here:

```
>>> rtr1.model
'iosV'
```

When you call **rtr1.model**, the interpreter displays the value assigned to the variable `model` within the object. You can also create more attributes that aren't defined during initialization, as shown in this example:

[Click here to view code image](#)

```
>>> rtr1.desc = 'virtual router'
>>> rtr1.desc
'virtual router'
```

This example shows how flexible objects are, but you typically want to define any attributes as part of a class to automate object creation instead of manually assigning values. When building a class, you can instantiate as many objects as you want by just providing a new variable and passing over some data. Here is another example of creating a second router object **rtr2**:

[Click here to view code image](#)

```
>>> rtr2= Router('isr4221', '16.9.5',
'10.10.10.5')
>>> rtr2.model
'isr4221'
```

Methods

Attributes describe an object, and methods allow you to interact with an object. Methods are functions you define as part of a class. In the previous section, you created an object and applied some attributes to it. [Example 4-1](#) shows how you can work with an object by using methods. A method that allows you to see the details hidden within an object without typing a bunch of commands over and over would be a useful method to add to a class. Building on the previous example, [Example 4-1](#) adds a new function called **getdesc()** to format and print the key attributes of your router. Notice that you pass **self** to this function only, as **self** can access the attributes applied during initialization.

Example 4-1 Router Class Example

[Click here to view code image](#)

```
class Router:
    '''Router Class'''
    def __init__(self, model, swversion,
ip_add):
        '''initialize values'''
        self.model = model
        self.swversion = swversion
        self.ip_add = ip_add

    def getdesc(self):
        '''return a formatted description of
the router'''
        desc = f'Router Model           :
{self.model}\n'
                f'Software Version       :
{self.swversion}\n'
                f'Router Management Address:
{self.ip_add}'
        return desc

rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
rtr2 = Router('isr4221', '16.9.5',
'10.10.10.5')

print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
print('Rtr2\n', rtr2.getdesc(), sep='')
```

There are two routers instantiated in this example: **rtr1** and **rtr2**. Using the **print** function, you can call the **getdesc()** method to return formatted text about the object's attributes. The following output would be displayed:

[Click here to view code image](#)

```
Rtr1
Router Model           :iosV
Software Version       :15.6.7
Router Management Address:10.10.10.1

Rtr2
Router Model           :isr4221
Software Version       :16.9.5
Router Management Address:10.10.10.5
```

Inheritance



Inheritance in Python classes allows a child class to take on attributes and methods of another class. In the previous section, [Example 4-1](#) creates a class for routers, but what about switches? If you look at the **Router** class, you see that all of the attributes apply to a switch as well, so why not reuse the code already written for a new **Switch** class? The only part of [Example 4-1](#) that wouldn't work for a switch is the **getdesc()** method, which prints information about a router. When you use inheritance, you can replace methods and attributes that need to be different. To inherit in a class, you create the class as shown earlier in this chapter, but before the colon, you add parentheses that include the class from which you want to pull attributes and methods. It is important to note that the parent class must come before

the child class in the Python code. [Example 4-2](#) shows how this works, creating a second class named **Switch**, using the **Router** class as parent. In addition, it creates a different **getdesc()** method that prints text about a switch rather than about a router.

Example 4-2 Router Class and Switch Class with Inheritance

[Click here to view code image](#)

```
class Router:
    '''Router Class'''
    def __init__(self, model, swversion,
ip_add):
        '''initialize values'''
        self.model = model
        self.swversion = swversion
        self.ip_add = ip_add

    def getdesc(self):
        '''return a formatted description of
the router'''
        desc = (f'Router Model           :
{self.model}\n'
                f'Software Version       :
{self.swversion}\n'
                f'Router Management Address:
{self.ip_add}')
        return desc

class Switch(Router):
    def getdesc(self):
        '''return a formatted description of
the switch'''
        desc = (f'Switch Model           :
{self.model}\n'
                f'Software Version       :
{self.swversion}\n'
                f'Switch Management Address:
{self.ip_add}')
        return desc

rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
rtr2 = Router('isr4221', '16.9.5',
'10.10.10.5')
sw1 = Switch('Cat9300', '16.9.5', '10.10.10.8')

print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
print('Rtr2\n', rtr2.getdesc(), '\n', sep='')
print('Sw1\n', sw1.getdesc(), '\n', sep='')
```

You can add another variable named **sw1** and instantiate the **Switch** class just as you did the **Router** class, by passing in attributes. If you create another **print** statement using the newly created **sw1** object, you see the output shown in [Example 4-3](#).

Example 4-3 *Code Results of Using Class Inheritance*

[Click here to view code image](#)

```
Rtr1
Router Model           :iosV
Software Version      :15.6.7
Router Management Address:10.10.10.1

Rtr2
Router Model           :isr4221
Software Version      :16.9.5
Router Management Address:10.10.10.5

Sw1
Switch Model          :Cat9300
Software Version      :16.9.5
Switch Management Address:10.10.10.8
```

To learn more about classes, methods, and inheritance, you can refer to the Python documentation.

<https://docs.python.org/3/tutorial/classes.html>

WORKING WITH PYTHON MODULES



A central goal of OOP is to allow you to build modular software that breaks code up into smaller, easier-to-understand pieces. One big file with thousands of lines of code would be extremely difficult to maintain and work with. If you are going to break up your code into functions and classes, you can also separate that code into smaller chunks that hold key structures and classes

and allow them to be physically moved into other files, called *modules*, that can be included in your main Python code with the **import** statement. Creating modular code provides the following benefits:

- **Easier readability/maintainability:** Code written in a modular fashion is inherently easier to read and follow. It's like chapters in a book providing groupings of similar concepts and topics. Even the best programmers struggle to understand line after line of code, and modularity makes maintaining and modifying code much easier.
- **Low coupling/high cohesion:** Modular code should be written in such a way that modules do not have interdependencies. Each module should be self-contained so that changes to one module do not affect other modules or code. In addition, a module should only include functions and capabilities related to what the module is supposed to do. When you spread your code around multiple modules, bouncing back and forth, it is really difficult to follow. This paradigm is called low coupling/high cohesion modular design.
- **Code reusability:** Modules allow for easy reusability of your code, which saves you time and makes it possible to share useful code.
- **Collaboration:** You often need to work with others as you build functional code for an organization. Being able to split up the work and have different people work on different modules speeds up the code-production process.

There are a few different ways you can use modules in Python. The first and easiest way is to use one of the many modules that are included in the Python standard library or install one of thousands of third-party modules by using **pip**. Much of the functionality you might need or think of has probably already been written, and using modules that are already available can save you a lot of time. Another way to use modules is to build them in the Python language by simply writing some code in your editor, giving the file a name, and appending a `.py` extension. Using your own custom modules does add a bit of processing overhead to your application, as Python is an interpreted language and has to convert your text into machine-readable instructions on the fly. Finally, you can program a module in the C language, compile it, and then add its capabilities to your Python program. Compared to writing your own modules in Python, this method results in faster runtime for your code, but it is a lot more work. Many of the third-party modules and

those included as part of the standard library in Python are built this way.

Importing a Module

All modules are accessed the same way in Python: by using the **import** command. Within a program—by convention at the very beginning of the code—you type **import** followed by the module name you want to use. The following example uses the **math** module from the standard library:

[Click here to view code image](#)

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__loader__', '__name__',
 '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh',
 'atan', 'atan2',
 'atanh', 'ceil', 'comb', 'copysign', 'cos',
 'cosh', 'degrees',
 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1',
 'fabs', 'factorial',
 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd',
 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt',
 'ldexp', 'lgam-
ma', 'log', 'log10', 'log1p', 'log2', 'modf',
 'nan', 'perm', 'pi',
 'pow', 'prod', 'radians', 'remainder', 'sin',
 'sinh', 'sqrt',
 'tan', 'tanh', 'tau', 'trunc']
```

After you import a module, you can use the **dir()** function to get a list of all the methods available as part of the module. The ones in the beginning with the `__` are internal to Python and are not generally useful in your programs. All the others, however, are functions that are now available for your program to access. As shown in [Example 4-4](#), you can use the **help()** function to get more details and read the documentation on the **math** module.

Example 4-4 *math* Module Help

[Click here to view code image](#)


```
>>> help(math)
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is
    automatically generated from the Python
    source files. It may be incomplete,
    incorrect or include features that
    are considered implementation detail and
    may vary between Python
    implementations. When in doubt, consult
    the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the
    mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in
        radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of
        x.

    asin(x, /)
        Return the arc sine (measured in
        radians) of x.
    -Snip for brevity-
```

You can also use **help()** to look at the documentation on a specific function, as in this example:

[Click here to view code image](#)

```
>>> help(math.sqrt)
Help on built-in function sqrt in module math:
```

```
sqrt(x, /)
    Return the square root of x.
```

If you want to get a square root of a number, you can use the **sqrt()** method by calling **math.sqrt** and passing a value to it, as shown here:

```
>>> math.sqrt(15)
3.872983346207417
```

You have to type a module's name each time you want to use one of its capabilities. This isn't too painful if you're using a module with a short name, such as **math**, but if you use a module with a longer name, such as the **calendar** module, you might wish you could shorten the module name. Python lets you do this by adding **as** and a short version of the module name to the end of the **import** command. For example, you can use this command to shorten the name of the **calendar** module to **cal**.

```
>>> import calendar as cal
```

Now you can use **cal** as an alias for **calendar** in your code, as shown in this example:

[Click here to view code image](#)

```
>>> print(cal.month(2020, 2, 2, 1))
```

```
February 2020
Mo Tu We Th Fr Sa Su
      1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29
```

Importing a whole module when you need only a specific method or function adds unneeded overhead. To help with this, Python allows you to import specific methods by using the **from** syntax. Here is an example of importing the **sqrt()** and **tan()** methods:

[Click here to view code image](#)

```
>>> from math import sqrt,tan
>>> sqrt(15)
3.872983346207417
```

As you can see here, you can import more than one method by separating the methods you want with commas.

Notice that you no longer have to use **math.sqrt** and can just call **sqrt()** as a function, since you imported only the module functions you needed. Less typing is always a nice side benefit.

The Python Standard Library

The Python standard library, which is automatically installed when you load Python, has an extensive range of prebuilt modules you can use in your applications. Many are built in C and can make life easier for programmers looking to solve common problems quickly. Throughout this book, you will see many of these modules used to interact programmatically with Cisco infrastructure. To get a complete list of the modules in the standard library, go to at <https://docs.python.org/3/library/>. This documentation lists the modules you can use and also describes how to use them.

Importing Your Own Modules

As discussed in this chapter, modules are Python files that save you time and make your code readable. To save the class example from earlier in this chapter as a

module, you just need to save all of the code for defining the class and the attributes and functions as a separate file with the .py extension. You can import your own modules by using the same methods shown previously with standard library modules. By default, Python looks for a module in the same directory as the Python program you are importing into. If it doesn't find the file there, it looks through your operating system's **path** statements. To print out the paths your OS will search through, consider this example of importing the **sys** module and using the **sys.path** method:

[Click here to view code image](#)

```
>>> import sys

>>> sys.path

['', '/Users/chrijack/Documents',
 '/Library/Frameworks/Python.
framework/Versions/3.8/lib/python38.zip',
 '/Library/Frameworks/
Python.framework/Versions/3.8/lib/python3.8',
 '/Library/Frameworks/
Python.framework/Versions/3.8/lib/python3.8/lib-
dynload', '/
Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/
site-packages']
```

Depending on your OS (this output is from a Mac), the previous code might look different from what you see here, but it should still show you what Python sees, so it is useful if you are having trouble importing a module.

If you remove the class from the code shown in [Example 4-2](#) and store it in a separate file named device.py, you can import the classes from your new module and end up with the following program, which is a lot more readable while still operating exactly the same:

[Click here to view code image](#)

```
from device import Router, Switch

rtrl = Router('iosV', '15.6.7', '10.10.10.1')
```

```
rtr2 = Router('isr4221', '16.9.5', '10.10.10.5')
sw1 = Switch('Cat9300', '16.9.5', '10.10.10.8')

print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
print('Rtr2\n', rtr2.getdesc(), '\n', sep='')

print('Sw1\n', sw1.getdesc(), '\n', sep='')
```

When you execute this program, you get the output shown in [Example 4-5](#). If you compare these results with the results shown in [Example 4-3](#), you see that they are exactly the same. Therefore, the device module is just Python code that is stored in another file but used in your program.

Example 4-5 *Code Results of device.py Import as a Module*

[Click here to view code image](#)

```
Rtr1
Router Model           :iosV
Software Version       :15.6.7
Router Management Address:10.10.10.1

Rtr2
Router Model           :isr4221
Software Version       :16.9.5
Router Management Address:10.10.10.5

Sw1
Switch Model           :Cat9300
Software Version       :16.9.5
Router Management Address:10.10.10.8
```

Useful Python Modules for Cisco Infrastructure

This chapter cannot cover every single module that you might find valuable when writing Python code to interact with Cisco infrastructure. As you become more familiar with Python, you will come to love and trust a wide range of standard library and third-party modules. The following list includes many that are widely used to

automate network infrastructure. Many of these modules are used throughout this book, so you will be able to see them in action. The following list provides a description of each one, how to install it (if it is not part of the standard library), and the syntax to use in your Python **import** statement:



- General-purpose standard library modules:
 - **pprint:** The pretty print module is a more intelligent print function that makes it much easier to display text and data by, for example, aligning data for better readability. Use the following command to import this module:

```
from pprint import pprint
```

- **sys:** This module allows you to interact with the Python interpreter and manipulate and view values. Use the following command to import this module:

```
import sys
```

- **os:** This module gives you access to the underlying operating system environment and file system. It allows you to open files and interact with OS variables. Use the following command to import this module:

```
import os
```

- **datetime:** This module allows you to create, format, and work with calendar dates and time. It also enables timestamps and other useful additions to logging and data. Use the following command to import this module:

```
import datetime
```

- **time:** This module allows you to add time-based delays and clock capabilities to your Python apps. Use the following command to import this module:

```
import time
```

- **Modules for working with data:**

- **xmltodict:** This module translates XML-formatted files into native Python dictionaries (key/value pairs) and back to XML, if needed. Use the following command to install this module:

```
pip install xmltodict
```

Use the following command to import this module:

```
import xmltodict
```

- **csv:** This is a standard library module for understanding CSV files. It is useful for exporting Excel spreadsheets into a format that you can then import into Python as a data source. It can, for example, read in a CSV file and use it as a Python list data type. Use the following command to import this module:

```
import csv
```

- **json:** This is a standard library module for reading JSON-formatted data sources and easily converting them to dictionaries. Use the following command to import this module:

```
import json
```

- **PyYAML:** This module converts YAML files to Python objects that can be converted to Python dictionaries or lists. Use the following command to install this module:

```
pip install PyYAML
```

Use the following command to import this module:

```
import yaml
```

- **pyang:** This isn't a typical module you import into a Python program. It's a utility written in Python that you can use to verify your YANG models, create YANG code, and transform YANG

models into other data structures, such as XSD (XML Schema Definition). Use the following command to install this module:

```
pip install pyang
```

- **Tools for API interaction:**

- **requests:** This is a full library to interact with HTTP services and used extensively to interact with REST APIs. Use the following command to install this module:

```
pip install requests
```

Use the following command to import this module:

```
import requests
```

- **ncclient:** This Python library helps with client-side scripting and application integration for the NETCONF protocol. Use the following command to install this module:

```
pip install ncclient
```

Use the following command to import this module:

```
from ncclient import manager
```

- **netmiko:** This connection-handling library makes it easier to initiate SSH connections to network devices. This module is intended to help bridge the programmability gap between devices with APIs and those without APIs that still rely on command-line interfaces and commands. It relies on the paramiko module and works with multiple vendor platforms. Use the following command to install this module:

```
pip install netmiko
```

Use the following command to import this module:

[Click here to view code image](#)

```
from netmiko import ConnectHandler
```


- **pysnmp:** This is a Python implementation of an SNMP engine for network management. It allows you to interact with older infrastructure components without APIs but that do support SNMP for management. Use the following command to install this module:

```
pip install pysnmp
```

Use the following command to import this module:

```
import pysnmp
```

- **Automation tools:**

- **napalm:** napalm (Network Automation and Programmability Abstraction Layer with Multivendor Support) is a Python module that provides functionality that works in a multivendor fashion. Use the following command to install this module:

```
pip install napalm
```

Use the following command to import this module:

```
import napalm
```

- **nornir:** This is an extendable, multithreaded framework with inventory management to work with large numbers of network devices. Use the following command to install this module:

```
pip install nornir
```

Use the following command to import this module:

[Click here to view code image](#)

```
from nornir.core import InitNornir
```

- **Testing tools:**

- **unittest:** This standard library testing module is used to test the functionality of Python code. It is often used for automated code testing and as part of test-driven development methodologies. Use the following command to import this module:

```
import unittest
```

- **pyats:** This module was a gift from Cisco to the development community. Originally named Genie, it was an internal testing framework used by Cisco developers to validate their code for Cisco products. pyats is an incredible framework for constructing automated testing for infrastructure as code. Use the following command to install this module:

[Click here to view code image](#)

```
pip install pyats (just installs the core fra  
documentation for more options)
```

Many parts of the **pyats** framework can be imported. Check the documentation on how to use it.

Chapter 5, “Working with Data in Python,” places more focus on techniques and tools used to interact with data in Python. This will round out the key Python knowledge needed to follow along with the examples in the rest of the book.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 4-2](#) lists these key topics and the page number on which each is found.



Table 4-2 Key Topics

Key Topic Element	Description	Page Number
Paragraph	Defining functions	<u>88</u>
Paragraph	The value of object-oriented programming	<u>92</u>
Paragraph	Defining classes	<u>92</u>
Paragraph	Inheritance	<u>94</u>
Paragraph	Python modules	<u>96</u>
Bulleted list	Common Python modules	<u>10</u> <u>1</u>

DEFINE KEY TERMS

There are no key terms for this chapter.

Chapter 5

Working with Data in Python

This chapter covers the following topics:

- **File Input and Output:** This section shows how to work with test files in Python.
- **Parsing Data:** This section discusses how to parse data into native Python objects.
- **Error Handling in Python:** This section discusses how to use try-except-else-finally to work through errors in Python input.
- **Test-Driven Development:** This section discusses using software testing to validate function.
- **Unit Testing:** This section discusses how to use the internal Python module unittest to automate Python code testing.

There are numerous ways to ingest data into a Python program. You can get input from the user, pull data from a website or an API, or read data from a file. The trick is being able to convert data from a data source into native Python structures and objects so that you can use it to automate your infrastructure. This chapter discusses a number of ways to use built-in and third-party modules to transform different types of data into Python dictionaries, lists, and other data collection objects. The rest of the book provides more detail on how to use these techniques to interact with Cisco infrastructure; this chapter provides a foundation for understanding how these data formats differ and how best to interact with them.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you

are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 5-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 5-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
File Input and Output	1, 2
Parsing Data	3–6
Error Handling in Python	7, 8
Test-Driven Development	9
Unit Testing	10, 11

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. When parsing a text file, what determines the end of a line?
 1. Return code

2. Nothing; Python sees it as one big string
3. \n or EOF
4. All of the above

2. What syntax would you use to open a text file to be written to?

1. `data = open("text.txt", "w")`
2. `data = load("text.txt", "w")`
3. `load("text.txt", "w")`
4. `open("text.txt", "w")`

3. Which of the following do you use to write to a CSV file in Python?

1. with `open("text.csv", "a")` as filehandle:
`csv_writer = csv.writer(filehandle)`
`csv_writer.writerow(data)`
2. with `open("text.csv", "a")` as filehandle:
`csv_writer.writerow(data)`
3. with `open("text.csv", "a")` as filehandle:
`csv_writer = csv.writer(filehandle)`
`csv_writer.writerow(data)`
4. with `open("text.csv", "a")` as filehandle:
`csv_writer = csv.writer(f)`
`csv_writer.writerow(data)`

4. Which module is imported to read XML data?

1. `xmlm`
2. `xmldict`
3. `XMLParse`
4. None of the above

5. Which methods are used for converting a native JSON file to Python and then back to JSON?
(Choose two.)

1. `load()` and `dump()`
2. `loads()` and `dump()`
3. `loads()` and `dumps()`
4. `load()` and `dumps()`

6. What does YAML stand for?

1. Yet Another Markup Language
2. YAML Ain't Markup Language
3. The name of its creator
4. None of the above

7. What is the syntax for error handling in Python?

1. `try-except-else-finally`
2. `raise ErrorMessage`
3. `assertErrorValue`

4. All of the above

8. When does the **finally** block execute?

1. After the **try** block is successful
2. After the **except** block
3. At the end of every **try** block
4. When an error code stops the **else** block

9. Test-driven development requires that developers:

1. Create a unit test for every bit of code they write
2. Know how to use DevOps tools for automated testing
3. Create a simple test that fails and then write code that allows the test to succeed
4. Completely unnecessary in an Agile development shop

10. What is the difference between a unit test and an integration test? (Choose two.)

1. An integration test is for validation of how different parts of the application work together.
2. An integration test verifies that the application operates as expected.
3. A unit test verifies API functionality.
4. A unit test is most specific in scope and tests small bits of code.

11. Which class is inherited as part of a unit test?

1. `unittest.testcase`
2. `unittest.TestCase`
3. `unittest`
4. `TestCase`

FOUNDATION TOPICS

FILE INPUT AND OUTPUT

Pulling data from a file in Python is very straightforward. To extract data from a text file, you can use native Python capabilities. Binary files, on the other hand, need to be processed by some module or another external program before it is possible to extract the data from them. The vast majority of your needs will be addressed through text files, so that is what this section focuses on.

From Python's perspective, a text file can be thought of as a sequence of lines. Each line, as it is read in to Python, is typically 79 characters long (per PEP 8

convention), with a newline character at the end (`\n` for Python). There are just two functions that you need to know when working with a text file: **open()** and **close()**.

To open a file and read in its contents, you have to first tell Python the name of the file you want to work with. You do this by using the **open()** function and assigning the output to a Python object (the variable **readdata** in this case). The function returns a file handle, which Python uses to perform various operations on the file. The code looks as follows:

[Click here to view code image](#)

```
readdata = open("textfile.txt", "r")
```

The **open()** function requires two arguments: the name of the file as a string and the mode that you want to open the file. In the preceding example, it opens the file in read mode. There are numerous options you can use when you set mode, and you can combine them in some cases to fine-tune how you want Python to handle the file. The following are some of the options:

- **r**: Open for reading (default)
- **w**: Open for writing, truncating the file first
- **x**: Open for exclusive creation, failing if the file already exists
- **a**: Open for writing, appending to the end of the file if it exists
- **b**: Open in binary mode
- **t**: Open in text mode (default)
- **+**: Open for updating (reading and writing)

With the previous code, you now have a file handling object named **readdata**, and you can use methods to interact with the file methods. To print the contents of the file, you can use the following:

[Click here to view code image](#)


```
print(readdata.read())
```

```
Line one of a text file
```

```
Line two of a text file, just like line one, but  
the second one.
```

```
Third line of a text file.
```

When using the **open()** function, you have to remember to close the file when you are finished reading from it. If you don't, the file will stay open, and you might run in to file lock issues with the operating system while the Python app is running. To close the file, you simply use the **close()** method on the **readdata** object:

```
readdata.close()
```

Keeping track of the state of the file lock and whether you opened and closed it can be a bit of a chore. Python provides another way you can use to more easily work with files as well as other Python objects. The **with** statement (also called a *context manager* in Python) uses the **open()** function but doesn't require direct assignment to a variable. It also has better exception handling and automatically closes the file for you when you have finished reading in or writing to the file. Here's an example:

[Click here to view code image](#)

```
with open("textfile.txt", "r") as data:  
    print(data.read())
```

This is much simpler code, and you can use all of the same methods to interact with the files as before. To write to a file, you can use the same structure, but in this case, because you want to append some data to the file, you need to change how you open the file to allow for writing. In this example, you can use **"a+"** to allow

reading and appending to the end of the file. Here is what the code would look like:

[Click here to view code image](#)

```
with open("textfile.txt", "a+") as data:
    data.write('\nFourth line added by Python')
```

Notice the newline in front of the text you are appending to the file. It appears here so that it isn't just tacked on at the very end of the text. Now you can read the file and see what you added:

[Click here to view code image](#)

```
with open ("textfile.txt", "r") as data:
    print(data.read())

Line one of a text file
Line two of a text file, just like line one, but
the second one.
Third line of a text file.
Fourth line added by Python
```

PARSING DATA

Imagine a world where all the data is in nice, neatly formatted cells, completely structured, and always consistent. Unfortunately, data is not so easily accessible, as there are a multitude of types, structures, and formats. This is why it is essential that you learn how to parse data in some of the more common forms within your Python programs.

Comma-Separated Values (CSV)

A CSV file is just a plaintext spreadsheet or database file. All of those spreadsheets or databases that you have with infrastructure information can be easily exported as CSV files so that you can use them as source data in Python.

Each line in a CSV file represents a row, and commas are used to separate the individual data fields to make it easier to parse the data. Python has a built-in CSV module that you can import that understands the CSV format and simplifies your code. The following is an example of a typical CSV file (in this case named routerlist.csv):

[Click here to view code image](#)

```
"router1", "192.168.10.1", "Nashville"  
"router2", "192.168.20.1", "Tampa"  
"router3", "192.168.30.1", "San Jose"
```

This example shows a common asset list or device inventory, such as one that you might pull from a network management system or simply keep track of locally. To start working with this data, you have to import the CSV module, and then you need to create a **reader** object to read your CSV file into. You first have to read the file into a file handle, and then you run the CSV read function on it and pass the results on to a **reader** object. From there, you can begin to use the CSV data as you wish. You can create another variable and pass the **reader** object variable to the built-in **list()** function. Here is what the code would look like:

[Click here to view code image](#)

```
>>> import csv  
  
>>> samplefile = open('routerlist.csv')  
  
>>> samplereader = csv.reader(samplefile)  
  
>>> sampledata = list(samplereader)  
  
>>> sampledata  
[['router1', '192.168.10.1', 'Nashville'],  
 ['router2',  
 '192.168.20.1', 'Tampa'], ['router3',  
 '192.168.30.1', 'San Jose ']]
```

In this example, you now have a list of lists that includes each row of data. If you wanted to manipulate this data, you could because it's now in a native format for Python. Using list notation, you can extract individual pieces of information:

[Click here to view code image](#)

```
>>> sampledata[0]
['router1', '192.168.10.1', 'Nashville']
>>> sampledata[0][1]
'192.168.10.1'
```

Using **with**, you can iterate through the CSV data and display information in an easier way:

[Click here to view code image](#)

```
import csv

with open("routerlist.csv") as data:
    csv_list = csv.reader(data)
    for row in csv_list:
        device = row[0]
        location = row[2]
        ip = row[1]
        print(f"{device} is in {location.rstrip()}
and has IP
{ip}.")
```

Notice the **rstrip** function used to format the **location** variable? You use it because the last entry in your CSV file will have a whitespace character at the very end when it is read into Python because it is at the very end of the file. If you don't get rid of it (by using **rstrip**), your formatting will be off.

The output of this code is as follows:

[Click here to view code image](#)

```
router1 is in Nashville and has IP 192.168.10.1.  
router2 is in Tampa and has IP 192.168.20.1.  
router3 is in San Jose and has IP 192.168.30.1.
```

If you want to add a fourth device to the CSV file, you can follow a process very similar to what you did with text files. [Example 5-1](#) shows how to add a little interaction from the command line to fill in the fields and create a Python list with details on the new router. Instead using of a **reader** object, this example uses a **writer** object to store the formatted CSV data and then write it to the file.

Example 5-1 Code and Input for a CSV File

[Click here to view code image](#)

```
import csv  
  
print("Please add a new router to the list")  
hostname = input("What is the hostname? ")  
ip = input("What is the ip address? ")  
location = input("What is the location? ")  
  
router = [hostname, ip, location]  
  
with open("routerlist.csv", "a") as data:  
    csv_writer = csv.writer(data)  
    csv_writer.writerow(router)  
  
<Below is interactive from the terminal after  
running the above code>  
Please add a new router to the list  
What is the hostname? router4  
What is the ip address? 192.168.40.1  
What is the location? London
```

If you run the code shown in [Example 5-1](#) and input details for router 4, now when you display the router list, you have the new router included as well:

[Click here to view code image](#)

```
router1 is in Nashville and has IP 192.168.10.1.  
router2 is in Tampa and has IP 192.168.20.1.
```

```
router3 is in San Jose and has IP 192.168.30.1.  
router4 is in London and has IP 192.168.40.1.
```

JavaScript Object Notation (JSON)



JavaScript Object Notation (JSON) is a data structure that is derived from the Java programming language, but it can be used as a portable data structure for any programming language. It was built to be an easily readable and standard way for transporting data back and forth between applications. JSON is heavily used in web services and is one of the core data formats you need to know how to use in order to interact with Cisco infrastructure. The data structure is built around key/value pairs that simplify mapping of data and its retrieval. [Example 5-2](#) shows an example of JSON.

Example 5-2 JSON

[Click here to view code image](#)

```
{  
  "interface": {  
    "name": "GigabitEthernet1",  
    "description": "Router Uplink",  
    "enabled": true,  
    "ipv4": {  
      "address": [  
        {  
          "ip": "192.168.1.1",  
          "netmask": "255.255.255.0"  
        }  
      ]  
    }  
  }  
}
```

In [Example 5-2](#), you can see the structure that JSON provides. **interface** is the main data object, and you can see that its value is multiple key/value pairs. This nesting

capability allows you to structure very sophisticated data models. Notice how similar to a Python dictionary the data looks. You can easily convert JSON to lists (for a JSON array) and dictionaries (for JSON objects) with the built-in JSON module. There are four functions that you work with to perform the conversion of JSON data into Python objects and back.

- **load()**: This allows you to import native JSON and convert it to a Python dictionary from a file.
- **loads()**: This will import JSON data from a string for parsing and manipulating within your program.
- **dump()**: This is used to write JSON data from Python objects to a file.
- **dumps()**: This allows you to take JSON dictionary data and convert it into a serialized string for parsing and manipulating within Python.

The **s** at the end of **dump** and **load** refers to a string, as in *dump string*. To see this in action, you load the JSON file and map the file handle to a Python object (data) like so:

[Click here to view code image](#)

```
import json

with open("json_sample.json") as data:
    json_data = data.read()

json_dict = json.loads(json_data)
```

The object **json_dict** has taken the output of **json.loads(json_data)** and now holds the **json** object as a Python dictionary:

[Click here to view code image](#)

```
>>> type(json_dict)
<class 'dict'>

>>> print(json_dict)
{'interface': {'name': 'GigabitEthernet1',
'description':
```

```
'Router Uplink', 'enabled': True, 'ipv4':
{'address':
[{'ip': '192.168.0.2', 'netmask':
'255.255.255.0'}]}}
```

You can now modify any of the key/value pairs, as in this example, where the description is changed:

[Click here to view code image](#)

```
>>> json_dict["interface"]["description"] =
"Backup Link"

>>> print(json_dict)

{'interface': {'name': 'GigabitEthernet1',
'description': 'Backup
Link', 'enabled': True, 'ipv4': {'address':
[{'ip': '192.168.0.2',
'netmask': '255.255.255.0'}]}}
```

In order to save the new **json** object back to a file, you have to use the **dump()** function (without the **s**) to convert the Python dictionary back into a JSON file object. To make it easier to read, you can use the **indent** keyword:

[Click here to view code image](#)

```
with open("json_sample.json", "w") as fh:
    json.dump(json_dict, fh, indent = 4)
```

Now if you load the file again and print, you can see the stored changes, as shown in [Example 5-3](#).

Example 5-3 *Loading the JSON File and Printing the Output to the Screen*

[Click here to view code image](#)

```
>>> with open ("json_sample.json") as data:
    json_data = data.read()
    print(json_data)
{
  "interface": {
    "name": "GigabitEthernet1",
    "description": "Backup Link",
    "enabled": true,
    "ipv4": {
```



```
        "address": [  
            {  
                "ip": "192.168.0.2",  
                "netmask": "255.255.255.0"  
            }  
        ]  
    }  
}>>>
```



Extensible Markup Language (XML)

Extensible Markup Language (XML) is a very common data format that is used heavily in configuration automation. Parsing XML is similar to using other data formats, in that Python natively understands and can support XML encoding and decoding. The following is a very simple example of what XML structure looks like.

```
<device>  
  <Hostname>Rtr01</Hostname>  
  <IPv4>192.168.1.5</IPv4>  
  <IPv6> </IPv6>  
</device>
```

It should come as no surprise that XML looks a bit like HTML syntax; it was designed to work hand-in-hand with HTML for data transport and storage between web services and APIs. XML has a tree structure, with the root element being at the very top. There is a parent/child relationship between elements. In the preceding example, **device** is the root element that has **Hostname**, **IPv4**, and **IPv6** as child elements. Just like with HTML, a tag has meaning and is used to enclose the relationships of the elements with a start tag (<>) and a closing tag (</>). It's not all that different from JSON in

that a tag acts as a key with a value. You can also assign attributes to a tag by using the following syntax:

```
attribute name="some value"
```

This works the same as an element in that it can provide a way to represent data. [Example 5-4](#) shows an example of an IETF interface YANG model in XML.

Example 5-4 *YANG Model Represented in XML*

[Click here to view code image](#)

```
<?xml version="1.0" encoding="UTF-8" ?>
<interface xmlns="ietf-interfaces">
  <name>GigabitEthernet2</name>
  <description>Wide Area Network</description>
  <enabled>true</enabled>
  <ipv4>
    <address>
      <ip>192.168.1.5</ip>
      <netmask>255.255.255.0</netmask>
    </address>
  </ipv4>
</interface>
```

To work with this, you can use the native XML library, but it has a bit of a learning curve and can be a little hard to use if you just want to convert XML into something you can work with in Python. To make it easier, you can use a module called `xmltodict` to convert XML into an ordered dictionary in Python. This is a special class of dictionary that does not allow elements to change order. Since dictionaries use key/value pairs, where the key/value pairs are stored is normally not a problem, but in the case of XML, order matters. [Example 5-5](#) reads in the XML from [Example 5-4](#) and converts it to an ordered dictionary.

Example 5-5 *Reading in XML and Printing the Imported Dictionary to the Command Line*

[Click here to view code image](#)

```
import xmltodict

with open("xml_sample.xml") as data:
    xml_example = data.read()

xml_dict = xmltodict.parse(xml_example)

>>> print(xml_dict)
OrderedDict([('interface',
OrderedDict([('xmlns', 'ietf-interfaces'),
('name',
'GigabitEthernet2'), ('description', 'Wide Area
Network'), ('enabled', 'true'),
('ipv4', OrderedDict([('address',
OrderedDict([('ip', '192.168.0.2'), ('netmask',
'255.255.255.0')]))]))]))])
```

Now that you have the XML in a Python dictionary, you can modify an element, as shown here:

[Click here to view code image](#)

```
xml_dict["interface"]["ipv4"]["address"]["ip"] =
"192.168.55.3"
```

You can see your changes in XML format by using the **unparse** function (see [Example 5-6](#)). You can use the **pretty=True** argument to format the XML to make it a bit easier to read. XML doesn't care about whitespace, but humans do for readability.

Example 5-6 *Printing from the Command Line with the unparse Function*

[Click here to view code image](#)

```
>>> print(xmltodict.unparse(xml_dict,
pretty=True))
<?xml version="1.0" encoding="utf-8"?>
<interface xmlns="ietf-interfaces">
  <name>GigabitEthernet2</name>
  <description>Wide Area
Network</description>
  <enabled>true</enabled>
  <ipv4>
```

```
<address>
  <ip>192.168.55.3</ip>
  <netmask>255.255.255.0</netmask>
</address>
</ipv4>
</interface>
```

To write these changes back to your original file, you can use the following code:

[Click here to view code image](#)

```
with open("xml_sample.xml", "w") as data:
    data.write(xmltodict.unparse(xml_dict,
    pretty=True))
```

YAML Ain't Markup Language (YAML)

YAML is an extremely popular human-readable format for constructing configuration files and storing data. It was built for the same use cases as XML but has a much simpler syntax and structure. It uses Python-like indentation to differentiate blocks of information and was actually built based on JSON syntax but with a whole host of features that are unavailable in JSON (such as comments). If you have ever used Docker or Kubernetes, you have undoubtedly run into YAML files. The following is an example of YAML:

[Click here to view code image](#)

```
---
interface:
  name: GigabitEthernet2
  description: Wide Area Network
  enabled: true
  ipv4:
    address:
      - ip: 172.16.0.2
      netmask: 255.255.255.0
```

Notice that a YAML object has minimal syntax, all data that is related has the same indentation level, and key/value pairs are used to store data. YAML can also represent a list by using the - character to identify elements, as in the following example of IP addresses.

[Click here to view code image](#)

```
---
addresses:
  - ip: 172.16.0.2
    netmask: 255.255.255.0
  - ip: 172.16.0.3
    netmask: 255.255.255.0
  - ip: 172.16.0.4
    netmask: 255.255.255.0
```

To work with YAML in Python, you need to install and import the PyYaml module. Once you import it into your code, you can convert YAML to Python objects and back again. YAML objects are converted to dictionaries, and YAML lists automatically become Python lists. The two functions that perform this magic are **yaml.load** to convert from YAML objects into Python and **yaml.dump** to convert Python objects back to YAML. Just as in the other data examples, you can load a YAML file and then pass it to **yaml.load** to work its magic. The latest PyYaml module requires that you add an argument to tell it which loader you want to use. This is a security precaution so that your code will not be vulnerable to arbitrary code execution from a bad YAML file. Here is what the code looks like:

[Click here to view code image](#)

```
import yaml

with open("yaml_sample.yaml") as data:
    yaml_sample = data.read()
```

```
yaml_dict = yaml.load(yaml_sample,
Loader=yaml.FullLoader)
```

The variable **yaml_dict** is now a dictionary object containing your YAML file. Had this key/value been a YAML list, it would have created a list instead, like this:

[Click here to view code image](#)

```
>>> type(yaml_dict)
<class 'dict'>

>>> yaml_dict
{'interface': {'name': 'GigabitEthernet2',
'description': 'Wide
Area Network', 'enabled': True, 'ipv4':
{'address': [{'ip':
'192.168.0.2', 'netmask': '255.255.255.0'}]}}
```

As before, you can modify this object to your liking. For example, you can change the interface name to **GigabitEthernet1**, as shown in [Example 5-7](#).

Example 5-7 *Changing an Interface Name Within the Python Dictionary and Printing the Results*

[Click here to view code image](#)

```
>>> yaml_dict["interface"]["name"] =
"GigabitEthernet1"

>>> print(yaml.dump(yaml_dict,
default_flow_style=False))
interface:
  description: Wide Area Network
  enabled: true
  ipv4:
    address:
      - ip: 192.168.0.2
        netmask: 255.255.255.0
  name: GigabitEthernet1
```

To write these changes back to your file, use the following code.

[Click here to view code image](#)

```
with open("yaml_sample.yaml", "w") as data:
    data.write(yaml.dump(yaml_dict,
                        default_flow_style=False))
```



ERROR HANDLING IN PYTHON

Whenever you are working with code, errors are bound to happen. In Python, errors often halt the execution of code, and the interpreter spits out some type of cryptic message. What if you wanted Python to tell the users what they did wrong and let them try again or perform some other task to recover from the error? That's where the **try-except-else-finally** code blocks come into play.

You have seen quite a bit of file access in this chapter. What happens if you ask the user for the filename instead of hard-coding it? If you did this, you would run the risk of a typo halting your program. In order to add some error handling to your code, you can use the **try** statement. [Example 5-8](#) shows an example of how this works.

Example 5-8 try-except-else-finally *Code Example*

[Click here to view code image](#)

```
x = 0
while True:

    try:
        filename = input("Which file would you
like to open? :")
        with open(filename, "r") as fh:
            file_data = fh.read()
    except FileNotFoundError:
```

```

        print(f'Sorry, {filename} doesn't
exist! Please try again.')
    else:
        print(file_data)
        x = 0
        break
    finally:
        x += 1
        if x == 3:
            print('Wrong filename 3
times.\nCheck name and Rerun.')
            break

```

In this example, a variable keeps track of the number of times the **while** loop will be run. This is useful for building in some logic to make sure the program doesn't drive the users crazy by constantly asking them to enter a filename. Next is an infinite **while** loop that uses the fact that the Boolean True will always result in continuing looping through the code. Next is the **try** statement, which contains the block of code you want to subject to error handling. You ask the user to enter a filename to open, and it is stored in the **filename** variable. This variable is used with **open()** to open a read-only text file and use the file handle object **fh**. The file handle object uses **read()** to store the text file in the **file_data** variable. If Python can't find the file specified, the **except FileNotFoundError** block of code is executed, printing an error message with the file's name and informing the user to try again. The **else** block runs only if an exception does not occur and the filename can be found. The **file_data** is printed, **x** is set to **0** (to empty the counter), the loop is stopped, and the **finally** block is run. The **finally** block runs regardless of whether an exception occurs each time through the loop. The **x** variable is incremented each time through the loop, and if the user gets the wrong filename three times, a message is printed, saying the user tried three times and to check the file. At this point, the loop is broken, and the script is halted.

Here is what the program output would look like with a valid **test.txt** file in the script directory:

[Click here to view code image](#)

```
Which file would you like to open? :test
Sorry, test doesn't exist! Please try again.
Which file would you like to open? :test.txt
Test file with some text.
Two lines long.
```

Here is what the output would look like with three wrong choices:

[Click here to view code image](#)

```
Which file would you like to open? :test
Sorry, test doesn't exist! Please try again.
Which file would you like to open? :test2
Sorry, test2 doesn't exist! Please try again.
Which file would you like to open? :test3
Sorry, test3 doesn't exist! Please try again.
Wrong filename 3 times.
Check name and Rerun.
```

There are quite a few other error-handling capabilities available in Python, and if you want to try to make your applications more user friendly, it would be worth your time to explore them. The latest documentation can be found at <https://docs.python.org/3/tutorial/errors.html>. This documentation discusses custom errors and provides more examples of types of errors you can use with the previous sample code.

TEST-DRIVEN DEVELOPMENT

Test-driven development (TDD) is an interesting concept that at first glance may seem completely backward. The idea is that you build a test case first, before any software has been created or modified. The goal is to streamline

the development process by focusing on only making changes or adding code that satisfies the goal of the test. In normal testing, you test after the software is written, which means you spend your time chasing errors and bugs more than writing code. By writing the test first, you spend your time focused on writing only what is needed and making your code simple, easier to understand, and hopefully bug free. [Figure 5-1](#) shows the TDD process in action.

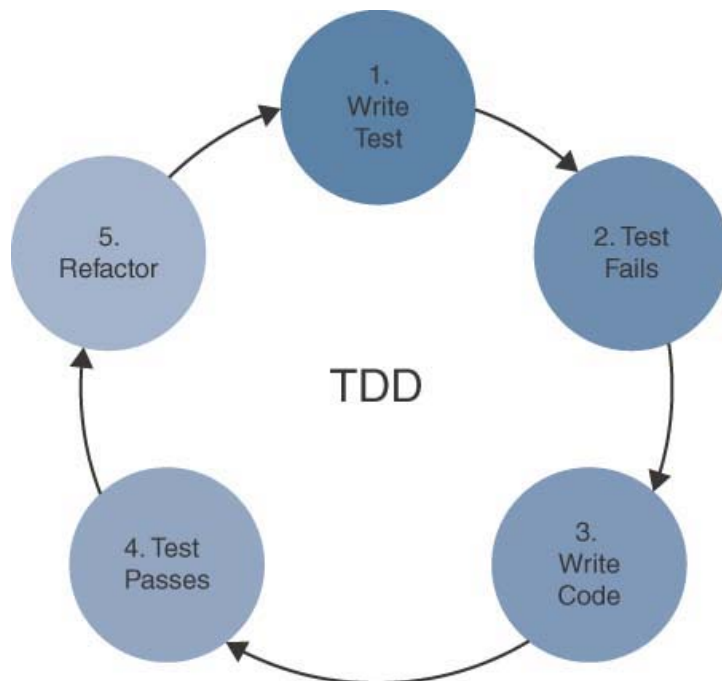


Figure 5-1 *Test-Driven Development in Action*



The following are the five steps of TDD:

Step 1. Write a test: Write a test that tests for the new class or function that you want to add to your code. Think about the class name and structure you will need in order to call the new capability that doesn't exist yet—and nothing more.

Step 2. Test fails: Of course, the test fails because you haven't written the part that works yet. The idea here is to think about the class or function you want and test for its intended output. This initial test failure shows you exactly where you should focus your code writing to get it to pass. This is like starting with your end state in mind, which is the most effective way to accomplish a goal.

Step 3. Write some code: Write only the code needed to make the new function or class successfully pass. This is about efficiency and focus.

Step 4. Test passes: The test now passes, and the code works.

Step 5. Refactor: Clean up the code as necessary, removing any test stubs or hard-coded variables used in testing. Refine the code, if needed, for speed.

TDD may seem like a waste of time initially. Why write tests for stuff you know isn't going to pass? Isn't all of this testing just wasted effort? The benefit of this style of development is that it starts with the end goal in mind, by defining success right away. The test you create is laser focused on the application's purpose and a clear outcome. Many programmers add too much to their code by trying to anticipate future needs or building in too much complexity for an otherwise simple problem. TDD works extremely well with the iterative nature of Agile development, with the side benefit of having plenty of test cases to show that the software works.

UNIT TESTING

Testing your software is not optional. Every script and application that you create have to go through testing of some sort. Maybe it's just testing your syntax in the interactive interpreter or using an IDE and trying your

code as you write it. While this is software testing, it's not structured and often is not repeatable. Did you test all options? Did you validate your expectations? What happens if you send unexpected input to a function? These are some of the reasons using a structured and automated testing methodology is crucial to creating resilient software.

A unit test is a type of test that is conducted on small, functional aspects of code. It's the lowest level of software testing and is interested in the logic and operation of only a single function in your code. That's not to say that you can't perform multiple tests at the same time. Computers are great at performing repetitive tasks, but the goal is for each test to be on one function at a time so that the testing is specific and consistent. Remember that a unit is the smallest testable part of your software.

There are other types of testing that you may hear about, such as integration testing and functional testing. The differences between these types of testing and unit testing come down to the scope of the test. As mentioned, a unit test is testing a small piece of code, such as a method or function. An integration test, on the other hand, tests how one software component works with the rest of the application. It is often used when modules of an application are developed by separate teams or when a distributed application has multiple components working together. A functional test (also called an end-to-end test) is the broadest in scope from a testing perspective. This is where the entire system is tested against the functional specifications and requirements of the software application. [Figure 5-2](#) shows a testing pyramid to put it in perspective.

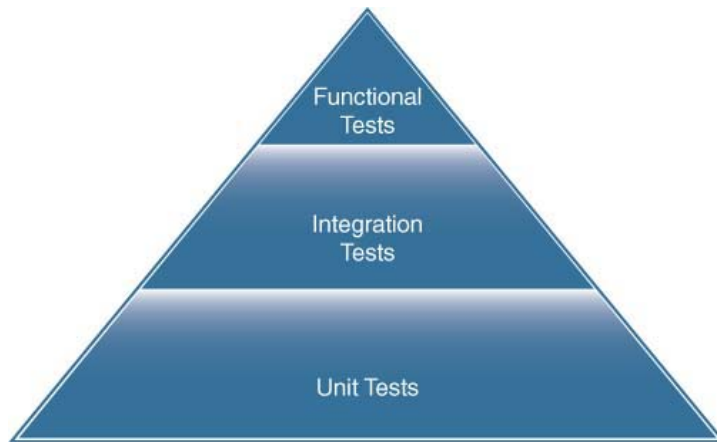


Figure 5-2 *Testing Pyramid*



Python has a built-in unit test module, named `unittest`. This module is quite full featured and can support a tremendous number of test cases. There are other testing modules that you can use, such as `Nose` and `PyTest` (who comes up with these names?), but for the purposes of the 200-901 DevNet Associate DEVASC exam, you need to know how `unittest` works. In order to use `unittest`, you need a bit of code to test. Here is a simple function that computes the area of a circle:

```
from math import pi

def area_of_circle(r):
    return pi*(r**2)
```

You import from the `math` module the `pi` method to make it a little easier. A function is defined with the name `area_of_circle` that takes the argument `r`. The function computes the radius of a circle and returns the value. This is very simple, but what happens if the function is called, and odd values are passed to it? You guessed it: lots of errors. So in order to test this function, you can create a unit test.

Certain conventions must be followed for a unit test. While you can create a unit test that has the code that you want to test all in the same file, it's a better idea to use object-oriented principles when building tests. In this case, the function you want to test is saved as `areacircle.py`, so following good practices you should name your unit test file `test_areacircle.py`. This makes it easy to differentiate the two. You should also import the `unittest` module, and from `areacircle` you can import the **`area_of_circle`** function. Import the **`pi`** method from `math` so that you can test your results. The **`import`** statements would look as follows:

[Click here to view code image](#)

```
import unittest
from areacircle import area_of_circle
from math import pi
```

Next, you need to create a class for your test. You can name it whatever you want, but you need to inherit **`unittest.TestCase`** from the `unittest` module. This is what enables the test function methods to be assigned to your **`test`** class. Next, you can define your first test function. In this case, you can test various inputs to validate that the math in your function under **`test`** is working as it should. You will notice a new method called **`assertAlmostEqual()`**, which takes the function you are testing, passes a value to it, and checks the returned value against an expected value. You can add a number of tests to this function. This is what the test now looks like with the additional code:

[Click here to view code image](#)

```
class
Test_Area_of_Circle_input(unittest.TestCase):
    def test_area(self):
        # Test radius >= 0
```

```
        self.assertAlmostEqual(area_of_circle(1),
pi)
        self.assertAlmostEqual(area_of_circle(0),
0)
        self.assertAlmostEqual(area_of_circle(3.5),
pi * 3.5**2)
```

You can go to the directory where these two scripts reside and enter **python -m unittest test_areacircle.py** to run the test. If you don't want to type all that, you can add the following to the bottom of the **test_areacircle.py** script to allow the unittest module to be launched when you run the test script:

```
if __name__ == '__main__':
    unittest.main()
```

All this does is check to see if the script is being run directly (because the **__main__** special case is an attribute for all Python scripts run from the command line) and call the **unittest.main()** function. After executing the function, you should see the following results:

[Click here to view code image](#)

```
.
-----
-----
Ran 1 test in 0.000s

OK
```

The dot at the top shows that 1 test ran (even though you had multiple checks in the same function) to determine whether the values submitted produced an error. Since all are valid for the function, the unit test came back successfully.

Now you can check to see if a negative number causes a problem. Create a new function under your previous **test_area** function. Name this function **test_values**. (The test at the beginning is required, or unittest will ignore the function and not check it.) You can use the **assertRaises** check, which will be looking for a **ValueError** exception for the function **area_of_circle**, and pass it a value of **-1**. The following function can be added to your code:

[Click here to view code image](#)

```
def test_values(self):
    # Test that bad values are caught
    self.assertRaises(ValueError,
        area_of_circle, -1)
```

[Example 5-9](#) shows the output of the test with this additional check.



Example 5-9 *Output from Adding a New Test That Fails*

[Click here to view code image](#)

```
.F
=====
FAIL: test_values
(__main__.Test_Area_of_Circle_input)
-----
Traceback (most recent call last):
  File
"/Users/chrijack/Documents/ccnadevnet/test_areacircle.py",
  line 14, in
    test_values
      self.assertRaises(ValueError,
area_of_circle, -1)
AssertionError: ValueError not raised by
area_of_circle
-----
```



```
-----  
Ran 2 tests in 0.001s  
  
FAILED (failures=1)
```

The first check is still good, so you see one dot at the top, but next to it is a big **F** for fail. You get a message saying that the **test_value** function is where it failed, and see that your original function did not catch this error. This means that the code is giving bad results. A radius of **-1** is not possible, but the function gives you the following output:

```
>>> area_of_circle(-1)  
3.141592653589793
```



To fix this, you go back to your original function and some error-checking code. You use a simple **if** statement to check for a negative number, and you raise a **ValueError** with a message to the user about invalid input:

[Click here to view code image](#)

```
from math import pi  
  
def area_of_circle(r):  
    if r < 0:  
        raise ValueError('Negative radius value  
error')  
    return pi*(r**2)
```

Now when you try the test from the interpreter, you see an error raised:

[Click here to view code image](#)

```
>>> area_of_circle(-1)

Traceback (most recent call last):

  File "<pyshell>", line 1, in <module>

    File
"/Users/chrijack/Documents/ccnadevnet/areacircle.py",
  line 5, in area_of_circle

    raise ValueError('Negative radius value
error')

ValueError: Negative radius value error
```

If you rerun the unit test, you see that it now passes the new check because an error is raised:

[Click here to view code image](#)

```
..
-----
-----
Ran 2 tests in 0.000s

OK
```

This simple example barely scratches the surface of how you can use unit testing to check your software, but it does show you how a unit test is constructed and, more importantly, what it does to help you construct resilient code. Many more tests can be conducted; see the documentation at <https://docs.python.org/3/library/unittest.html>.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “Final Preparation,” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. Table 5-2 lists these key topics and the page number on which each is found.



Table 5-2 Key Topics

Key Topic Element	Description	Page Number
Section	JavaScript Object Notation (JSON)	<u>113</u>
Section	Extensible Markup Language (XML)	<u>115</u>
Section	Error Handling in Python	<u>11</u> <u>9</u>
Steps	Test-driven development	<u>121</u>
Paragraph	Unit test in Python	<u>12</u> <u>3</u>
<u>Example 5-9</u>	Output from Adding a New Test That Fails	<u>12</u> <u>5</u>
Paragraph	Fixing a test that fails	<u>12</u> <u>5</u>

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

test-driven development (TDD)

unit test

integration test

functional test

ADDITIONAL RESOURCES

Reading Data from a File in Python:

<https://developer.cisco.com/learning/lab/coding-204-reading-a-file/step/1>

Useful Python Libraries for Network

Engineers: <https://www.youtube.com/watch?v=Y4vfA11fP00>

Python unittest Module—How to Test Your Python Code?

<https://saralgyaan.com/posts/python-unittest-module-how-to-test-your-python-code/>

Chapter 6

Application Programming Interfaces (APIs)

This chapter covers the following topics:

- **Application Programming Interfaces (APIs):** This section describes what APIs are and what they are used for.
- **Representational State Transfer (REST) APIs:** This section of provides a high-level overview of the RESTful APIs and how they function as well as the benefits of using RESTful APIs.
- **RESTful API Authentication:** This section covers various aspects of the API authentication methods and the importance of API security.
- **Simple Object Access Protocol (SOAP):** This section examines SOAP and common examples of when and where this protocol is used.
- **Remote-Procedure Calls (RPCs):** This section provides a high-level overview of RPCs, why they are used, and the components involved.

Software developers use application programming interfaces (APIs) to communicate with and configure networks. APIs are used to communicate with applications and other software. They are also used to communicate with various components of a network through software. You can use APIs to configure or monitor specific components of a network, and there are multiple different types of APIs. This chapter focuses on two of the most common APIs: northbound and southbound APIs. This chapter explains the differences between these type of APIs through the lens of network automation.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly

or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 6-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 6-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Application Programming Interfaces (APIs)	1, 2
Representational State Transfer (REST) APIs	3
RESTful API Authentication	4
Simple Object Access Protocol (SOAP)	5, 6
Remote-Procedure Calls (RPCs)	7

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following is a sample use case of a southbound API?

1. Pushing network configuration changes down to devices
2. Increasing security
3. Streaming telemetry
4. Sending information to the cloud

2. What are some benefits of using asynchronous APIs? (Choose two.)

1. Not having to wait on a response to process data
2. Reduced processing time
3. Increased processing time
4. Data function reuse

3. What are the HTTP functions used for API communication? (Choose three.)

1. GET
2. SOURCE
3. PURGE
4. PATCH
5. PUT

4. True or false: RESTful API authentication can use API keys or custom tokens.

1. True
2. False

5. What does SOAP stand for?

1. Software Operations and Procedures
2. Software Operations Authentication Protocol
3. Simple Object Access Protocol
4. Simple Operations Automation Platform
5. Support Object Abstract Protocol

6. What are the main components of SOAP messages? (Choose all that apply.)

1. Envelope
2. Header
3. Destination
4. Body
5. Fault
6. Authentication
7. Source

7. Remote-procedure calls (RPCs) behave similarly to which of the following?

1. Synchronous API
2. Asynchronous API

FOUNDATION TOPICS

APPLICATION PROGRAMMING INTERFACES (APIS)

For communicating with and configuring networks, software developers commonly use application programming interfaces (APIs). APIs are mechanisms used to communicate with applications and other software. They are also used to communicate with various components of a network through software. A developer can use APIs to configure or monitor specific components of a network. Although there are multiple different types of APIs, this chapter focuses on two of the most common APIs: northbound and southbound APIs. The following sections explain the differences between these two API types through the lens of network automation, and [Figure 6-1](#) illustrates the typical basic operations of northbound and southbound APIs.

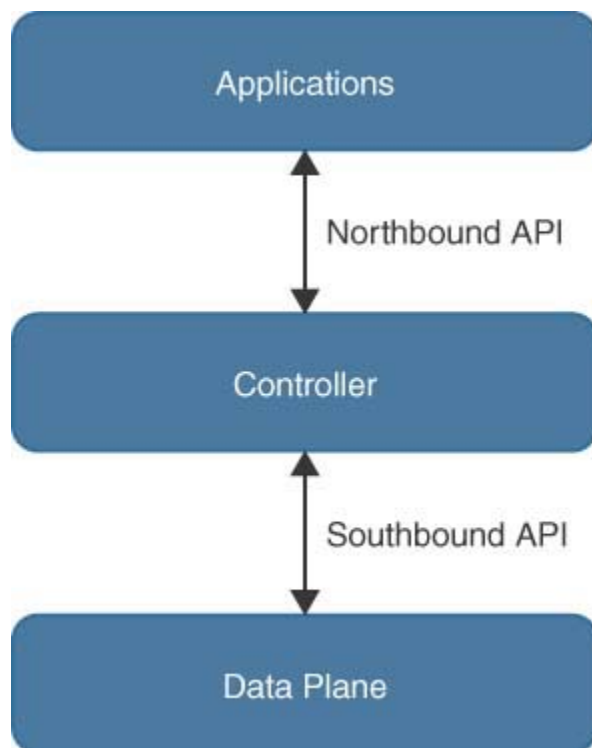


Figure 6-1 Basic API Operations

Northbound APIs

Northbound APIs are often used for communication from a network controller to its management software. For example, Cisco DNA Center has a software graphical user interface (GUI) that is used to manage its own network controller. Typically, when a network operator logs into a controller to manage the network, the information that is passed to the management software leverages a northbound REST-based API. Best practices suggest that the traffic should be encrypted using TLS between the software and the controller. Most types of APIs have the ability to use encryption to secure the data in flight.

Note

RESTful APIs are covered in an upcoming section of this chapter and in depth in [Chapter 7, “RESTful API Requests and Responses.”](#)

Southbound APIs

If a network operator makes a change to a switch’s configuration in the management software of the controller, those changes will then be pushed down to the individual devices using a southbound API. These devices can be routers, switches, or even wireless access points. APIs interact with the components of a network through the use of a programmatic interface. Southbound APIs can modify more than just the data plane on a device.



Synchronous Versus Asynchronous APIs

APIs can handle transactions either in a synchronous manner or an asynchronous manner. A **synchronous**

API causes an application to wait for a response from the API in order to continue processing data or function normally. This can lead to interruptions in application processing as delay in responses or failed responses could cause the application to hang or stop performing the way it was intended to work. This might occur, for example, if an application relies on some piece of information to be retrieved from another API before it can continue functioning. For example, uploading videos to YouTube was originally a synchronous use case. While the videos were uploading, users couldn't use the rest of the GUI or change the names of the videos or make other changes. Users had to wait until the process completed prior to doing any other work within the YouTube application. [Figure 6-2](#) provides an example of a synchronous process.

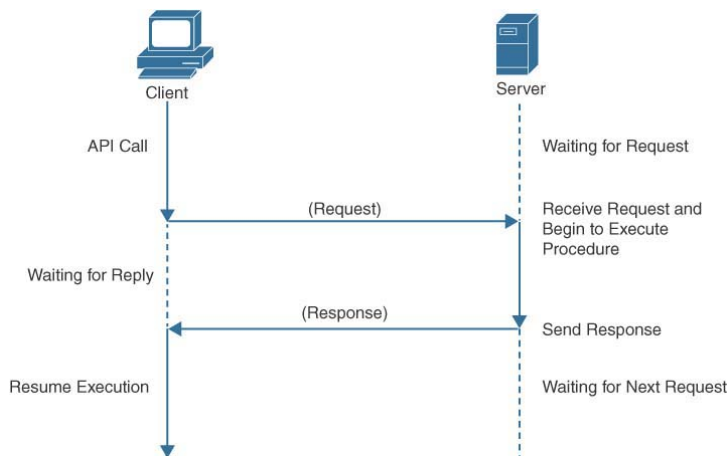


Figure 6-2 *Synchronous API Call Example*

Asynchronous APIs do exactly the opposite of synchronous APIs in that they do not wait until a response is received prior to continuing to function and process other aspects of data. Asynchronous APIs provide a callback function so that the API response can be sent back at another time, without the application having to wait for the entire transaction to complete. As an example of an asynchronous API, today you can upload a video to YouTube, and while it's uploading,

users can change the title, add hashtags, and even retrieve the URL to which the video will be posted once it is finished being uploaded.

In summary, the main difference between synchronous and asynchronous APIs is that a synchronous API waits for other aspects of the code to complete prior to moving on and processing additional code. An asynchronous API, on the other hand, provides the ability to continue processing code and provides a callback so that an application doesn't have to wait for the API call to complete before it can continue processing other API calls. An asynchronous API provides a better user experience as the users do not have to wait on certain aspects of information to be received prior to being able to use the application for other things. [Figure 6-3](#) illustrates an asynchronous API call and how the application can continue processing while waiting for the response from the initial API call.

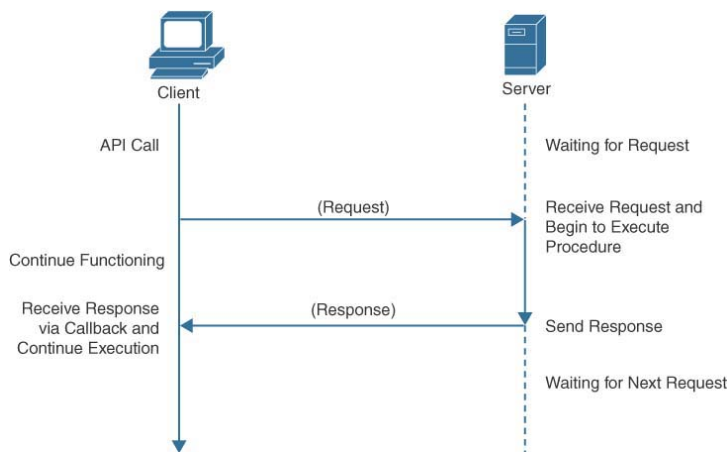


Figure 6-3 *Asynchronous API Call Example*

Representational State Transfer (REST) APIs

An API that uses REST is often referred to a RESTful API. RESTful APIs use HTTP methods to gather and manipulate data. Because there is a defined structure for how HTTP works, HTTP offers a consistent way to interact with APIs from multiple vendors. REST uses

different HTTP functions to interact with data. [Table 6-2](#) lists some of the most common HTTP functions and their associated use cases.

HTTP functions are very similar to the functions that most applications and databases use to store or alter data, whether it is stored in a database or within an application. These functions are called CRUD functions; CRUD is an acronym that stands for CREATE, READ, UPDATE, and DELETE. For example, in an SQL database, the CRUD functions are used to interact with or manipulate the data stored in the database. [Table 6-3](#) lists the CRUD functions and their associated actions and use cases.



Table 6-2 HTTP Functions and Sample Use Cases

HTTP Function	Action	Use Case
GET	Requests data from a destination	Viewing a website
POST	Submits data to a specific destination	Submitting login credentials
PUT	Replaces data at a specific destination	Updating an NTP server
PATCH	Appends data to a specific destination	Adding an NTP server
DELETE	Removes data from a specific destination	Removing an NTP server



Table 6-3 CRUD Functions and Sample Use Cases

CRUD Function	Action	Use Case
C R E A T E	Inserts data inside a database or an application	Creating a customer's home address in a database
R E A D	Retrieves data from a database or an application	Pulling up a customer's home address from a database
U P D A T E	Modifies or replaces data in a database or an application	Changing a street address stored in a database
D E L E T E	Removes data from a database or an application	Removing a customer from a database

Whether you are trying to learn how APIs interact with applications or controllers, test code and outcomes, or become a full-time developer, one of the most important pieces of interacting with any software via APIs is testing. Testing code helps ensure that developers are accomplishing the desired outcome. This chapter covers some tools and resources that make it possible to practice using APIs and REST functions and hone your

development skills in order to become a more efficient network engineer with coding skills.

Note

Chapter 7 provides more detail on HTTP and CRUD functions as well as response codes.



RESTful API Authentication

As mentioned earlier in this chapter, it is important to be able to interact with a software controller using RESTful APIs and to be able to test code to see if the desired outcomes are accomplished when executing the code. Keep in mind that APIs are software interfaces into an application or a controller. Many APIs require authentication; such APIs are just like devices in that the user needs to authenticate to gain access to utilize the APIs. Once a user has authenticated, any changes that a developer has access to make via the API are then able to impact the application. This means if a RESTful API call is used to delete data, that data will be removed from the application or controller just as if a user were logged into the device via the CLI and deleted it. It is best practice to use a test lab or a Cisco DevNet sandbox while learning or practicing API concepts to prevent accidental impacts in a production or lab environment.

Note

Cisco DevNet is covered in [Chapter 1, “Introduction to Cisco DevNet Associate Certification.”](#)

Basic Authentication

Basic authentication, illustrated in [Figure 6-4](#), is one of the simplest and most common authentication methods

used in APIs. The downfall of basic authentication is that the credentials are passed unencrypted. This means that if the transport is simple HTTP, it is possible to sniff the traffic and capture the username and password with little to no effort. The lack of encryption means that the credentials are in simple plaintext base 64 encoding in the HTTP header. However, basic authentication is more commonly used with SSL or TLS to prevent such attacks.

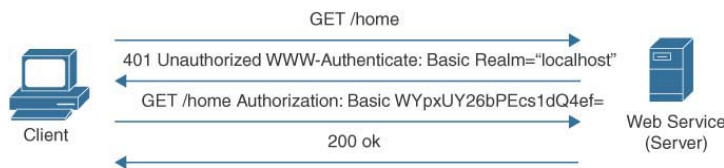


Figure 6-4 *Basic Authentication Example*

Another big issue with basic authentication is that the password is sent back and forth with each request, which increases the opportunity for an attacker to capture the traffic containing the password. This is yet another reason to use encryption on this type of transaction.

API Keys

Some APIs use API keys for authentication. An **API key** is a predetermined string that is passed from the client to the server. It is intended to be a pre-shared secret and should not be well known or easy to guess because it functions just like a password. Anyone with this key can access the API in question and can potentially cause a major outage and gain access to critical or sensitive data. An API key can be passed to the server in three different ways:

- String
- Request header
- Cookie

Example 6-1 provides an example of a string-based API key. This type of API key is sent with every API call and is often used as a one-off method of authentication. When

you're looking to do multiple API calls, it isn't convenient to manually enter the API key string every time. This is where the request header or cookie options come into play.

Example 6-1 *String-Based API Key Example*

[Click here to view code image](#)

```
GET /something?api_key=abcdef12345
```

Request headers are frequently used when a user is making multiple API calls and doesn't want to keep having to put the API key into each API individually. This approach is typically seen in Postman and Python scripts. The header must include the string or token in the header of each API call. [Example 6-2](#) shows the request header option for API key authentication.

Example 6-2 *Request Header API Key Example*

```
GET /something HTTP/1.1  
X-API-Key: abcdef12345
```

Finally, one of the most common methods for recurring API calls is to use cookies. A cookie stores the API key string and can be reused and stored over and over. This is synonymous with a header. [Example 6-3](#) shows an API key cookie that uses the same key as the previous examples.

Example 6-3 *Cookie API Key Example*

[Click here to view code image](#)

```
GET /something HTTP/1.1  
Cookie: X-API-KEY=abcdef12345
```

Note

Later chapters provide detailed examples of the authentication methods introduced in this chapter.

Custom Tokens

A custom **token** allows a user to enter his or her username and password once and receive a unique auto-generated and encrypted token. The user can then use this token to access protected pages or resources instead of having to continuously enter the login credentials. Tokens can be time bound and set to expire after a specific amount of time has passed, thus forcing users to reauthenticate by reentering their credentials. A token is designed to show proof that a user has previously authenticated. It simplifies the login process and reduces the number of times a user has to provide login credentials. A token is stored in the user's browser and gets checked each time the user tries to access information requiring authentication. Once the user logs out of the web browser or website, the token is destroyed so it cannot be compromised. [Figure 6-5](#) provides an overview of token-based authentication between a client and a server.

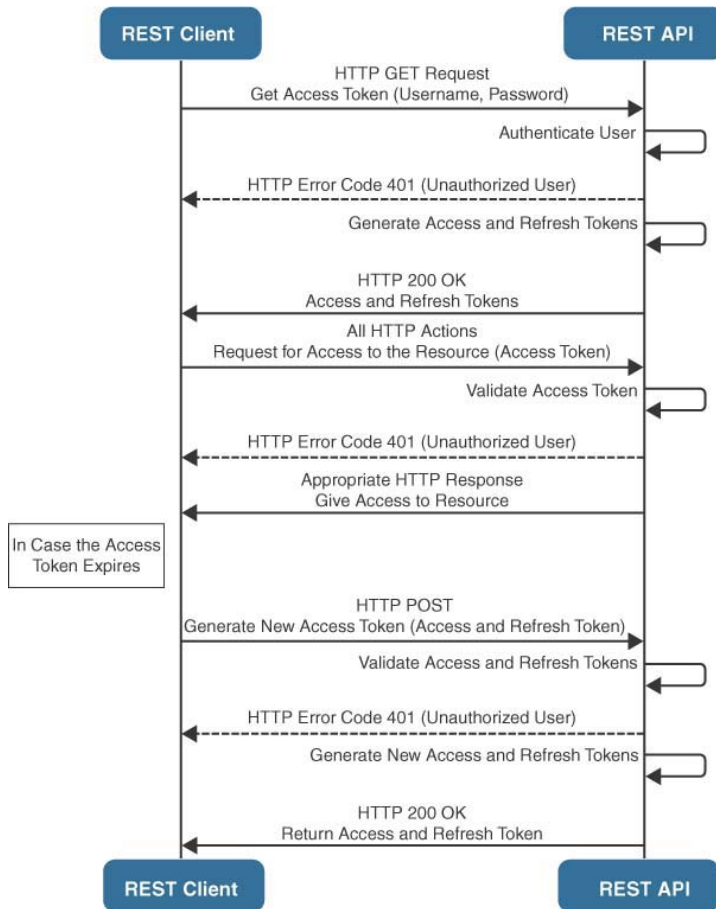


Figure 6-5 *Token-Based Authentication Example*

Simple Object Access Protocol (SOAP)



Simple Object Access Protocol (SOAP) is used to access web services. Although HTTP is the most commonly deployed transport for SOAP, SOAP can use either Simple Mail Transfer Protocol (SMTP) or HTTP. SOAP is used to exchange data between applications that were built on different programming languages, such as Java, .NET, and PHP. SOAP greatly simplifies the life of a developer, eliminating the need to know how to develop in each of these specific programming languages. It makes it possible to exchange data between these applications in a more simplified manner, without

requiring a developer to be expert in all the different languages. SOAP is based on XML. Because most programming languages today have libraries for working with XML, SOAP can act as an intermediary specification between the different applications.

SOAP uses XML to communicate between web services and clients. Because SOAP is platform and operating system independent, it can work with both Windows and Linux platforms. SOAP messages, which typically consist of the following four main components, are sent between the web applications and the clients (see [Figure 6-6](#)):

- Envelope
- Header
- Body
- Fault (optional)

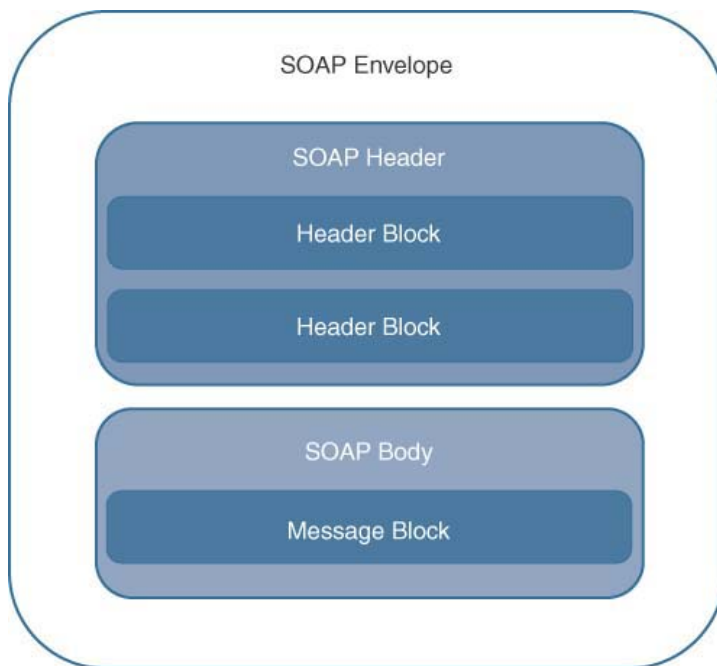


Figure 6-6 *SOAP Message Format*

The SOAP envelope encloses the XML data and identifies it as a SOAP message. The envelope indicates the beginning and the end of the SOAP message. The next portion of a SOAP message is the SOAP header, and it

can contain multiple header blocks. Header blocks are targeted to specific SOAP receiver nodes. If a SOAP message contains a header, it must come before the body element. The SOAP body contains the actual message that is designated for the SOAP receiver. Every SOAP envelope must contain at least one body element. Typically, SOAP messages are automatically generated by the web service when it's called by the client. [Figure 6-7](#) illustrates the high-level communication that occurs between a client and a server or web service.

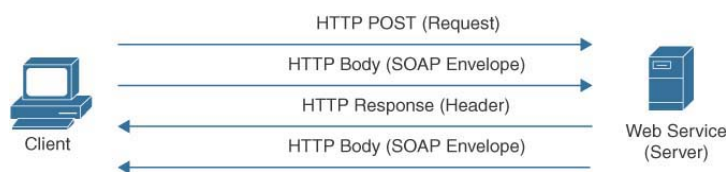


Figure 6-7 *High-Level SOAP Communication*

Another potentially beneficial aspect of SOAP is that because it primarily uses HTTP, it is efficient in passing through firewalls without requiring that any additional ports be allowed or open for the web service traffic to be permitted. This can save time and reduce some operational overhead. To reiterate, the benefit of SOAP is its capability to work between different languages while using a simple common HTTP and XML structure.

[Example 6-4](#) shows a sample SOAP message that is being used to leverage an HTTP GET to retrieve the price for Cisco's stock, using the ticker symbol CSCO.

Example 6-4 *Sample SOAP Message*

[Click here to view code image](#)

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml;
charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-
envelope"

<?xml version="1.0"?>
```

```

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
envelope" xmlns:m="http://
www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>CSCO</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>

```

The World Wide Web Consortium (W3C) recommends using the current SOAP specification, version 1.2. The previous version of SOAP is 1.1. Although it is possible for a SOAP node to support both versions of the protocol, a protocol binding must be used for the version of SOAP that the client intends to use.

As mentioned briefly at the beginning of this section, SOAP can include optional fault messages. [Table 6-4](#) lists the options that are available for the optional fault messages as well as which of them are optional.

Table 6-4 SOAP Fault Options

Fault Element	Description	Optional
faultCode	Specifies the fault code of an error	No
faultString	Describes an error	No
faultActor	Specifies who caused a fault	Yes
detail	Applies specific error messages	Yes

[Table 6-4](#) shows the options available in SOAP version 1.2. **faultString** provides a description of an error

message that is generated. This is not an optional field; rather, it is mandatory in the communications between the client and the web service. **faultActor** specifies which node caused a fault. Although this field would provide some additional information related to who caused the fault at hand, this field is optional. The **detail** element provides application-specific error messages; that is, this element is populated with information provided by the application. SOAP fault messages can contain a variety of **faultCode** options, which indicate what errors were generated as well as potentially who caused each error (the sender or receiver). [Table 6-5](#) lists the available SOAP fault codes and their associated use cases.

Table 6-5 SOAP Fault Codes

SOAP Fault Code Description	
V e r s i o n M i s m a t c h	The faulting node found an invalid element information item instead of the expected envelope element information item. The namespace, local name, or both did not match the envelope element information item required by this recommendation.
M u s t U n d e r	This is a child element of the SOAP header. If this attribute is set, any information that was not understood triggers this fault code.

s t a n d	
D a t a E n c o d i n g U n k n o w n	A SOAP header block or SOAP body child element information item targeted at the faulting SOAP node is scoped.
S e n d e r	The message was incorrectly formed or did not contain the information needed to succeed. For example, the message might have lacked the proper authentication or payment information. This code generally indicates that the message is not to be resent without change.
R e c e i v e r	The message could not be processed for reasons attributable to the processing of the message rather than to the contents of the message itself. For example, processing could include communicating with an upstream SOAP node, which did not respond. The message could succeed, however, if resent at a later point in time.

The fault message shown in [Example 6-5](#) was generated because the Detail value wasn't interpreted correctly due to the typo in the XML `<m:MaxTime>P5M</m:MaxTime>`. The value P5M caused the issue in this case because the code was expecting it to be 5PM.

The XML code and value should be `<m:MaxTime>5PM</m:MaxTime>` in this case.

Example 6-5 Sample SOAP Fault

[Click here to view code image](#)

```
<env:Envelope
xmlns:env="http://www.w3.org/2003/05/soap-
envelope"

xmlns:m="http://www.example.org/timeouts"

xmlns:xml="http://www.w3.org/XML/1998/namespace">

  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
        <env:Subcode>
          <env:Value>m:MessageTimeout</env:Value>
        </env:Subcode>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="en">Sender
Timeout</env:Text>
      </env:Reason>
      <env:Detail>
        <m:MaxTime>P5M</m:MaxTime>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Note

The examples used in this chapter are all based on SOAP version 1.2.

Remote-Procedure Calls (RPCs)

Remote-procedure calls (RPCs) make it possible to execute code or a program on a remote node in a network. RPCs behave as if the code were executed locally on the same local node, even though the code is executed on a remote address space, such as another

system on the network. Most remote and local calls are very similar in nature and can be distinguished from one another based on whether they are local or remote. RPCs are sometimes also known as function or subroutine calls. Using an RPC is a very common way of executing specific commands, such as executing GET or POST operations to a set API or URL.

When a client sends a request message, the RPC translates it and then sends it to the server. A request may be a procedure or a function call destined to a remote server. When a server receives the request, it sends back a response to the client. While this communication is happening, the client is blocked, allowing the server time to process the call. Once the call is processed and a response has been sent back to the client, the communication between the client and server is unblocked so the client can resume executing the procedure call. This can be considered a security mechanism to prevent the flooding of RPCs to brute-force the server and cause denial-of-service (DoS) attacks or exhaustion of resources. Figure 6-8 showcases the high-level RPC communications between a client and a server.

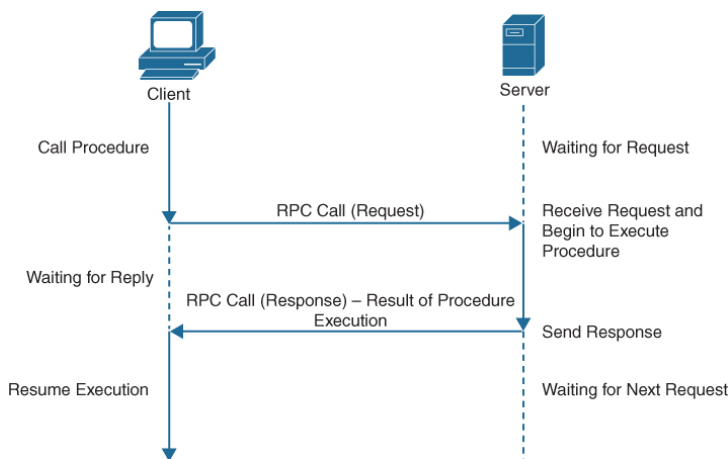


Figure 6-8 High-Level RPC Communications

As mentioned earlier in this section, an RPC call is blocked during the waiting periods. Once a procedure is executed and the response is sent from the server and received on the client, the execution of the procedure continues. (This means that RPC calls are typically synchronous. There are also asynchronous RPC calls, but the focus of this section is on synchronous RPC calls.)

Now that the high-level communications of RPC have been covered, let's look at an example of an RPC request message. There are different versions of RPC messages. However, the most common is XML-RPC; XML-RPC was also the most common version prior to SOAP becoming available. [Example 6-6](#) shows a simple RPC call with XML-RPC that uses a GET to retrieve the name of the 21st state added to the United States.

Example 6-6 *Sample XML-RPC Request Message*

[Click here to view code image](#)

```
<?xml version="1.0"?>
<methodCall>

<methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>21</i4></value>
    </param>
  </params>
</methodCall>
```

You can see in [Example 6-6](#) that the format of XML is very similar to that of SOAP, making these messages simple for humans to read and digest and also to build. [Example 6-7](#) shows an example of an XML-RPC reply or response message, in which the response to the GET message from [Example 6-6](#) is Illinois.

Example 6-7 *Sample XML-RPC Reply Message*

[Click here to view code image](#)

```

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Illinois</string>
    </value>
  </param>
</params>
</methodResponse>

```

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 6-6](#) lists these key topics and the page number on which each is found.



Table 6-6 Key Topics for Chapter 6

Key Topic Element	Description	Page Number
Section	Synchronous Versus Asynchronous APIs	131
Table 6-2	HTTP Functions and Sample Use Cases	133
Table 6-3	CRUD Functions and Sample Use Cases	133

Section	RESTful API Authentication	133
Paragraph h	SOAP structure and components	136

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Representational State Transfer (REST) APIs

synchronous API

asynchronous API

CRUD functions

API key

API token

Simple Object Access Protocol (SOAP)

remote-procedure call (RPC)

Chapter 7

RESTful API Requests and Responses

This chapter covers the following topics:

- **RESTful API Fundamentals:** This section covers the basics of RESTful APIs and details operations such as GET, POST, PUT, and DELETE. Other topics include REST headers and data formats such as XML, JSON, and YAML.
- **REST Constraints:** This section covers the six architectural constraints of REST in detail.
- **REST Tools:** This section covers sequence diagrams and tools such as Postman, curl, HTTPie, and the Python Requests library that are used to make basic REST calls.

Application programming interfaces (**APIs**) are the foundation of the new generation of software, including networking, cloud, mobile, and Internet of Things (IoT) software. APIs connect pieces of software together, “gluing” together any required information components around a system and enabling two pieces of software to communicate with each other.

REST, which stands for Representational State Transfer, refers to a particular style of API building. Most modern services and networking products today rely on REST for their APIs simply because REST is based on HTTP (which happens to be the protocol that powers nearly all Internet connections). REST is lightweight, flexible, and scalable, and its popularity has been growing.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 7-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 7-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
RESTful API Fundamentals	1–5
REST Constraints	6, 7
REST Tools	8

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. In HTTP, in order to make a successful GET request to the server, the client needs to include at least which of the following? (Choose two.)
 1. URL

2. Method
3. Headers
4. Body

2. Which of the following is not an HTTP method?

1. GET
2. HEAD
3. TRIGGER
4. PATCH

3. Webhooks are like which of the following? (Choose two.)

1. Remote procedure calls
2. Callback functions
3. State-altering functions
4. Event-triggered notifications

4. Which response code indicates that a resource has moved?

1. 201
2. 301
3. 401
4. 501

5. Which of the following model the interactions between various objects in a single use case?

1. REST APIs
2. Sequence diagrams
3. Excel sheets
4. Venn diagrams

6. Which REST API architectural constraint allows you to download code and execute it?

1. Client/server
2. Statelessness
3. Code on demand
4. Layered systems

7. Rate limiting is an essential REST API design method for developers. Rate-limiting techniques are used to _____.

1. increase security
2. have business impact
3. enhance efficiency end to end
4. do all the above

8. To add HTTP headers to a Python request, you can simply pass them in as which of the following?

1. list

2. dict
3. tuple
4. set

FOUNDATION TOPICS

RESTFUL API FUNDAMENTALS

An **application programming interface** (API) is a set of functions and procedures intended to be used as an interface for software components to communicate with each other. An API may be for a web app, an operating system, a database system, computer hardware, or any software library. A common example of an API is the Google Maps API, which lets you interface with Google Maps so that you can display maps in your application, query locations, and so on. [Figure 7-1](#) shows a simple way to visualize an API.

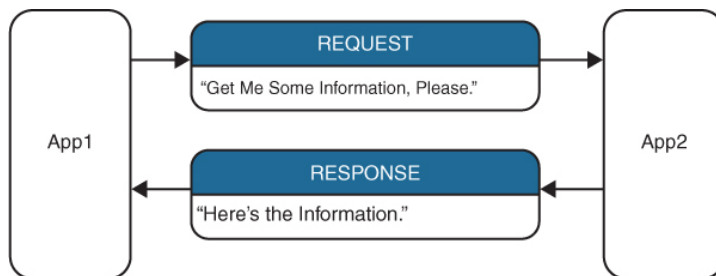


Figure 7-1 APIs are a contract between two communicating applications

The following sections look at the different API types.

API Types

APIs can be broadly classified into three categories, based on the type of work that each one provides:

- **Service API:** In a service API, an application can call on another application to solve a particular problem (see [Figure 7-2](#)). Usually these systems can exist independently. For example, in a payment system, an application can call the API to accept payments via credit cards. As

another example, with a user-management system, an application can call an API to validate and authenticate users.

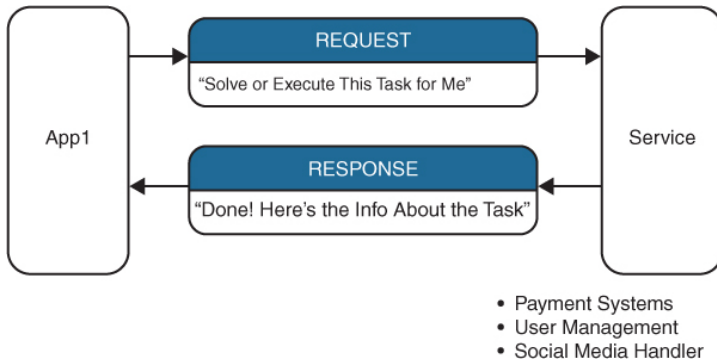


Figure 7-2 Service API Providing a Complete Service to the Calling Application

- **Information API:** An information API allows one application to ask another application for information. *Information* in this context can refer to data gathered over time, telemetry data, or a list of devices that are currently connected. [Figure 7-3](#) provides a visual representation of an information API.

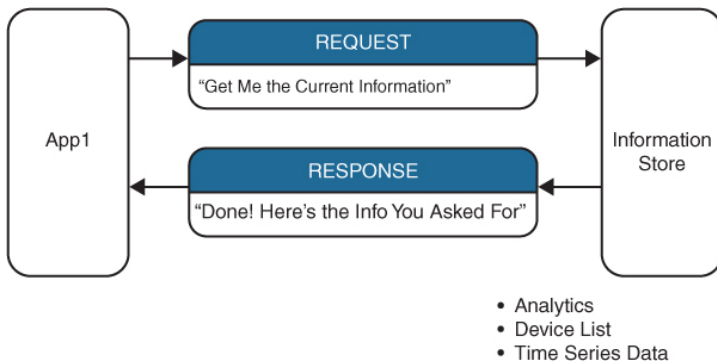


Figure 7-3 Information API Providing Information or Analysis of Data That Has Been Collected

- **Hardware API:** Application developers use hardware APIs to gain access to the features of hardware devices. Usually these APIs encompass some kind of hardware or sensors, and an application can call this kind of API to get the GPS location or real-time sensor data such as temperature or humidity. [Figure 7-4](#) provides a visual representation of what a hardware API does.



Figure 7-4 *Hardware API Providing Access to Hardware in Order to Get or Set Data*

API Access Types

There are typically three ways APIs can be accessed:

- **Private:** A private API is for internal use only. This access type gives a company the most control over its API.
- **Partner:** A partner API is shared with specific business partners. This can provide additional revenue streams without compromising quality.
- **Public:** A public API is available to everyone. This allows third parties to develop applications that interact with an API and can be a source for innovation.

Regardless of how they are accessed, APIs are designed to interact through a network. Because the most widely used communications network is the Internet, most APIs are designed based on web standards. Not all remote APIs are web APIs, but it's fair to assume that web APIs are remote.

Thanks to the ubiquity of HTTP on the web, most developers have adopted it as the protocol underlying their APIs. The greatest benefit of using HTTP is that it reduces the learning curve for developers, which encourages use of an API. HTTP has several features that are useful in building a good API, which will be apparent as we start exploring the basics in the next section.

HTTP Basics

A web browser is a classic example of an HTTP client. Communication in HTTP centers around a concept called the request/response cycle, in which the client sends the server a request to do something. The server, in turn, sends the client a response saying whether or not the server can do what the client asked. [Figure 7-5](#) provides a very simple illustration of how a client requests data from a server and how the server responds to the client.

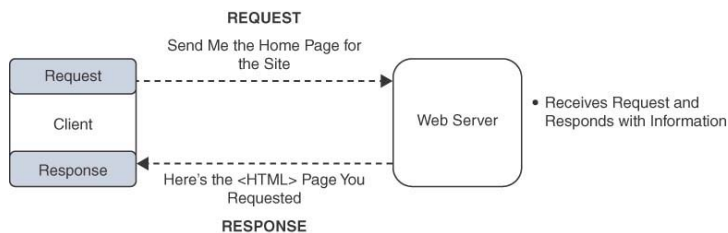


Figure 7-5 *Simple HTTP Request/Response Cycle*

Now let's look at a request from the HTTP point of view, where the client (web browser) makes a request (GET /index.html) to the server (developer.cisco.com). The server eventually responds to the client with the actual HTML page, which then gets rendered by the browser. [Figure 7-6](#) provides a very simple representation of how a client sends a GET request requesting the page from the server and how the server responds with the HTML page to the client.

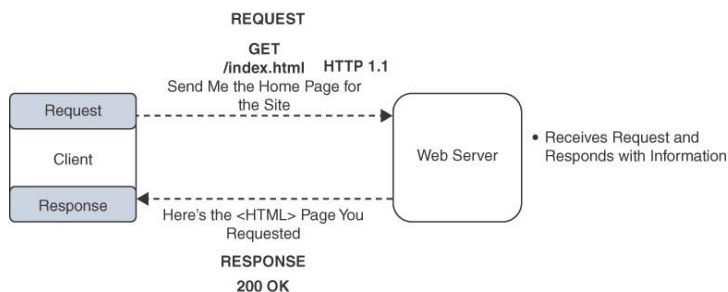


Figure 7-6 *Simple HTTP GET Request with 200 OK Response*

In HTTP, in order to make a successful request to the server, the client needs to include four items:

- URL (uniform resource locator)
- Method
- List of headers
- Body

The following sections look at each of these items in detail.

Uniform Resource Locator (URL)

An URL is similar to a house address in that it defines the location where a service resides on the Internet. A URL typically has four components, as shown in [Figure 7-7](#):



- Protocol
- Server/host address
- Resource
- Parameters

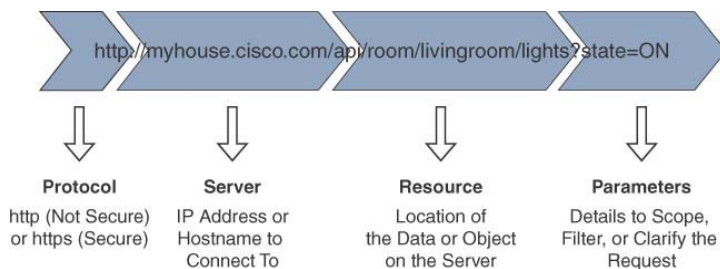


Figure 7-7 *Anatomy of an HTTP URL*

These four components are shown in the [Figure 7-7](#).

As you can see, the server or host address is the unique server name, `/api/rooms/livingroom` defines a resource to access, and `lights?state=ON` is the parameter to send in order to take some action.

Method

HTTP defines a set of request methods, outlined in [Table 7-2](#). A client can use one of these request methods to send a request message to an HTTP server.



Table 7-2 Request Methods

MethodExplanation	
GET	A client can use a GET request to get a web resource from the server.
HEAD	A client can use a HEAD request to get the header that a GET request would have obtained. Because the header contains the last-modified date of the data, it can be used to check against the local cache copy.
POST	A client can use a POST request to post data or add new data to the server.
PUT	A client can use a PUT request to ask a server to store or update data.
PATCH	A client can use a PATCH request to ask a server to partially store or update data.
DELETE	A client can use a DELETE request to ask a server to delete data.

T R A C E	A client can use a TRACE request to ask a server to return a diagnostic trace of the actions it takes.
O P T I O N S	A client can use an OPTIONS request to ask a server to return a list of the request methods it supports.
C O N N E C T	A client can use a CONNECT request to tell a proxy to make a connection to another host and simply reply with the content, without attempting to parse or cache it. This request is often used to make SSL connection through the proxy.

REST Methods and CRUD

As you have seen, REST is an architectural paradigm that allows developers to build RESTful services. These RESTful applications make use of HTTP requests for handling all four CRUD operations: CREATE, READ, UPDATE, and DELETE. These four operations are the operations most commonly used in manipulating data. The HTTP methods map in a one-to-one way to the CRUD operations, as shown in [Table 7-3](#).

Table 7-3 Mapping HTTP Methods to CRUD Operations

HTTP Method Operation Explanation		
P O S T	C R E A T E	Used to create a new object or resource. Example: Add new room to a house

GET	RENDER	Used to retrieve resource details from the system. Example: Get a list of all the rooms or all the details of one room
PUT	UPDATE	Typically used to replace or update a resource. Can be used to modify or create a resource. Example: Update details of a room
PATCH	UPDATE	Used to modify some details about a resource. Example: Change the dimensions of a room
DELETE	DELETE	Used to remove a resource from the system. Example: Delete a room from a house.

Deep Dive into GET and POST

GET is the most common HTTP request method. A client can use the GET request method to request (or “get”) a resource from an HTTP server. GET requests, which have special qualities, fetch information, and that’s it; they have no side effects, make no modifications to the system, create nothing, and destroy nothing. GET requests should, in other words, be safe and idempotent. (Idempotent means that no matter how many times you perform an action, the state of the system you’re dealing with remains the same.)

A GET request message has the following components, as shown in [Figure 7-8](#):

GET	Request URI	HTTP Version
Request Header (Optional)		
Request Body (Optional)		

Figure 7-8 *Syntax of a GET Request*

- **GET:** The keyword GET must be all uppercase.
- **Request URI:** Specifies the path of the resource requested, which must begin from the root / of the document base directory.
- **HTTP version:** Either HTTP/1.0 or HTTP/1.1. This client negotiates the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
- **Request headers (optional):** The client can use optional request headers (such as accept and accept language) to negotiate with the server and ask the server to deliver the preferred contents (such as in the language the client prefers).
- **Request body (optional):** A GET request message has an optional request body, which contains the query string (explained later in this chapter).

The POST request method is used to post additional data to the server (for example, submitting HTML form data or uploading a file). Issuing an HTTP URL from the browser always triggers a GET request. To trigger a POST request, you can use an HTML form with attribute method="post" or write your own code. For submitting HTML form data, the POST request is the same as the GET request except that the URL-encoded query string is sent in the request body rather than appended behind the request URI.

The POST request has the following components, as shown in [Figure 7-9](#):

POST	Request URI	HTTP Version
Content Type		
Content Length		
Request Header (Optional)		
Request Body (Optional)		

Figure 7-9 *Syntax of a POST Request*

- **POST:** The keyword POST must be all uppercase.
- **Request URI:** Specifies the path of the resource requested, which must begin from the root / of the document base directory.
- **HTTP version:** Either HTTP/1.0 or HTTP/1.1. This client negotiates the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
- **Request headers (optional):** The client can use optional request headers, such as content type and content length to inform the server of the media type and the length of the request body, respectively.
- **Request body (optional):** A POST request message has an optional request body, which contains the query string (explained later in this chapter).

HTTP Headers

The HTTP headers and parameters provide of a lot of information that can help you trace issues when you encounter them. HTTP headers are an essential part of an API request and response as they represent the metadata associated with the API request and response. Headers carry information for the following:

- Request and response body
- Request authorization
- Response caching
- Response cookies

In addition, HTTP headers have information about HTTP connection types, proxies, and so on. Most of these headers are for managing connections between a client, a server, and proxies.

Headers are classified as request headers and response headers. You have to set the request headers when sending a request API and have to set the assertion against the response headers to ensure that the correct headers are returned.

Request Headers

The request headers appear as name:value pairs. Multiple values, separated by commas, can be specified as follows:

[Click here to view code image](#)

```
request-header-name: request-header-value1,  
request-header-value2, ...
```

The following are some examples of request headers:

[Click here to view code image](#)

```
Host: myhouse.cisco.com  
Connection: Keep-Alive  
Accept: image/gif, image/jpeg, */*  
Accept-Language: us-en, fr, cn
```

Response Headers

The response headers also appear as name:value pairs. As with request headers, multiple values can be specified as follows:

[Click here to view code image](#)

```
response-header-name: response-header-value1,  
response-header-value2, ...
```

The following are some examples of response headers:

[Click here to view code image](#)

```
Content-Type: text/html
Content-Length: 35
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
The response message body contains the resource
data requested.
```

The following are some examples of request and response headers:

- **Authorization:** Carries credentials containing the authentication information of the client for the resource being requested.
- **WWW-Authenticate:** This is sent by the server if it needs a form of authentication before it can respond with the actual resource being requested. It is often sent along with response code 401, which means “unauthorized.”
- **Accept-Charset:** This request header tells the server which character sets are acceptable by the client.
- **Content-Type:** This header indicates the media type (text/HTML or application/JSON) of the client request sent to the server by the client, which helps process the request body correctly.
- **Cache-Control:** This header is the cache policy defined by the server. For this response, a cached response can be stored by the client and reused until the time defined in the Cache-Control header.

Response Codes

The first line of a response message (that is, the status line) contains the response status code, which the server generates to indicate the outcome of the request. Each status code is a three-digit number:

- **1xx (informational):** The request was successfully received; the server is continuing the process.
- **2xx (success):** The request was successfully received, understood, accepted, and serviced.
- **3xx (redirection):** Further action must be taken to complete the request.
- **4xx (client error):** The request cannot be understood or is unauthorized or the requested resource could not be found.
- **5xx (server error):** The server failed to fulfill a request.

Table 7-4 describes some commonly encountered status codes.



Table 7-4 HTTP Status Codes

Status Code	Meaning	Explanation
200	Continue	The server received the request and is in the process of giving the response.
200	Okay	The request is fulfilled.
301	Moved permanently	The resource requested has been permanently moved to a new location. The URL of the new location is given in the Location response header. The client should issue a new request to the new location, and the application should update all references to this new location.
302	Found and redirect (or temporarily)	This is the same as code 301, but the new location is temporary in nature. The client should issue a new request, but applications need not update the references.
304	Not modified	In response to the if-modified-since conditional GET request, the server notifies that the resource requested has not been modified.
400	Bad	The server could not interpret or understand the

00	request	request; there is probably a syntax error in the request message.
401	Authentication required	The requested resource is protected and requires the client's credentials (username and password). The client should resubmit the request with the appropriate credentials (username and password).
403	Forbidden	The server refuses to supply the resource, regardless of the identity of the client.
404	Not found	The requested resource cannot be found on the server.
405	Method not allowed	The request method used (for example, POST, PUT, DELETE) is a valid method. However, the server does not allow that method for the resource requested.
408	Request timeout	The request sent to the server took longer than the website's server was prepared to wait.
414	Request URI too large	The URI requested by the client is longer than the server is willing to interpret.
500	Internal server error	The server is confused; this may be caused by an error in the server-side program responding to the request.
501	Method not	The request method used is invalid; this could be caused by a typing error, such as Get in place of GET.

	imple- m- ente- d	
502	Bad gateway	The proxy or gateway indicates that it received a bad response from the upstream server.
503	Service unavailable	The server cannot respond due to overloading or maintenance. The client can try again later.
504	Gateway timeout	The proxy or gateway indicates that it received a timeout from an upstream server.



Now that we have looked at HTTP methods and return codes, let's look at the data that is sent or received during a GET or POST. Let's look at an example and see how the same information is represented in the various data types. For this example, refer to [Figure 7-7](#). In this example, you are making a GET request to the server at myhouse.com to change the state of lights in the living room to ON.

The data sent and received in a RESTful connection requires structured data formatting. For the house example, you now see a response from the server that includes information about the house. Standard data formats include XML, JSON, and YAML, which are described in the following sections.

XML

Extensible Markup Language (XML) is a markup language that encodes information between descriptive tags. XML is a superset of Hypertext Markup Language (HTML), which was originally designed to describe the formatting of web pages served by servers through HTTP. The encoded information is defined within user-defined schemas that enable any data to be transmitted between systems. An entire XML document is stored as text, and it is both machine readable and human readable.

Example 7-1 shows a sample XML response document. As you can see, with XML, you can assign some meaning to the tags in the document. You can extract the various attributes from the response by simply locating the content surrounded by `<study_room>` and `</study_room>`; this content is technically known as the `<study_room>` *element*.

Example 7-1 XML Data Format

[Click here to view code image](#)

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <home>this is my house</home>
  <home>located in San Jose, CA</home>
  <rooms>
    <living_room>true</living_room>
    <kitchen>false</kitchen>
    <study_room>
      <size>20x30</size>
    </study_room>
    <study_room>
      <desk>true</desk>
    </study_room>
    <study_room>
      <lights>On</lights>
    </study_room>
  </rooms>
</root>
```

JSON

JSON, short for **JavaScript Object Notation**, is pronounced like the name “Jason.” The JSON format is derived from JavaScript object syntax, but it is entirely text based. It is a key: value data format that is typically rendered in curly braces {} and square brackets []. JSON is readable and lightweight, and it is easy for humans to understand.

A key/value pair has a colon (:) that separates the key from the value, and each such pair is separated by a comma in the document or the response.

JSON **keys** are valid strings. The value of a key is one of the following data types:

- String
- Number
- Object
- Array
- Boolean (true or false)
- Null

Example 7-2 shows a sample JSON response document, and you can see the full response. If you are interested in seeing the status of the lights in the study_room, then you look at all the values that are present and follow the various key/value pairs (such as “lights”: “On”) and extract the various values from the response by locating the correct keys and corresponding values.

Example 7-2 *JSON Data Format*

```
{
  "home": [
    "this is my house",
    "located in San Jose, CA"
  ],
  "rooms": {
    "living_room": "true",
    "kitchen": "false",
    "study_room": [
      {
        "size": "20x30"
      }
    ]
  }
}
```



```
    },
    {
      "desk": true
    },
    {
      "lights": "On"
    }
  ]
}}
```

YAML

YAML is an acronym that stands for “YAML Ain’t Markup Language.” According to the official YAML site (<https://yaml.org>), “YAML is a human-friendly data serialization standard for all programming languages.”

YAML is a data serialization language designed for human interaction. It’s a strict superset of JSON, another data serialization language. But because it’s a strict superset, it can do everything that JSON can do and more. One significant difference is that newlines and indentation mean something in YAML, whereas JSON uses brackets and braces to convey similar ideas. YAML uses three main data formats:

- **Scalars:** The simplest is a *keyvalue* view.
- **Lists/sequences:** Data can be ordered by indexes.
- **Dictionary mappings:** These are similar to scalars but can contain nested data, including other data types.

Example 7-3 shows a sample YAML response document. As you can see, the response is very straightforward and human readable. If you are interested in seeing the status of the lights in `study_room`, you find the `study_room` section and then look for the value of `lights`.

Example 7-3 *YAML Data Format*

```
---
home:
- this is my house
- located in San Jose, CA
```

```
rooms:
  living_room: 'true'
  kitchen: 'false'
  study_room:
    - size: 20x30
    - desk: true
    - lights: 'On'
```

Webhooks

Webhooks are user-defined HTTP callbacks. A webhook is triggered by an event, such as pushing code to a repository or typing a keyword in a chat window. An application implementing webhooks sends a POST message to a URL when a specific event happens. Webhooks are also referred to as *reverse APIs*, but perhaps more accurately, a webhook lets you skip the request step in the request/response cycle. No request is required for a webhook, and a webhook sends data when triggered.

For security reasons, the REST service may perform some validation to determine whether the receiver is valid. A simple validation handshake performs validation, but this is just one way of validating.

The validation token is a unique token specified by the server. Validation tokens can be generated or revoked on the server side through the configuration UI. When the server sends data to a webhook URL, it includes a validation token in the request HTTP header. The webhook URL should consist of the same validation token value in the HTTP response header. In this way, the server knows that it is sending to a validated endpoint and not a rogue endpoint. [Figure 7-10](#) illustrates the flow of webhooks.

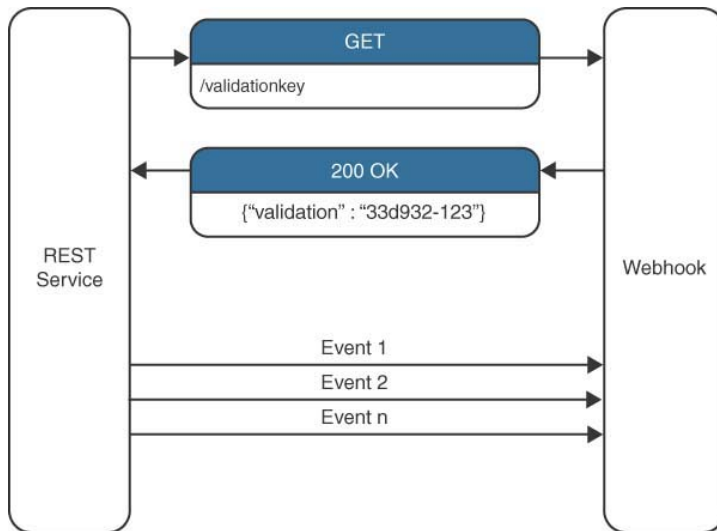


Figure 7-10 *Webhook Validation and Event Flow*

Tools Used When Developing with Webhooks

You will face a particular difficulty when developing an application that consumes webhooks. When using a public service that provides webhooks, you need a publicly accessible URL to configure the webhook service. Typically, you develop on localhost, and the rest of the world has no access to your application, so how do you test your webhooks? ngrok (<http://ngrok.com>) is a free tool that allows you to tunnel from a public URL to your application running locally.

Sequence Diagrams

Now that you understand the fundamentals of REST API (request, response, and webhooks), authentication, data exchange, and constraints that go with rest APIs, it's time to introduce sequence diagrams. A sequence diagram models the interactions between various objects in a single use case. It illustrates how the different parts of a system interact with each other to carry out a function and the order in which the interactions occur when a particular use case is executed. In simpler terms, a sequence diagram shows how different parts of a system work in a sequence to get something done.

Figure 7-11 is a sequence diagram for the example we've been looking at, where a user wants to get list of all rooms in the house. For this example, assume that there is a web application with a user interface that renders the list of all the rooms and the various attributes of the rooms.

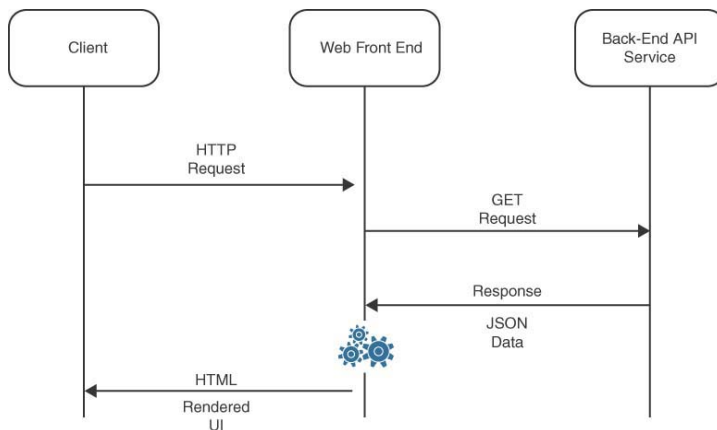


Figure 7-11 Sequence Diagram Showing End-to-End Flow

The sequence of events that occur is as follows:

1. The client browser points to <http://myhouse.cisco.com/> (the HTTP GET request sent), which is the web application.
2. The server sends out a REST API request to get all the rooms to the back-end service (/API/getallrooms) to get all the details of the house.
3. The back-end API service returns data in JSON format.
4. The web application processes the JSON and renders the data in the user interface.
5. The client sees the data.



REST CONSTRAINTS

REST defines six architectural constraints that make any web service a truly RESTful API. These are constraints also known as Fielding's constraints (see

<https://www.ics.uci.edu/~fielding/pubs/dissertation/toip.htm>). They generalize the web's architectural principles and represent them as a framework of constraints or an architectural style. These are the REST constraints:

- Client/server
- Stateless
- Cache
- Uniform interface
- Layered system
- Code on demand

The following sections discuss these constraints in some detail.

Client/Server

The client and server exist independently. They must have no dependency of any sort on each other. The only information needed is for the client to know the resource URIs on the server. The interaction between them is only in the form of requests initiated by the client and responses that the server sends to the client in response to requests. The client/server constraint encourages separation of concerns between the client and the server and allows them to evolve independently.

Stateless

REST services have to be stateless. Each individual request contains all the information the server needs to perform the request and return a response, regardless of other requests made by the same API user. The server should not need any additional information from previous requests to fulfill the current request. The URI identifies the resource, and the body contains the state of the resource. A stateless service is easy to scale horizontally, allowing additional servers to be added or removed as necessary without worry about routing

subsequent requests to the same server. The servers can be further load balanced as necessary.

Cache

With REST services, response data must be implicitly or explicitly labeled as cacheable or non-cacheable. The service indicates the duration for which the response is valid. Caching helps improve performance on the client side and scalability on the server side. If the client has access to a valid cached response for a given request, it avoids repeating the same request. Instead, it uses its cached copy. This helps alleviate some of the server's work and thus contributes to scalability and performance.

Note

GET requests should be cacheable by default. Usually browsers treat all GET requests as cacheable.

POST requests are not cacheable by default but can be made cacheable by adding either an Expires header or a Cache-Control header to the response.

PUT and DELETE are not cacheable at all.

Uniform Interface

The uniform interface is a contract for communication between a client and a server. It is achieved through four subconstraints:

- **Identification of resources:** As we saw earlier in the chapter, resources are uniquely identified by URIs. These identifiers are stable and do not change across interactions, even when the resource state changes.
- **Manipulation of resources through representations:** A client manipulates resources by sending new representations of the resource to the service. The server controls the resource representation and can accept or reject the new resource representation sent by the client.
- **Self-descriptive messages:** REST request and response messages contain all information needed for the service and the client to interpret the message and handle it appropriately. The messages are quite

verbose and include the method, the protocol used, and the content type. This enables each message to be independent.

- **Hypermedia as the Engine of Application State (HATEOS):** Hypermedia connects resources to each other and describes their capabilities in machine-readable ways. Hypermedia refers to the hyperlinks, or simply links, that the server can include in the response. Hypermedia is a way for a server to tell a client what HTTP requests the client might want to make in the future.

Layered System

A layered system further builds on the concept of client/server architecture. A layered system indicates that there can be more components than just the client and the server, and each system can have additional layers in it. These layers should be easy to add, remove, or change. Proxies, load balancers, and so on are examples of additional layers.

Code on Demand

Code on demand is an optional constraint that gives the client flexibility by allowing it to download code. The client can request code from the server, and then the response from the server will contain some code, usually in the form of a script, when the response is in HTML format. The client can then execute that code.

REST API Versioning

Versioning is a crucial part of API design. It gives developers the ability to improve an API without breaking the client's applications when new updates are rolled out. Four strategies are commonly employed with API versioning:

- **URI path versioning:** In this strategy, the version number of the API is included in the URL path.
- **Query parameter versioning:** In this strategy, the version number is sent as a query parameter in the URL.
- **Custom headers:** REST APIs are versioned by providing custom headers with the version number included as an attribute. The main difference between this approach and the two previous ones is that it doesn't clutter the URI with versioning information.

- **Content negotiation:** This strategy allows you to version a single resource representation instead of versioning an entire API, which means it gives you more granular control over versioning. Another advantage of this approach is that it doesn't require you to implement URI routing rules, which are introduced by versioning through the URI path.

Pagination



When a request is made to get a list, it is almost never a good idea to return all resources at once. This is where a pagination mechanism comes into play. There are two popular approaches to pagination:

- Offset-based pagination
- Keyset-based pagination, also known as continuation token or cursor pagination (recommended)

A really simple approach to offset-based pagination is to use the parameters **offset** and **limit**, which are well known from databases.

Example 7-4 shows how query parameters are passed in the URI in order to get data based on offset and to limit the number of results returned.

Example 7-4 *Pagination: Offset and Limit*

[Click here to view code image](#)

```
# returns the devices between 100-115
```

Usually if the parameters are not specified, the default values are used. Never return all resources. One rule of thumb is to model the limit based on the design of your store retrieval performance.

Example 7-5 shows a URI where no parameters are passed, which results in the default number of results.

Example 7-5 *Pagination: No Parameters Yields the Default*

[Click here to view code image](#)

```
# returns the devices 0 to 200
```

Note that the data returned by the service usually has links to the next and the previous pages, as shown in [Example 7-6](#).

Example 7-6 *Pagination Response Containing Links*

[Click here to view code image](#)

```
GET /devices?offset=100&limit=10
{
  "pagination": {
    "offset": 100,
    "limit": 10,
    "total": 220,
  },
  "device": [
    //...
  ],
  "links": {
    "next": "http://myhouse.cisco.com/devices?
offset=110&limit=10",
    "prev": "http://myhouse.cisco.com/devices?
offset=90&limit=10"
  }
}
```

Rate Limiting and Monetization

Rate limiting is an essential REST API design method for developers. Rate-limiting techniques are used to increase security, business impact, and efficiency across the board or end to end. Let's look at how rate limiting helps with each of them:

- **Security:** Allowing unlimited access to your API is essentially like handing over the master key to a house and all the rooms therein. While it's great when people want to use your API and find it useful, open access can decrease value and limit business success. Rate

limiting is a critical component of an API's scalability. Processing limits are typically measured in transactions per second (TPS). If a user sends too many requests, API rate limiting can throttle client connections instead of disconnecting them immediately. Throttling enables clients to keep using your services while still protecting your API. Finally, keep in mind that there is always a risk of API requests timing out, and the open connections also increase the risk of DDoS attacks. (DDoS stands for distributed denial of service. A DDoS attack consists of a website being flooded by requests during a short period of time, with the aim of overwhelming the site and causing it to crash.)

- **Business impact:** One approach to API rate limiting is to offer a free tier and a premium tier, with different limits for each tier. Limits could be in terms of sessions or in terms of number of APIs per day or per month. There are many factors to consider when deciding what to charge for premium API access. API providers need to consider the following when setting up API rate limits:
 - Are requests throttled when they exceed the limit?
 - Do new calls and requests incur additional fees?
 - Do new calls and requests receive a particular error code and, if so, which one?
- **Efficiency:** Unregulated API requests usually and eventually lead to slow page load times for websites. Not only does this leave customers with an unfavorable opinion but it can lower your service rankings.

Rate Limiting on the Client Side

As discussed in the previous section, various rate-limiting factors can be deployed on the server side. As a good programming practice, if you are writing client-side code, you should consider the following:

- Avoid constant polling by using webhooks to trigger updates.
- Cache your own data when you need to store specialized values or rapidly review very large data sets.
- Query with special filters to avoid re-polling unmodified data.
- Download data during off-peak hours.



REST TOOLS

Understanding and testing REST API architecture when engaging in software development is crucial for any development process. The following sections explore a

few of the most commonly used tools in REST API testing and how to use some of their most important features. Based on this information, you will get a better idea of how to determine which one suits a particular development process the best.

Postman

One of the most intuitive and popular HTTP clients is a tool called Postman

(<https://www.getpostman.com/downloads/>). It has a very simple user interface and is very easy to use, even if you're just starting out with RESTful APIs. It can handle the following:

- Sending simple GETs and POSTs
- Creating and executing collections (to group together requests and run those requests in a predetermined sequence)
- Writing tests (scripting requests with the use of dynamic variables, passing data between requests, and so on)
- Chaining, which allows you to use the output of response as an input to another request
- Generating simple code samples in multiple programming languages
- Importing and executing collections created by the community

Now, let's take a look at several Postman examples.

Figure 7-12 shows the user interface of Postman calling an HTTP GET to the Postman Echo server, and **Figure 7-13** shows how easy it is to send a POST request using the same interface.

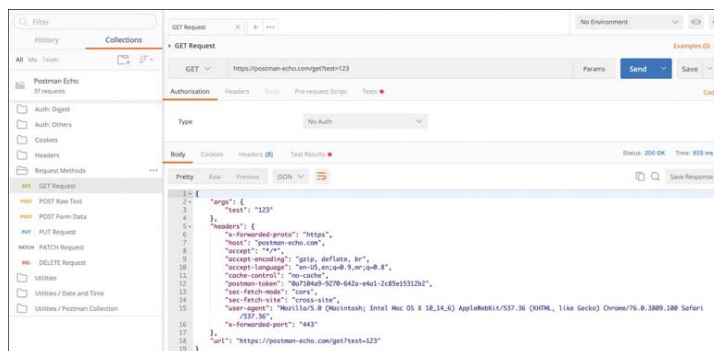


Figure 7-12 Postman: HTTP GET from the Postman Echo Server

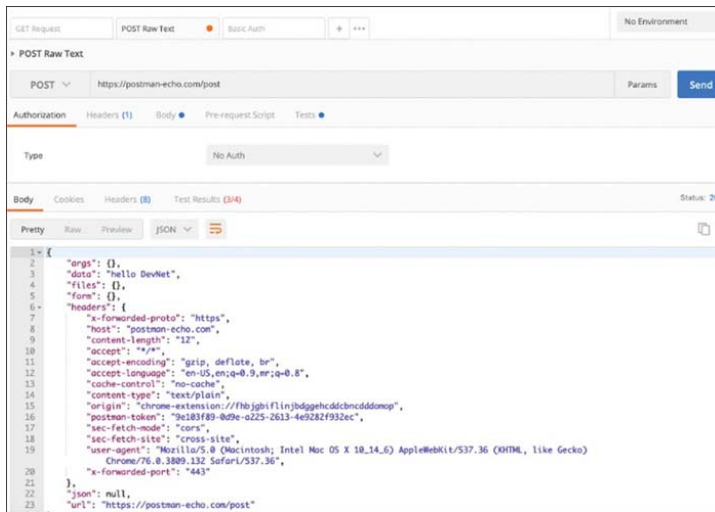


Figure 7-13 Postman: HTTP POST to the Postman Echo Server

Figure 7-14 illustrates collections. A collection lets you group individual requests together. You can then organize these requests into folders. Figure 7-14 shows the user interface of Postman, with a default collection that interacts with the Postman Echo Server. Using this interface is a very good way to learn about various options for sending or getting REST-based information.

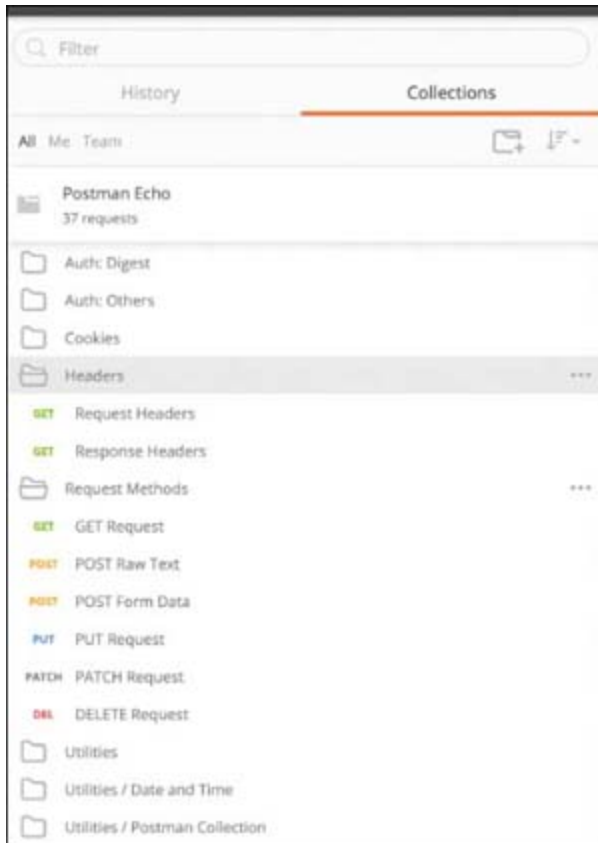


Figure 7-14 *Postman Collection*

It is possible to generate code for any REST API call that you try in Postman. After a GET or POST call is made, you can use the Generate Code option and choose the language you prefer. [Figure 7-15](#) shows an example of generating Python code for a simple GET request.

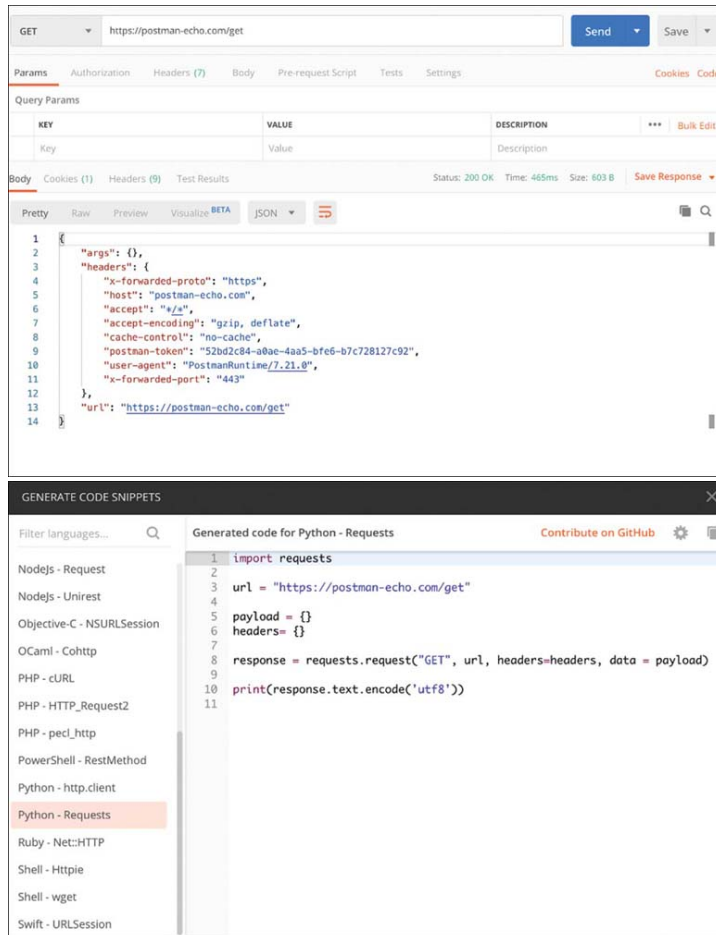


Figure 7-15 *Postman Automatic Code Generation*

Postman also has other helpful features, such as environments. An environment is a key/value pair. The key represents the name of the variable, which allows you to customize requests; by using variables, you can easily switch between different setups without changing your requests.

Finally, Postman stores a history of past calls so you can quickly reissue a call. Postman even includes some nice touches such as autocompletion for standard HTTP headers and support for rendering a variety of payloads, including JSON, HTML, and even multipart payloads.

You can find Postman examples at <https://learning.postman.com/docs/postman/launching>

[-postman/introduction/](#).

curl

curl is an extensive command-line tool that can be downloaded from <https://curl.haxx.se>. curl can be used on just about any platform on any hardware that exists today. Regardless of what you are running and where, the most basic curl commands just work.

With curl, you commonly use a couple of different command-line options:

- **-d:** This option allows you to pass data to the remote server. You can either embed the data in the command or pass the data using a file.
- **-H:** This option allows you to add an HTTP header to the request.
- **-insecure:** This option tells curl to ignore HTTPS certificate validation.
- **-c:** This option stores data received by the server. You can reuse this data in subsequent commands with the `-b` option.
- **-b:** This option allows you to pass cookie data.
- **-X:** This option allows you to specify the HTTP method, which normally defaults to GET.

Now let's take a look at some examples of how to use curl. [Example 7-7](#) shows how to use curl to call a simple GET request.

Example 7-7 *Sample HTTP GET Using curl*

[Click here to view code image](#)

```
$ curl -sD - https://postman-echo.com/get?
test=123
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 04:59:34 GMT
ETag: W/"ca-42Kz98xXW2nwFREN74xZNS6JeJk"
Server: nginx
set-cookie:
sails.sid=s%3AxZUPHE30jk1yts3qrUFqTj_MzBQZZR5n.NrjPkNm0Wp1J7%2F%2BX907VU

TFpKHpJySLzBytRbnlzYCw; Path=/; HttpOnly
Vary: Accept-Encoding
Content-Length: 202
Connection: keep-alive
```

```
{"args":{"test":"123"},"headers":{"x-forwarded-  
proto":"https","host":"postman-  
echo.com","accept":"*/*","user-  
agent":"curl/7.54.0","x-forward-  
ed-port":"443"},"url":"https://postman-  
echo.com/get?test=123"}
```

Example 7-8 shows how to use curl to call a simple POST request.

Example 7-8 *Sample HTTP POST Using curl*

[Click here to view code image](#)

```
$ curl -sD - -X POST https://postman-  
echo.com/post -H 'cache-control: no-cache'  
-H 'content-type: text/plain' -d 'hello  
DevNet'  
HTTP/1.1 200 OK  
Content-Type: application/json; charset=utf-8  
Date: Tue, 27 Aug 2019 05:16:58 GMT  
ETag: W/"13a-01MLfkxl7vDVWfb06pyVxdZlaug"  
Server: nginx  
set-cookie:  
sails.sid=s%3AwiFXmSNJpzY00NduxUCAE8IodwNg9Z2Y.j%2Bj5%2B0mch8XEq8j01vzH8  
  
kjNBi8ecJij1rGT8D1nBhE; Path=/; HttpOnly  
Vary: Accept-Encoding  
Content-Length: 314  
Connection: keep-alive  
  
{"args":{},"data":"hello DevNet","files":  
{}, "form":{},"headers":{"x-forwarded-  
proto":"https","host":"postman-  
echo.com","content-  
length":"12","accept":"*/*","ca  
che-control":"no-cache","content-  
type":"text/plain","user-  
agent":"curl/7.54.0","x-  
forwarded-  
port":"443"},"json":null,"url":"https://postman-  
echo.com/post"}
```

Example 7-9 shows how to use curl to call a simple GET request with Basic Auth sent via the header.

Example 7-9 Basic Auth Using curl

[Click here to view code image](#)

```
$ curl -sD - -X GET https://postman-echo.com/basic-auth -H 'authorization: Basic cG9zdG1hbJpwYXNzd29yZA==' -H 'cache-control: no-cache'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 05:21:00 GMT
ETag: W/"16-sJz8uwjdDv0wvm7//BYdNw8vMbU"
Server: nginx
set-cookie: sails.sid=s%3A4i3UW5-DQCMpey8Z1Ayrqq0izt4KZR5-.Bl8QDnt44B690E8J06qyC-s8oyCLpUfEsFxLEFTSWSC4; Path=/; HttpOnly
Vary: Accept-Encoding
Content-Length: 22
Connection: keep-alive
{"authenticated":true}
```

HTTPIe

HTTPIe is a modern, user-friendly, and cross-platform command-line HTTP client written in Python. It is designed to make CLI interaction with web services easy and user friendly. Its simple HTTP commands enable users to send HTTP requests using intuitive syntax. HTTPIe is used primarily for testing, trouble-free debugging, and interacting with HTTP servers, web services, and RESTful APIs. For further information on HTTPIe documentation, downloading, and installation, see <https://httpie.org/doc>:

- HTTPIe comes with an intuitive UI and supports JSON.
- It uses expressive and intuitive command syntax.
- HTTPIe allows for syntax highlighting, formatting, and colored terminal output.
- HTTPIe allows you to use HTTPS, proxies, and authentication.
- It provides support for forms and file uploads.
- It provides support for arbitrary request data and headers.
- It enables Wget-like downloads and extensions.

Now let's take a look at some examples of using HTTPie. [Example 7-10](#) shows how to use HTTPie to call a simple GET request.

Example 7-10 *Sample HTTP GET Using HTTPie*

[Click here to view code image](#)

```
$ http https://postman-echo.com/get?test=123
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 179
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 05:27:17 GMT
ETag: W/"ed-mB0Pm0M3ExozL3fgwq7U1H9aozQ"
Server: nginx
Vary: Accept-Encoding
set-cookie:
sails.sid=s%3AYCeNAWJG7Kap5wvKpg8HY1ZI5SHZoqEf.r7Gi96fe5g7%2FSp0jaJk%2Fa

VRpHZp30j5tDxiM8TPZ%2Bpc; Path=/; HttpOnly

{
  "args": {
    "test": "123"
  },
  "headers": {
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "host": "postman-echo.com",
    "user-agent": "HTTPie/1.0.2",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "url": "https://postman-echo.com/get?
test=123"
}
```

Python Requests

Requests is a Python module that you can use to send HTTP requests. It is an easy-to-use library with a lot of possibilities ranging from passing parameters in URLs to sending custom headers and SSL verification. The Requests library is a very handy tool you can use whenever you programmatically start using any APIs.

Here you will see how to use this library to send simple HTTP requests in Python as way to illustrate its ease of use.

You can use Requests with Python versions 2.7 and 3.x. Requests is an external module, so it needs to be installed before you can use it. [Example 7-11](#) shows the command you use to install the Requests package for Python.

Example 7-11 *Installing the Requests Package for Python*

```
$ pip3 install requests
```

To add HTTP headers to a request, you can simply pass them in a Python dict to the headers parameter. Similarly, you can send your own cookies to a server by using a dict passed to the cookies parameter. [Example 7-12](#) shows a simple Python script that uses the Requests library and does a GET request to the Postman Echo server.

Example 7-12 *Simple HTTP GET Using Python Requests*

[Click here to view code image](#)

```
import requests
url = "https://postman-echo.com/get"
querystring = {"test":"123"}
headers = {}
response = requests.request("GET", url,
headers=headers, params=querystring)
print(response.text)
```

[Example 7-13](#) shows a simple Python script that uses the Requests library and does a POST request to the Postman Echo server. Notice that the headers field is populated with the content type and a new field call

payload that sends some random text to the service. The response to the request is stored in a response object called response. Everything in the response can be parsed, and further actions can be taken. This example simply prints the values of a few attributes of the response.

Example 7-13 *Simple HTTP POST Using Python Requests*

[Click here to view code image](#)

```
import requests
url = "https://postman-echo.com/post"
payload = "hello DevNet"
headers = {'content-type': "text/plain"}
response = requests.request("POST", url,
data=payload, headers=headers)
print(response.text)
```

[Example 7-14](#) shows a simple Python script that uses the Requests library and does a GET request to the Postman Echo server. One difference you will notice between [Example 7-11](#) and [Example 7-14](#) is related to authentication. With the Requests library, authentication is usually done by passing the ‘authorization’ keyword along with the type and key.

Example 7-14 *Basic Auth Using Python Requests*

[Click here to view code image](#)

```
import requests
url = "https://postman-echo.com/basic-auth"
headers = {
    'authorization': "Basic
cG9zdG1hbJpwYXNzd29yZA=="
}
response = requests.request("GET", url,
headers=headers)
print(response.text)
```

REST API Debugging Tools for Developing APIs

As you start playing with RESTful APIs, you are bound to encounter errors. You can use several techniques to determine the nature of a problem. As you saw in [Table 7-3](#), RESTful APIs use several mechanisms to indicate the results of REST calls and errors that occur during processing. You can use these methods to start your debugging journey for a RESTful application. Usually the error code returned is the biggest hint you can receive. Once you have this information, you can use tools like Postman and curl to make simple API calls and see the sent and response headers. In addition, other tools that are built in to web browsers can allow you to see traces and do other types of debugging. Most browsers include some type of developer tools, such as Safari's Web Development Tools, Chrome's DevTools, and Firefox's Developer Tools. Such tools are included with browsers by default and enable you to inspect API calls quickly. Finally, if you plan on building your own test environment or sandbox, you might want to use tools like Simple JSON Server (an open-source server that you can clone and run in your environment for playing with and learning about RESTful APIs).

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions in the Pearson Test Prep Software Online.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 7-5](#) lists these key topics and the page number on which each is found.



Table 7-5 Key Topics

Key Topic Element	Description	Page
List	The elements of a URL	<u>14</u> 9
Table 7-2	Request Methods	<u>15</u> 0
Table 7-4	HTTP Status Codes	<u>15</u> 4
Paragraphs	Data formats and XML, YAML, and JSON data	<u>15</u> 5
Section	REST Constraints	<u>16</u> 0
Paragraph	Pagination	<u>16</u> 2
Section	REST Tools	<u>16</u> 4

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

API

REST

CRUD

YAML

JSON

webhook

Chapter 8

Cisco Enterprise Networking Management Platforms and APIs

This chapter covers the following topics:

- **What Is an SDK?:** This section covers what an SDK is and what it is used for.
- **Cisco Meraki:** This section covers the Cisco Meraki platform and the REST APIs it exposes.
- **Cisco DNA Center:** This section covers Cisco DNA Center and the REST APIs that it publicly exposes.
- **Cisco SD-WAN:** This section covers Cisco SD-WAN and the REST APIs exposed through Cisco vManage.

In Chapter 7, “RESTful API Requests and Responses,” you learned about REST API concepts. This chapter begins exploring software development kits (SDKs) as well as Cisco enterprise networking products, their APIs, and the public SDKs that come with them. In particular, this chapter explores the Cisco Meraki, Cisco DNA Center, and Cisco SD-WAN platforms and the REST APIs they expose. This chapter provides a short introduction to each of these solutions and shows authentication and authorization API calls for each platform. This chapter also covers basic API calls, such as for obtaining a list of devices and client health status. API tools such as **curl** and **Postman** are used throughout the chapter. Python SDKs and scripts are also explored as an introduction to network programmability and automation.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 8-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 8-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
What Is an SDK?	1–2
Cisco Meraki	3–5
Cisco DNA Center	6–8
Cisco SD-WAN	9–10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. What are some of the features of a good SDK?
(Choose three.)

1. Is easy to use
2. Is well documented
3. Integrates well with other SDKs
4. Impacts hardware resources

2. What are the advantages of using an SDK? (Choose two.)

1. Quicker integration
2. Faster development
3. Advanced customization
4. Error handling

3. What APIs does the Cisco Meraki platform provide to developers? (Choose two.)

1. Captive Portal API
2. Scanning API
3. Access Point API
4. Infrastructure API

4. What is the name of the Cisco Meraki Dashboard API authentication header?

1. X-Cisco-Meraki-API-Key
2. X-Cisco-Meraki-Token
3. X-Cisco-Meraki-Session-key
4. Bearer

5. What is the base URL for the Cisco Meraki Dashboard API?

1. <https://api.meraki.com/api/v0>
2. <https://api.meraki.com/v1/api>
3. <https://api.meraki.cisco.com/api/v0>
4. <https://api.meraki.cisco.com/v1/api>

6. What type of authentication do the Cisco DNA Center platform APIs use?

1. No auth
2. API key
3. Basic auth
4. Hash-based message authentication

7. When specifying the **timestamp** parameter with the Cisco DNA Center APIs, what format should the time be in?

1. UNIX time
2. OS/2
3. OpenVMS
4. SYSTEMTIME

8. What is the output of the multivendor SDK for Cisco DNA Center platform?

1. Device driver
2. Device package
3. Software driver
4. Software package

9. Which component of the Cisco SD-WAN fabric exposes a public REST API interface?

1. vSmart
2. vBond
3. vManage
4. vEdge

10. When initially authenticating to the Cisco SD-WAN REST API, how are the username and password encoded?

1. application/postscript
2. application/xml
3. application/json
4. application/x-www-form-urlencoded

FOUNDATION TOPICS

WHAT IS AN SDK?

An SDK (software development kit) or devkit is a set of software development tools that developers can use to create software or applications for a certain platform, operating system, computer system, or device. An SDK typically contains a set of libraries, APIs, documentation, tools, sample code, and processes that make it easier for developers to integrate, develop, and extend the platform. An SDK is created for a specific programming language, and it is very common to have the same functionality exposed through SDKs in different programming languages.

Chapter 6, “Application Programming Interfaces (APIs),” describes what an API is, and Chapter 7 covers the most common API framework these days, the REST API framework. As a quick reminder, an application

programming interface (API) is, as the name implies, a programming interface that implements a set of rules developers and other software systems can use to interact with a specific application.

As you will see in this chapter and throughout this book, where there is an API, there is usually also an SDK. Software developers can, of course, implement and spend time on developing code to interact with an API (for example, building their own Python classes and methods to authenticate, get data from the API, or create new objects in the API), or they can take advantage of the SDK, which makes all these objects already available.

Besides offering libraries, tools, documentation, and sample code, some SDKs also offer their own integrated development environments (IDEs). For example, the SDKs available for mobile application development on Google Android and Apple iOS also make available an IDE to give developers a complete solution to create, test, debug, and troubleshoot their applications.



A good SDK has these qualities:

- Is easy to use
- Is well documented
- Has value-added functionality
- Integrates well with other SDKs
- Has minimal impact on hardware resources

In order to be used by developers, an SDK should be easy to use and follow best practices for software development in the programming language for which the SDK was developed. For example, for Python development, there are Python Enhancement Proposals (PEPs), which are documents that provide guidance and spell out best practices for how Python code should be organized,

packaged, released, deprecated, and so on. PEP8 is a popular standard for styling Python code and is extensively used in the developer community.

Documenting the SDK inline as well as having external documentation is critical to developer adoption and the overall quality of the SDK. Having good, up-to-date documentation of the SDK makes the adoption and understanding of the code and how to use it much easier. A good SDK also adds value by saving development time and providing useful features. Integrating with other SDKs and development tools should be easy and scalable, and the code should be optimized for minimal hardware resource utilization as well as execution time.



SDKs provide the following advantages:

- Quicker integration
- Faster and more efficient development
- Brand control
- Increased security
- Metrics

As mentioned previously, there are significant development time savings when adopting SDKs, as the functionality and features provided by an SDK don't have to be developed in house. This leads to quicker integration with the API and quicker time to market. In addition, brand control can be enforced with the SDK. The look and feel of applications developed using the SDK can be uniform and in line with the overall brand design. For example, applications developed for iOS using the Apple SDK have a familiar and uniform look and feel because they use the building blocks provided by the SDK. Application security best practices can be enforced through SDKs. When you develop using a security-conscious SDK, the applications developed have the SDK's security features integrated automatically. Sets

of metrics and logging information can be included with an SDK to provide better insights into how the SDK is being used and for troubleshooting and performance tweaking.

It is critical to ensure a great experience for all developers when interacting with an API. In addition, offering a great SDK with an API is mandatory for success.

Cisco has been developing applications and software since its inception. As the requirements for integrations with other applications and systems have grown, APIs have been developed to make it easier for developers and integrators to create and develop their own solutions and integrations. Throughout the years, software architectures have evolved, and currently all Cisco solutions provide some type of API. As mentioned earlier, where there is an API, there is usually also an SDK.

The starting point in exploring all the SDKs that Cisco has to offer is <https://developer.cisco.com>. As you will see in the following sections of this chapter and throughout this book, there are several SDKs developed by Cisco and third parties that take advantage of the APIs that currently exist with all Cisco products.

CISCO MERAKI

Meraki became part of Cisco following its acquisition in 2012. The Meraki portfolio is large, comprising wireless, switching, security, and video surveillance products. The differentiating factor for Meraki, compared to similar products from Cisco and other vendors, is that management is cloud based. Explore all the current Cisco Meraki products and offerings at <https://meraki.cisco.com>.



From a programmability perspective, the Meraki cloud platform provides several APIs:

- Captive Portal API
- Scanning API
- MV Sense Camera API
- Dashboard API

The Cisco Meraki cloud platform also provides webhooks, which offer a powerful and lightweight way to subscribe to alerts sent from the Meraki cloud when an event occurs. (For more about webhooks, see [Chapter 7](#).) A Meraki alert includes a JSON-formatted message that can be configured to be sent to a unique URL, where it can be further processed, stored, and acted upon to enable powerful automation workflows and use cases.

The Captive Portal API extends the power of the built-in Meraki splash page functionality by providing complete control of the content and authentication process that a user interacts with when connecting to a Meraki wireless network. This means Meraki network administrators can completely customize the portal, including the onboarding experience for clients connecting to the network, how the web page looks and feels, and the authentication and billing processes.

The Scanning API takes advantage of Meraki smart devices equipped with wireless and BLE (Bluetooth Low Energy) functionality to provide location analytics and report on user behavior. This can be especially useful in retail, healthcare, and enterprise environments, where business intelligence and information can be extracted about trends and user engagement and behavior. The Scanning API delivers data in real time and can be used to detect Wi-Fi and BLE devices and clients. The data is exported to a specified destination server through an

HTTP POST of JSON documents. At the destination server, this data can then be further processed, and applications can be built on top of the received data. Taking into consideration the physical placement of the access points on the floor map, the Meraki cloud can estimate the location of the clients connected to the network. The geolocation coordinates of this data vary based on a number of factors and should be considered as a best-effort estimate.

The MV Sense Camera API takes advantage of the powerful onboard processor and a unique architecture to run machine learning workloads at the edge. Through the MV Sense API, object detection, classification, and tracking are exposed and become available for application integration. You can, for example, extract business insight from video feeds at the edge without the high cost of compute infrastructure that is typically needed with computer imaging and video analytics. Both REST and MQTT API endpoints are provided, and information is available in a request or subscribe model. MQ Telemetry Transport (MQTT) is a client/server publish/subscribe messaging transport protocol. It is lightweight, simple, open, and easy to implement. MQTT is ideal for use in constrained environments such as Internet of Things (IoT) and machine-to-machine communication where a small code footprint is required.

The Meraki APIs covered so far are mostly used to extract data from the cloud platform and build integrations and applications with that data. The Dashboard API, covered next, provides endpoints and resources for configuration, management, and monitoring automation of the Meraki cloud platform. The Dashboard API is meant to be open ended and can be used for many purposes and use cases. Some of the most common use cases for the Dashboard API are as follows:

- Provisioning new organizations, administrators, networks, devices, and more
- Configuring networks at scale
- Onboarding and decommissioning of clients
- Building custom dashboards and applications

To get access to the Dashboard API, you first need to enable it. Begin by logging into the Cisco Meraki dashboard at <https://dashboard.meraki.com> using your favorite web browser and navigating to Organization > Settings. From there, scroll down and locate the section named Dashboard API Access and make sure you select Enable Access and save the configuration changes at the bottom of the page. Once you have enabled the API, select your username at the top-right corner of the web page and select My Profile. In your profile, scroll down and locate the section named Dashboard API Access and select Generate New API Key. The API key you generate is associated with your account. You can generate, revoke, and regenerate your API key in your profile. Make sure you copy and store your API key in a safe place, as whoever has this key can impersonate you and get access through the Dashboard API to all the information your account has access to. For security reasons, the API key is not stored in plaintext in your profile, so if you lose the key, you will have to revoke the old one and generate a new one. If you believe that your API key has been compromised, you can generate a new one to automatically revoke the existing API key.

Every Dashboard API request must specify an API key within the request header. If a missing or incorrect API key is specified, the API returns a 404 HTTP error message. Recall from [Chapter 7](#) that HTTP error code 404 means that the API resource you were trying to reach could not be found on the server. This error code prevents information leakage and unauthorized discovery of API resources.

The key for the authentication request header is X-Cisco-Meraki-API-Key, and the value is the API key you obtained previously.

In order to mitigate abuse and denial-of-service attacks, the Cisco Meraki Dashboard API is limited to 5 API calls per second per organization. In the first second, a burst of an additional 5 calls is allowed, for a maximum of 15 API calls in the first 2 seconds per organization. If the rate limit has been exceeded, an error message with HTTP status code 429 is returned. The rate-limiting technique that the Dashboard API implements is based on the token bucket model. The token bucket is an algorithm used to check that the data that is transmitted in a certain amount of time complies with set limits for bandwidth and burstiness. Based on this model, if the number of API requests crosses the set threshold for a certain amount of time, you have to wait a set amount of time until you can make another request to the API. The time you have to wait depends on how many more requests you have performed above the allowed limit; the more requests you have performed, the more time you have to wait.

The Cisco Meraki Dashboard API uses the base URL <https://api.meraki.com/api/v0>. Keep in mind that the API will evolve, and different versions will likely be available in the future. Always check the API documentation for the latest information on all Cisco APIs, including the Meraki APIs, at <https://developer.cisco.com>.

To make it easier for people to become comfortable with the Meraki platform, the Dashboard API is organized to mirror the structure of the Meraki dashboard. When you become familiar with either the API or the GUI, it should be easy to switch between them. The hierarchy of the Dashboard API looks as follows:

- Organizations

- Networks
 - Devices
 - Uplink

Most Dashboard API calls require either the organization ID or the network ID as part of the endpoint. (You will see later in this chapter how to obtain these IDs and how to make Dashboard API calls.) When you have these IDs, you can build and make more advanced calls to collect data, create and update new resources, and configure and make changes to the network. Remember that all API calls require an API key.

If your Meraki dashboard contains a large number of organizations, networks, and devices, you might have to consider pagination when making API calls. Recall from [Chapter 7](#) that pagination is used when the data returned from an API call is too large and needs to be limited to a subset of the results. The Meraki Dashboard API supports three special query parameters for pagination:

- **perPage:** The number of entries to be returned in the current request
- **startingAfter:** A value used to indicate that the returned data will start immediately after this value
- **endingBefore:** A value used to indicate that the returned data will end immediately before this value

While the types of the **startingAfter** and **endingBefore** values differ based on API endpoints, they generally are either timestamps specifying windows in time for which the data should be returned or integer values specifying IDs and ranges of IDs.

The Dashboard API also supports action batches, which make it possible to submit multiple configuration requests in a single transaction and are ideal for initial provisioning of a large number of devices or performing large configuration changes throughout the whole network. Action batches also provide a mechanism to avoid hitting the rate limitations implemented in the API for high-scale configuration changes as you can

implement all the changes with one or a small number of transactions instead of a large number of individual API requests. Action batch transactions can be run either synchronously, waiting for the API call return before continuing, or asynchronously, in which case the API call does not wait for a return as the call is placed in a queue for processing. (In [Chapter 7](#) you saw the advantages and disadvantages of both synchronous and asynchronous APIs.) With action batches, you can be confident that all the updates contained in the transaction were submitted successfully before being committed because batches are run in an atomic fashion: all or nothing.

After you have enabled the Meraki Dashboard API, generated the API key, and saved it in a safe place, you are ready to interact with the API. For the rest of this chapter, you will use the always-on Cisco DevNet Meraki Sandbox, which can be found at <https://developer.cisco.com/sandbox>. The API key for this sandbox is 15da0c6ffff295f16267f88f98694cf29a86ed87.

At this point, you need to obtain the organization ID for this account. As you saw in [Chapter 7](#), there are several ways you can interact with an API: You can use tools like curl and Postman, or you can interact with the API through programming languages and the libraries that they provide. In this case, you will use curl and Postman to get the organization ID for the Cisco DevNet Sandbox account and then the Cisco Meraki Python SDK.

As mentioned earlier, the base URL for the Dashboard API is <https://api.meraki.com/api/vo>. In order to get the organizations for the account with the API key mentioned previously, you have to append the /organizations resource to the base URL. The resulting endpoint becomes <https://api.meraki.com/api/vo/organizations>. You also need to include the X-Cisco-Meraki-API-Key header for

authentication purposes. This header will contain the API key for the DevNet Sandbox Meraki account. The **curl** command should look as follows in this case:

[Click here to view code image](#)

```
curl -I -X GET \  
  --url \  
'https://api.meraki.com/api/v0/organizations' \  
  -H 'X-Cisco-Meraki-API-Key: \  
15da0c6fffff295f16267f88f98694cf29a86ed87'
```

The response should look as shown in [Example 8-1](#).

Example 8-1 Headers of the *GET Organizations REST API Call*

[Click here to view code image](#)

```
HTTP/1.1 302 Found  
Server: nginx  
Date: Sat, 17 Aug 2019 19:05:25 GMT  
Content-Type: text/html; charset=utf-8  
Transfer-Encoding: chunked  
Connection: keep-alive  
Cache-Control: no-cache, no-store, max-age=0,  
must-revalidate  
Pragma: no-cache  
Expires: Fri, 01 Jan 1990 00:00:00 GMT  
X-Frame-Options: sameorigin  
X-Robots-Tag: none  
Location:  
https://n149.meraki.com/api/v0/organizations  
X-UA-Compatible: IE=Edge,chrome=1  
X-Request-Id: 87654dbd16ae23fbc7e3282a439b211c  
X-Runtime: 0.320067  
Strict-Transport-Security: max-age=15552000;  
includeSubDomains
```

You can see in [Example 8-1](#) that the response code for the request is 302. This indicates a redirect to the URL value in the Location header. Redirects like the one in [Example 8-1](#) can occur with any API call within the Dashboard API, including POST, PUT, and DELETE. For

GET calls, the redirect is specified through a 302 status code, and for any non-GET calls, the redirects are specified with 307 or 308 status codes. When you specify the **-I** option for **curl**, only the headers of the response are displayed to the user. At this point, you need to run the **curl** command again but this time specify the resource as <https://n149.meraki.com/api/v0/organizations>, remove the **-I** flag, and add an Accept header to specify that the response to the call should be in JSON format. The command should look like this:

[Click here to view code image](#)

```
curl -X GET \  
  
  --url  
'https://n149.meraki.com/api/v0/organizations' \  
  
  -H 'X-Cisco-Meraki-API-Key:  
    15da0c6ffff295f16267f88f98694cf29a86ed87'\  
  
  -H 'Accept: application/json'
```

The response in this case contains the ID of the DevNet Sandbox organization in JSON format:

[Click here to view code image](#)

```
[  
  {  
    "name" : "DevNet Sandbox",  
    "id" : "549236"  
  }  
]
```

Now let's look at how you can obtain the organization ID for the Cisco DevNet Sandbox Meraki account by using Postman. As mentioned in [Chapter 7](#), Postman is a popular tool used to explore APIs and create custom requests; it has extensive built-in support for different authentication mechanisms, headers, parameters, collections, environments, and so on. By default,

Postman has the Automatically Follow Redirects option enabled in Settings, so you do not have to change the <https://api.meraki.com/api/v0/organizations> resource as it is already done in the background by Postman. If you disable the Automatically Follow Redirects option in the Postman settings, you should see exactly the same behavior you just saw with **curl**. **Figure 8-1** shows the Postman client interface with all the fields (the method of the call, the resource URL, and the headers) populated so that the Cisco Meraki REST API returns all the organizations to which this account has access.

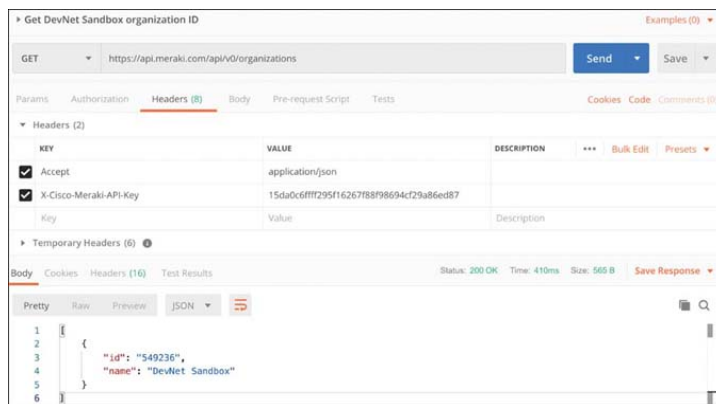


Figure 8-1 *GET Organizations REST API Call in Postman*

In the body of the response, you see the same JSON-formatted output as before, with the same organization ID for the DevNet Sandbox account.

Let's explore the Meraki Dashboard API further and obtain the networks associated with the DevNet Sandbox organization. If you look up the API documentation at <https://developer.cisco.com/meraki/api/#/rest/api-endpoints/networks/get-organization-networks>, you see that in order to obtain the networks associated with a specific organization, you need to do a GET request to <https://api.meraki.com/api/v0/organizations/{organizationId}/networks>, where {organizationId} is the ID you obtained previously, in your first interaction with the

Dashboard API. You have also discovered that the base URL for the DevNet Sandbox organization is <https://n149.meraki.com/api/v0>. You can modify the API endpoint with this information to use the following **curl** command:

[Click here to view code image](#)

```
curl -X GET \  
  
  --url \  
'https://n149.meraki.com/api/v0/organizations/549236/  
  
networks' \  
  
  -H 'X-Cisco-Meraki-API-Key:  
15da0c6fffff295f16267f88f98694c  
f29a86ed87' \  
  
  -H 'Accept: application/json'
```

The response from the API should contain a list of all the networks that are part of the DevNet Sandbox organization and should look similar to the output in [Example 8-2](#).

Example 8-2 *List of All the Networks in a Specific Organization*

[Click here to view code image](#)

```
[  
  {  
    "timeZone" : "America/Los_Angeles",  
    "tags" : " Sandbox ",  
    "organizationId" : "549236",  
    "name" : "DevNet Always On Read Only",  
    "type" : "combined",  
    "disableMyMerakiCom" : false,  
    "disableRemoteStatusPage" : true,  
    "id" : "L_646829496481099586"  
  },  
  {  
    "organizationId" : "549236",  
    "tags" : null,  
    "timeZone" : "America/Los_Angeles",  
    "id" : "N_646829496481152899",  
    "disableRemoteStatusPage" : true,  
    "name" : "test - mx65",  
    "disableMyMerakiCom" : false,  
  }  
]
```



```
"type" : "appliance"
}, ... omitted output
```

The output in [Example 8-2](#) shows a list of all the networks that are part of the DevNet Sandbox organization with an ID of 549236. For each network, the output contains the same information found in the Meraki dashboard. You should make a note of the first network ID returned by the API as you will need it in the next step in your exploration of the Meraki Dashboard API.

Now you can try to get the same information—a list of all the networks that are part of the DevNet Sandbox organization—by using Postman. As you’ve seen, Postman by default does the redirection automatically, so you can specify the API endpoint as <https://api.meraki.com/api/v0/organizations/549236/networks>. You need to make sure to specify the GET method, the X-Cisco-Meraki-API-Key header for authentication, and the Accept header, in which you specify that you would like the response from the API to be in JSON format. [Figure 8-2](#) shows the Postman client interface with all the information needed to obtain a list of all the networks that belong to the organization with ID 549236.

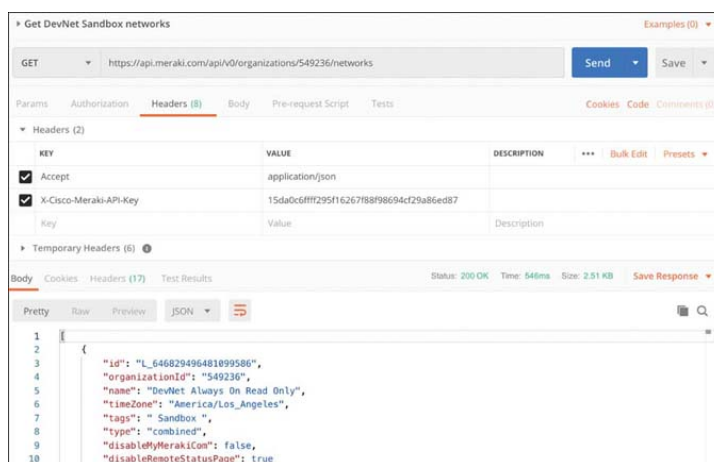


Figure 8-2 *GET Networks REST API Call in Postman*

The body of the response contains exactly the same information as before: a complete list of all the networks that are part of the DevNet Sandbox organization.

Next, you can obtain a list of all devices that are part of the network that has the name “DevNet Always On Read Only” and ID L_646829496481099586. Much as in the previous steps, you start by checking the API documentation to find the API endpoint that will return this data to you. The API resource that contains the information you are looking for is

/networks/{networkId}/devices, as you can see from the API documentation at the following link:

<https://developer.cisco.com/meraki/api/#/rest/api-endpoints/devices/get-network-devices>. You add the

base URL for the DevNet Sandbox account,

<https://n149.meraki.com/api/v0>, and populate

{networkId} with the value you obtained in the previous step. Combining all this information, the endpoint that

will return the information you are seeking is

https://n149.meraki.com/api/v0/networks/L_646829496481099586/devices. The **curl** command in this case is

as follows:

[Click here to view code image](#)

```
curl -X GET \  
  
  --url  
'https://n149.meraki.com/api/v0/networks/L_646829496481099586/  
  
  devices' \  
  
  -H 'X-Cisco-Meraki-API-Key:  
15da0c6ffff295f16267f88f98694cf29a86ed87' \  
  
  -H 'Accept: application/json'
```

And the response from the API should be similar to the one in [Example 8-3](#).

Example 8-3 *List of All the Devices in a Specific Network*

[Click here to view code image](#)

```
[
  {
    "wan2Ip" : null,
    "networkId" : "L_646829496481099586",
    "lanIp" : "10.10.10.106",
    "serial" : "QYYY-WWWW-ZZZZ",
    "tags" : " recently-added ",
    "lat" : 37.7703718,
    "lng" : -122.3871248,
    "model" : "MX65",
    "mac" : "e0:55:3d:17:d4:23",
    "wan1Ip" : "10.10.10.106",
    "address" : "500 Terry Francois, San
Francisco"
  },
  {
    "switchProfileId" : null,
    "address" : "",
    "lng" : -122.098531723022,
    "model" : "MS220-8P",
    "mac" : "88:15:44:df:f3:af",
    "tags" : " recently-added ",
    "serial" : "QAAA-BBBB-CCCC",
    "networkId" : "L_646829496481099586",
    "lanIp" : "192.168.128.2",
    "lat" : 37.4180951010362
  }
]
```

Notice from the API response that the “DevNet Always On Read Only” network has two devices: an MX65 security appliance and an eight-port MS-220 switch. The output also includes geographic coordinates, MAC addresses, serial numbers, tags, model numbers, and other information. The same information is available in the Meraki GUI dashboard.

Following the process used so far, you can obtain the same information but this time using Postman. Since the redirection is automatically done for you, the API endpoint for Postman is

https://api.meraki.com/api/v0/networks/L_646829496481099586/devices. You populate the two headers Accept and X-Cisco-Meraki-API-Key with their respective values, select the GET method, and click the Send button. If everything went well, the response code should be 200 OK, and the body of the response should contain exactly the same information found using **curl**. **Figure 8-3** shows the Postman client interface with all the headers and fields needed to get a list of all devices that are part of the network with ID L_646829496481099586.

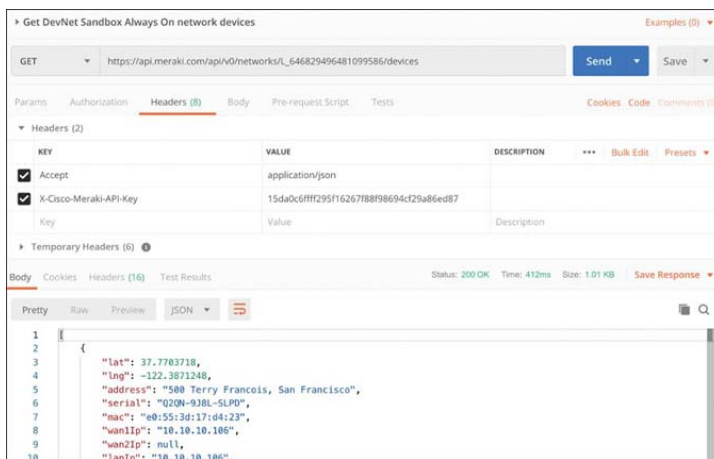


Figure 8-3 GET Devices REST API Call in Postman

So far, you have explored the Meraki Dashboard API using **curl** and Postman. You first obtained the organization ID of the DevNet Sandbox Meraki account and then, based on that ID, you obtained all the networks that are part of the organization and then used one of the network IDs you obtained to find all the devices that are part of that specific network. This is, of course, just a subset of all the capabilities of the Meraki Dashboard API, and we leave it as an exercise for you to explore in more depth all the capabilities and functionalities of the API.

As a final step in this section, let's take a look at the Meraki Python SDK. As of this writing, there are two

Meraki SDKs for the Dashboard API: one is Python based and the other is Node.js based. The Meraki Python SDK used in this book is version 1.0.2; it was developed for Python 3 and implements a complete set of classes, methods, and functions to simplify how users interact with the Dashboard API in Python.

In order to get access to the SDK, you need to install the `meraki-sdk` module. As a best practice, always use virtual environments with all Python projects. Once a virtual environment is activated, you can run **`pip install meraki-sdk`** to get the latest version of the SDK. In this section, you follow the same three steps you have followed in other examples in this chapter: Get the organization ID for the DevNet Sandbox account, get a list of all the networks that are part of this organization, and get all the devices associated to the “DevNet Always on Read Only” network. The Python 3 code to accomplish these three tasks might look as shown in [Example 8-4](#).

You need to import the `MerakiSdkClient` class from the `meraki_sdk` module. You use the `MerakiSdkClient` class to create an API client by passing the API key as a parameter and creating an instance of this class called `MERAKI`.

Example 8-4 *Python Script That Uses `meraki_sdk`*

[Click here to view code image](#)

```
#!/usr/bin/env python
from meraki_sdk.meraki_sdk_client import
MerakiSdkClient

#Cisco DevNet Sandbox Meraki API key
X_CISCO_MERAKI_API_KEY =
'15da0c6ffff295f16267f88f98694cf29a86ed87'

#Establish a new client connection to the
Meraki REST API
MERAKI =
MerakiSdkClient(X_CISCO_MERAKI_API_KEY)
```

```

#Get a list of all the organizations for the
Cisco DevNet account
ORGS = MERAKE.organizations.get_organizations()
for ORG in ORGS:
    print("Org ID: {} and Org Name:
    {}".format(ORG['id'], ORG['name']))

PARAMS = {}
PARAMS["organization_id"] = "549236" # Demo
Organization "DevNet Sandbox"

#Get a list of all the networks for the Cisco
DevNet organization
NETS =
MERAKE.networks.get_organization_networks(PARAMS)

for NET in NETS:
    print("Network ID: {0:20s}, Name:
    {1:45s}, Tags: {2:<10s}".format(\
        NET['id'], NET['name'],
        str(NET['tags'])))

#Get a list of all the devices that are part of
the Always On Network
DEVICES =
MERAKE.devices.get_network_devices("L_646829496
481099586")
for DEVICE in DEVICES:
    print("Device Model: {0:9s}, Serial:
    {1:14s}, MAC: {2:17}, Firmware:{3:12s}" \
        .format(DEVICE['model'],
        DEVICE['serial'], DEVICE['mac'], \
        DEVICE['firmware']))

```

After you instantiate the MerakiSdkClient class, you get an API client object that provides access to all the methods of the class. You can find the documentation for the Python SDK at <https://developer.cisco.com/meraki/api/#/python/guides/python-sdk-quick-start>. This documentation covers all the classes and methods, their parameters, and input and output values for the Python SDK implementation. Unlike with the **curl** and Postman examples earlier in this chapter, you do not need to determine the exact API resource that will return the information you are interested in; however, you do need to know how the MerakiSdkClient API class is organized and what

methods are available. There's a consistent one-to-one mapping between the Dashboard API and the Python SDK, so when you are familiar with one of them, you should find the other one very easy to understand. There is also no need to pass the API key through the X-Cisco-Meraki-API-Key header. This is all automatically handled by the SDK as well as all the redirects that had to manually be changed for the **curl** examples.

Obtaining the organization ID for the DevNet Sandbox account is as easy as invoking the **organizations.get_organizations()** method of the API client class. The **ORGS** variable in [Example 8-4](#) contains a list of all the organizations that the DevNet Sandbox account is a member of. Next, you iterate within a **for** loop through all these organizations and display to the console the organization ID and the organization name.

Next, you create an empty dictionary called **PARAMS** and add to it a key called `organization_id` with the value 549236. Remember that this was the organization ID for the DevNet Sandbox account. You still use the Meraki API client instance, but in this case, you invoke the **networks.get_organization_networks()** method. Just as in the case of the REST API calls with **curl** and Postman earlier in this section, where you had to specify the organization ID when building the endpoint to obtain the list of networks, the **get_organization_networks()** method takes as input the params dictionary, which contains the same organization ID value but in a Python dictionary format. The **NETS** variable stores the output of the API call. In another iterative loop, information about each network is displayed to the console.

Finally, you get the list of devices that are part of the network and have the ID `L_646829496481099586`. Recall from earlier that this ID is for the “DevNet Always

on Read Only” network. In this case, you use the `devices.get_network_devices()` method of the Meraki API client instance and store the result in the `DEVICES` variable. You iterate over the `DEVICES` variable and, for each device in the list, extract and print to the console the device model, the serial number, the MAC address, and the firmware version.

Running the Python 3 script discussed here should result in output similar to that shown in [Figure 8-4](#).

```

Org ID: 549236 and Org Name: DevNet Sandbox
Network ID: L_646829496481099586, Name: DevNet Always On Read Only, Tags: Sandbox
Network ID: N_646829496481152899, Name: test - mx65, Tags: None
Network ID: L_646829496481103251, Name: DNENT3-Derrick.Robba@anm.com, Tags: None
Network ID: L_646829496481103256, Name: DNSMB3, Tags: None
Network ID: N_646829496481157538, Name: API Test - Postman 1242, Tags: test
Network ID: N_646829496481157539, Name: API Test - Postman 12420, Tags: test
Network ID: L_646829496481103262, Name: DNSMB2, Tags: None
Network ID: L_646829496481103267, Name: DNENT1-jll@nycap.rr.com-ken@ken-farrell.co.uk, Tags: None
Network ID: N_646829496481157582, Name: API Test - Postman 124201, Tags: test
Network ID: L_646829496481103275, Name: DNENT2-jll@nycap.rr.com-Mark.Porter@anm.com, Tags: None
Network ID: L_646829496481103292, Name: DNSMB1, Tags: None
Network ID: L_646829496481103296, Name: DNSMB5, Tags: None
Network ID: L_646829496481103298, Name: DNSMB4-ellopeza@cisco.com, Tags: None
Device Model: MX65, Serial: Q2QN-9J8L-SLPD, MAC: e0:55:3d:17:d4:23, Firmware:wired-13-28
Device Model: MS220-8P, Serial: Q2HP-F5K5-R88R, MAC: 88:15:44:df:f3:af, Firmware:switch-9-37

```

Figure 8-4 Output of the Python Script from Example 8-4

CISCO DNA CENTER

Cisco Digital Network Architecture (DNA) is an open, extensible, software-driven architecture from Cisco that accelerates and simplifies enterprise network operations. Behind this new architecture is the concept of intent-based networking, a new era in networking, in which the network becomes an integral and differentiating part of the business. With Cisco DNA and the products behind it, network administrators define business intents that get mapped into infrastructure configurations by a central SDN controller. In the future, an intent-based network will dynamically adjust itself based on what it continuously learns from the traffic it transports as well as the business inputs it gets from the administrator.

Cisco DNA Center is the network management and command center for Cisco DNA. With Cisco DNA Center, you can provision and configure network devices in

minutes, define a consistent policy throughout a network, get live and instantaneous statistics, and get granular networkwide views. Multidomain and multivendor integrations are all built on top of a secure platform.



From a programmability perspective, Cisco DNA Center provides a set of REST APIs and SDKs through the Cisco DNA Center platform that are grouped in the following categories:

- Intent API
- Integration API
- Multivendor SDK
- Events and notifications

The Intent API is a northbound REST API that exposes specific capabilities of the Cisco DNA Center platform. The main purpose of the Intent API is to simplify the process of creating workflows that consolidate multiple network actions into one. An example is the SSID provisioning API, which is part of the Intent API. When configuring an SSID on a wireless network, several operations need to be completed, including creating a wireless interface, adding WLAN settings, and adding security settings. The SSID provisioning API combines all these operations and makes them available with one API call. This results in a drastic reduction in overall wireless SSID deployment time and also eliminates errors and ensures a consistent configuration policy. The Intent API provides automation capabilities and simplified workflows for QoS policy configuration, software image management and operating system updates for all devices in the network, overall client health status, and monitoring of application health. Application developers and automation engineers can take advantage of this single northbound integration

layer to develop tools and applications on top of the network.

One of the main goals of the Cisco DNA Center platform is to simplify and streamline end-to-end IT processes. The Integration API was created for exactly this purpose. Through this API, Cisco DNA Center platform publishes network data, events, and notifications to external systems and at the same time can consume information from these connected systems. Integrations with IT service management systems like Service Now, BMC Remedy, and other ticketing systems are supported. Automatic ticket creation and assignment based on network issues that are flagged by Cisco DNA Center are now possible. Cisco DNA Center can even suggest remediation steps based on the machine learning algorithms that are part of the assurance capabilities. You can see how a typical support workflow can be improved with Cisco DNA Center platform in the future. Cisco DNA Center detects a network issue, automatically creates a ticket and assigns it to the right support personnel, along with a possible solution to the issue. The support team reviews the ticket and the suggested solution and can approve either immediately or during a maintenance window the remediation of the problem that Cisco DNA Center suggested. IP Address Management (IPAM) integrations are also supported by the Integration API. It is possible to seamlessly import IP address pools of information from IPAM systems such as Infoblox and BlueCat into Cisco DNA Center. Synchronization of IP pool/subpool information between Cisco DNA Center and IPAM systems is also supported. Through the Integration API that Cisco DNA Center provides, developers can integrate with any third-party IPAM solution.

Data as a service (DaaS) APIs that are part of the Integration API allow Cisco DNA Center to publish insights and data to external tools such as Tableau and

similar reporting solutions. IT administrators have the option to build dashboards and extract business-relevant insights.

The Integration API is also used for cross-domain integrations with other Cisco products, such as Cisco Meraki, Cisco Stealthwatch, and Cisco ACI. The idea is to deliver a consistent intent-based infrastructure across the data center, WAN, and security solutions.

Cisco DNA Center allows customers to have their non-Cisco devices managed by DNA Center through a multivendor SDK. Cisco DNA Center communicates with third-party devices through device packages. The device packages are developed using the multivendor SDK and implement southbound interfaces based on CLI, SNMP, or NETCONF.

Cisco DNA Center also provides webhooks for events and notifications that are generated in the network or on the Cisco DNA Center appliance itself. You have the option of configuring a receiving URL to which the Cisco DNA Center platform can publish events. Based on these events, the listening application can take business actions. For instance, if some of the devices in a network are out of compliance, the events that the platform generates can be interpreted by a custom application, which might trigger a software upgrade action in Cisco DNA Center. This completes the feedback loop in the sense that a notification generated by Cisco DNA Center is interpreted by a third-party custom application and acted upon by sending either an Intent API or Integration API call back to Cisco DNA Center to either remedy or modify the network, based on the desired business outcome. This mechanism of publishing events and notifications also saves on processing time and resources; before this capability existed, the custom application had to poll continuously to get the status of an event. By subscribing through the webhook, polling

can now be avoided entirely, and the custom application receives the status of the event right when it gets triggered.

Next, let's focus on the Cisco DNA Center Platform Intent API. As of this writing, the Intent API is organized into several distinct categories:

- **Know Your Network category:** This category contains API calls pertaining to sites, networks, devices, and clients:
 - With the Site Hierarchy Intent API, you can get information about, create, update, and delete sites as well as assign devices to a specific site. (Sites within Cisco DNA Center are logical groupings of network devices based on a geographic location or site.)
 - The Network Health Intent API retrieves data regarding network devices, their health, and how they are connected.
 - The Network Device Detail Intent API retrieves detailed information about devices. Different parameters can be passed to limit the scope of the information returned by the API, such as timestamp, MAC address, and UUID. Besides all the detailed information you can retrieve for all the devices in the network, you can also add, delete, update, or sync specified devices.
 - The Client Health Intent API returns overall client health information for both wired and wireless clients.
 - The Client Detail Intent API returns detailed information about a single client.
- **Site Management category:** This category helps provision enterprise networks with zero-touch deployments and manage the activation and distribution of software images in the network:
 - The Site Profile Intent API gives you the option to provision NFV and ENCS devices as well as retrieve the status of the provisioning activities.
 - The Software Image Management (SWIM) API enables you to completely manage the lifecycle of software images running within a network in an automated fashion. With this API, you can retrieve information about available software images, import images into Cisco DNA Center, distribute images to devices, and activate software images that have been deployed to devices.
 - The Plug and Play (PnP) API enables you to manage all PnP-related workflows. With this API, you can create, update, and delete PnP workflows and PnP server profiles, claim and unclaim devices, add and remove virtual accounts, and retrieve information about all PnP-related tasks.
- **Connectivity category:** This category contains APIs that provide mechanisms to configure and manage both non-fabric wireless and Cisco SDA wired fabric devices. For fabric devices, you can add and

remove border devices to the fabric and get details about their status. For non-fabric wireless devices, you can create, update, and delete wireless SSIDs, profiles, and provisioning activities.

- **Operational Tools category:** This category includes APIs for the most commonly used tools in the Cisco DNA Center toolbelt:
 - The Command Runner API enables the retrieval of all valid keywords that Command Runner accepts and allows you to run read-only commands on devices to get their real-time configuration.
 - The Network Discovery API provides access to the discovery functionalities of Cisco DNA Center. You can use this API to create, update, delete, and manage network discoveries and the credentials needed for them. You can also retrieve network discoveries, network devices that were discovered as part of a specific network discovery task, and credentials associated with these discoveries.
 - The Template Programmer API can be used to manage configuration templates. You can create, view, edit, delete, version, add commands, check contents for errors, deploy, and check the status of template deployments.
 - The Path Trace API provides access to the Path Trace application in Cisco DNA Center. Path Trace can be used to troubleshoot and trace application paths throughout the network and provide statistics at each hop. The API gives you access to initiating, retrieving, and deleting path traces.
 - The File API enables you to retrieve files such as digital certificates, maps, and SWIM files from Cisco DNA Center.
 - The Task API provides information about the network actions that are being run asynchronously. Each of these background actions can take from seconds to minutes to complete, and each has a task associated with it. You can query the Task API about the completion status of these tasks, get the task tree, retrieve tasks by their IDs, and so on.
 - The Tag API gives you the option of creating, updating, and deleting tags as well as assigning tags to specific devices. Tags are very useful in Cisco DNA Center; they are used extensively to group devices by different criteria. You can then apply policies and provision and filter these groups of devices based on their tags.

The Cisco DNA Center platform APIs are rate limited to five API requests per minute.

So far in this section, we've covered all the APIs and the multivendor SDK offered by Cisco DNA Center. Next, we will start exploring the Intent API, using Cisco DNA Center version 1.3 for the rest of the chapter. As API resources and endpoints exposed by the Cisco DNA

Center platform might change in future versions of the software, it is always best to start exploring the API documentation for any Cisco product at <https://developer.cisco.com/docs/dna-center/api/1-3-0-x/>.

In Cisco DNA Center version 1.3, the REST API is not enabled by default. Therefore, you need to log in to DNA Center with a super-admin role account, navigate to Platform > Manage > Bundles, and enable the DNA Center REST API bundle. The status of the bundle should be active, as shown in [Figure 8-5](#).

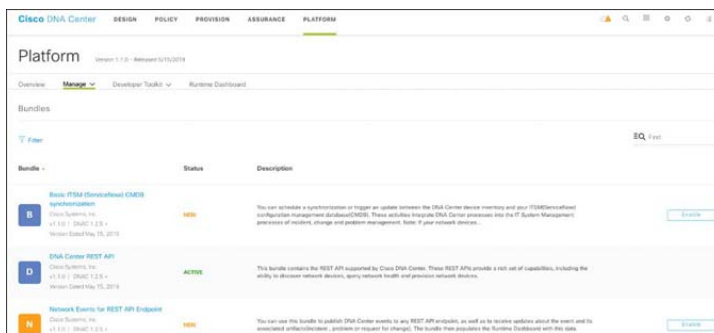


Figure 8-5 Cisco DNA Center Platform Interface

For this section, you can use the always-on DevNet Sandbox for Cisco DNA Center at <https://sandboxdnac2.cisco.com>. The username for this sandbox is **devnetuser**, and the password is **Cisco123!**. You need to get authorized to the API and get the token that you will use for all subsequent API calls. The Cisco DNA Center platform API authorization is based on basic auth. Basic auth, as you learned in [Chapter 7](#), is an authorization type that requires a username and password to access an API endpoint. In the case of Cisco DNA Center, the username and password mentioned previously are base-64 encoded and then transmitted to the API service in the Authorization header. There are many online services that can do both encoding and decoding of base-64 data for you, or as a fun challenge you can look up how to do it manually. The username

devnetuser and the password **Cisco123!** become **ZGV2-bmVodXNlcjpaXNjbzEyMyE=** when they are base-64 encoded. The only missing component is the resource that you need to send the authorization request to. You verify the documentation and see that the authorization resource is `/system/api/v1/auth/token` and requires the API call to be POST. With this information, you can build the authorization API endpoint, which becomes <https://sandboxnac2.cisco.com/system/api/v1/auth/token>.

Next, you will use **curl**, a command-line tool that is useful in testing REST APIs and web services. Armed with the authorization API endpoint, the Authorization header containing the base-64-encoded username and password, and the fact that the API call needs to be a POST call, you can now craft the authorization request in **curl**. The authorization API call with **curl** should look as follows:

[Click here to view code image](#)

```
curl -X POST \  
  
https://sandboxnac2.cisco.com/dna/system/api/v1/auth/token \  
 \  
  -H 'Authorization: Basic \  
ZGV2bmV0dXNlcjpaXNjbzEyMyE='
```

The result should be JSON formatted with the key `Token` and a value containing the actual authorization token. It should look similar to the following:

[Click here to view code image](#)

```
{"Token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Y2U3MmV0dXNlcjpaXNjbzEyMyE="}
```

```
DA4OWYwZmYzNyJdLCJ0ZW5hbnRjZCI6IjViNmNmZGZjNDMwOTkwMDA4OWYwZ-  
mYzMCIsImV4cCI6MTU2NjU0Mzk3OCwidXNlcm5hbWUiOiJkZXZuZXRlc2VyIn0.  
Qv6vU6d1tqFGx9GETj6S1Da8Ts6uJNk9624onLSNSnU" }
```

Next, you can obtain an authorization token by using Postman. Make sure you select POST as the verb for the authorization API call and the endpoint <https://sandboxnac2.cisco.com/dna/system/api/v1/auth/token>. This is a POST call because you are creating new data in the system—in this case, a new authorization token. Under the Authorization tab, select Basic Auth as the type of authorization, and in the Username and Password fields, make sure you have the correct credentials (devnetuser and Cisco123!). Since you have selected basic auth as the authorization type, Postman does the base-64 encoding for you automatically. All you need to do is click Send, and the authorization API call is sent to the specified URL. If the call is successfully completed, the status code should be 200 OK, and the body of the response should contain the JSON-formatted token key and the corresponding value. [Figure 8-6](#) shows the Postman client interface with the information needed to successfully authenticate to the always-on Cisco DevNet DNA Center Sandbox.

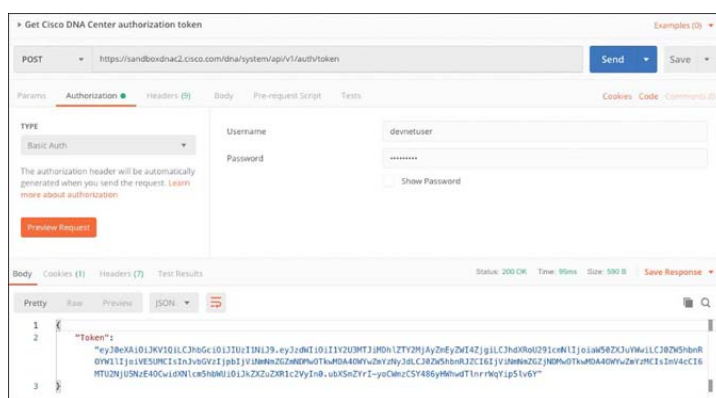


Figure 8-6 *Authenticating to Cisco DNA Center over the REST API*

The body of the response for the Postman request should look as follows:

[Click here to view code image](#)

```
{
  "Token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Y2U3MTJiMDhlZiTY2MjAyZmEyZWI4ZjgiLCJhdXRoU291cmNlIjoiaW50ZXJyYWwiLCJ0ZW5hbnROYyW1lIjoive5UMCI6IjVjbGVzIjpbIjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLmV4cCI6MTU2NjU5NzE4OCwidXN1cm5hbWUiOiJkZXZuZXRlc2VyIn0.ubXSmZyYrI-yoCWmzCSY486y-HWhwdTlnrrWqYip5lv6Y"
}
```

As with the earlier **curl** example, this token will be used in all subsequent API calls performed in the rest of this chapter. The token will be passed in the API calls through a header that is called **X-Auth-Token**.

Let's now get a list of all the network devices that are being managed by the instance of Cisco DNA Center that is running in the always-on DevNet Sandbox you've just authorized with. If you verify the Cisco DNA Center API documentation on <https://developer.cisco.com/docs/dna-center/api/1-3-0-x/>, you can see that the API resource that will return a complete list of all network devices managed by Cisco DNA Center is `/dna/intent/api/v1/network-device`. [Figure 8-7](#) shows the online documentation for Cisco DNA Center version 1.3.



Figure 8-7 Cisco DNA Center Platform API Documentation (<https://developer.cisco.com>)

With all this information in mind, you can craft the **curl** request to obtain a list of all the network devices managed by the Cisco DevNet always-on DNA Center Sandbox. The complete URL is <https://sandboxdnac2.cisco.com/dna/intent/api/v1/network-device>. You need to retrieve information through the API, so we need to do a GET request; don't forget the X-Auth-Token header containing the authorization token. The **curl** command should look as follows, and it should contain a valid token:

[Click here to view code image](#)

```
curl -X GET \

https://sandboxdnac2.cisco.com/dna/intent/api/v1/network-
device \

-H 'X-Auth-Token:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJzdWIiOiI1Y2U3MTJiMDhlZTY2MjAyZmEyZWl4ZjgiLCJhdXRoU291c-
mNlIjoiaW50ZXJuYWwiLCJ0ZW5hbnROYW11IjoiveUMCIIsInJvbGVzIjpbI-
jViNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLCJ0ZW5hbnRJCi6IjViNmNmZG-
ZjNDMwOTkwMDA4OWYwZmYzMCIsImV4cCI6MTU2NjYwODAxMSwidXNlcm5hbWUiOi-
JkZXZuZXRlc2VyIn0.YXc_2o8FDzSQ1YBhUxUIoxwzYXXWYeNJRkBooKBlIHI'
```

The response to this **curl** command should look as shown in [Example 8-5](#).

Example 8-5 *List of Network Devices Managed by the Always-On Cisco DNA Center Sandbox Instance*

[Click here to view code image](#)

```
{
  "response" : [
    {
      "type" : "Cisco 3504 Wireless LAN
Controller",
      "roleSource" : "AUTO",
      "apManagerInterfaceIp" : "",
      "lastUpdateTime" : 1566603156991,
      "inventoryStatusDetail" : "<status>
<general code=\"SUCCESS\"/></status>",
      "collectionStatus" : "Managed",
      "serialNumber" : "FCW2218MOB1",
      "location" : null,
      "waasDeviceMode" : null,
      "tunnelUdpPort" : "16666",
      "reachabilityStatus" : "Reachable",
      "lastUpdated" : "2019-08-23 23:32:36",
      "tagCount" : "0",
      "series" : "Cisco 3500 Series Wireless
LAN Controller",
      "snmpLocation" : "",
      "upTime" : "158 days, 13:59:36.00",
      "lineCardId" : null,
      "id" : "50c96308-84b5-43dc-ad68-
cda146d80290",
      "reachabilityFailureReason" : "",
      "lineCardCount" : null,
      "managementIpAddress" : "10.10.20.51",
      "memorySize" : "3735302144",
      "errorDescription" : null,
      "snmpContact" : "",
      "family" : "Wireless Controller",
      "platformId" : "AIR-CT3504-K9",
      "role" : "ACCESS",
      "softwareVersion" : "8.5.140.0",
      "hostname" : "3504_WLC",
      "collectionInterval" : "Global
Default",
      "bootDateTime" : "2019-01-19
02:33:05",
      "instanceTenantId" : "SYS0",
      "macAddress" : "50:61:bf:57:2f:00",
      "errorCode" : null,
      "locationName" : null,
    }
  ]
}
```

```
        "softwareType" : "Cisco Controller",
        "associatedWlcIp" : "",
        "instanceUuid" : "50c96308-84b5-43dc-
ad68-cda146d80290",
        "interfaceCount" : "8"
    },
    ... omitted output
],
"version" : "1.0"
}
```

From this verbose response, you can extract some very important information about the network devices managed by Cisco DNA Center. The data returned in the response is too verbose to fully include in the previous output, so just a snippet is included for your reference. As of this writing, in the complete response output, you can see that there are 14 devices in this network:

- One AIR-CT3504-K9 wireless LAN controller
- One WS-C3850-24P-L Catalyst 3850 switch
- Two C9300-48U Catalyst 9300 switches
- Ten AIR-AP141N-A-K9 wireless access points

For each device, you can see extensive information such as the hostname, uptime, serial number, software version, management interface IP address, reachability status, hardware platform, and role in the network. You can see here the power of the Cisco DNA Center platform APIs. With one API call, you were able to get a complete status of all devices in the network. Without a central controller like Cisco DNA Center, it would have taken several hours to connect to each device individually and run a series of commands to obtain the same information that was returned with one API call in less than half a second. These APIs can save vast amounts of time and bring numerous possibilities in terms of infrastructure automation. The data returned by the API endpoints can be extremely large, and it might take a long time to process a request and return a complete response. As mentioned in [Chapter 7](#), pagination is an API feature that

allows for passing in parameters to limit the scope of the request. Depending on the Cisco DNA Center platform API request, different filter criteria can be considered, such as management IP address, MAC address, and hostname.

Now you will see how to obtain the same information you just got with **curl** but now using Postman. The same API endpoint URL is used:

<https://sandboxnac2.cisco.com/dna/intent/api/v1/network-device>. In this case, it is a GET request, and the X-Auth-Token header is specified under the Headers tab and populated with a valid token. If you click Send and there aren't any mistakes with the request, the status code should be 200 OK, and the body of the response should be very similar to that obtained with the **curl** request. **Figure 8-8** shows how the Postman interface should look in this case.

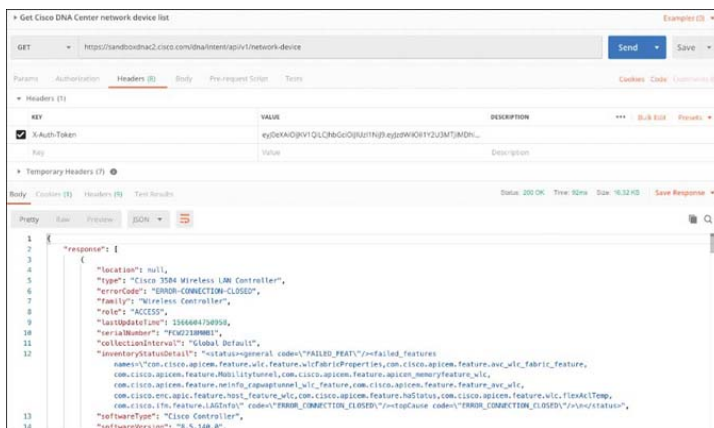


Figure 8-8 Getting a List of Network Devices

Now you can try to obtain some data about the clients that are connected to the network managed by Cisco DNA Center. Much like network devices, network clients have associated health scores, provided through the Assurance feature to get a quick overview of client network health. This score is based on several factors, including onboarding time, association time, SNR (signal-to-noise ratio), and RSSI (received signal

strength indicator) values for wireless clients, authentication time, connectivity and traffic patterns, and number of DNS requests and responses. In the API documentation, you can see that the resource providing the health status of all clients connected to the network is `/dna/intent/api/v1/client-health`. This API call requires a parameter to be specified when performing the call. This parameter, called `timestamp`, represents the UNIX epoch time in milliseconds. UNIX epoch time is a system for describing a point in time since January 1, 1970, not counting leap seconds. It is extensively used in UNIX and many other operating systems. The `timestamp` provides the point in time for which the client health information should be returned in the API response. For example, if I retrieved the health status of all the clients connected to the network on Thursday, August 22, 2019 8:41:29 PM GMT, the UNIX time, in milliseconds, would be `1566506489000`. Keep in mind that based on the data retention policy set in Cisco DNA Center, client data might not be available for past distant timeframes.

With the information you now have, you can build the API endpoint to process the API call:
<https://sandboxnac2.cisco.com/dna/intent/api/v1/client-health?timestamp=1566506489000>. The authorization token also needs to be included in the call as a value in the X-Auth-Token header. The **curl** command should look as follows:

[Click here to view code image](#)

```
curl -X GET \  
  
https://sandboxnac2.cisco.com/dna/intent/api/v1/client-  
health?timestamp=1566506489000 \  
  
-H 'X-Auth-Token:  
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
eyJzdWIiOiI1Y2U3MTJiMDhlZTY2MjAyZmEyZWl4ZjgiLCJhdXRoU291c  
mNlIjoiaW50ZXJuYWwiLCJ0ZW5hbnROYWllIjoieVE5UMCIiInJvbnVzIjpbI
```

```
jViNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLcJ0ZW5hbnRjZCI6IjViNmNmZG-  
ZjNDMwOTkwMDA4OWYwZmYzMCI6MTU2NjYxODkyOCwidXNlcm5hbWUio  
iJkZXZuZXRlc2VyIn0.7JNXdgSMi3Bju8v8QU_L5nmBKYOTivinAjP8ALT_opw'
```

The API response should look similar to [Example 8-6](#).

From this response, you can see that there are a total of 82 clients in the network, and the average health score for all of them is 27. To further investigate why the health scores for some of the clients vary, you can look into the response to the `/dna/intent/api/v1/client-detail` call. This API call takes as input parameters the timestamp and the MAC address of the client, and it returns extensive data about the status and health of that specific client at that specific time.

Now you can try to perform the same API call but this time with Postman. The API endpoint stays the same: <https://sandboxnac2.cisco.com/dna/intent/api/v1/client-health?timestamp=1566506489000>. In this case, you are trying to retrieve information from the API, so it will be a GET call, and the X-Auth-Token header contains a valid token value. Notice that the Params section of Postman gets automatically populated with a timestamp key, with the value specified in the URL: 1566506489000. Click Send, and if there aren't any errors with the API call, the body of the response should be very similar to the one obtained previously with **curl**. The Postman window for this example should look as shown in [Figure 8-9](#).

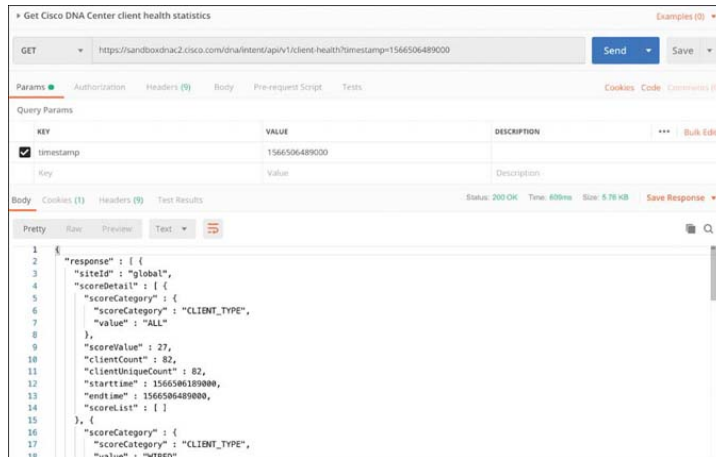


Figure 8-9 Viewing Client Health in Cisco DNA Center

Example 8-6 List of Clients and Their Status

[Click here to view code image](#)

```

{
  "response" : [ {
    "siteId" : "global",
    "scoreDetail" : [ {
      "scoreCategory" : {
        "scoreCategory" : "CLIENT_TYPE",
        "value" : "ALL"
      },
      "scoreValue" : 27,
      "clientCount" : 82,
      "clientUniqueCount" : 82,
      "starttime" : 1566506189000,
      "endtime" : 1566506489000,
      "scoreList" : [ ]
    }, ... output omitted
  ]
}

```

So far in this section, you've explored the Cisco DNA Center platform API and seen how to authorize the API calls and obtain a token, how to get a list of all the devices in the network, and how to get health statistics for all the clients in the network using both **curl** and Postman. Next let's explore the Cisco DNA Center Python SDK. The SDK has been developed for Python 3 and maps all the Cisco DNA Center APIs into Python

classes and methods to make it easier for developers to integrate with and expand the functionality of Cisco DNA Center. Installing the SDK is as simple as issuing the command **pip install dnacentersdk** from the command prompt. At this writing, the current Cisco DNA Center SDK version is 1.3.0. The code in [Example 8-7](#) was developed using this version of the SDK and Python 3.7.4. The code in [Example 8-7](#) uses the SDK to first authorize to the API and then retrieve a list of all the network devices and client health statuses.

Example 8-7 *Python Script That Exemplifies the Use of the Cisco DNA Center Python SDK*

[Click here to view code image](#)

```
#!/usr/bin/env python
from dnacentersdk import api

# Create a DNACenterAPI connection object;
# it uses DNA Center sandbox URL, username and
password
DNAC = api.DNACenterAPI(username="devnetuser",
                        password="Cisco123!",

base_url="https://sandboxdnac2.cisco.com")

# Find all devices
DEVICES = DNAC.devices.get_device_list()

# Print select information about the retrieved
devices
print('{0:25s}{1:1}{2:45s}{3:1}
{4:15s}'.format("Device Name", "|", \
               "Device Type", "|", "Up Time"))
print('-'*95)
for DEVICE in DEVICES.response:
    print('{0:25s}{1:1}{2:45s}{3:1}
{4:15s}'.format(DEVICE.hostname, \
               "|", DEVICE.type, "|", DEVICE.upTime))
print('-'*95)

# Get the health of all clients on Thursday,
August 22, 2019 8:41:29 PM GMT
CLIENTS =
DNAC.clients.get_overall_client_health(timestamp="
1566506489000")

# Print select information about the retrieved
```

```

client health statistics
print('{0:25s}{1:1}{2:45s}{3:1}
{4:15s}'.format("Client Category", "|", \
    "Number of Clients", "|", "Clients Score"))
print('-'*95)
for CLIENT in CLIENTS.response:
    for score in CLIENT.scoreDetail:
        print('{0:25s}{1:1}{2:<45d}{3:1}{4:
<15d}'.format(
            score.scoreCategory.value, "|",
            score.clientCount, "|", \
            score.scoreValue))
print('-'*95)

```

First, this example imports the **api** class from **dnacentersdk**. Next, it instantiates the **api** class and creates a connection to the always-on Cisco DevNet Sandbox DNA Center instance and stores the result of that connection in a variable called **DNAC**. If the connection was successfully established, the **DNAC** object has all the API endpoints mapped to methods that are available for consumption.

DNAC.devices.get_device_list() provides a list of all the devices in the network and stores the result in the **DEVICES** dictionary. The same logic applies to the client health status.

DNAC.clients.get_overall_client_health(timestamp='1566506489000') returns a dictionary of health statuses for all the clients at that specific time, which translates to Thursday, August 22, 2019 8:41:29 PM GMT. When the data is extracted from the API and stored in the variables named **DEVICES** for all the network devices and **CLIENTS** for all the client health statuses, a rudimentary table is displayed in the console, with select information from both dictionaries. For the **DEVICES** variable, only the device name, device type, and device uptime are displayed, and for the **CLIENTS** variable, only the client health category, number of clients, and client score for each category are displayed.

The output of the Python script should look similar to that in [Figure 8-10](#).

Device Name	Device Type	Up Time
3504_WLC	Cisco 3504 Wireless LAN Controller	159 days, 13:27:47.00
leaf1.lab.local	Cisco Catalyst 9300 Switch	127 days, 5:31:13.60
leaf2.lab.local	Cisco Catalyst 9300 Switch	126 days, 12:29:10.39
spine1.abc.in.lab.local	Cisco Catalyst38xx stack-able ethernet switch	159 days, 13:37:57.53
T1-1	Cisco 1140 Unified Access Point	158days 11:05:55.340
T1-10	Cisco 1140 Unified Access Point	158days 11:05:55.350
T1-2	Cisco 1140 Unified Access Point	158days 11:05:55.340
T1-3	Cisco 1140 Unified Access Point	158days 11:05:55.340
T1-4	Cisco 1140 Unified Access Point	158days 11:05:54.340
T1-5	Cisco 1140 Unified Access Point	158days 11:05:54.340
T1-6	Cisco 1140 Unified Access Point	158days 11:05:55.340
T1-7	Cisco 1140 Unified Access Point	158days 11:05:55.340
T1-8	Cisco 1140 Unified Access Point	158days 11:05:55.350
T1-9	Cisco 1140 Unified Access Point	158days 11:05:55.350
Client Category	Number of Clients	Clients Score
ALL	82	27
WIRED	2	100
WIRELESS	80	25

Figure 8-10 Output of the Python Script from Example 8-7

CISCO SD-WAN

Cisco SD-WAN (Software-Defined Wide Area Network) is a cloud-first architecture for deploying WAN connectivity. Wide-area networks have been deployed for a long time, and many lessons and best practices have been learned throughout the years. Applying all these lessons to software-defined networking (SDN) resulted in the creation of Cisco SD-WAN. An important feature of SDN is the separation of the control plane from the data plane.

The control plane includes a set of protocols and features that a network device implements so that it can determine which network path to use to forward data traffic. Spanning tree protocols and routing protocols such as OSPF (Open Shortest Path First), EIGRP (Enhanced Interior Gateway Routing Protocol), and BGP (Border Gateway Protocol) are some of the protocols that make up the control plane in network devices. These protocols help build the switching or routing tables in network devices to enable them to determine how to forward network traffic.

The data plane includes the protocols and features that a network device implements to forward traffic to its destination as quickly as possible. Cisco Express Forwarding (CEF) is a proprietary switching mechanism that is part of the data plane. It was developed specifically to increase the speed with which data traffic is forwarded through network devices. You can read more about the control and data planes in [Chapter 17, “Networking Components.”](#)

Historically, the control plane and data plane were part of the network device architecture, and they worked together to determine the path that the data traffic should take through the network and how to move this traffic as fast as possible from its source to its destination. As mentioned previously, software-defined networking (SDN) suggests a different approach.

SDN separates the functionality of the control plane and data plane in different devices, and several benefits result. First, the cost of the resulting network should be lower as not all network devices have to implement expensive software and hardware features to accommodate both a control plane and data plane. The expensive intelligence from the control plane is constrained to a few devices that become the brains of the network, and the data plane is built with cheaper devices that implement only fast forwarding. Second, the convergence of this new network, which is the amount of time it takes for all devices to agree on a consistent view of the network, should be much lower than in the case of the non-SDN architectures of the past. In networks of similar sizes, the ones built with network devices that implement both the control plane and the data plane in their architecture take much longer to exchange all the information needed to forward data traffic than do the networks that implement separate control and data plane functionality in their architecture. Depending on the size of a network, this could mean waiting for

thousands of devices to exchange information through their control plane protocols and settle on a certain view of the network or wait for tens of SDN controllers to accomplish the same task; the convergence time improvements are massive.

Cisco currently has two SD-WAN offerings. The first one, based on the Viptela acquisition, is called Cisco SD-WAN; the second one, based on the Meraki acquisition, is called Meraki SD-WAN. We already covered Cisco Meraki at the beginning of this chapter; this section covers Cisco SD-WAN based on the Viptela acquisition.

You've already seen some of the advantages that SDN brings to WAN connectivity. Based on this new architecture and paradigm, the Cisco SD-WAN offering contains several products that perform different functions:



- **vManage:** Cisco vManage is a centralized network management system that provides a GUI and REST API interface to the SD-WAN fabric. You can easily manage, monitor, and configure all Cisco SD-WAN components through this single pane of glass.
- **vSmart:** Cisco vSmart is the brains of the centralized control plane for the overlay SD-WAN network. It maintains a centralized routing table and centralized routing policy that it propagates to all the network Edge devices through permanent DTLS tunnels.
- **vBond:** Cisco vBond is the orchestrator of the fabric. It authenticates the vSmart controllers and the vEdge devices and coordinates connectivity between them. The vBond orchestrator is the only component in the SD-WAN fabric that needs public IP reachability to ensure that all devices can connect to it.
- **vEdge:** Cisco vEdge routers, as the name implies, are Edge devices that are located at the perimeter of the fabric, such as in remote offices, data centers, branches, and campuses. They represent the data plane and bring the whole fabric together and route traffic to and from their site across the overlay network.

All the components of the Cisco SD-WAN fabric run as virtual appliances, and the vEdges are also available as hardware routers.

Separating the WAN fabric this way makes it more scalable, faster to converge, and cheaper to deploy and maintain. On top of a transport-independent underlay that supports all types of WAN circuits (MPLS, DSL, broadband, 4G, and so on), an overlay network is being built that runs OMP (Overlay Management Protocol). Much like BGP, OMP propagates throughout the network all the routing information needed for all the components of the fabric to be able to forward data according to the routing policies configured in vManage.

Cisco vManage provides a REST API interface that exposes the functionality of the Cisco SD-WAN software and hardware features. The API resources that are available through the REST interface are grouped in the following collections:

- **Administration:** For management of users, groups, and local vManage instance
- **Certificate Management:** For management of SSL certificates and security keys
- **Configuration:** For creation of feature and device configuration templates and creation and configuration of vManage clusters
- **Device Inventory:** For collecting device inventory information, including system status
- **Monitoring:** For getting access to status, statistics, and related operational information about all the devices in the network every 10 minutes from all devices
- **Real-Time Monitoring:** For gathering real-time monitoring statistics and traffic information approximately once per second
- **Troubleshooting Tools:** For API calls used in troubleshooting, such as to determine the effects of applying a traffic policy, updating software, or retrieving software version information

Cisco vManage exposes a self-documenting web interface for the REST API, based on the OpenAPI specification. This web interface is enabled by default and can be accessed at https://vManage_IP_or_hostname:port/apidocs. vManage_IP_or_hostname is the IP address or hostname of the Cisco vManage server, and the port is 8443 by default. The rest of this chapter uses the always-

on Cisco DevNet SD-WAN Sandbox, available at <https://sandboxsdwan.cisco.com>. The username for this vManage server is **devnetuser**, and the password is **Cisco123!**. At this writing, this sandbox is running Cisco SD-WAN version 18.3 for all components of the fabric. Because this sandbox will be updated in time, or you might be using a different version, you should check <https://developer.cisco.com> for the latest information on all Cisco REST APIs documentation and changes. After you specify the credentials, the self-documenting web interface of <https://sandboxsdwan.cisco.com:8443/apidocs> looks as shown in [Figure 8-11](#).

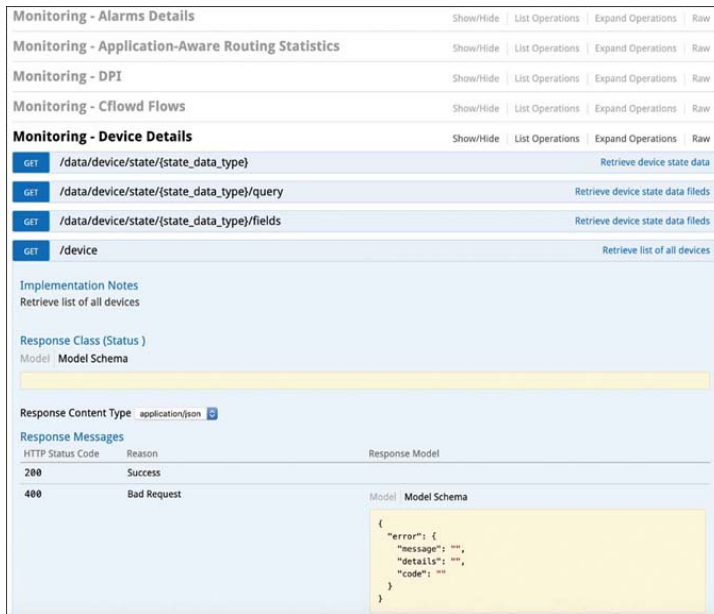


Figure 8-11 Cisco SD-WAN OpenAPI Specification-Based Interface

This web interface displays a list of all the REST API resources available, the methods associated with each one of them, and the model schema of the responses. The option of trying out each API call and exploring the returned data is also available.

Let's explore the Cisco vManage REST API next. The API documentation can be found at https://sdwan-docs.cisco.com/Product_Documentation/Command_Reference/Command_Reference/vManage_REST_APIs. At this link, you can find all the information needed on how to interact with the REST API, all the resources available, and extensive explanations.

You need to establish a connection to the Cisco vManage instance. The initial connection is established through an authorization request to https://sandboxsdwan.cisco.com:8443/j_security_check. The information sent over this POST call is URL form encoded and contains the username and password mentioned previously. The **curl** request for this authorization request should look as follows:

[Click here to view code image](#)

```
curl -c - -X POST -k \  
  
https://sandboxsdwan.cisco.com:8443/j_security_check \  
\  
-H 'Content-Type: application/x-www-form- \  
urlencoded' \  
-d 'j_username=devnetuser&j_password=Cisco123!'
```

The **-c** option passed to the **curl** request specifies that the returned authorization cookie should be printed to the console. The **-k** option bypasses SSL certificate verification as the certificate for this sandbox is self-signed. The output of the command should look as follows:

[Click here to view code image](#)

```
# Netscape HTTP Cookie File  
  
# https://curl.haxx.se/docs/http-cookies.html  
  
# This file was generated by libcurl! Edit at your  
own risk.
```



```
#HttpOnly_sandboxswan.cisco.com. FALSE / TRUE
0 JSESSIONID.
v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.4854266f-
a8ad-4068-9651-
d4e834384f51
```

The long string after JSESSIONID is the value of the authorization cookie that will be needed in all subsequent API calls.

Figure 8-12 shows the same API call in Postman.

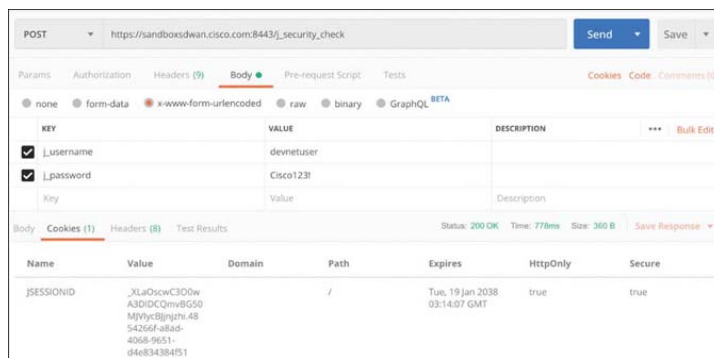


Figure 8-12 Cisco SD-WAN REST API Authorization Call

The status code of the response should be 200 OK, the body should be empty, and the JSESSIONID cookie should be stored under the Cookies tab. The advantage with Postman is that it automatically saves the JSESSIONID cookie and reuses it in all API calls that follow this initial authorization request. With **curl**, in contrast, you have to pass in the cookie value manually. To see an example, you can try to get a list of all the devices that are part of this Cisco SD-WAN fabric. According to the documentation, the resource that will return this information is `/dataservice/device`. It will have to be a GET call, and the JSESSIONID cookie needs to be passed as a header in the request. The **curl** command to get a list of all the devices in the fabric should look like as follows:

[Click here to view code image](#)

```
curl -X GET -k \  
  
https://sandboxsdwan.cisco.com:8443/dataservice/device  
\   
  
-H 'Cookie:  
JSESSIONID=v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.4  
854266f-a8ad-4068-9651-d4e834384f51'
```

The response from the always-on Cisco DevNet vManage server should look similar to the one in [Example 8-8](#).

Example 8-8 *List of Devices That Are Part of the Cisco SD-WAN Fabric*

[Click here to view code image](#)

```
{  
  ... omitted output  
  "data" : [  
    {  
      "state" : "green",  
      "local-system-ip" : "4.4.4.90",  
      "status" : "normal",  
      "latitude" : "37.666684",  
      "version" : "18.3.1.1",  
      "model_sku" : "None",  
      "connectedVManages" : [  
        "\"4.4.4.90\""  
      ],  
      "statusOrder" : 4,  
      "uuid" : "4854266f-a8ad-4068-9651-  
d4e834384f51",  
      "deviceId" : "4.4.4.90",  
      "reachability" : "reachable",  
      "device-groups" : [  
        "\"No groups\""  
      ],  
      "total_cpu_count" : "2",  
      "certificate-validity" : "Valid",  
      "board-serial" : "01",  
      "platform" : "x86_64",  
      "device-os" : "next",  
      "timezone" : "UTC",  
      "uptime-date" : 1567111260000,  
      "host-name" : "vmanage",  
      "device-type" : "vmanage",  
      "personality" : "vmanage",  
      "domain-id" : "0",  
      "isDeviceGeoData" : false,  
      "lastupdated" : 1567470387553,
```

```
        "site-id" : "100",
        "controlConnections" : "5",
        "device-model" : "vmanage",
        "validity" : "valid",
        "system-ip" : "4.4.4.90",
        "state_description" : "All daemons
up",
        "max-controllers" : "0",
        "layoutLevel" : 1,
        "longitude" : "-122.777023"
    },
    ... omitted output
]
}
```

The output of this GET API call is too verbose to be fully displayed. We encourage you to explore this API call and observe the full response on your own computer.

The body of the response is in JSON format and contains information about all the devices in the SD-WAN fabric. As of this writing, this specific fabric contains the following:

- One Cisco vManage server
- One Cisco vSmart server
- One Cisco vBond server
- Four Cisco vEdge routers

For each device, the response includes status, geographic coordinates, role, device ID, uptime, site ID, SSL certificate status, and more. You can build the same request in Postman and send it to vManage as shown in [Figure 8-13](#). The body of the response is very similar to the one received from the **curl** command.

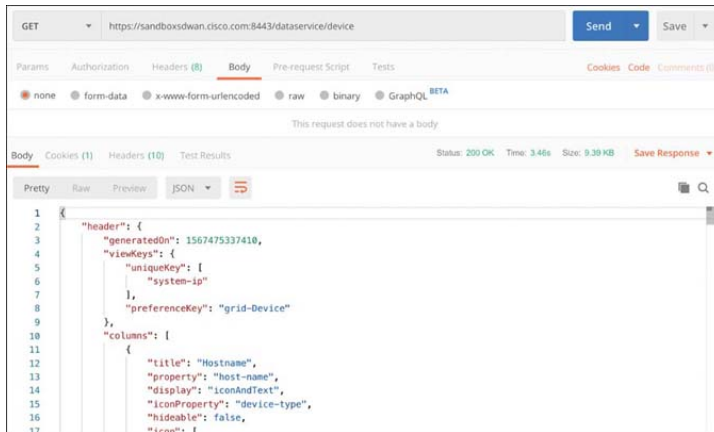


Figure 8-13 Getting a List of All the Devices in the Cisco SD-WAN Fabric

While exploring the Cisco SD-WAN REST API, let's get a list of all the device templates that are configured on the Cisco DevNet vManage server. According to the API documentation, the resource that will return this information is `/dataservice/template/device`. You pass in the `JSESSIONID` value in the cookie header and build the following **curl** command:

[Click here to view code image](#)

```

curl -X GET -k \

https://sandboxsdwan.cisco.com:8443/dataservice/template/device \

-H 'Cookie:
JSESSIONID=v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.48
54266f-a8ad-4068-9651-d4e834384f51'

```

The response from the vManage server at <https://sandboxsdwan.cisco.com> should look as shown in [Example 8-9](#).

Example 8-9 List of Device Configuration Templates

[Click here to view code image](#)

```

{
  "data" : [
    {

```

```
        "templateDescription" : "VEDGE BASIC
TEMPLATE01",
        "lastUpdatedOn" : 1538865915509,
        "templateAttached" : 15,
        "deviceType" : "vedge-cloud",
        "templateId" : "72babaf2-68b6-4176-
92d5-fa8de58e19d8",
        "configType" : "template",
        "devicesAttached" : 0,
        "factoryDefault" : false,
        "templateName" :
"VEDGE_BASIC_TEMPLATE",
        "lastUpdatedBy" : "admin"
    }
],
... output omitted
}
```

The response contains details about the only device template available on this vManage server. The template is called VEDGE_BASIC_TEMPLATE, it is of type vedge-cloud (which means it can be applied to vEdge devices), and it currently has no devices attached to it.

The same information is returned by vManage when using Postman to get the list of all device templates. As before, the JSESSIONID cookie is already included with Postman and does not need to be specified again. [Figure 8-14](#) shows the Postman client interface with all the parameters needed to retrieve a list of all the device configuration templates available on a specific vManage instance.

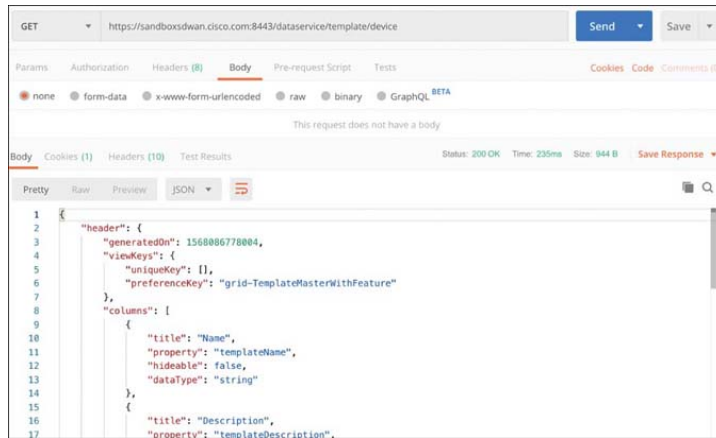


Figure 8-14 Getting Device Configuration Templates

Next, let's use Python to build a script that will go through the same steps: Log in to vManage, get a list of all the devices in the SD-WAN fabric, and get a list of all device templates available. No SDK will be used in this case; this will help you see the difference between this code and the Python code you used earlier in this chapter. Since no SDK will be used, all the API resources, payloads, and handling of data will have to be managed individually.

The Python **requests** library will be used extensively in this sample code. You should be familiar with this library from [Chapter 7](#). [Example 8-10](#) shows a possible version of the Python 3 script that accomplishes these tasks. The script was developed using Python 3.7.4 and version 2.22.0 of the requests library. The **json** library that comes with Python 3.7.4 was also used to deserialize and load the data returned from the REST API into a Python object; in this case, that object is a list. In this code, first, the **import** keyword makes the two libraries **requests** and **json** available for use within the script. Since the connection in the script is made to an instance of vManage that is in a sandbox environment and that uses a self-signed SSL certificate, the third and fourth lines of the script disable the warning messages that are generated by the **requests** library when connecting to

REST API endpoints that are secured with self-signed SSL certificates. Next, the script specifies the vManage hostname and the username and password for this instance; this example uses the same vManage server used earlier in this chapter. The code then specifies the base URL for the vManage REST API endpoint: <https://sandboxsdwan.cisco.com:8443>. The code shows the authentication resource (`j_security_check`) and the login credentials, and then the login URL is built as a combination of the base URL and the authentication API resource. In the next line, a new **request** session instance is created and stored in the **SESS** variable.

Example 8-10 *Python Script Showcasing How to Interact with the Cisco SD-WAN REST API*

[Click here to view code image](#)

```
#!/usr/bin/env python
import json
import requests
from requests.packages.urllib3.exceptions
import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

# Specify Cisco vManage IP, username and
password
VMANAGE_IP = 'sandboxsdwan.cisco.com'
USERNAME = 'devnetuser'
PASSWORD = 'Cisco123!'

BASE_URL_STR =
'https://{}:8443/'.format(VMANAGE_IP)

# Login API resource and login credentials
LOGIN_ACTION = 'j_security_check'
LOGIN_DATA = {'j_username' : USERNAME,
'j_password' : PASSWORD}
# URL for posting login data
LOGIN_URL = BASE_URL_STR + LOGIN_ACTION

# Establish a new session and connect to Cisco
vManage
SESS = requests.session()
LOGIN_RESPONSE = SESS.post(url=LOGIN_URL,
data=LOGIN_DATA, verify=False)
```

```

# Get list of devices that are part of the
fabric and display them
DEVICE_RESOURCE = 'dataservice/device'
# URL for device API resource
DEVICE_URL = BASE_URL_STR + DEVICE_RESOURCE

DEVICE_RESPONSE = SESS.get(DEVICE_URL,
verify=False)
DEVICE_ITEMS =
json.loads(DEVICE_RESPONSE.content)['data']

print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}
{6:16s}{7:1}{8:7s}'\
      .format("Host-Name", "|", "Device Model",
"|", "Device ID", \
      "|", "System IP", "|", "Site ID"))
print('-'*105)

for ITEM in DEVICE_ITEMS:
    print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}
{6:16s}{7:1}{8:7s}'\
          .format(ITEM['host-name'], "|",
ITEM['device-model'], "|", \
          ITEM['uuid'], "|", ITEM['system-
ip'], "|", ITEM['site-id']))
print('-'*105)
# Get list of device templates and display them
TEMPLATE_RESOURCE =
'dataservice/template/device'
# URL for device template API resource
TEMPLATE_URL = BASE_URL_STR + TEMPLATE_RESOURCE

TEMPLATE_RESPONSE = SESS.get(TEMPLATE_URL,
verify=False)
TEMPLATE_ITEMS =
json.loads(TEMPLATE_RESPONSE.content)['data']

print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}
{6:16s}{7:1}{8:7s}'\
      .format("Template Name", "|", "Device
Model", "|", "Template ID", \
      "|", "Attached devices", "|", "Template
Version"))
print('-'*105)

for ITEM in TEMPLATE_ITEMS:
    print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}
{6:<16d}{7:1}{8:<7d}'\
          .format(ITEM['templateName'], "|",
ITEM['deviceType'], "|", \
          ITEM['templateId'], "|"
ITEM['devicesAttached'], "|", \
          ITEM['templateAttached']))
print('-'*105)

```


Using this new session, a POST request is sent to the login URL, containing the username and password as payload and disabling the SSL certificate authenticity verification by specifying `verify=False`. At this point, a session is established to the DevNet Sandbox vManage instance. This session can be used to interact with the vManage REST API by getting, creating, modifying, and deleting data.

The code specifies the API resource that will return a list of all the devices in the SD-WAN fabric:

`dataservice/device`. The complete URL to retrieve the devices in the fabric is built on the next line by combining the base URL with the new resource. The **DEVICE_URL** variable will look like <https://sandboxsdwan.cisco.com:8443/dataservice/device>. Next, the same session that was established earlier is used to perform a GET request to the `device_url` resource. The result of this request is stored in the variable aptly named **DEVICE_RESPONSE**, which contains the same JSON-formatted data that was obtained in the previous **curl** and Postman requests, with extensive information about all the devices that are part of the SD-WAN fabric. From that JSON data, only the list of devices that are values of the **data** key are extracted and stored in the **DEVICE_ITEMS** variable.

Next, the header of a rudimentary table is created. This header contains the fields Host-Name, Device Model, Device ID, System IP, and Site ID. From the extensive list of information contained in the **DEVICE_ITEMS** variable, only these five fields will be extracted and displayed to the console for each device in the fabric. The code next prints a series of delimiting dashes to the console to increase the readability of the rudimentary table. The next line of code has a **for** loop that is used to iterate over each element of the **DEVICE_ITEMS** list

and extract the hostname, device model, device ID, system IP address, and site ID for each device in the fabric and then display that information to the console. The code then prints a series of dashes for readability purposes. Next, the same logic is applied to GET data from the API but this time about all the device templates that are configured on this instance of vManage. The URL is built by concatenating the base URL with the device template resource, dataservice/template/device. The same session is reused once more to obtain the data from the REST API. In the case of the device templates, only the template name, the type of device the template is intended for, the template ID, the number of attached devices to each template, and the template version are extracted and displayed to the console.

If you run this script in a Python 3.7.4 virtual environment with the **requests** library version 2.22.0 installed, you get output similar to that shown in [Figure 8-15](#).

Host-Name	Device Model	Device ID	System IP	Site ID
vmanage	vmanage	4854266f-a8ad-4068-9651-d4e834384f51	4.4.4.90	100
vsmart	vsmart	da6c566f-eb5f-4731-a89a-ff745661027c	4.4.4.70	100
vbond	vedge-c cloud	455407de-9327-467e-a002-d3444659dbb2	4.4.4.80	100
vedge01	vedge-c cloud	4a19e049-0052-47e9-83af-81a582577ffe	4.4.4.60	200
vedge02	vedge-c cloud	f3d4159b-4172-462c-9c8d-8db76c31521d	4.4.4.61	300
vedge03.cisco.com	vedge-c cloud	100faf9-8b36-4312-bf97-743b26bd0211	4.4.4.62	555
vedge04	vedge-c cloud	46c18a49-f6f3-4588-a49a-0b1cc387f179	4.4.4.63	400
Template Name	Device Model	Template ID	Attached devices	Template version
VEDGE_BASIC_TEMPLATE	vedge-c cloud	72babaf2-68b6-4176-92d5-fa8de58e19d8	1	15

Figure 8-15 Output of the Python Script from Example 8-10

This chapter has explored several Cisco solutions and their REST APIs. Authentication and authorization methods have been explained, and basic information has been obtained from the APIs. This chapter has provided a basic introduction to these extensive APIs and the features that they expose. We encourage you to continue your exploration of these APIs and build your own use cases, automation, and network programmability projects.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 8-2](#) lists these key topics and the page number on which each is found.



Table 8-2 Key Topics

Key Topic Element	Description	Page Number
L i s t	Qualities of a good SDK	<u>1</u> 7 7
L i s t	Advantages of SDKs	<u>1</u> 7 7
L i s t	The Meraki cloud platform provides several APIs from a programmability perspective	<u>1</u> 7 <u>8</u>
L i s t	Cisco DNA Center REST APIs and SDKs	<u>1</u> 9 0

L i s t	Cisco SD-WAN products that perform different functions	<u>2</u> <u>0</u> <u>2</u>
------------------	--	----------------------------------

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

software development kit (SDK)

Python Enhancement Proposals (PEP)

Bluetooth Low Energy (BLE)

MQ Telemetry Transport (MQTT)

Cisco Digital Network Architecture (DNA)

data as a service (DaaS)

Software Image Management (SWIM) API

Plug and Play (PnP) API

Cisco Software Defined-WAN (SD-WAN)

Open Shortest Path First (OSPF)

Enhanced Interior Gateway Routing Protocol (EIGRP)

Border Gateway Protocol (BGP)

Cisco Express Forwarding (CEF)

software-defined networking (SDN)

Overlay Management Protocol (OMP)

Chapter 9

Cisco Data Center and Compute Management Platforms and APIs

This chapter covers the following topics:

- **Cisco ACI:** This section describes Cisco ACI and the APIs it exposes.
- **Cisco UCS Manager:** This section covers Cisco UCS Manager and the public APIs that come with it.
- **Cisco UCS Director:** This section goes over Cisco UCS Director and its APIs.
- **Cisco Intersight:** This section introduces Cisco Intersight and its REST API interface.

This chapter begins exploring Cisco data center technologies and the SDKs and APIs that are available with them. First, it provides an introduction to Cisco Application Centric Infrastructure (ACI) and its components. This chapter also looks at the Cisco ACI REST API and the resources exposed over the API, as well as how to use a popular Python library called `acitoolkit` to extract data from the API. This chapter examines next Cisco Unified Computing System (UCS) and how all its components work together to offer one of the most comprehensive and scalable data center compute solutions available today. This chapter also provides an overview of Cisco UCS Manager, the XML API it provides, and how to interact with this API by using `curl` commands and the Cisco UCS Manager SDK. Cisco UCS Director takes data center automation to the next level, and this chapter covers the tasks and workflows that are available with it. The chapter also discusses the Cisco UCS Director SDK and its components and the `curl` commands that are used to

interact and extract data from the REST API. Finally, the chapter covers Cisco Intersight, a software as a service (SaaS) product that takes Cisco UCS management into the cloud. The chapter wraps up by covering the REST API interface of Cisco Intersight and the Python SDK that comes with it.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 9-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 9-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Cisco ACI	1–3
Cisco UCS Manager	4–6
Cisco UCS Director	7–8
Cisco Intersight	9–10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the

answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. On what family of switches does Cisco ACI run?

1. Cisco Catalyst 9000
2. Cisco Nexus 9000
3. Cisco Nexus 7000
4. Cisco Catalyst 6800

2. True or false: An ACI bridge domain can be associated with multiple VRF instances.

1. True
2. False

3. What Cisco ACI REST API endpoint is used for authentication?

1. https://APIC_IP_or_Hostname/api/aaaLogin.json
2. https://APIC_IP_or_Hostname/api/login
3. https://APIC_IP_or_Hostname/api/v1/aaaLogin
4. https://APIC_IP_or_Hostname/api/v1/login.json

4. In Cisco UCS Manager, what is the logical construct that contains the complete configuration of a physical server?

1. Server profile
2. Service profile
3. Template profile
4. None of the above

5. What is the Cisco UCS Manager Python SDK library called?

1. ucmsdk
2. ucssdk
3. ucsm
4. ciscoucsm

6. What is the managed object browser called in Cisco UCS Manager?

1. Mobrowser
2. UCSMobrowser
3. Visore
4. UCSVisore

7. What is a Cisco UCS Director workflow?

1. The atomic unit of work in Cisco UCS Director
2. A single action with inputs and outputs
3. A collection of predefined tasks
4. A series of tasks arranged to automate a complex operation

8. What is the name of the header that contains the Cisco UCS Director REST API access key?

1. X-Cloupia-Access-Key
2. X-Cloupia-Request-Key
3. X-Cloupia-Secret-Key
4. X-Cloupia-API-Access

9. How are the managed objects organized in Cisco Intersight?

1. Management information table
2. Hierarchical management information tree
3. Management Information Model
4. Hierarchical Managed Objects Model

10. What does the Cisco Intersight REST API key contain?

1. keyId and keySecret
2. token
3. accessKey and secretKey
4. cookie

FOUNDATION TOPICS

CISCO ACI

Cisco Application Centric Infrastructure (ACI) is the SDN-based solution from Cisco for data center deployment, management, and monitoring. The solution is based on two components: the Cisco Nexus family of switches and Cisco Application Policy Infrastructure Controller (APIC).



The Cisco Nexus 9000 family of switches can run in two separate modes of operation, depending on the software

loaded on them. The first mode is called standalone (or NX-OS) mode, which means the switches act like regular Layer 2/Layer 3 data center devices that are usually managed individually. In the second mode, ACI mode, the Cisco Nexus devices are part of an ACI fabric and are managed in a centralized fashion. The central controller for the ACI fabric is the Cisco Application Policy Infrastructure Controller (APIC). This controller is the main architectural component of the Cisco ACI solution and provides a single point of automation and management for the Cisco ACI fabric, policy enforcement, and health monitoring. The Cisco APIC was built on an API-first architecture from its inception. On top of this API, a command-line interface (CLI) and a graphical user interface (GUI) have been developed. The API is exposed through a REST interface and is accessible as a northbound interface for users and developers to integrate and develop their own custom solutions on top of the Cisco APIC and Cisco ACI fabric. The Cisco APIC interacts with and manages the Cisco Nexus switches through the OpFlex protocol, which is exposed as a southbound interface. From an SDN controller perspective (similar to the Cisco DNA Center controller described in [Chapter 8, “Cisco Enterprise Networking Management Platforms and APIs”](#)), a northbound interface specifies the collection of protocols that a user can use to interact with and program the controller, while a southbound interface specifies the protocols and interfaces that the controller uses to interact with the devices it manages. Some of the features and capabilities of the Cisco APIC are as follows:

- Application-centric network policy for physical, virtual, and cloud infrastructure
- Data model-based declarative provisioning
- Designed around open standards and open APIs
- Cisco ACI fabric inventory and configuration
- Software image management
- Fault, event, and performance monitoring and management

- Integration with third-party management systems such as VMware, Microsoft, and OpenStack
- Cloud APIC appliance for Cisco cloud ACI deployments in public cloud environments

A minimum of three APICs in a cluster are needed for high availability.



The Cisco ACI fabric is built in a leaf-and-spine architecture. As the name implies, some of the Cisco Nexus switches that are part of the ACI fabric are called *leaves* and perform a function similar to that of an access switch, to which both physical and virtual endpoint servers are connected, and some of the switches are called *spines* and perform a function similar to that of a distribution switch to which all the access switches are connected. [Figure 9-1](#) provides a visual representation of how all the Cisco ACI fabric components come together.

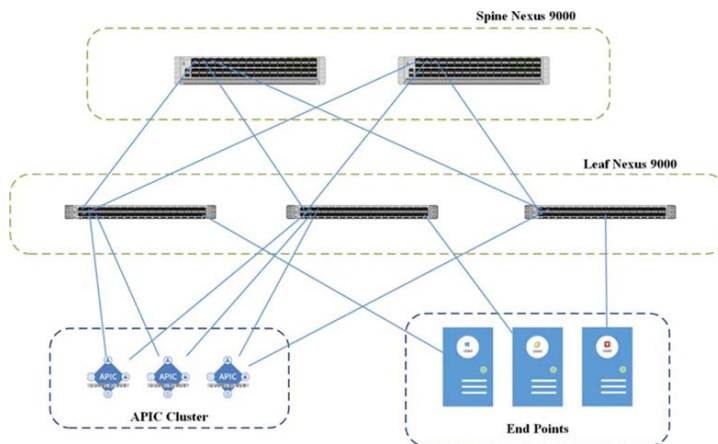


Figure 9-1 Cisco ACI Fabric Architecture

It is very important to choose the right switches for the right functions as not all Cisco Nexus 9000 switches support all functions in a leaf-and-spine architecture. The leaf switches connect to all the spine switches and to endpoint devices, including the Cisco APICs. The Cisco APICs never connect to spine switches. Spine switches

can only connect to leaf switches and are never interconnected with each other. The ACI fabric provides consistent low-latency forwarding across high-bandwidth links (40 Gbps, 100 Gbps, and 400 Gbps). Data traffic with the source and destination on the same leaf switch is handled locally. When the traffic source and destination are on separate leaf switches, they are always only one spine switch away. The whole ACI fabric operates as a single Layer 3 switch, so between a data traffic source and destination, it will always be at most one Layer 3 hop.



The configuration of the ACI fabric is stored in the APIC using an object-oriented schema. This configuration represents the logical model of the fabric. The APIC compiles the logical model and renders the policies into a concrete model that runs in the physical infrastructure. Figure 9-2 shows the relationship between the logical model, the concrete model, and the operating system running on the switches.

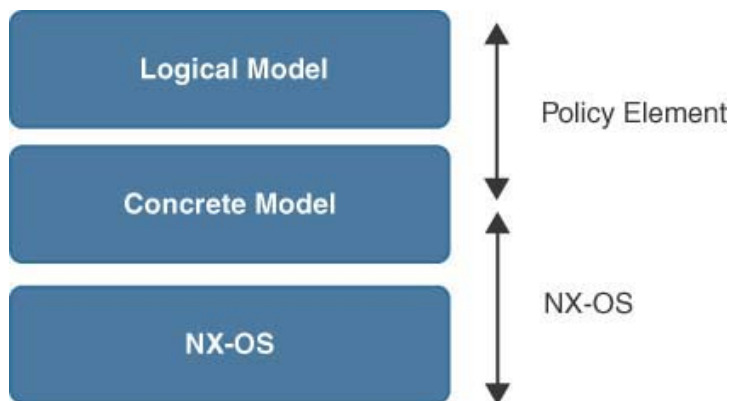


Figure 9-2 *Relationship Between the Logical Model, the Concrete Model, and the Operating System*

Each of the switches contains a complete copy of the concrete model. When a policy that represents a

configuration is created in the APIC, the controller updates the logical model. It then performs the intermediate step of creating a complete policy that it pushes into all the switches, where the concrete model is updated. The Cisco Nexus 9000 switches can only execute the concrete model when running in ACI mode. Each switch has a copy of the concrete model. If by any chance, all the APIC controllers in a cluster go offline, the fabric keeps functioning, but modifications to the fabric policies are not possible.

The ACI policy model enables the specification of application requirements. When a change is initiated to an object in the fabric, the APIC first applies that change to the policy model. This policy model change triggers a change to the concrete model and the actual managed endpoint. This management framework is called the *model-driven framework*. In this model, the system administrator defines the desired state of the fabric but leaves the implementation up to the APIC. This means that the data center infrastructure is no longer managed in isolated, individual component configurations but holistically, enabling automation and flexible workload provisioning. In this type of infrastructure, network-attached services can be easily deployed as the APIC provides an automation framework to manage the complete lifecycle of these services. As workloads move and changes happen, the controller reconfigures the underlying infrastructure to ensure that the policies are still in place for the end hosts.



The Cisco ACI fabric is composed of physical and logical components. These components are recorded in the Management Information Model (MIM) and can be represented in a hierarchical management information tree (MIT). Each node in the MIT represents a managed

object (MO). An MO can represent a concrete object, such as a switch, an adapter, a power supply, or a logical object, such as an application profile, an endpoint group, or an error message. All the components of the ACI fabric can be represented as managed objects.

Figure 9-3 provides an overview of the MIT and its elements.

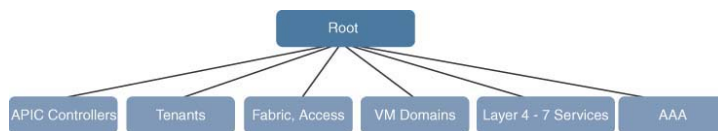


Figure 9-3 Cisco ACI Management Information Tree

The MIT hierarchical structure starts at the top with the root object and contains parent and child nodes. Each node in the tree is an MO, and each object in the fabric has a distinguished name (DN). The DN describes the object and specifies its location in the tree. The following managed objects contain the policies that control the operation of the fabric:

- **APICs:** These are the clustered fabric controllers that provide management, application, and policy deployment for the fabric.
- **Tenants:** Tenants represent containers for policies that are grouped for a specific access domain. The following four kinds of tenants are currently supported by the system:
 - **User:** User tenants are needed by the fabric administrator to cater to the needs of the fabric users.
 - **Common:** The common tenant is provided by the system and can be configured by the fabric administrator. This tenant contains policies and resources that can be shared by all tenants. Examples of such resources are firewalls, load balancers, and intrusion detection systems.
 - **Infra:** The infra tenant is provided by the system and can be configured by the fabric administrator. It contains policies that manage the operation of infrastructure resources.
 - **Management:** The management tenant is provided by the system and can be configured by the fabric administrator. This tenant contains policies and resources used for in-band and out-of-band configuration of fabric nodes.

- **Access policies:** These policies control the operation of leaf switch access ports, which provide fabric connectivity to resources such as virtual machine hypervisors, compute devices, storage devices, and so on. Several access policies come built in with the ACI fabric by default. The fabric administrator can tweak these policies or create new ones, as necessary.
- **Fabric policies:** These policies control the operation of the switch fabric ports. Configurations for time synchronization, routing protocols, and domain name resolution are managed with these policies.
- **VM domains:** Virtual machine (VM) domains group virtual machine controllers that require similar networking policy configurations. The APIC communicates with the VM controller to push network configurations all the way to the VM level.
- **Integration automation framework:** The Layer 4 to Layer 7 service integration automation framework enables a system to respond to services coming online or going offline.
- **AAA policies:** Access, authentication, and accounting (AAA) policies control user privileges, roles, and security domains for the ACI fabric.

The hierarchical policy model fits very well with the REST API interface. As the ACI fabric performs its functions, the API reads and writes to objects in the MIT. The API resources represented by URLs map directly into the distinguished names that identify objects in the MIT.

Next, let's explore the building blocks of the Cisco ACI fabric policies.



Building Blocks of Cisco ACI Fabric Policies

Tenants are top-level MOs that identify and separate administrative control, application policies, and failure domains. A tenant can represent a customer in a managed service provider environment or an organization in an enterprise environment, or a tenant can be a convenient grouping of objects and policies. A tenant's sublevel objects can be grouped into two categories: tenant networking and tenant policy. [Figure 9-4](#) provides a graphical representation of how a tenant

is organized and the main networking and policy components.

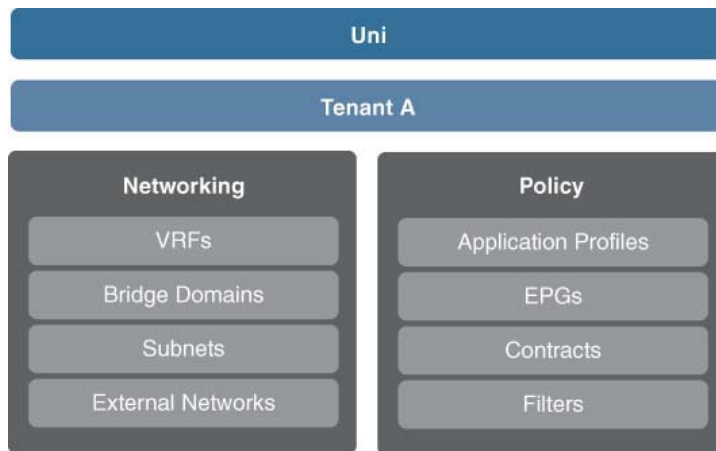


Figure 9-4 Cisco ACI Tenant Components

The tenant networking objects provide Layer 2 and Layer 3 connectivity between the endpoints and consist of the following constructs: VRF (virtual routing and forwarding) instances, bridge domains, subnets, and external networks. [Figure 9-5](#) displays in more detail how the tenant networking constructs are organized.

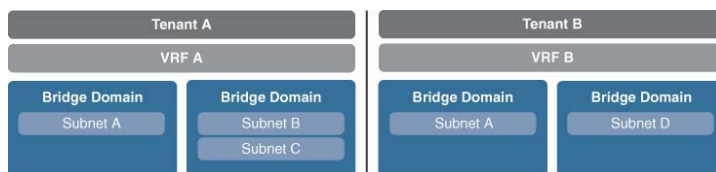


Figure 9-5 Cisco ACI Tenant Networking Components

VRF instances, also called *contexts* and *private networks*, are isolated routing tables. A VRF instance defines a Layer 3 address domain. A tenant can contain one or multiple VRF instances. VRF instances exist on any leaf switch that has a host assigned to the VRF instance. All the endpoints within a Layer 3 domain must have unique IP addresses because traffic can flow between these devices if allowed by the policy.

Bridge domains represent the Layer 2 forwarding domains within the fabric and define the unique MAC address space and flooding domain for broadcast, unknown unicast, and multicast frames. Each bridge domain is associated with only one VRF instance, but a VRF instance can be associated with multiple bridge domains. Bridge domains can contain multiple subnets, which is different from regular VLANs, which are usually associated with only one subnet each.

Subnets are the Layer 3 networks that provide IP address space and gateway services for endpoints to be able to connect to the network. Each subnet is associated with only one bridge domain. Subnets can be the following:

- **Public:** A subnet can be exported to a routed connection.
- **Private:** A subnet is confined within its tenant.
- **Shared:** A subnet can be shared and exposed in multiple VRF instances in the same tenant or across tenants as part of a shared service.

External bridged networks connect the ACI fabric to legacy Layer 2/Spanning Tree Protocol networks. This is usually needed as part of the migration process from a traditional network infrastructure to an ACI network.

External routed networks create a Layer 3 connection with a network outside the ACI fabric. Layer 3 external routed networks can be configured using static routes or routing protocols such as BGP, OSPF, and EIGRP.

The tenant policy objects are focused on the policies and services that the endpoints receive. The tenant policy consists of application profiles, endpoint groups (EPGs), contracts, and filters. [Figure 9-6](#) shows how the tenant policy objects, application profiles, and EPGs are organized in different bridge domains.

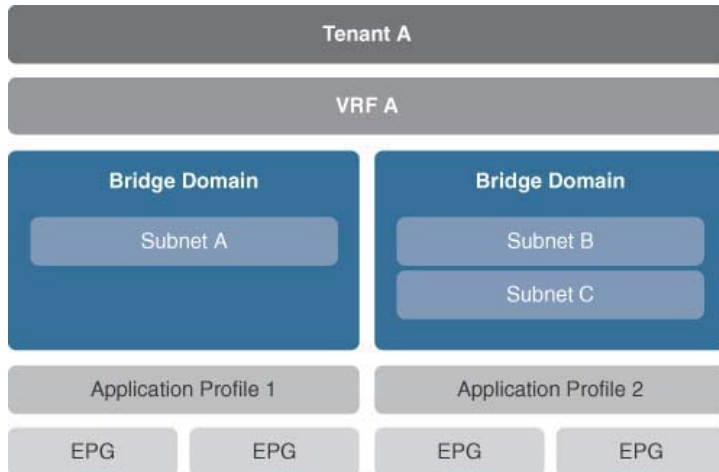


Figure 9-6 Cisco ACI Tenant Policy Components

An application profile defines the policies, services, and relationships between EPGs. An application profile contains one or more EPGs. Applications typically contain multiple components, such as a web-based front end, an application logic layer, and one or more databases in the back end. The application profile contains as many EPGs as necessary, and these EPGs are logically related to providing the capabilities of the application.



The EPG is the most important object in the policy model. An EPG is a collection of endpoints that have common policy requirements, such as security, virtual machine mobility, QoS, or Layer 4 to Layer 7 services. In the Cisco ACI fabric, each endpoint has an identity represented by its address, a location, and attributes, and it can be physical or virtual. Endpoint examples include servers, virtual machines, clients on the internet, and network-attached storage devices. Rather than configure and manage endpoints individually, you can place them in EPGs and manage them as a group. Policies apply to EPGs and never to individual endpoints. Each EPG can only be related to one bridge domain.

Contracts define the policies and services that get applied to EPGs. Contracts can be used for redirecting service to a Layer 4 to Layer 7 device, assigning QoS values, and controlling the traffic flow between EPGs. EPGs can only communicate with other EPGs based on contract rules. Contracts specify the protocols and ports allowed between EPGs. If there is no contract, inter-EPG communication is disabled by default. For intra-EPG communication, no contract is required as this traffic is always allowed by default. The relationship between an EPG and a contract can be either a consumer or a provider. EPG providers expose contracts with which a consumer EPG must comply. When an EPG consumes a contract, the endpoints in the consuming EPG can initiate communication with any endpoint from the provider EPG. [Figure 9-7](#) displays this contractual relationship between providing and consuming EPGs.

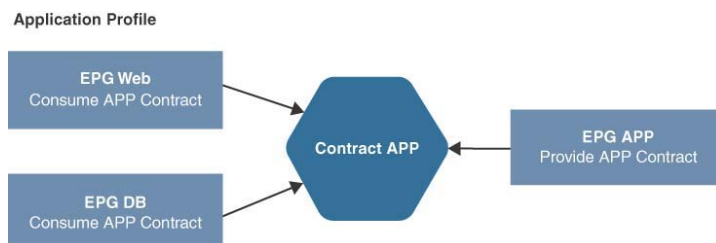


Figure 9-7 Cisco ACI Application Profiles and Contracts

Filters are the objects that define protocols and port numbers used in contracts. Filter objects can contain multiple protocols and ports, and contracts can consume multiple filters.

APIC REST API

As mentioned previously, the APIC REST API is a programmatic interface that uses the REST architecture. The API accepts and returns HTTP or HTTPS messages that contain JSON or XML documents. Any programming language can be used to generate the

messages and the JSON or XML documents that contain the API methods or managed object (MO) attributes. Whenever information is retrieved and displayed, it is read through the REST API, and whenever configuration changes are made, they are written through the REST API. The REST API also provides a way of subscribing to push-based event notification, so that when a change occurs in the MIT, an event can be sent through a web socket.



The generic APIC REST API URI looks as follows:

`https://APIC_Host:port/api/{mo|class}/{dn|class name}.{xml|json}?[options]`

Since the REST API matches one to one the MIT, defining the URI to access a certain resource is important. First, you need to define the protocol (http or https) and the hostname or IP address of the APIC instance. Next, **/api** indicates that the API is invoked. After that, the next part of the URI specifies whether the operation will be for an MO or a class. The next component defines either the fully qualified domain name for MO-based queries or the class name for class-based queries. The final mandatory part of the request is the encoding format, which can be either XML or JSON. (The APIC ignores Content-Type and other headers, so the method just explained is the only one accepted.) The complete Cisco ACI REST API documentation with information on how to use the API, all of the API endpoints, and operations available can be found at <https://developer.cisco.com/docs/aci/>.

APIC REST API username- and password-based authentication uses a special URI, including `aaaLogin`, `aaaLogout`, and `aaaRefresh` as the DN targets of a POST

operation. The payloads contain a simple XML or JSON document containing the MO representation of an aaaUser object. The following examples use the Cisco DevNet always-on APIC instance available at <https://sandboxapicdc.cisco.com> with a username value of **admin** and a password of **ciscopsdt** to show how to interact with an ACI fabric using the APIC REST API interface. Using **curl**, the authentication API call should look as shown in [Example 9-1](#).

Example 9-1 *curl Command for Cisco APIC Authentication*

[Click here to view code image](#)

```
curl -k -X POST \  
  
https://sandboxapicdc.cisco.com/api/aaaLogin.json \  
\  
-d '{  
  "aaaUser" : {  
    "attributes" : {  
      "name" : "admin",  
      "pwd" : "ciscopsdt"  
    }  
  }  
}'
```

The returned information from the APIC should look as shown in [Example 9-2](#).

Example 9-2 *Cisco APIC Authentication Response*

[Click here to view code image](#)

```
{  
  "totalCount" : "1",  
  "imdata" : [  
    {  
      "aaaLogin" : {  
        "attributes" : {  
          "remoteUser" : "false",  
          "firstLoginTime" : "1572128727",  
          "version" : "4.1(1k)",  
          "buildTime" : "Mon May 13  
16:27:03 PDT 2019",
```

```

        "siteFingerprint" :
"Z29SSG/BAVFY04Vv",
        "guiIdleTimeoutSeconds" :
"1200",
        "firstName" : "",
        "userName" : "admin",
        "refreshTimeoutSeconds" : "600",
        "restTimeoutSeconds" : "90",
        "node" : "topology/pod-1/node-
1",
        "creationTime" : "1572128727",
        "changePassword" : "no",
        "token" :
"pRgAAAAAAAAAAAAAAAAAGNPf39fZd71fV6DJwidJoqxJmHt1Fephm-
w6Q0I5byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVRfFZruCEgqsPg8+dbjb8kWX02FJLcw9Qp
sg98s5Qf0aMDQWHSyqw00bK0GxxglLeQbkgxM8/fgOAFZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA==",
        "unixUserId" : "15374",
        "lastName" : "",
        "sessionId" :
"1IZw4uthRVSmyWH/+S9aA==",
        "maximumLifetimeSeconds" :
"86400"
    }
    ...omitted output
}

```

The response to the POST operation contains an authentication token that will be used in subsequent API operations as a cookie named **APIC-cookie**.

Next, let's get a list of all the ACI fabrics that are being managed by this APIC instance. The URI for this GET operation is <https://sandboxapicdc.cisco.com/api/node/class/fabricPod.json>, and the **APIC-cookie** header, are specified for authentication purposes. The **curl** request should look similar to the one shown in [Example 9-3](#).

Example 9-3 *curl Command to Get a List of All ACI Fabrics*

[Click here to view code image](#)

```
curl -k -X GET \
https://sandboxapicdc.cisco.com/api/node/class/fabricPod.json
-H 'Cookie: APIC-
cookie=pRgAAAAAAAAAAAAAAAAAGNPf39fZd71fV6DJwidJoxJmHt1Fepmw6Q0
I5byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVRfFZruCEgqsPg8+dbjb8kWX02FJLcw9Qpsg
98s5Qf0aMDQWHSyqw00bK0GxxglLeQbkgxM8/fg0AFZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA=='
```

The response received from this instance of the APIC should look like the one in [Example 9-4](#).

Example 9-4 *curl Command Response with a List of All ACI Fabrics*

[Click here to view code image](#)

```
{
  "totalCount" : "1",
  "imdata" : [
    {
      "fabricPod" : {
        "attributes" : {
          "id" : "1",
          "monPolDn" : "uni/fabric/monfab-
default",
          "dn" : "topology/pod-1",
          "status" : "",
          "childAction" : "",
          "modTs" : "2019-10-
26T18:01:13.491+00:00",
          "podType" : "physical",
          "lcOwn" : "local"
        }
      }
    }
  ]
}
```

From this response, we can see that the always-on Cisco DevNet Sandbox APIC instance manages only one ACI

fabric, called pod-1. Next, let's find out more information about pod-1 and discover how many devices are part of this fabric. The API URI link for the resource that will return this information is

<https://sandboxapicdc.cisco.com/api/node/class/topology/pod-1/topSystem.json>. We specify again **APIC-cookie**, and the **GET** request should look like the one in [Example 9-5](#).

Example 9-5 *curl Command to Get ACI Pod Information*

[Click here to view code image](#)

```
curl -k -X GET \  
  
https://sandboxapicdc.cisco.com/api/node/class/topology/pod-1/topSystem.json \  
-H 'Cookie: APIC-  
cookie=pRgAAAAAAAAAAAAAAAAAGNPf39fZd71fV6DJwidJoqxJmHt1Fephmw6Q0  
  
I5byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVRfFZuCEgqsPg8+dbjb8kWX02FJLcw9Qpsg  
  
98s5Qf0aMDQWHSyqw00bK0GxxglLeQbkgxM8/fg0AFZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA=='
```

The redacted response from the APIC should look similar to the one shown in [Example 9-6](#).

Example 9-6 *REST API Response Containing Details About the Cisco ACI Pod*

[Click here to view code image](#)

```
{  
  "imdata" : [  
    {  
      "topSystem" : {  
        "attributes" : {  
          "role" : "controller",  
          "name" : "apic1",  
          "fabricId" : "1",  
          "inbMgmtAddr" : "192.168.11.1",  
          "oobMgmtAddr" : "10.10.20.14",  
          "systemUpTime" :
```

```
"00:04:33:38.000",
  "siteId" : "0",
  "state" : "in-service",
  "fabricDomain" : "ACI Fabric1",
  "dn" : "topology/pod-1/node-
1/sys",
  "podId" : "1"
}
},
{
  "topSystem" : {
    "attributes" : {
      "state" : "in-service",
      "siteId" : "0",
      "address" : "10.0.80.64",
      "fabricDomain" : "ACI Fabric1",
      "dn" : "topology/pod-1/node-
101/sys",
      "id" : "101",
      "podId" : "1",
      "role" : "leaf",
      "fabricId" : "1",
      "name" : "leaf-1"
    }
  }
},
{
  "topSystem" : {
    "attributes" : {
      "podId" : "1",
      "id" : "102",
      "dn" : "topology/pod-1/node-
102/sys",
      "address" : "10.0.80.66",
      "fabricDomain" : "ACI Fabric1",
      "siteId" : "0",
      "state" : "in-service",
      "role" : "leaf",
      "name" : "leaf-2",
      "fabricId" : "1"
    }
  }
},
{
  "topSystem" : {
    "attributes" : {
      "fabricId" : "1",
      "name" : "spine-1",
      "role" : "spine",
      "podId" : "1",
      "id" : "201",
      "dn" : "topology/pod-1/node-
201/sys",
```



```
        "state" : "in-service",
        "siteId" : "0",
        "fabricDomain" : "ACI Fabric1",
        "address" : "10.0.80.65"
    }
}
],
"totalCount" : "4"
}
```

From the response, we can see that this ACI fabric is made up of four devices: an APIC, two leaf switches, and one spine switch. Extensive information is returned about each device in this response, but it was modified to extract and display just a subset of that information. You are encouraged to perform the same steps and explore the APIC REST API either using the Cisco DevNet sandbox resources or your own instance of APIC.



As of this writing, there are several tools and libraries available for Cisco ACI automation. An ACI Python SDK called Cobra can be used for advanced development. For basic day-to-day configuration and monitoring tasks and for getting started with ACI automation, there is also a Python library called **acitoolkit**. The acitoolkit library exposes a subset of the APIC object model that covers the most common ACI workflows.

Next, we will use acitoolkit to build a Python script that retrieves all the endpoints from an ACI fabric. Additional information about the endpoints—such as the EPGs they are members of, application profiles that are applied to those EPGs, and tenant membership, encapsulation, MAC, and IP addresses—will be displayed for each endpoint. A Python script that uses acitoolkit and accomplishes these tasks might look as the one shown in [Example 9-7](#).

Example 9-7 acitoolkit Example

[Click here to view code image](#)

```
#!/usr/bin/env python
import sys
import acitoolkit.acitoolkit as aci

APIC_URL = 'https://sandboxapicdc.cisco.com'
USERNAME = 'admin'
PASSWORD = 'ciscopsdt'

# Login to APIC
SESSION = aci.Session(APIC_URL, USERNAME,
                      PASSWORD)
RESP = SESSION.login()
if not RESP.ok:
    print('Could not login to APIC')
    sys.exit()

ENDPOINTS = aci.Endpoint.get(SESSION)
print('{0:19s}{1:14s}{2:10s}{3:8s}{4:17s}'
      '{5:10s}'.format(
        "MAC ADDRESS",
        "IP ADDRESS",
        "ENCAP",
        "TENANT",
        "APP PROFILE",
        "EPG"))
print('-'*80)

for EP in ENDPOINTS:
    epg = EP.get_parent()
    app_profile = epg.get_parent()
    tenant = app_profile.get_parent()
    print('{0:19s}{1:14s}{2:10s}{3:8s}{4:17s}'
          '{5:10s}'.format(
            EP.mac,
            EP.ip,
            EP.encap,
            tenant.name,
            app_profile.name,
            epg.name))
```

The latest version of acitoolkit can be found at <https://github.com/datacenter/acitoolkit>. Follow the steps at this link to install acitoolkit. The acitoolkit library supports Python 3, and version 0.4 of the library

is used in [Example 9-7](#). The script was tested successfully with Python 3.7.4.

First, the `acitoolkit` library is imported; it will be referenced in the script using the short name `aci`. Three variables are defined next: `APIC_URL` contains the URL for the APIC instance that will be queried (in this case, the Cisco DevNet always-on APIC sandbox), and `USERNAME` and `PASSWORD` contain the login credentials for the APIC instance. Next, using the `Session` method of the `aci` class, a connection is established with the APIC. The `Session` method takes as input the three variables defined previously: the `APIC_URL`, `USERNAME`, and `PASSWORD`. Next, the success of the login action is verified. If the response to the login action is not okay, a message is displayed to the console (“Could not login to APIC”), and the script execution ends. If the login was successful, all the endpoints in the ACI fabric instance are stored in the `ENDPOINTS` variable. This is done by using the `get` method of the `aci.Endpoint` class and passing in the current session object. Next, the headers of the table—with the information that will be extracted—are displayed to the console. As mentioned previously, the MAC address, the IP address, the encapsulation, the tenant, the application profile, and the EPG will be retrieved from the APIC for all the endpoints in the fabric. The `for` iterative loop will go over one endpoint at a time and, using the `get_parent()` method, will go one level up in the MIT and retrieve the parent MO of the endpoint, which is the EPG. Recall that all endpoints in an ACI fabric are organized in EPGs. The parent object for an endpoint is hence the EPG of which that endpoint is a member. Going one level up in the MIT, the parent object of the EPG is the application profile, and going one more level up, the parent object of the application profile is the tenant object. The `epg`, `app_profile`, and `tenant` variables contain the respective EPG, application profile, and tenant values for each endpoint in the fabric.

The last line of code in the script displays to the console the required information for each endpoint. The output of the script should look similar to the output shown in [Figure 9-8](#).

MAC ADDRESS	IP ADDRESS	ENCAP	TENANT	APP PROFILE	EPG
44:CD:BB:C0:00:00	10.193.101.3	vlan-201	Heroes	Save_The_Planet	app
45:CD:BB:C0:00:00	2222:66:3	vlan-201	Heroes	Save_The_Planet	app
45:CD:BB:C0:00:00	10.193.102.3	vlan-200	Heroes	Save_The_Planet	web
45:CD:BB:C0:00:00	2222:66:3	vlan-202	Heroes	Save_The_Planet	db
44:CD:BB:C0:00:00	2222:65:3	vlan-200	Heroes	Save_The_Planet	web
44:CD:BB:C0:00:00	10.193.101.3	vlan-202	Heroes	Save_The_Planet	db
46:CD:BB:C0:00:00	10.193.102.4	vlan-201	Heroes	Save_The_Planet	app
46:CD:BB:C0:00:00	2222:66:4	vlan-200	Heroes	Save_The_Planet	web
46:CD:BB:C0:00:00	2222:66:4	vlan-202	Heroes	Save_The_Planet	db
42:CD:BB:C0:00:00	10.193.101.1	vlan-126	SnV	Chaos	Database
43:CD:BB:C0:00:00	2222:66:1	vlan-126	SnV	Chaos	Database
42:CD:BB:C0:00:00	2222:65:1	vlan-125	SnV	Chaos	Web
43:CD:BB:C0:00:00	10.193.102.1	vlan-125	SnV	Chaos	Web
43:CD:BB:C0:00:00	10.193.102.1	vlan-128	SnV	Power_Up	Database
43:CD:BB:C0:00:00	2222:66:1	vlan-121	SnV	Evolution_X	Web
43:CD:BB:C0:00:00	2222:66:1	vlan-127	SnV	Power_Up	Web
42:CD:BB:C0:00:00	10.193.101.1	vlan-121	SnV	Evolution_X	Web
42:CD:BB:C0:00:00	2222:65:1	vlan-124	SnV	Rescue	Database
43:CD:BB:C0:00:00	10.193.102.1	vlan-123	SnV	Rescue	Web
42:CD:BB:C0:00:00	2222:65:1	vlan-122	SnV	Evolution_X	Database
43:CD:BB:C0:00:00	10.193.102.1	vlan-124	SnV	Rescue	Database
42:CD:BB:C0:00:00	2222:65:1	vlan-127	SnV	Power_Up	Web
42:CD:BB:C0:00:00	10.193.101.1	vlan-123	SnV	Rescue	Web
43:CD:BB:C0:00:00	10.193.102.1	vlan-122	SnV	Evolution_X	Database
42:CD:BB:C0:00:00	10.193.101.1	vlan-128	SnV	Power_Up	Database
44:CD:BB:C0:00:00	2222:66:2	vlan-125	SnV	Chaos	Web
44:CD:BB:C0:00:00	2222:66:2	vlan-127	SnV	Power_Up	Web
44:CD:BB:C0:00:00	10.193.102.2	vlan-122	SnV	Evolution_X	Database
44:CD:BB:C0:00:00	2222:66:2	vlan-123	SnV	Rescue	Web
44:CD:BB:C0:00:00	2222:66:2	vlan-128	SnV	Power_Up	Database
44:CD:BB:C0:00:00	10.193.102.2	vlan-124	SnV	Rescue	Database
44:CD:BB:C0:00:00	2222:66:2	vlan-126	SnV	Chaos	Database
44:CD:BB:C0:00:00	10.193.102.2	vlan-121	SnV	Evolution_X	Web

Figure 9-8 Output of the Python Script from Example 9-7

UCS MANAGER

Cisco Unified Computing System (UCS) encompasses most of the Cisco compute products. The first UCS products were released in 2009, and they quickly established themselves as leaders in the data center compute and server market. Cisco UCS provides a unified server solution that brings together compute, storage, and networking into one system. While initially the UCS solution took advantage of network-attached storage (NAS) or storage area networks (SANs) in order to support requirements for large data stores, with the release of Cisco HyperFlex and hyperconverged servers, large storage data stores are now included with the UCS solution.

Cisco UCS B-series blade servers, C-series rack servers, S-series storage servers, UCS Mini, and Cisco HyperFlex hyperconverged servers can all be managed through one interface: UCS Manager. UCS Manager provides unified, embedded management of all software and hardware components of Cisco UCS. Cisco UCS Manager software runs on a pair of hardware appliances called *fabric interconnects*. The two fabric interconnects form an active/standby cluster that provides high availability. The UCS infrastructure that is being managed by UCS Manager forms a UCS fabric that can include up to 160 servers. The system can scale to thousands of servers by integrating individual UCS Manager instances with Cisco UCS Central in a multidomain Cisco UCS environment.



UCS Manager participates in the complete server lifecycle, including server provisioning, device discovery, inventory, configuration, diagnostics, monitoring, fault detection, and auditing and statistics collection. All infrastructure that is being managed by UCS Manager is either directly connected to the fabric interconnects or connected through fabric extenders. Fabric extenders, as the name implies, have the function of offering additional scalability in connecting servers back to the fabric interconnects. They are zero-management, low-cost, and low-power devices that eliminate the need for expensive top-of-rack Ethernet and Fibre Channel switches. [Figure 9-9](#) shows how all these components connect to each other.

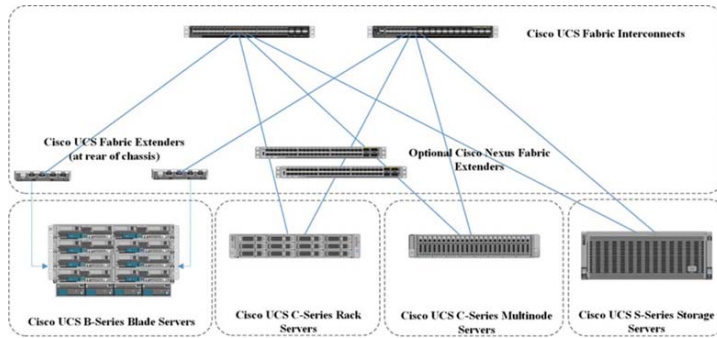


Figure 9-9 *Cisco Unified Computing System Connectivity*

All Cisco UCS servers support Cisco SingleConnect technology. Cisco SingleConnect is a revolutionary technology that supports all traffic from the servers (LAN, SAN, management, and so on) over a single physical link. The savings that this technology brings only as part of the cabling simplification is orders of magnitude higher than competing products.

Cisco UCS Manager provides an HTML 5 graphical user interface (GUI), a command-line interface (CLI), and a comprehensive API. All Cisco UCS fabric functions and managed objects are available over the UCS API. Developers can take advantage of the extensive API and can enhance the UCS platform according to their unique requirements. Tools and software integrations with solutions from third-party vendors like VMware, Microsoft, and Splunk are already publicly available. We will briefly see later in this chapter how the Cisco UCS PowerTool for UCS Manager and the Python software development kit (SDK) can be used to automate and programmatically manage Cisco UCS Manager.

With Cisco UCS Manager, the data center servers can be managed using an infrastructure-as-code framework. This is possible through another innovation that is included with the Cisco UCS solution: the service profile. The service profile is a logical construct in UCS Manager that contains the complete configuration of a physical

server. All the elements of a server configuration—including RAID levels, BIOS settings, firmware revisions and settings, adapter settings, network and storage settings, and data center connectivity—are included in the service profile. When a service profile is associated with a server, Cisco UCS Manager automatically configures the server, adapters, fabric extenders, and fabric interconnects to match the configuration specified in the service profile. With service profiles, infrastructure can be provisioned in minutes instead of days. With service profiles, you can even pre-provision servers and have their configurations ready before the servers are even connected to the network. Once the servers come online and get discovered by UCS Manager, the service profiles can be automatically deployed to the server.



The UCS Manager programmatic interface is the XML API. The Cisco UCS Manager XML API accepts XML documents over HTTP or HTTPS connections. Much as with Cisco ACI, the configuration and state information for Cisco UCS is stored in a hierarchical tree structure known as the management information tree (MIT). The MIT, which contains all the managed objects in the Cisco UCS system, is accessible through the XML API. Any programming language can be used to generate XML documents that contain the API methods. One or more managed objects can be changed with one API call. When multiple objects are being configured, the API operation stops if any of the MOs cannot be configured, and a full rollback to the state of the system before the change was initiated is done. API operations are transactional and are done on the single data model that represents the whole system. Cisco UCS is responsible for all endpoint communications, making UCS Manager the single source of truth. Users cannot communicate directly with the endpoints, relieving developers from

administering isolated, individual component configurations. All XML requests are asynchronous and terminate on the active Cisco UCS Manager.

All the physical and logical components that make up Cisco UCS are represented in a hierarchical management information tree (MIT), also known as the Management Information Model (MIM). Each node in the tree represents a managed object (MO) or a group of objects that contains its administrative and operational states. At the top of the hierarchical structure is the **sys** object, which contains all the parent and child nodes in the tree. Each object in Cisco UCS has a unique distinguished name that describes the object and its place in the tree. The information model is centrally stored and managed by a process running on the fabric interconnects that is called the Data Management Engine (DME). When an administrative change is initiated to a Cisco UCS component, the DME first applies that change to the information model and then applies the change to the actual managed endpoint. This approach is referred to as a *model-driven framework*.

A specific managed object in the MIT can be identified by its distinguished name (DN) or by its relative name (RN). The DN specifies the exact managed object on which the API call is operating and consists of a series of relative names:

```
DN = {RN} / {RN} / {RN} / {RN} . . .
```

A relative name identifies an object in the context of its parent object.

The Cisco UCS Manager XML API model includes the following programmatic entities:

- **Classes:** Classes define the properties and states of objects in the MIT.
- **Methods:** Methods define the actions that the API performs on one or more objects.

- **Types:** Types are object properties that map values to the object state.

Several types of methods are available with the XML API:

- **Authentication methods:** These methods, which include the following, are used to authenticate and maintain a session:
 - **aaaLogin:** Login method
 - **aaaRefresh:** Refreshes the authentication cookie
 - **aaaLogout:** Exits the session and deactivates the corresponding authentication cookie
- **Query methods:** These methods, which include the following, are used to obtain information on the current configuration state of an object:
 - **configResolveDn:** Retrieves objects by DN
 - **configResolveClass:** Retrieves objects of a given class
 - **configResolveParent:** Retrieves the parent object of an object
- **Configuration methods:** These methods, which include the following, are used to make configuration changes to managed objects:
 - **configConfMo:** Affects a single MO
 - **configConfMos:** Affects multiple subtrees

Since the query methods available with the XML API can return large sets of data, filters are supported to limit this output to subsets of information. Four types of filters are available:

- **Simple filters:** These true/false filters limit the result set of objects with the Boolean value of True or False.
- **Property filters:** These filters use the values of an object's properties as the inclusion criteria in a result set (for example, equal filter, not equal filter, greater than filter)
- **Composite filters:** These filters are composed of two or more component filters (for example, AND filter, OR filter)
- **Modifier filter:** This filter changes the results of a contained filter. Currently only the NOT filter is supported. This filter negates the result of a contained filter.

External applications can get Cisco UCS Manager state change information either by regular polling or by event subscription. Full event subscription is supported with the API and is the preferred method of notification.

Polling usually consumes a lot of resources and should be used only in limited situations.

Cisco UCS Manager provides a managed object browser called Visore. Visore can be accessed by navigating to <https://<UCS-Manager-IP>/visore.html>. The web interface looks as shown in [Figure 9-10](#).

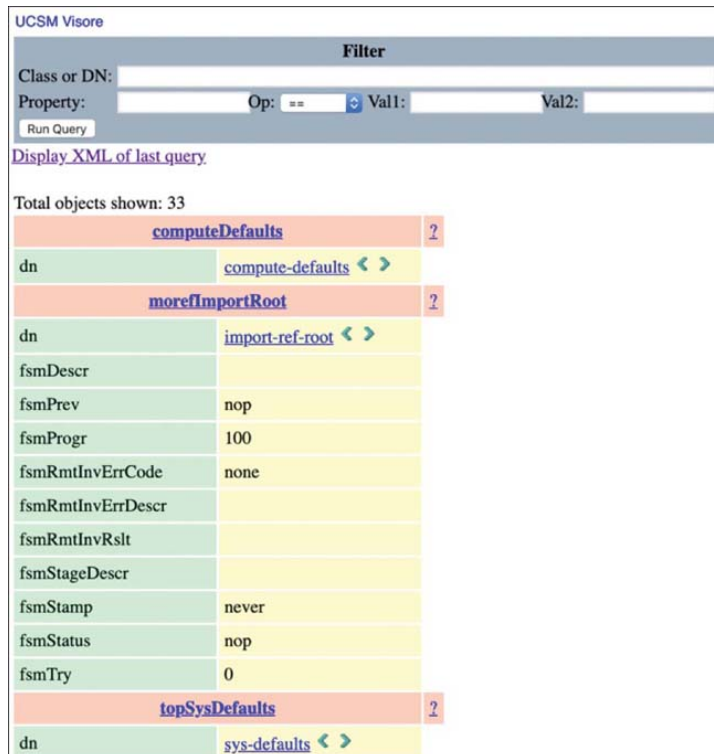


Figure 9-10 Cisco UCS Manager Visore Interface

The whole MIT tree can be explored, and also queries for specific DNs can be run from this interface. Additional developer resources regarding Cisco UCS Manager can be found on the Cisco DevNet website, at <https://developer.cisco.com/site/ucs-dev-center/>.

Next, let's explore the Cisco UCS Manager XML API. The complete documentation of the Cisco UCS Manager information model for different releases can be found at <https://developer.cisco.com/site/ucs-mim-ref-api-picker/>. At this site, you can find all the managed objects,

all the methods, all the types, all the fault and FSM rules, and extensive documentation for each of them.



In order for data center administrators and developers to become more familiar with the Cisco UCS system, Cisco has released a software emulator. Cisco UCS Platform Emulator is the Cisco UCS Manager application bundled into a virtual machine (VM). The VM includes software that emulates hardware communications for the Cisco UCS system. The Cisco UCS Platform Emulator can be used to create and test a supported Cisco UCS configuration or to duplicate an existing Cisco UCS environment for troubleshooting and development purposes. The Cisco UCS Platform Emulator is delivered as an .ova file and can run in nearly any virtual environment. The complete Cisco UCS Manager information model documentation is also bundled within the UCS Platform Emulator.

As usual, the Cisco DevNet team makes available to the larger DevNet community a series of sandboxes for easier product discovery and development. So far in this chapter, we have used always-on sandboxes. In this example, we will use a reservable sandbox. As the name suggests, reservable sandboxes can be reserved up to 7 days and are available only to the person who makes the reservation. At this writing, there is a Cisco UCS Manager sandbox that can be used to explore the XML API. It is called **UCS Management** and can be found at <https://developer.cisco.com/sandbox>. This sandbox takes advantage of the Cisco UCS Platform Emulator version 3.2(2.5).

At this point, to authenticate and get an authentication cookie, we can use the **curl** command as follows:

[Click here to view code image](#)

```
curl -k -X POST https://10.10.20.110/nuova \  
  -H 'Content-Type: application/xml' \  
  -d '<aaaLogin inName="ucspe" inPassword="ucspe">  
</aaaLogin>'
```

The IP address of the Cisco UCS Manager is **10.10.20.110**, the XML API resource is **/nuova**, and the authentication method used is **aaaLogin**. The username and password are passed in an XML document within the **inName** and **inPassword** variables. In this case, both the username and password are **ucspe**. The **Content-Type** header specifies the type of data that the **POST** call will send to the XML API (which is, of course, **XML** in this case).

The response should be similar to the following one:

[Click here to view code image](#)

```
<aaaLogin cookie="" response="yes"  
  outCookie="1573019916/7c901636-  
  c461-487e-bbd0-c74cd68c27be"  
  outRefreshPeriod="600"  
  outPriv="aaa,admin,ext-lan-config,ext-lan-  
  policy,ext-lan-  
  qos,ext-lan-security,ext-san-config,ext-san-  
  policy,ext-san-  
  security,fault,operations,pod-config,pod-  
  policy,pod-qos,pod-  
  security,read-only" outDomains="org-root"  
  outChannel="noencssl"  
  outEvtChannel="noencssl" outSessionId=""  
  outVersion="3.2(2.5)"  
  outName="" />
```

aaaLogin specifies the method used to log in, the **"yes"** value confirms that this is a response, **outCookie** provides the session cookie, **outRefreshPeriod** specifies the recommended cookie refresh period (where the default is 600 seconds), and the **outPriv** value specifies the privilege level associated with the account.

Next, let's get a list of all the objects that are part of the compute class and are being managed by this instance of Cisco UCS Manager. In order to accomplish this, we can

use the **configFindDnsByClassId** method. This method finds distinguished names and returns them sorted by class ID. The **curl** command should look similar to the following one:

[Click here to view code image](#)

```
curl -k -X POST https://10.10.20.110/nuova \  
-H 'Content-Type: application/xml' \  
-d '<configFindDnsByClassId  
  classId="computeItem"  
  cookie="1573019916/7c901636-c461-487e-bbd0-  
c74cd68c27be" />'
```

The XML API endpoint, <https://10.10.20.110/nuova>, and the **Content-Type** header, **application/xml**, stay the same. The XML data that is being sent to the Cisco UCS Manager server is different. First, the **configFindDnsByClassId** method is specified, and then the two mandatory variables for **classId** and the **cookie** are passed in. The **classId** specifies the object class that in this case is the **computeItem** class, and the **cookie** is being populated with the value of the authentication cookie obtained previously.

The response in this case, as shown in [Example 9-8](#), contains a complete list of all the compute items that are being managed by the 10.10.20.110 instance of Cisco UCS Manager.

Example 9-8 *List of Compute Items That Are Being Managed by Cisco UCS Manager*

[Click here to view code image](#)

```
<configFindDnsByClassId  
cookie="1573019916/7c901636-c461-487e-bbd0-  
c74cd68c27be"  
response="yes" classId="computeItem">  
  <outDns>  
    <dn value="sys/chassis-  
4/blade-8"/>
```

```

5/blade-8"/>
6/blade-8"/>
6/blade-1"/>
3/blade-1"/>
... omitted output
<dn value="sys/rack-unit-9"/>
<dn value="sys/rack-unit-8"/>
<dn value="sys/rack-unit-7"/>
<dn value="sys/rack-unit-6"/>
<dn value="sys/rack-unit-5"/>
<dn value="sys/rack-unit-4"/>
<dn value="sys/rack-unit-3"/>
<dn value="sys/rack-unit-2"/>
<dn value="sys/rack-unit-1"/>
</outDns>
</configFindDnsByClassId>

```

In our exploration of the Cisco UCS Manager XML API, let's now get more information about a specific compute object. The method to retrieve a single managed object for a specified DN is **configResolveDn**. The **curl** command for this API request should look as shown in [Example 9-9](#).

Example 9-9 *Using a curl Command to Retrieve Information About a Compute Object*

[Click here to view code image](#)

```

curl -k -X POST https://10.10.20.110/nuova \
-H 'Content-Type: application/xml' \
-d '<configResolveDn
  cookie="1573019916/7c901636-c461-487e-bbd0-
  c74cd68c27be"
  dn="sys/chassis-4/blade-8" />'

```

Much as in the previous call, the API endpoint and **Content-Type** header stay the same. The XML data that is being sent with the request contains the method, **configResolveDn**, the authentication cookie, and the DN for which additional information is requested, which

in this case is the blade number 8 in chassis number 4:
sys/chassis-4/blade-8.

The response contains extensive information about the blade server in slot 8 from the chassis with number 4, as shown in [Example 9-10](#).

Example 9-10 *curl Command Response Containing Information About a Compute Object*

[Click here to view code image](#)

```
<configResolveDn dn="sys/chassis-4/blade-8"
cookie="1573019916/7c901636-c461-487e-bbd0-
c74cd68c27be" response="yes">
  <outConfig>
    <computeBlade
adminPower="policy" adminState="in-service"
assetTag=""
      assignedToDn=""
      association="none"
availability="available"
availableMemory="49152"
      chassisId="4"
      checkPoint="discovered"
connPath="A,B" connStatus="A,B" descr=""
      discovery="complete"
      discoveryStatus=""
dn="sys/chassis-4/blade-8" fltAggr="0"
fsmDescr=""
      fsmFlags=""
      fsmPrev="DiscoverSuccess"
fsmProgr="100" fsmRmtInvErrCode="none"
      fsmRmtInvErrDescr=""
      fsmRmtInvRslt=""
fsmStageDescr="" fsmStamp="2019-11-
06T04:02:03.896"
      fsmStatus="nop"
      fsmTry="0" intId="64508"
kmipFault="no" kmipFaultDescription=""
      lc="undiscovered"
      lcTs="1970-01-01T00:00:00.000"
localId="" lowVoltageMemory="not-applicable"
      managingInst="A"
memorySpeed="not-applicable" mfgTime="not-
applicable"
      model="UCSB-
B200-M4" name=""
numOf40GAdaptorsWithOldFw="0"

numOf40GAdaptorsWithUnknownFw="0"
```

```

        numOfAdaptors="1"
numOfCores="8" numOfCoresEnabled="8"
numOfCpus="2"
        numOfEthHostIfs="0"
numOfFcHostIfs="0" numOfThreads="16"
operPower="off"
        operPwrTransSrc="unknown"
operQualifier="" operSolutionStackType="none"
        operState="unassociated"
operability="operable"
        originalUuid="1b4e28ba-2fa1-
11d2-
        0408-b9a761bde3fb"
partNumber="" policyLevel="0"
policyOwner="local"
        presence="equipped"
revision="0" scaledMode="none" serial="SRV137"
        serverId="4/8"
        slotId="8" totalMemory="49152"
usrLbl=""
        uuid="1b4e28ba-2fa1-11d2-0408-
b9a761bde3fb"
        vendor="Cisco Systems Inc"
vid=""/>
    </outConfig>
</configResolveDn>

```

While interacting with the Cisco UCS Manager XML API this way is possible, you can see that it becomes cumbersome very quickly. The preferred way of working with the XML API is either through Cisco UCS PowerTool suite or the Cisco UCS Python SDK.

The Cisco UCS PowerTool suite is a PowerShell module that helps automate all aspects of Cisco UCS Manager. The PowerTool cmdlets work on the Cisco UCS MIT. The cmdlets can be used to execute read, create, modify, and delete operations on all the managed objects in the MIT. The Cisco UCS PowerTool suite enables easy integration with existing IT management processes and tools. The PowerTool suite can be downloaded for Windows via PS Gallery and for Linux from <https://community.cisco.com/t5/cisco-developed-ucs-integrations/cisco-ucs-powertool-core-suite-for-powershell-core-modules-for/ta-p/3985798>.



Cisco UCS Python SDK is a Python module that helps automate all aspects of Cisco UCS management, including server, network, storage, and hypervisor management. The Cisco UCS Python SDK works on the Cisco UCS Manager MIT, performing create, read, modify, or delete actions on the managed objects in the tree. Python versions 2.7 and higher and version 3.5 and higher are supported. The Cisco UCS Python module for UCS Manager is called **ucsmsdk** and can be installed using pip by issuing the following command at the command prompt: **pip install ucsmsdk**. As of this writing, the current version of the ucsmsdk module is 0.9.8.

The Cisco UCS Python SDK provides a utility called `convert_to_ucs_python` that gives administrators and developers the option of recording all the interactions with the Cisco UCS Manager GUI and saving them into an XML file. Running this XML file through the `convert_to_ucs_python` tool automatically generates Python code corresponding to the actions that were performed in the GUI. Using this process, data center automation efforts can be sped up orders of magnitude, and simple tasks such as creating a new VLAN or complex tasks such as configuring a service policy template can be automated within seconds.

Next, let's explore the Cisco UCS Python SDK and see how to connect to a Cisco UCS Manager instance, retrieve a list of all the compute blades in the system, and extract specific information from the returned data. The sample Python code is built in Python 3.7.4 using version 0.9.8 of the ucsmsdk module.

First, the **UcsHandle** class is imported. An instance of this class is used to connect to Cisco UCS Manager. The

Cisco UCS Manager IP address, username, and password are passed in as parameters to the instance of the **UcsHandle** class that is called **HANDLE**. Several methods are available with the **UcsHandle** class. In this script only three are used:

- **HANDLE.login()**: This method is used to log in to Cisco UCS Manager.
- **HANDLE.query_classid()**: This method is used to query the MIT for objects with a specific class ID.
- **HANDLE.logout()**: This method is used to log out from the Cisco UCS Manager.

The **BLADES** variable contains a dictionary of all the compute blades that are being managed by the 10.10.20.110 instance of Cisco UCS Manager. Within a **for** loop, specific information regarding the DN, serial number, administrative state, model number, and total amount of memory for each blade is extracted and displayed to the console. The Python script using the Cisco UCS Manager SDK that accomplishes all of these tasks looks as shown in [Example 9-11](#).

Example 9-11 *ucsmsdk Python Example*

[Click here to view code image](#)

```
#!/usr/bin/env python
from ucsmSdk.ucshandle import UcsHandle
HANDLE = UcsHandle("10.10.20.110", "ucspe",
                  "ucspe")

# Login into Cisco UCS Manager
HANDLE.login()

# Retrieve all the compute blades
BLADES = HANDLE.query_classid("ComputeBlade")

print('{0:23s}{1:8s}{2:12s}{3:14s}
      {4:6s}'.format(
    "DN",
    "SERIAL",
    "ADMIN STATE",
    "MODEL",
    "TOTAL MEMORY"))
print('-'*70)
```

```

# Extract DN, serial number, admin state,
# model, and total memory for each blade
for BLADE in BLADES:
    print('{0:23s}{1:8s}{2:12s}{3:14s}
{4:6s}'.format(
        BLADE.dn,
        BLADE.serial,
        BLADE.admin_state,
        BLADE.model,
        BLADE.total_memory))

HANDLE.logout()

```

The results of running this script look as shown in [Figure 9-11](#).

DN	SERIAL	ADMIN STATE	MODEL	TOTAL MEMORY
sys/chassis-3/blade-1	SRV126	in-service	UCSB-EX-M4-1	49152
sys/chassis-3/blade-3	SRV127	in-service	UCSB-EX-M4-1	49152
sys/chassis-3/blade-5	SRV128	in-service	UCSB-EX-M4-1	49152
sys/chassis-3/blade-7	SRV129	in-service	UCSB-EX-M4-1	49152
sys/chassis-4/blade-1	SRV130	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-2	SRV131	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-3	SRV132	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-4	SRV133	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-5	SRV134	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-6	SRV135	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-7	SRV136	in-service	UCSB-B200-M4	49152
sys/chassis-4/blade-8	SRV137	in-service	UCSB-B200-M4	49152
sys/chassis-5/blade-1	SRV138	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-2	SRV139	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-3	SRV140	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-4	SRV141	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-5	SRV142	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-6	SRV143	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-7	SRV144	in-service	UCSB-B200-M5	49152
sys/chassis-5/blade-8	SRV145	in-service	UCSB-B200-M5	49152
sys/chassis-6/blade-1	SRV146	in-service	UCSB-B200-M4	49152
sys/chassis-6/blade-2	SRV147	in-service	UCSB-B200-M4	49152
sys/chassis-6/blade-3	SRV148	in-service	UCSB-B200-M4	49152
sys/chassis-6/blade-4	SRV149	in-service	UCSB-B200-M4	49152
sys/chassis-6/blade-5	SRV150	in-service	UCSB-B200-M4	49152
sys/chassis-6/blade-6	SRV151	in-service	UCSB-B200-M4	49152

Figure 9-11 Output of the Python Script from Example 9-11

CISCO UCS DIRECTOR

Automation delivers the essential scale, speed, and repeatable accuracy needed to increase productivity and respond quickly to business requirements in a data center environment. Cisco UCS Director replaces manual configuration and provisioning processes with

orchestration in order to optimize and simplify delivery of data center resources.

This open private-cloud platform delivers on-premises infrastructure as a service (IaaS) from the core to the edge of the data center. Automated workflows configure, deploy, and manage infrastructure resources across Cisco and third-party computing, network, and storage resources and converged and hyperconverged infrastructure solutions. Cisco UCS Director supports the industry's leading converged infrastructure solutions, including NetApp FlexFod and FlexPod Express, EMC VSPEX, EMC VPLEX, and VCE Block. It delivers unified management and orchestration for a variety of hypervisors across bare-metal and virtualized environments.

A self-service portal, a modern service catalog, and more than 2500 multivendor tasks enable on-demand access to integrated services across data center resources. Cisco UCS Director allows IT professionals and development teams to order and manage infrastructure services on demand.

Cisco UCS Director is supported by a broad ecosystem. Third-party hardware and solution vendors can use the southbound APIs and the SDKs provided with them to develop integrations into the Cisco UCS Director management model. Northbound APIs can be used by DevOps and IT operations management tools to interact with Cisco UCS Director and perform all the functions provided by the solution in a programmable and automated fashion.



Cisco UCS Director provides comprehensive visibility and management of data center infrastructure

components. From a data center management perspective, the following are some of the tasks that can be performed using Cisco UCS Director:

- Create, clone, and deploy service profiles and templates for all Cisco UCS servers and compute applications.
- Manage, monitor, and report on data center components such as Cisco UCS domains or Cisco Nexus devices.
- Monitor usage, trends, and capacity across a converged infrastructure on a continuous basis.
- Deploy and add capacity to converged infrastructures in a consistent, repeatable manner.

Cisco UCS Director also enables the creation of workflows that provide automation services. These automation workflows can be published and made available to the end users of the data center resources through on-demand portals. Once built and validated, these workflows perform the same way every time, no matter who triggers them. A data center administrator can run them, or role-based access control can be implemented to enable users and customers to run these workflows on a self-service basis. From an infrastructure automation perspective, some of the use cases that Cisco UCS Director can help automate include the following:

- Virtual machine provisioning and lifecycle management.
- Compute, network, and storage resources configuration and lifecycle management.
- Bare-metal server provisioning, including operating system installation.

Cisco UCS Director supports Cisco ACI by offering automation workflows that orchestrate the APIC configuration and management tasks. It also supports multitenancy and the ability to define contracts between different container tiers.

Cisco UCS Director can be managed using Cisco Intersight, which is covered later in this chapter. Cisco UCS Director is a 64-bit appliance that uses the standard templates Open Virtualization Format (OVF) for

VMware vSphere and Virtual Hard Disk (VHD) for Microsoft Hyper-V and can be downloaded from www.cisco.com.



Next, let's go over some essential concepts needed to understand how the Cisco UCS Director orchestrator works. First, there is the concept of a task. A *task* is an atomic unit of work in Cisco UCS Director; it cannot be decomposed into smaller actions and represents a single action with inputs and outputs. Cisco UCS Director has a task library that contains hundreds of predefined tasks, such as an SSH command task (executing a command in a Secure Shell session), an inventory collection task (gathering information about available devices), a new VM provisioning task (creating a new virtual machine), and many more. In the event that there is no suitable predefined task, the system offers the option of creating custom tasks, as described later in this section.

The second concept is the workflow. A *workflow* is a series of tasks arranged to automate a complex operation. The simplest workflow contains a single task, but workflows can contain any number of tasks. Workflows are at the heart of Cisco UCS Director orchestration. They automate processes of any level of complexity. Workflows are built using the Workflow Designer, which is a drag-and-drop interface. In Workflow Designer, the tasks are arranged in sequence and define inputs and outputs to those tasks. Loops and conditionals can be implemented using flow of control tasks. Every time a workflow is executed, a service request is generated. Workflows can be scheduled for later execution, and Cisco UCS Director stores details of completed service requests. A service request can have one of several states, depending on its execution status: scheduled, running, blocked, completed, or failed.

Finally, libraries and catalogs are collections of predefined tasks and workflows that can be used as building blocks for more advanced workflows.



Let's now explore the programmability and extensibility of Cisco UCS Director. The Cisco UCS Director SDK is a collection of technologies that enable developers to extend the capabilities of Cisco UCS Director, access Cisco UCS Director data, and invoke Cisco UCS Director's automation and orchestration operations from any application. The Cisco UCS Director SDK includes the Open Automation component. Scripting technologies include the Cisco UCS Director PowerShell API, custom tasks bundled in Cisco UCS Director script modules, and the ability to write custom tasks using Cloupiascript, a server-side JavaScript implementation.

The Cisco UCS Director SDK makes the following possible:

- Accessing Cisco UCS Director programmatically by using the Cisco UCS Director REST API
- Customizing Cisco UCS Director by creating custom workflow tasks
- Extending Cisco UCS Director by using Cisco UCS Director Open Automation to build connectors that support additional devices and systems

Cisco UCS Director provides the Cisco UCS Director Open Automation module to enable developers to enhance the functionality of the Cisco UCS Director appliance. Open Automation can be used to add modules to Cisco UCS Director. A module is the topmost logical entry point into Cisco UCS Director. In order to add or extend the functionality of the system, a new module must be developed and deployed on Cisco UCS Director. A module developed using Open Automation behaves the same way as any Cisco UCS Director built-in feature or module. Open Automation is a Java SDK and framework

that contains all the resources needed to develop new modules. Some of the use cases for Open Automation are the following:

- Adding the ability to control a new type of device with Cisco UCS Director
- Designing custom menus for displaying new devices or components
- Taking inventory of new devices
- Developing custom Cisco UCS Director reports and report actions
- Developing tasks that can be used in workflows

Custom tasks enable developers to perform customized operations on Cisco UCS Director resources. Custom tasks are written using Cloupiascript, a language similar to JavaScript. Custom tasks can be used like any other task, including in workflows that orchestrate how the system works. Script bundles are collections of custom tasks that are included with each Cisco UCS Director release and are used for a variety of specific applications. Script bundles can be downloaded, and the custom tasks that are contained in a bundle can be imported into Cisco UCS Director. The main goal with custom tasks is to expand the range of tasks that is available for use in orchestration workflows.

Script modules are used to integrate third-party JARs (Java Archives) and custom libraries with Cisco UCS Director to add custom functionality to the Cisco UCS Director user interface. Some script module operations are already defined in Cisco UCS Director, such as creating advanced controls to collect user input in workflows and context workflow mapping, which enables an administrator to attach workflows to custom actions in a report in Cisco UCS Director. Script modules can be exported and reused in different instances of Cisco UCS Director. Script modules, although named similarly to script bundles, have in fact a very different role. Script bundles, as we've seen previously, are packaged collections of workflow tasks that are released with Cisco UCS Director. Script modules, on the other hand, make it

possible to add custom functionality to Cisco UCS Director.

Cisco UCS Director PowerShell console is a Cisco-developed application that provides a PowerShell interface to the Cisco UCS Director REST API. The console provides a set of PowerShell cmdlets wrapped in a module to internally invoke the REST APIs over HTTP. Each cmdlet performs a single operation. Cmdlets can be chained together to accomplish more advanced automation and data center management tasks. [Figure 9-12](#) shows the relationship between the PowerShell console, Cisco UCS Director, and the infrastructure that is being managed by it.

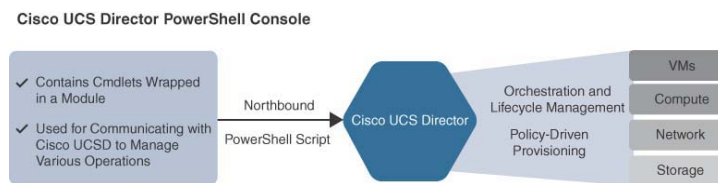


Figure 9-12 Cisco UCS Director PowerShell Console

Cisco UCS Director offers a REST API that enables applications to consume or manipulate the data stored in Cisco UCS Director. Applications use HTTP or HTTPS requests from the REST API to perform Create/Read/Update/Delete (CRUD) operations on Cisco UCS Director resources. With an API call, a developer can execute Cisco UCS Director workflows and change the configuration of switches, adapters, policies, and any other hardware and software components. The API accepts and returns HTTP messages that contain JavaScript Object Notation (JSON) or Extensible Markup Language (XML) documents.



To access the Cisco UCS Director REST API, a valid user account and an API access key are needed. Cisco UCS

Director uses the API access key to authenticate an API request. The access key is a unique security access code that is associated with a specific Cisco UCS Director user account. In order to retrieve the API access key for a specific user, you first log in to Cisco UCS Director with that specific user account. Then hover the mouse over the user icon in the top-right corner and select Edit My Profile from the drop-down list. On the Edit My Profile page, select Show Advanced Settings and retrieve the API access key from the REST API Access Key area. There is also an option to regenerate the access key, if necessary.

Within the user advanced settings is an option to enable the developer menu. By enabling the developer menu, access to the REST API browser and the Report Metadata features is turned on. The REST API browser becomes visible under the Orchestration tab of Cisco UCS Director and provides API information and API code generation capabilities for all available APIs. The Report Metadata option becomes available on all the pages of the Cisco UCS Director GUI; when selected, it returns the API code that the GUI is using to retrieve the information that is displayed to the user in that specific page. This code includes a complete URL that is ready to paste into a browser to send the request to Cisco UCS Director. Both the REST API browser and the Report Metadata features are extremely valuable to developers as they provide ready-to-use sample code and API calls to all the resources available in Cisco UCS Director.

Figure 9-13 shows the Cisco UCS Director REST API browser web interface.

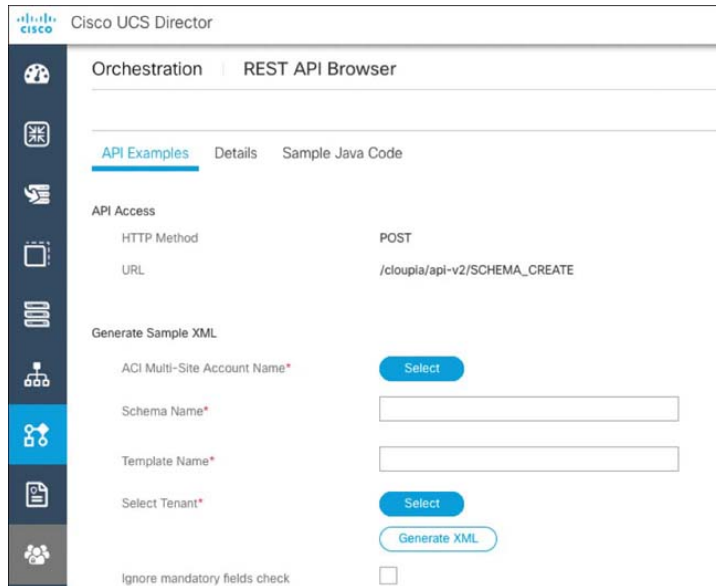


Figure 9-13 Cisco UCS Director REST API Browser



Each REST API request must be associated with an HTTP header called **X-Cloupia-Request-Key**, with its value set to the REST API access key retrieved previously. The REST API request must contain a valid URL of the following format:

https://Cisco_UCS_Director/app/api/rest?formatType=json&opName=operationName&opData=operationData

where

- **Cisco_UCS_Director:** This is the IP address or hostname of the Cisco UCS Director VM.
- **formatType:** This can be either JSON or XML; it is JSON in this case. (Only JSON is discussed throughout the rest of this chapter.)
- **opName:** This is the API operation name that is associated with the request (for example, userAPIGetMyLoginProfile), as explored later in this chapter.
- **opData:** This contains the parameters or the arguments associated with the operation. Cisco UCS Director uses JSON encoding for the parameters. If an operation doesn't require any parameters, the empty

set {} should be used. When building the URL, escape characters should be encoded as appropriate.

Next, let's explore the Cisco UCS Director REST API by using **curl** to construct API calls. Programming guides and complete documentation of the Cisco UCS REST API can be found at the following link:

<https://www.cisco.com/c/en/us/support/servers-unified-computing/ucs-director/products-programming-reference-guides-list.html>. The Cisco DevNet team makes available a reservable sandbox called "UCS Management" for learning purposes. This sandbox contains an installation of Cisco UCS Director and is available at <https://developer.cisco.com/sandbox>. Cisco UCS Director version 6.7 has been used in the following interactions with the REST API. The operation with the name **userAPIGetMyLoginProfile** is used to retrieve the profile of the user with the specific access key that is passed in the request in order to identify the group to which the user belongs. The **curl** command for this operation looks as shown in [Example 9-12](#).

Example 9-12 *curl Command to Retrieve the User Profile in Cisco UCS Director*

[Click here to view code image](#)

```
curl -k -L -X GET \  
  -g 'https://10.10.10.66/app/api/rest?formatType=json&opName=userAPIGetMyLoginProfile&opData={}' \  
  -H 'X-Cloupia-Request-Key:8187C34017C3479089C66678F32775FE'
```

For this request, the **-g** parameter disables the **curl** check for nested braces {}, the **-k** or **-insecure** parameter allows **curl** to proceed and operate even if the server uses self-signed SSL certificates, and the **-L** parameter allows **curl** to follow the redirects sent by the server. The URL for the request follows the requirements discussed previously, using the **/app/api/rest** endpoint

to access the REST API and then passing the **formatType**, **opName**, and **opData** as parameters. The HTTP header for authentication is named **X-Cloupia-Request-Key** and contains the value of the access key for the admin user for the Cisco UCS Director instance that runs on the server with IP address **10.10.10.66**. The response from this instance of Cisco UCS Director looks as shown in [Example 9-13](#).

The operation name is contained in the response and is indeed **userAPIGetMyLoginProfile**, **serviceName** specifies the name of the back-end service (which is in most cases **InfraMgr**), **serviceResult** contains a set of name/value pairs or a JSON object if the request was successful, and **serviceError** contains the error message. If the request succeeds, the **serviceError** value is set to null, and if the operation fails, **serviceError** contains the error message.

Example 9-13 *REST API Response Containing User Profile Information*

[Click here to view code image](#)

```
{
  "opName" : "userAPIGetMyLoginProfile",
  "serviceName" : "InfraMgr",
  "serviceResult" : {
    "email" : null,
    "groupName" : null,
    "role" : "Admin",
    "userId" : "admin",
    "groupId" : 0,
    "firstName" : null,
    "lastName" : null
  },
  "serviceError" : null}
```

As mentioned previously, Cisco UCS Director tasks and workflows can have any number of input and output variables. In order to retrieve the inputs for a specific workflow, the **userAPIGetWorkflowInputs**

operation can be used with the name of the desired workflow in the **paramo** field. Cisco UCS Director comes by default with a large number of predefined workflows. One of them is the “VMware OVF Deployment,” which, as the name implies, can deploy new VMware virtual machines based on OVF images. The **curl** command in [Example 9-14](#) contains the API call to retrieve all the inputs for this workflow.

Example 9-14 *curl Command to Retrieve Workflow Inputs in Cisco UCS Director*

[Click here to view code image](#)

```
curl -k -L -X GET \  
  -g 'http://10.10.10.66/app/api/rest?formatType=json&opName=userAPIGetWorkflowInputs&opData={param0:%22VMware%20OVF%20Deployment%22}' \  
  -H 'X-Cloupia-Request-Key:8187C34017C3479089C66678F32775FE'
```

Notice that the name of the workflow is passed in the API call in the **paramo** parameter and also that VMware OVF Deployment is encoded, using single quotation marks and spaces between the words. [Example 9-15](#) shows a snippet of the response.

The response contains similar fields to the response in [Example 9-14](#). **opName** is confirmed as **userAPIGetWorkflowInputs**, the back-end service that responded to the request is once again **InfraMgr**, **serviceError** is **null** (indicating that there were no errors in processing the request), and **serviceResult** contains a list called **details**, which includes all the inputs and their properties for the **VMware OVF Deployment** workflow.

Example 9-15 *curl Command Response Containing Workflow Inputs in Cisco UCS Director*

[Click here to view code image](#)

```

{
  "serviceResult" : {
    "details" : [
      {
        "inputFieldValidator" :
"VdcValidator",
        "label" : "vDC",
        "type" : "vDC",
        "inputFieldType" : "embedded-lov",
        "catalogType" : null,
        "isOptional" : false,
        "name" : "input_0_vDC728",
        "isMultiSelect" : false,
        "description" : "",
        "isAdminInput" : false
      },
      {
        "isAdminInput" : false,
        "description" : "",
        "label" : "OVF URL",
        "type" : "gen_text_input",
        "isMultiSelect" : false,
        "isOptional" : false,
        "inputFieldType" : "text",
        "catalogType" : null,
        "name" : "input_1_OVF_URL465",
        "inputFieldValidator" : null
      },
      ...omitted output
    ]
  },
  "serviceName" : "InfraMgr",
  "opName" : "userAPIGetWorkflowInputs",
  "serviceError" : null}

```

CISCO INTERSIGHT

The Cisco Intersight platform provides intelligent cloud-powered infrastructure management for Cisco UCS and Cisco HyperFlex platforms. Cisco UCS and Cisco HyperFlex use model-based management to provision servers and the associated storage and networking automatically. Cisco Intersight works with Cisco UCS Manager and Cisco Integrated Management Controller (IMC) to bring the model-based management of Cisco compute solutions into one unified management

solution. Cisco Intersight offers flexible deployment options either as software as a service (SaaS) on <https://intersight.com> or running a Cisco Intersight virtual appliance on premises. Some of the benefits of using Cisco Intersight are the following:

- It simplifies Cisco UCS and Cisco HyperFlex management with a single management platform.
- It makes it possible to scale across data center and remote locations without additional complexity.
- It automates the generation and forwarding of technical support files to the Cisco Technical Assistance Center to accelerate the troubleshooting process.
- Full programmability and automation capabilities are available through a REST API interface.
- A streamlined upgrade process is available for standalone Cisco UCS servers.

Getting started with Cisco Intersight involves the following steps:

Step 1. Log in to <https://intersight.com> with a Cisco ID account.

Step 2. Claim a device for the account. Endpoint devices connect to the Cisco Intersight portal through a device connector that is embedded in the management controller of each system.

Step 3. (Optional) Add additional users to the new account. Several roles are available, including read-only and admin roles. Custom roles can be created, if needed.



Cisco Intersight includes a REST API interface built on top of the OpenAPI specification. The API documentation, API schemas, and SDKs can be found at <https://intersight.com/apidocs>. At this writing, Python and PowerShell SDKs are available for download at the previous link. The API accepts and returns messages that

are encapsulated in JSON documents and are sent over HTTPS. The Intersight API is a programmatic interface to the Management Information Model that is similar to Cisco ACI and Cisco UCS Manager. Just like Cisco ACI and Cisco UCS Manager MIMs, the Cisco Intersight MIM is composed of managed objects. Managed objects or REST API resources are uniquely identified by URI (uniform resource identifier) or, as seen earlier in this chapter, distinguished name (DN). Example of managed objects include Cisco UCS servers; Cisco UCS fabric interconnects; Cisco HyperFlex nodes and clusters; server, network, and storage policies; alarms; statistics; users; and roles. Cisco Intersight managed objects are represented using a class hierarchy specified in the OpenAPI specification. All the API resources are descendants of the **mo.Mo** class. [Table 9-2](#) shows the properties that are common to all managed objects.

Table 9-2 Common Properties for All Managed Objects in Cisco Intersight

Property Name	Description
M o i d	A unique identifier of the managed object instance.
O b j e c t T y p e	The fully qualified class name of the managed object.
Ac co u nt M o i d	The Intersight account ID for the managed object.

Cr ea te Ti me	The time when the managed object was created.
M od Ti me	The time when the managed object was last modified. ModTime is automatically updated whenever at least one property of the managed object is modified.
O w ne rs	An array of owners, which represents effective ownership of the object.
Ta gs	An array of tags that allow the addition of key/value metadata to managed objects.
A nc es to rs	An array containing the MO references of the ancestors in the object containment hierarchy.
Pa re nt	The direct ancestor of the managed object in the containment hierarchy.

Every managed object has a unique **Moid** identifier assigned when the resource is created. The **Moid** is used to uniquely distinguish a Cisco Intersight resource from all other resources. The **Moid** is a 12-byte string set when a resource is created.

Each managed object can be addressed using a unique uniform resource identifier (URI) that includes the **Moid**. The URI can be used in any HTTPS request to address the managed object. A generic Cisco Intersight URI is of the following form:

`https://intersight.com/path[?query]`

The URI of a managed object includes the following:

- **https:** The HTTPS protocol
- **intersight.com:** The Cisco Intersight hostname
- *path:* The path, organized in hierarchical form
- *query:* An optional query after the question mark and typically used to limit the output of the response to only specific parameters

For example, the URI of an object with **Moid** **48601f85ae74b80001aee589** could be:

`https://intersight.com/api/v1/asset/DeviceRegistrations/48601f85ae74b80001aee589`

Every managed object in the Cisco Intersight information model supports tagging. Tagging is used to categorize objects by a certain common property, such as owner, geographic location, or environment. Tags can be set and queried through the Intersight API. Each tag consists of a key and an optional value. Both the key and the value are of type string.

Managed objects may include object relationships, which are dynamic links to REST resources. Cisco Intersight uses Hypermedia as the Engine of Application State (HATEOAS) conventions to represent object relationships. Object relationships can be links to self or links to other managed objects, which, taken as a whole, form a graph of objects. By using relationships as a first-class attribute in the object model, many classes of graphs can be represented, including trees and cyclic or bipartite graphs.

Intersight provides a rich query language based on the OData standard. The query language is represented using URL query parameters for **GET** results. Several types of data are supported with the Intersight queries, including string, number, duration, data and time, and time of day.



When a client sends an API request, the Intersight web service must identify and authenticate the client. The Intersight web service supports two authentication methods:

- API keys
- Session cookies

An Intersight API key is composed of a `keyId` and a `keySecret`. The API client uses the API key to cryptographically sign each HTTP request sent to the Intersight web service. The “**signature**” parameter is a base 64–encoded digital signature of the message HTTP headers and message content. API keys are generated in the Settings > API section of the Intersight web interface. As a best practice, it is recommended to generate separate API keys for each client application that needs access to the API.

Cookies are used primarily by the Intersight GUI client running in a browser. When accessing the Intersight web service, end users must first authenticate to <https://sso.cisco.com>. When authentication is successful, sso.cisco.com sends a signed SAML assertion to the Intersight web service, and Intersight generates a session cookie with a limited time span validity. The client must send the session cookie in each API request.



Included with the Cisco Intersight REST API documentation at <https://intersight.com/apidocs> are the API reference documentation and an embedded REST API client. [Figure 9-14](#) shows the web interface for the Cisco Intersight API reference documentation. In this figure, the **Get a list of**

'equipmentDeviceSummary' instances API call is selected. The query parameters that this specific API call supports are displayed, as are the API URI for the endpoint that will return the list of all the devices that are being managed by Cisco Intersight for this specific account. Much as with Postman, if the Send button is clicked, the API call is triggered, and the response is displayed in the Response Text window.

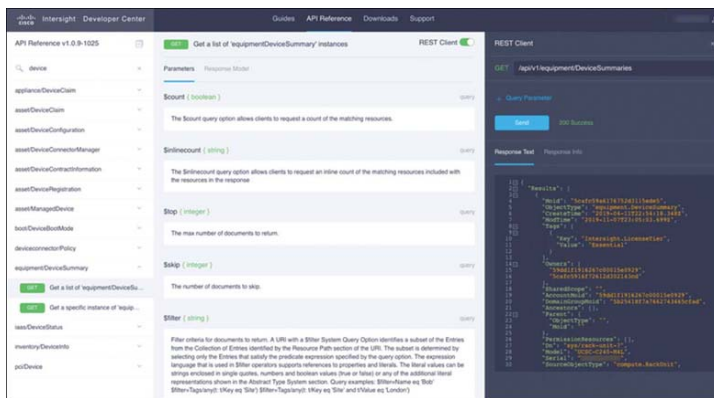


Figure 9-14 Cisco Intersight REST API Reference Documentation

Under the Downloads section of <https://intersight.com/apidocs>, the Cisco Intersight Python and PowerShell SDKs can be downloaded. The Python SDK covers all the functionality of the REST API and offers Python classes and methods that can be used to simplify Cisco Intersight automation projects. The Python sample code in [Example 9-16](#) was developed using the Intersight module version 1.0 and Python 3.7.4. This Python code replicates the earlier REST API call **equipmentDeviceSummary**, which returns a list of all the devices that are being managed by Cisco Intersight for a specific account.

Example 9-16 Intersight Python Module Example

[Click here to view code image](#)

```
#!/usr/bin/env python
from intersight.intersight_api_client import
```

```

IntersightApiClient
from intersight.apis import
equipment_device_summary_api

# Create an Intersight API Client instance
API_INSTANCE = IntersightApiClient(
    host="https://intersight.com/api/v1",\
    private_key="/Path_to/SecretKey.txt",\
    api_key_id="your_own_api_key_id")

# Create an equipment device handle
D_HANDLE =
equipment_device_summary_api.EquipmentDeviceSummaryApi(API_INSTANCE)

DEVICES =
D_HANDLE.equipment_device_summaries_get().results

print('{0:35s}{1:40s}{2:13s}{3:14s}'.format(
    "DN",
    "MODEL",
    "SERIAL",
    "OBJECT TYPE"))
print('-'*105)

# Loop through devices and extract data
for DEVICE in DEVICES:
    print('{0:35s}{1:40s}{2:13s}{
3:14s}'.format(
        DEVICE.dn,
        DEVICE.model,
        DEVICE.serial,
        DEVICE.source_object_type))

```

The first two lines of [Example 9-16](#) use the **import** keyword to bring in and make available for later consumption the **IntersightApiClient** Python class that will be used to create a connection to the Cisco Intersight platform and the **equipment_device_summary_api** file, which contains Python objects that are useful for retrieving equipment that is being managed by Intersight. Every Cisco Intersight REST API endpoint has a corresponding Python file containing classes and methods that can be used to programmatically process those REST API endpoints. Next, an instance of the **IntersightApiClient** class is created in order to

establish a connection and have a hook back to the Cisco Intersight platform. Three parameters need to be passed in to instantiate the class:

- **host:** This parameter specifies the Cisco Intersight REST API base URI.
- **private_key:** This parameter specifies the path to the file that contains the keySecret of the Intersight account that will be used to sign in.
- **api_key_id:** This parameter contains the keyId of the same Intersight account. As mentioned previously, both the keyId and keySecret are generated in the Intersight web interface, under Settings > API keys.

Next, an instance of the

EquipmentDeviceSummaryApi class is created and stored in the **D_HANDLE** variable. This Python class maps into the **/api/v1/equipment/DeviceSummaries** REST API resource. The **D_HANDLE** variable contains the handle to that REST API resource. The

equipment_device_summaries_get method that is available with the **EquipmentDeviceSummaryApi** class is invoked next, and the results are stored in the **DEVICES** variable, which contains a complete list of all the equipment that is being managed by Cisco Intersight for the user account with the keyId and keySecret with which the initial connection was established. The **for** loop iterates over the devices in the list and extracts for each one the distinguished name, model, serial number, and object type and displays this information to the console. The output of this Python script for a test user account looks as shown in [Figure 9-15](#).

DN	MODEL	SERIAL	OBJECT	TYPE
sys/rack-unit-7	UCSC-C240-M4L	FC-	T	compute.RackUnit
sys/rack-unit-8	UCSC-C240-M4L	FC-	7	compute.RackUnit
sys/rack-unit-3	HXAF220C-M5SX	WZF	3	compute.RackUnit
sys/rack-unit-1	HXAF220C-M5SX	WZF	6	compute.RackUnit
sys/rack-unit-6	HXAF220C-M5SX	WZF	H	compute.RackUnit
sys/rack-unit-5	HXAF220C-M5SX	WZF	J	compute.RackUnit
sys/rack-unit-2	HXAF220C-M5SX	WZF	7	compute.RackUnit
sys/rack-unit-4	HXAF220C-M5SX	WZF	F	compute.RackUnit
sys/rack-unit-8/adaptor-1	UCSC-PCIE-CSC-02	FC-	E	adapter.Unit
sys/rack-unit-7/adaptor-1	UCSC-PCIE-CSC-02	FC-	J	adapter.Unit
sys/rack-unit-5/adaptor-1	UCSC-ML0M-C400-03	FC-	4	adapter.Unit
sys/rack-unit-2/adaptor-1	UCSC-ML0M-C400-03	FC-	T	adapter.Unit
sys/rack-unit-6/adaptor-1	UCSC-ML0M-C400-03	FC-	H	adapter.Unit
sys/rack-unit-1/adaptor-1	UCSC-ML0M-C400-03	FC-	F	adapter.Unit
sys/rack-unit-4/adaptor-1	UCSC-ML0M-C400-03	FC-	w	adapter.Unit
sys/rack-unit-3/adaptor-1	UCSC-ML0M-C400-03	FC-	C	adapter.Unit
sys/switch-A	UCS-FI-6248UP	SSI	2	network.Element
sys/switch-B	UCS-FI-6248UP	SSI	7	network.Element
sys/rack-unit-1/adaptor-ML0M	UCSC-ML0M-CSC-02	FC-	B	adapter.Unit
sys/rack-unit-1	UCSC-C240-M4SX	FC-	A	compute.RackUnit
sys/rack-unit-1/adaptor-ML0M	UCSC-ML0M-CSC-02	FC-	J	adapter.Unit
sys/rack-unit-1	UCSC-C240-M4SX	FC-	2	compute.RackUnit
sys/rack-unit-1/adaptor-ML0M	UCSC-ML0M-CSC-02	FC-	8	adapter.Unit
sys/rack-unit-1	UCSC-C240-M4SX	FC-	0	compute.RackUnit
sys/rack-unit-1/adaptor-ML0M	UCSC-ML0M-CSC-02	FC-	B	adapter.Unit
sys/rack-unit-1	UCSC-C240-M4SX	FC-	Z	compute.RackUnit
sys/rack-unit-1/network-adaptor-L	Intel(R) I350 1 Gbps Network Controller			adapter.Unit
sys/rack-unit-1/network-adaptor-L	Intel(R) I350 1 Gbps Network Controller			adapter.Unit
sys/rack-unit-1/network-adaptor-L	Intel(R) I350 1 Gbps Network Controller			adapter.Unit
sys/rack-unit-1/network-adaptor-L	Intel(R) I350 1 Gbps Network Controller			adapter.Unit

Figure 9-15 Output of the Python Script from Example 9-16

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 9-3](#) lists these key topics and the page number on which each is found.

Table 9-3 Key Topics

Key Topic	Element Description	Page Number
Parag raph	Cisco Nexus 9000 switches	<u>2</u>
		<u>1</u>
		<u>6</u>
Parag raph	Cisco ACI fabric	<u>2</u>
		<u>1</u>

		<u>7</u>
Parag raph	The configuration of the ACI fabric	<u>2</u> <u>1</u> <u>8</u>
Parag raph	Physical and logical components of the Cisco ACI fabric	<u>2</u> <u>1</u> <u>9</u>
Parag raph	Cisco ACI fabric policies	<u>2</u> <u>2</u> <u>0</u>
Parag raph	Endpoint groups (EPGs)	<u>2</u> <u>2</u> <u>2</u>
Parag raph	The APIC REST API URI	<u>2</u> <u>2</u> <u>3</u>
Parag raph	Tools and libraries for Cisco ACI automation	<u>2</u> <u>2</u> <u>7</u>
Parag raph	UCS Manager and the server lifecycle	<u>2</u> <u>3</u> <u>0</u>
Parag raph	The UCS Manager programmatic interface	<u>2</u> <u>3</u> <u>1</u>
Parag raph	The Cisco software emulator	<u>2</u> <u>3</u> <u>4</u>
Parag raph	The Cisco UCS Python SDK	<u>2</u> <u>3</u> <u>7</u>
Parag raph	Cisco UCS Director	<u>2</u> <u>4</u> <u>0</u>

Parag raph	The Cisco UCS Director orchestrator	<u>2</u> 4 0
Parag raph	The programmability and extensibility of the Cisco UCS Director	<u>2</u> 4 1
Parag raph	Accessing the Cisco UCS Director REST API	<u>2</u> 4 2
Parag raph	REST API requests and the X-Cloupia-Request-Key header	<u>2</u> 4 3
Parag raph	The Cisco Intersight REST API interface	<u>2</u> 4 7
Parag raph	Client API requests and Intersight	<u>2</u> 4 9
Parag raph	The Cisco Intersight REST API documentation	<u>2</u> 4 9

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Application Centric Infrastructure (ACI)

Application Policy Infrastructure Controller (APIC)

Management Information Model (MIM)

management information tree (MIT)

managed object (MO)

distinguished name (DN)

virtual routing and forwarding (VRF) instance

endpoint group (EPG)

Unified Computing System (UCS)

Chapter 10

Cisco Collaboration Platforms and APIs

This chapter covers the following topics:

- **Introduction to the Cisco Collaboration Portfolio:** This section introduces the collaboration portfolio by functionality and provides an overview of the product offerings.
- **Webex Teams API:** This section introduces Webex Teams and the rich API set for managing and creating applications, integrations, and bots.
- **Cisco Finesse:** This section provides an overview of Cisco Finesse and API categories, and it provides sample code and introduces gadgets.
- **Webex Meetings APIs:** This section provides an introduction to the high-level API architecture of Webex Meetings along with the Meetings XML API for creating, updating, and deleting meetings.
- **Webex Devices:** This section provides an overview of the Webex Devices portfolio, xAPI, and sample applications to turn on presence detector on devices.
- **Cisco Unified Communications Manager:** This section provides an overview of Cisco Call Manager and Cisco Administrative XML (AXL), and it shows sample code for using the SDK.

Every day, millions of people rely on Cisco collaboration and Webex solutions to collaborate with their teams, partners, and customers. These products help them work smarter, connect across boundaries, and drive new innovative ideas forward. Cisco products offer secure, flexible, seamless, and intelligent collaboration. This chapter introduces the various products as well as how to integrate these collaboration products via APIs. It covers the following:

- Cisco Webex Teams
- Cisco Webex Devices

- Cisco Unified Communications Manager
- Cisco Finesse

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 10-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 10-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Introduction to the Cisco Collaboration Portfolio	1
Webex Teams API	2–4
Cisco Finesse	5, 6
Webex Meetings APIs	7
Webex Devices	8, 9
Cisco Unified Communications Manager	10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the

answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which of the following are part of the Cisco collaboration portfolio? (Choose three.)

1. Video Calling
2. Bots
3. Remote Expert
4. Connected Mobility Experience

2. How does Webex Teams allow you to access APIs? (Choose three.)

1. Integrations
2. Bots
3. Drones
4. Personal access tokens

3. Guest users of Webex Teams authenticate with guest tokens, which use _____.

1. Base64
2. No token
3. JWT
4. Sessions

4. Which of the following use webhooks in Webex Teams?

1. Bots
2. Guests
3. Nobody
4. Integrations

5. True or false: The Finesse desktop application is completely built using APIs.

1. False
2. True

6. Finesse implements the XMPP specification. The purpose of this specification is to allow the XMPP server (for Notification Service) to get information published to XMPP topics and then to send XMPP events to entities subscribed to the topic. The

Finesse Notification Service then sends XMPP over _____ messages to agents that are subscribed to certain XMPP nodes.

1. MQTT
 2. BOSH
 3. HTTP
 4. None of the above
7. Which of the following enables hosts/users to update the information for a scheduled meeting that they are able to edit?
1. SetMeeting
 2. ListMeeting
 3. CreateMeeting
 4. Layered systems
8. Which of the following is the application programming interface (API) for collaboration endpoint software?
1. MQTT
 2. TAPI
 3. DevNet
 4. xAPI
9. xAPI on a device can be accessed via which of the following protocol methods? (Choose all that apply.)
1. SSH
 2. FTP
 3. HTTP
 4. Websocket
10. Which of the following provides a mechanism for inserting, retrieving, updating, and removing data from Cisco Unified Communications Manager?
1. Session Initiation Protocol
 2. Administration XML
 3. Skinny
 4. REST API

FOUNDATION TOPICS

INTRODUCTION TO THE CISCO COLLABORATION PORTFOLIO

Cisco's collaboration portfolio is vast, but it can be logically broken down into essentially four high-level components:

- **Unified Communications Manager:** This product unifies voice, video, data, and mobile apps.
- **Unified Contact Center:** This product provides customers with personalized omnichannel experiences.
- **Cisco Webex:** This conferencing solution enables teamwork with intuitive solutions that bring people together.
- **Cisco collaboration endpoints:** These devices provide better-than-being-there experiences via new devices.

Figure 10-1 depicts the Cisco collaboration portfolio and the various products that fall into each of the categories.

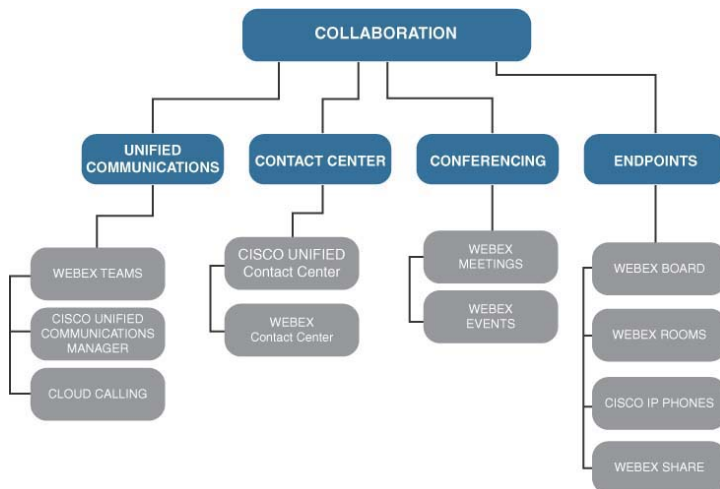


Figure 10-1 Rich Collaboration Portfolio

Unified Communications

People work together in different ways. And they use a lot of collaboration tools: IP telephony for voice calling, web and video conferencing, voicemail, mobility, desktop sharing, instant messaging and presence, and more.

Cisco Unified Communications solutions deliver seamless user experiences that help people work together more effectively—anywhere and on any device. They bring real-time communication from your phone system and conferencing solutions together with messaging and

chat and integrate with everyday business applications using APIs.

Unified Communications solutions are available as on-premises software, as partner-hosted solutions, and as a service (UCaaS) from cloud providers.

The following sections describe the products available under the Unified Communications umbrella.

Cisco Webex Teams

Cisco Webex Teams brings people and work together in a single reimagined workspace in and beyond the meeting. Webex Teams allows you to connect with people (inside and outside your organization) and places all your tools right in the center of your workflow. It breaks down the silos that exist for some across the collaboration experience.

Both Webex Teams and Webex Meetings have a Join button you can click to easily join a meeting. This helps ensure that meetings start on time and it streamlines the process of joining a meeting. Cisco Webex Teams also has features that will help you make decisions on which meetings to prioritize and when to join them:

- **Seeing invitee status/participants:** Every invitee can see who has accepted/declined the meeting and who's in the meeting live before even joining. There is no need to switch back and forth between your calendar and the meeting application.
- **Instantly switching between meetings:** If you need to move from one meeting to another, simply leave the meeting with one click and join the other one with another click.
- **Easily informing attendees when you are running late:** In Webex Teams, you can message people in the meeting to keep them posted on your status. Gone are the days of sending an email that no one will read because they are in the meeting.

Webex Teams provides a space for a team to start working—to discuss issues and share content—before a meeting. When the meeting starts, all the discussion and work from before the meeting are available right there in

the meeting space. You can simply share content from any device in the meeting—directly from the space or from another device, wirelessly and quickly.

An added benefit of joining a meeting through Webex Teams is that during the meeting, everyone is an equal participant and can mute noisy participants, record the meeting, and do other meeting management tasks without having to disrupt the flow of the meeting.

Cisco Webex Calling

Webex Calling is a cloud-based phone system that is optimized for midsize businesses and provides the essential business calling capabilities you are likely to need. With Webex Calling, there's no need to worry about the expense and complexity of managing phone system infrastructure on your premises anymore. Cisco takes care of the Webex Cloud so you can focus on what matters most.

You can choose from a wide range of Cisco IP phones to make and receive calls. Enjoy the calling features you are used to from a traditional phone system to help run your organization smoothly and never miss a call. If you are a mobile worker, or if you are out of the office, you can make and receive calls on your smartphone, computer, or tablet, using the Cisco Webex Teams app.

Webex Calling seamlessly integrates with Webex Teams and Webex Meetings, so you can take collaboration inside and outside your organization to a new level. Customers and business partners can make high-definition audio and video calls from the office, from home, or on the go. Screen sharing, file sharing, real-time messaging, and whiteboarding can turn any call into a productive meeting. You can also add Cisco Webex Board, Room, or Desk Device to get the most out of Webex Teams and Meetings and improve teamwork.

Cisco Webex Calling delivers all the features of a traditional PBX through a monthly subscription service. Important qualities include the following:

- An advanced set of enterprise-grade PBX features
- A rich user experience that includes the CiscoWebex Calling app, for mobile and desktop users, integrated with the Cisco Webex Teams collaboration app
- Support for an integrated user experience with Cisco Webex Meetings and Webex Devices, including Cisco IP Phones 6800, 7800, and 8800 Series desk phones and analog ATAs
- Delivery from a set of regionally distributed, geo-redundant data centers around the globe
- Service that is available across a growing list of countries in every region
- Protection of existing investment in any on-premises Cisco Unified Communications Manager (Unified CM) licenses, through Cisco Collaboration Flex Plan
- A smooth migration to the cloud at your pace, through support of cloud and mixed cloud and on-premises deployments

Cisco Unified Communications Manager (Unified CM)

Cisco Unified CM, often informally referred to as *Call Manager*, is the core of Cisco's collaboration portfolio. It delivers people-centric user and administrative experiences and supports a full range of collaboration services, including video, voice, instant messaging and presence, messaging, and mobility on Cisco as well as third-party devices. Unified CM is the industry leader in enterprise call and session management platforms, with more than 300,000 customers worldwide and more than 120 million Cisco IP phones and soft clients deployed.

Unified Contact Center

The Cisco Unified Contact Center (Unified CC), also called Finesse, is a next-generation desktop that is designed to provide the optimal user experience for agents. It is 100% browser based, so agent client machines do not need any Unified CC-specific applications. Cisco Finesse is a next-generation agent and supervisor desktop designed to provide a

collaborative experience for the various communities that interact with a customer service organization. It also helps improve the customer experience and offers a user-centric design to enhance customer care representative satisfaction.

Cisco Finesse provides the following:

- An agent and supervisor desktop that integrates traditional contact center functions into a thin-client desktop.
- A 100% browser-based desktop implemented through a Web 2.0 interface; no client-side installations are required.
- A single customizable interface that gives customer care providers quick and easy access to multiple assets and information sources.
- Open Web 2.0 APIs that simplify the development and integration of value-added applications and minimize the need for detailed desktop development expertise.

Cisco Webex

Cisco Webex is a conferencing solution that allows people to collaborate more effectively with each other anytime, anywhere, and from any device. Webex online meetings are truly engaging with high-definition video. Webex makes online meetings easy and productive with features such as document, application, and desktop sharing; integrated audio/video; active speaker detection; recording; and machine learning features.

Cisco Collaboration Endpoints

To support and empower the modern workforce, Cisco has a “no-compromise” collaboration solution for every room, on every desk, in every pocket, and into every application. Its portfolio of collaboration devices includes everything from voice to collaboration room devices for small businesses to very large enterprises. The majority of the portfolio has been redesigned to make collaboration more affordable, accessible, and easy to use. Many Cisco collaboration endpoints have received the Red Dot Award for excellence in design.

The Cisco collaboration endpoint portfolio includes the following devices:

- **Room-based devices:** These include video system or codec in the Cisco TelePresence MX Series, SX, and DX Series.
- **Cisco Webex Board:** Cisco Webex Board allows you to wirelessly present whiteboard and video or audio conference for team collaboration.
- **Webex Share:** The new Webex Share device allows easy, one-click wireless screen sharing from the Webex Teams software client to any external display with an HDMI port.
- **Collaboration desktop video devices:** A range of options are available, from entry-level HD video up to the lifelike DX video collaboration devices.
- **IP Phone portfolio:** Cisco IP Phone devices deliver superior voice communications, with select endpoints supporting HD video and a range of options that offer choices for businesses of various sizes and with unique needs and budgets. The complete portfolio also supports specialty use cases, such as in-campus wireless with WLAN handsets and an audio-conferencing endpoint for small to large conference rooms. The goal of the IP Phone portfolio is to deliver the highest-quality audio communications with affordable, scalable options for desktop video that are easy to use and administer so you can collaborate effectively and achieve your desired business results.
- **Cisco Headset 500 Series:** These headsets deliver surprisingly vibrant sound for open workspaces. Now users can stay focused in noisy environments with rich sound, exceptional comfort, and proven reliability. The headsets offer a lightweight form factor designed for workers who spend a lot of time collaborating in contact centers and open workspaces. With the USB headset adapter, the 500 Series delivers an enhanced experience, including automatic software upgrades, in-call presence indicator, and audio customizations that allow you to adjust how you hear the far end and how they hear you.

Cisco now provides more intelligent audio, video, and usability, offering new ways for users to bring their personal devices, such as smartphones or tablets, into a phone or video meeting to further enhance the collaboration experience.

API Options in the Cisco Collaboration Portfolio

The Cisco collaboration portfolio is rich in features. APIs are used to integrate and scale these products in building various applications. The following sections cover four categories of collaboration APIs:

- Webex Meetings APIs
- Webex Teams
- Contact Center (Finesse)
- Endpoints

WEBEX TEAMS API



Webex Teams makes it easy for everyone on a team to be in sync. Conversations in Webex Teams take place in virtual meeting rooms called *spaces*. Some spaces live for a few hours, while others become permanent fixtures of a team's workflow. Webex Teams allows conversations to flow seamlessly between messages, video calls, and real-time whiteboarding sessions.

Getting started with the Webex APIs is easy. These APIs allow developers to build integrations and bots for Webex Teams. APIs also allow administrators to perform administrative tasks.

Webex APIs provide applications with direct access to the Cisco Webex platform, giving you the ability to do the following:

- Administer the Webex Teams platform for an organization, add user accounts, and so on
- Create a Webex Teams team, space, and memberships
- Search for people in the company
- Post messages in a Webex Teams space
- Get Webex Teams space history or be notified in real time when new messages are posted by others

Figure 10-2 shows an overall picture of how Webex Teams is organized. Only an admin has the ability to add a new organization or a new user account for the organization. An organization is made up of teams, and a team can have one or more rooms. A person is an end user who can be added to a room. People communicate

with each other or with everyone else in a room via messages.

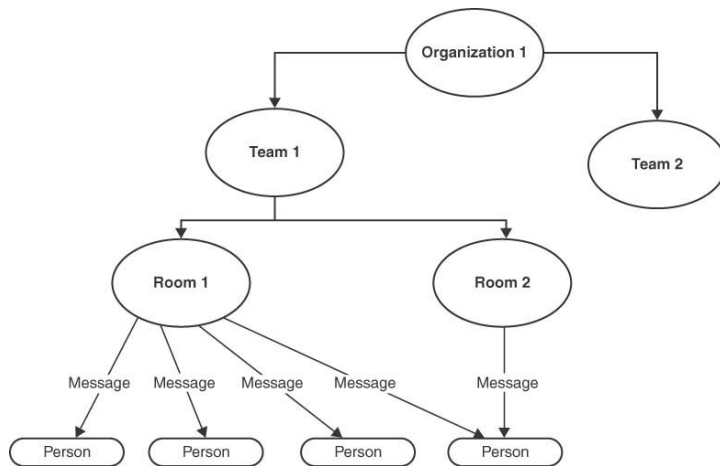


Figure 10-2 *Webex Teams Building Blocks: Organizations, Teams, Rooms, People, and Messages*

API Authentication

There are four ways to access the Webex Teams APIs:

- Personal access tokens
- Integrations
- Bots
- Guest issuers

The Webex Representational State Transfer (REST) API uses the Cisco Webex Common Identity (CI) account. Once you create an account to join Webex Teams, you have access to the Common Identity account, which allows you to use the APIs and SDKs.

Personal Access Tokens

When making requests to the Webex REST API, an authentication HTTP header is used to identify the requesting user. This header must include an access token, which may be a personal access token from the developer site (<https://developer.Webex.com>), a bot token, or an OAuth token from an integration or a guest issuer application. The API reference uses your personal

access token, which you can use to interact with the Webex API as yourself. This token has a short lifetime—it lasts only 12 hours after logging in to the site—so it shouldn't be used outside of app development. [Figure 10-3](#) shows the bearer token as obtained from the developer portal.



Figure 10-3 *Webex Teams: Getting a Personal Access Token*

Integrations

To perform actions on behalf of someone else, you need a separate access token that you obtain through an OAuth authorization grant flow. OAuth is supported directly into the platform. With a few easy steps, you can have a Webex Teams user grant permission to your app and perform actions on that person's behalf. [Figure 10-4](#) shows how third-party apps can access the platform.



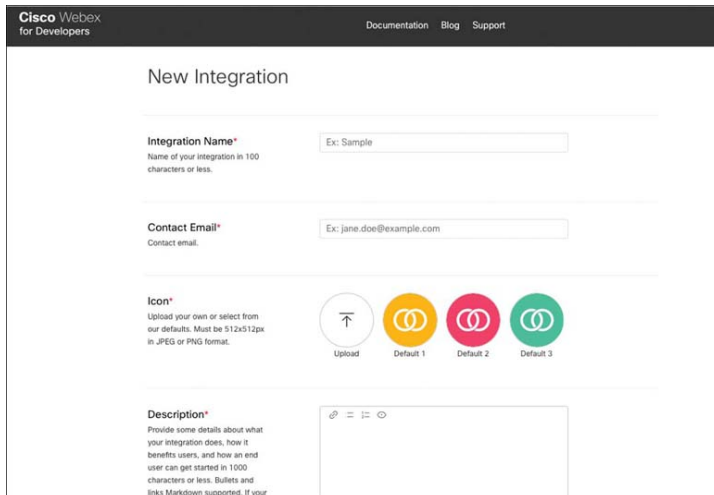
Figure 10-4 *Webex Teams: Third-Party Integrations*

You use an integration to request permission to invoke the Webex REST API on behalf of another Webex Teams

user. To provide security, the API supports the OAuth 2 standard, which allows a third-party integration to get a temporary access token for authenticating API calls instead of asking users for their password.

Here are a few easy steps to get started using an integration:

Step 1. Register an integration with Webex Teams at <https://developer.Webex.com/my-apps/new>. [Figure 10-5](#) shows the sample form on the portal that allows you to create a new integration.



The screenshot shows the 'New Integration' form on the Cisco Webex Developer Portal. The form includes the following fields and options:

- Integration Name***: A text input field with the placeholder 'Ex: Sample'. Below the field, it says 'Name of your integration in 100 characters or less.'
- Contact Email***: A text input field with the placeholder 'Ex: jane.doe@example.com'. Below the field, it says 'Contact email.'
- Icon***: A section for selecting an icon. It includes an 'Upload' button with an upward arrow icon and three default icons: 'Default 1' (yellow), 'Default 2' (red), and 'Default 3' (green). Below this section, it says 'Upload your own or select from our defaults. Must be 512x512px in JPEG or PNG format.'
- Description***: A rich text editor area with a toolbar. Below the editor, it says 'Provide some details about what your integration does, how it benefits users, and how an end user can get started in 1000 characters or less. Bullets and links Markdown supported. If your'

Figure 10-5 *Creating a New Integration via the Developer Portal*

Step 2. Request permission by using an OAuth grant flow by invoking the flow via <https://webexapis.com/v1/authorize> and providing a redirect URL to come back to. After the integration is created successfully, you see a screen like the one in [Figure 10-6](#), which also shows the full authorization URL.

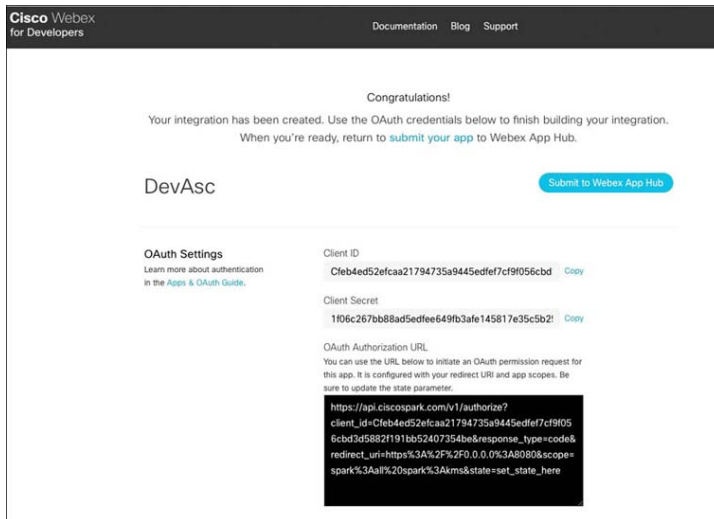


Figure 10-6 Successful Integration Results in OAuth Credentials

Step 3. On the screen shown the [Figure 10-7](#), click [Accept](#) to obtain the authorization code for an access token.

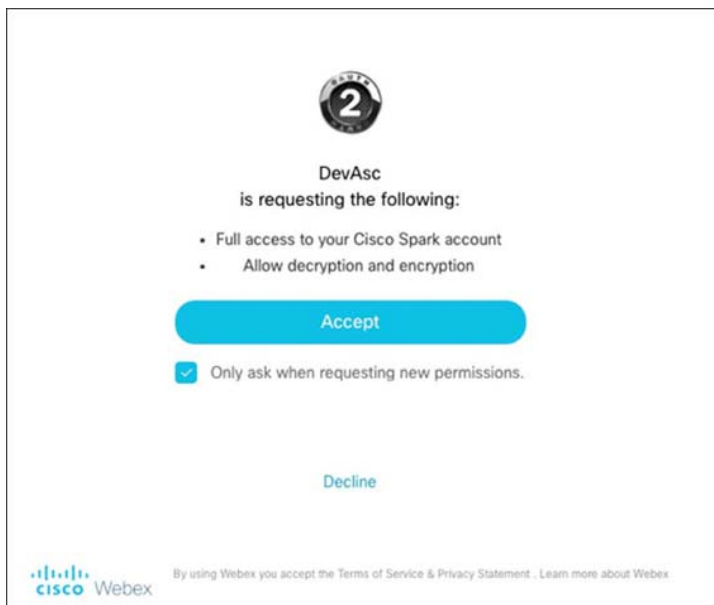


Figure 10-7 Using the OAuth Credentials and Accepting Permissions

The redirect URL contains a code parameter in the query string like so:

https://o.o.o.o:8080/?code=NzAwMGUyZDUtYjcxMSooYWM4LTg3ZDYtNzd hMDhhNWRjZGY5NGFmMjA3ZjEtYzRk_Pf84_1eb65f df-9643-417f-9974-ad72cae0e10f&state=set_state_here



Access Scopes

Scopes define the level of access that an integration requires. Each integration alerts the end user with the scope of the integration. Scopes determine what resources the Access Token has access to. [Table 10-2](#) lists and describes the scopes.

Table 10-2 Webex Teams Scopes API Definitions

ScopeDescription	
permission dialog.spark:all	Full access to your Webex Teams account
spark:people_read	Read your company directory
spark:rooms_read	List the titles of rooms that you're in
spark:rooms_write	Manage rooms on your behalf
spark:memberships_read	List the people in rooms that you're in
spark:memberships_write	Invite people to rooms on your behalf
spark:messages_read	Read the content of rooms that you're in
spark:messages_write	Post and delete messages on your

	behalf
spark:teams_read	List the teams you are a member of
spark:teams_write	Manage teams on your behalf
spark:team_members hips_read	List the people in the teams that you are in
spark:team_members hips_write	Add people to teams on your behalf
spark:webhooks_read	See all webhooks created on your behalf
spark:webhooks_write	Modify or delete webhooks created on your behalf

The following sections examine some of the APIs you can use to create rooms, add people, and send messages in a room.

Organizations API

An organization is a set of people in Webex Teams. Organizations may manage other organizations or may be managed themselves. The Organizations API can be accessed only by an administrator. **Table 10-3** shows the methods used with the Organizations API to get details about organizations.

Table 10-3 Webex Teams: Organization API

MethodAPIDescription		
G E T	<a href="https://webexapis.com/v1/org
anizations">https://webexapis.com/v1/org anizations	List all the organizations

GET	https://webexapis.com/v1/organizations/{orgId}	Get details about an organization
-----	---	-----------------------------------

Note

The host name <https://api.ciscospark.com> has now been changed to <https://webexapis.com>. The old <https://api.ciscospark.com> will continue to work.

Teams API

A team is a group of people with a set of rooms that is visible to all members of that team. The Teams API is used to manage teams—to create, delete, and rename teams. [Table 10-4](#) lists the various operations that can be performed on the Teams API.

Table 10-4 Webex Teams: Teams API

Method	API	Description
GET	https://webexapis.com/v1/teams	List all teams
POST	https://webexapis.com/v1/teams	Create a new team
GET	https://webexapis.com/v1/teams/{teamId}	Get details about a particular team
PUT	https://webexapis.com/v1/teams/{teamId}	Update details about a team
DELETE	https://webexapis.com/v1/teams/{teamId}	Delete a team

For example, say that you want to use the Teams API to create a new team named DevNet Associate Certification Room. To do so, you use the POST method and the API <https://webexapis.com/v1/teams>.

You can use a Python request to make the REST call. [Example 10-1](#) shows a Python script that sends a POST request to create a new team. It initializes variables such as the base URL, the payload, and the headers, and it calls the request.

Example 10-1 *Python Code to Create a New Team*

[Click here to view code image](#)

```
""" Create Webex Team """
import json
import requests

URL = "https://webexapis.com/v1/teams"
PAYLOAD = {
    "name": "DevNet Associate Certification
Team"
}
HEADERS = {
    "Authorization": "Bearer
MDAOY2V1MzktNDc2Ni00NzI5LWFiNmYtZmNmYzZM30TkyNjMxNmI0ND-
VmNDktNGE1-PF84_consumer",
    "Content-Type": "application/json"
}
RESPONSE = requests.request("POST", URL,
data=json.dumps(PAYLOAD), headers=HEADERS)
print(RESPONSE.text)
```

Rooms API

Rooms are virtual meeting places where people post messages and collaborate to get work done. The Rooms API is used to manage rooms—to create, delete, and rename them. [Table 10-5](#) lists the operations that can be performed with the Rooms API.

Table 10-5 Webex Teams: Rooms API

MethodAPIDescription		
GET	https://webexapis.com/v1/rooms	List all the rooms
POST	https://webexapis.com/v1/rooms	Create a new room
GET	https://webexapis.com/v1/rooms/{roomId}	Get room details
GET	https://webexapis.com/v1/rooms/{roomId}/meetingInfo	Get room meeting details
PUT	https://webexapis.com/v1/rooms/{roomId}	Update room details
DELETE	https://webexapis.com/v1/rooms/{roomId}	Delete a room

You can use the Rooms API to create a room. When you do, an authenticated user is automatically added as a member of the room. To create a room, you can use the POST method and the

API <https://webexapis.com/v1/rooms>.

Example 10-2 show Python request code that creates a room with the name DevAsc Team Room. It initializes variables such as the base URL, the payload, and the headers, and it calls the request. The header consists of the bearer token of the authenticated user or the integration along with other parameters.

Example 10-2 *Python Request to Create a New Room*

[Click here to view code image](#)

You can use the Rooms API to get a list of all the rooms that have been created. To do so, you can use the GET method and the API <https://webexapis.com/v1/rooms>.

Example 10-4 shows how to use the **curl** command to make the REST call. This script sends a GET request to list all rooms that a particular user belongs to.

Example 10-4 *curl Script for Getting a List of All Rooms*

[Click here to view code image](#)

```
$ curl -X GET \
  https://webexapis.com/v1/rooms \
  -H 'Authorization: Bearer
DeadBeefMTAtN2UzZi00YjRiLWIzMGEtMThjMzliNWQwZGEyZTljN-
WQxZTkNTRl_Pf84_1eb65fdf-9643-417f-9974-
ad72cae0e10f'
```

Memberships API

A membership represents a person’s relationship to a room. You can use the Memberships API to list members of any room that you’re in or create memberships to invite someone to a room. Memberships can also be updated to make someone a moderator or deleted to remove someone from the room. **Table 10-6** lists the operations that can be performed with respect to the Memberships API, such as listing memberships and adding a new member.

Table 10-6 Webex Teams: Memberships API

Method	API	Description
GET	https://webexapis.com/v1/memberships	List memberships
POST	https://webexapis.com/v1/mem	Add a new

ST	berships	member
GET	https://webexapis.com/v1/memberships/{membershipId}	Get details about a member
PUT	https://webexapis.com/v1/memberships/{membershipId}	Update details about a member
DELETE	https://webexapis.com/v1/memberships/{membershipId}	Delete a member

You can use the Memberships API to add a new member to a given room (that is, create a new membership) by using the POST method and the API <https://webexapis.com/v1/memberships>.

You can use Python to make a REST call. [Example 10-5](#) shows a **curl** script that sends a POST request to add a new member with email-id newUser@devasc.com to the room.

Example 10-5 *Python Script to Add a New Member to a Room*

[Click here to view code image](#)

```

""" Add new Member to a Webex Room """

import json
import requests
import pprint

URL = "https://webexapis.com/v1/memberships"
PAYLOAD = {
    "roomId" :
    "Y21zY29zcGFyazovL3VzL1JPT00vY2FhMzJiYTA0NTA0OC0xMmVhLWJiZ-
    WItYmY1MWQyNGRDEADB",
    "personEmail": "newUser@devasc.com",
    "personDisplayName": "Cisco DevNet",
    "isModerator": "false"
}
HEADERS = {
    "Authorization": "Bearer

```

```

MDA0Y2VlMzktNDc2Ni00NzI5LWFiNmYtZmNmYzZM30TkyNjMxNmI0ND-
    VmNDktNGE1_Pf84_consumer",
    "Content-Type": "application/json"
  }
  RESPONSE = requests.request("POST", URL,
    data=json.dumps(PAYLOAD), headers=HEADERS)
  pprint.pprint(json.loads(RESPONSE.text))

```

Messages API

Messages are communications that occur in a room. In Webex Teams, each message is displayed on its own line, along with a timestamp and sender information. You can use the Messages API to list, create, and delete messages.

Message can contain plaintext, rich text, and a file attachment. [Table 10-7](#) shows the API for sending messages to Webex Teams.

Table 10-7 Webex Teams: Message API

Method	API	Description
GET	https://webexapis.com/v1/messages	List messages
GET	https://webexapis.com/v1/messages/direct	List one-to-one message
POST	https://webexapis.com/v1/messages	Post a new message
GET	https://webexapis.com/v1/messages/{messageId}	Get details about a message
DELETE	https://webexapis.com/v1/messages/{messageId}	Delete a message

You can use the Messages API to add a new member to a given room (that is, create a new membership). To do so, you use the POST method and the

API <https://webexapis.com/v1/messages>.

You can use a Python request to make a REST call.

Example 10-6 shows a **curl** script that sends a POST message to add a new message to a particular room.

Example 10-6 Python Script to Add a New Message to a Room

[Click here to view code image](#)

```
""" Send Webex Message """

import json
import requests
import pprint

URL = "https://webexapis.com/v1/messages"
PAYLOAD = {
    "roomId" :
    "Y21zY29zcGFyazovL3VzL1JPT00vY2FhMzJiYTA0NTA0OC0xMwYhLWJiZ-

    WItYmY1MWQyNGRmMTU0",
    "text" : "This is a test message"
}
HEADERS = {
    "Authorization": "Bearer
NDkzODZkZDUtZDExNC00DM5LTk0YmYtZmY4NDI0ZTE5ZDA1MGI-

    5YTY30WUtZGYy_PF84_consumer",
    "Content-Type": "application/json",
}
RESPONSE = requests.request("POST", URL,
data=json.dumps(PAYLOAD), headers=HEADERS)
pprint.pprint(json.loads(RESPONSE.text))
```

Bots

A *bot* (short for chatbot) is a piece of code or an application that simulates a human conversation. Users communicate with a bot via the chat interface or by voice, just as they would talk to a real person. Bots help

users automate tasks, bring external content into the discussion, and gain efficiencies. Webex Teams has a rich set of APIs that make it very easy and simple for any developer to add a bot to any Teams room. In Webex, bots are similar to regular Webex Teams users. They can participate in one-to-one and group spaces, and users can message them directly or add them to a group space. A special badge is added to a bot's avatar in the Webex Teams clients so users know they're interacting with a bot instead of a human.

A bot can only access messages sent to it directly. In group spaces, bots must be @mentioned to access a message. In one-to-one spaces, a bot has access to all messages from the user. Bots are typically of the three types described in [Table 10-8](#).



Table 10-8 Bot Types

Type	Description
Notification bot	Events from external services are brought in and posted in Webex Teams. Examples of events include build complete, retail inventory status, and temperature today.
Control bot	External systems that have APIs allow third-party apps to be integrated to control them. For example, you could control turning lights on or off by invoking a lights bot.

ll e r b o t	
A s s i s t a n t b o t	Virtual assistants usually understand natural language, so a user can ask questions of bots as they would ask humans (for example, “@Merakibot, how many wifi devices are currently on floor 2?”)

Bot Frameworks and Tools

There are several bot frameworks that can greatly simplify the bot development process by abstracting away the low-level communications with the Webex REST API, such as creating and sending API requests and configuring webhooks. You can focus on building the interaction and business logic of a bot. These are two popular bot frameworks:

- **Flint:** Flint is an open-source bot framework with support for regex pattern matching for messages and more.
- **Botkit:** Botkit is a popular open-source bot framework with advanced conversational support as well as integrations with a comprehensive array of natural language processing and storage providers.

One of the greatest starting points for learning about and creating your own bots for Webex Teams is the DevNet Code Exchange, at

<https://developer.cisco.com/codeexchange/github/repo/howdyai/botkit>, which is shown in [Figure 10-8](#).

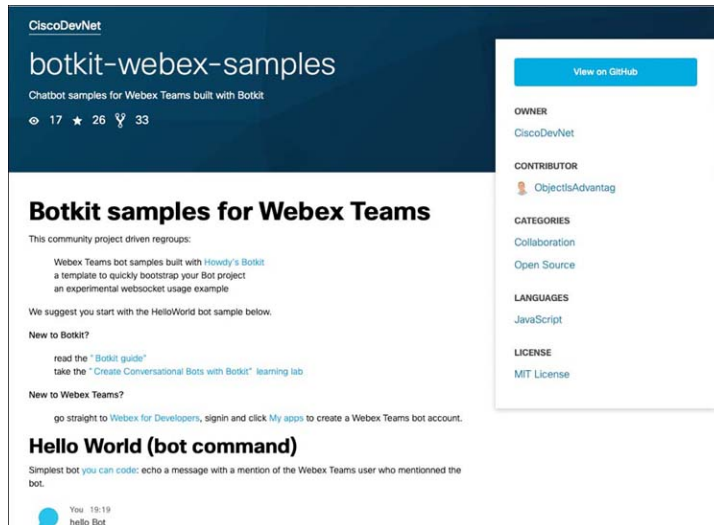


Figure 10-8 DevNet Code Exchange: Building Your First Bot

Guest Issuer

Guest issuer applications give guest users temporary access to users within the organization. Guest issuers can be created at <https://developer.Webex.com/my-apps/new/guest-issuer>. To create a new guest issuer, the only thing that is required is the name. A new guest issuer ID and shared secret will be generated and can be used subsequently. The main reason to use a guest issuer is to interact with users who do not have a Webex Teams account. These users might be visitors to a website who you'd like to message with over Webex Teams. Or they might be customers in a store with which you'd like to have a video call. These guest users can interact with regular Webex Teams users via tokens generated by a guest issuer application.

Guest users of Webex Teams authenticate by using guest tokens. Guest tokens use the JSON Web Token (JWT) standard to create and share authentication credentials between SDKs and widgets and the Webex REST API. These tokens are exchanged for an access authentication token that can be used for a limited time and limited purpose to interact with regular Webex Teams users.

Each guest token should be associated with an individual user of an application. The guest's activity within Webex Teams, such as message activity or call history, will persist, just as it would for a regular Webex Teams user. While guest users can interact with regular Webex Teams users, they are not allowed to interact with other guests. [Example 10-7](#) shows a Python code snippet that creates a JWT token from the guest issuer ID and secret and passes it in the authentication headers. It is then possible to use any of the APIs to interact with other users in the system.

Example 10-7 *Python Code to Generate a JWT Token for a Guest Issuer*



[Click here to view code image](#)

```
""" Generate JWT """

import base64
import time
import math
import jwt

EXPIRATION = math.floor(time.time()) + 3600 # 1
hour from now
PAYLOAD = {
    "sub": "devASC",
    "name": "devASC-guest",
    "iss": "GUEST_ISSUER_ID",
    "exp": EXPIRATION
}

SECRET = base64.b64decode('GUEST_ISSUE_SECRET')

TOKEN = jwt.encode(PAYLOAD, SECRET)

print(TOKEN.decode('utf-8'))
HEADERS = {
    'Authorization': 'Bearer ' +
    TOKEN.decode('utf-8')
}
```


Webex Teams SDKs

As of this writing, there is a variety of SDKs available; some of them are official Webex Teams SDKs, and others are from the community. The following is a selection of the Web Teams SDKs that are available:

- **Go (go-cisco-Webex-teams):** A Go client library (by jbogarin)
- **Java (spark-java-sdk):** A Java library for consuming the RESTful APIs (by Cisco Webex)
- **Node.js (ciscospark):** A collection of Node.js modules targeting the REST API (by Cisco Webex)
- **PHP (SparkBundle):** A Symfony bundle (by CiscoVE)
- **Python (Webexteamssdk):** An SDK that works with the REST APIs in native Python (by cmlccie)

The following are some advanced APIs:

- **SDK for Android:** Integrates messaging and calling into Android apps (by Cisco Webex)
- **SDK for Browsers:** Integrates calling into client-side JavaScript applications (by Cisco Webex)
- **SDK for iOS:** Integrates messaging and calling into iOS apps (by Cisco Webex)
- **SDK for Windows:** Integrates messaging and calling into Windows apps (by Cisco Webex)
- **Widgets:** Provides components that mimic the web user experience (by Cisco Webex)

CISCO FINESSE

The Cisco Finesse desktop is a call agent and supervisor desktop solution designed to meet the growing needs of agents, supervisors, and the administrators and developers who support them. The Cisco Finesse desktop runs in a browser, which means you install Cisco Unified Contact Center Express (Unified CCX), and agents start by simply typing in the URL for the Unified CCX server. The desktop is more than an agent state and call-control application. It is an OpenSocial gadget container, built to include third-party applications in a single agent desktop experience. Rather than switching between applications, agents have easy access to all applications and tools from

a single window, which increases their efficiency. **Figure 10-9** shows the architecture and high-level flow of Finesse, which involves the following steps:

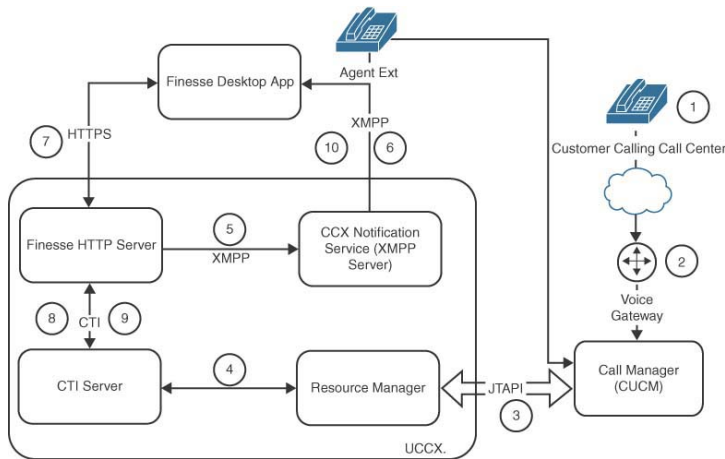


Figure 10-9 *Finesse High-Level Flow*

- Step 1.** The call arrives from either the PSTN or a VoIP connection to the gateway.
- Step 2.** The gateway then hands over the call to Unified CM, which invokes the application that was preregistered. In this case, it is handled by Unified CCX.
- Step 3.** Unified CM notifies the Unified CCX about the incoming call via JTAPI.
- Step 4.** After consulting the resource manager (routing the call to the agent based on skills, priority, and rebalancing), Unified CCX notifies Finesse via computer telephony integration (CTI) connection.
- Step 5.** Finesse performs internal processing and then publishes a notification to Notification Service.
- Step 6.** The Finesse desktop receives this notification from Notification Service via the Bidirectional-

streams Over Synchronous HTTP (BOSH) connection (Extensible Messaging and Presence Protocol [XMPP]).

Step 7. The agent makes a request to perform an operation (such as answer a call); the app makes this HTTP (REST) request to Finesse Web Services.

Step 8. Finesse processes the request and then, if necessary, requests action with Unified CCE via CTI connection. Where applicable, Unified CCE performs/forwards the requested action.

Step 9. Unified CCE notifies Finesse via CTI connection about whether the request was successful or caused an error.

Step 10. Finesse processes the notification (successful/error) and publishes the notification to Notification Service. The agent web browser receives this notification from Notification Service via the BOSH connection.

The Finesse agent goes through various states that specifically pertain to the agent workflow (see [Table 10-9](#)).

Table 10-9 User States in Finesse

State Description	
LO GI N	The agent is signing in to the system. This is an intermediate state.
LO GO UT	The agent is signed off the system.
RE AD Y	The agent is ready to take calls.

NO T_ RE AD Y	The agent is signed in but not ready to take calls. It could be on a break, or the shift might be over, or the agent might be in between calls.
RE SE RV ED	This is a transient state, as the agent gets chosen but has not answered the call.
TA LK IN G	The agent is on a call.
H OL D	The agent puts the call on hold.

Cisco Finesse API

The Cisco Finesse API is a modern, open-standards-based web API, exposed via REST. Each function available in the Cisco Finesse user interface has a corresponding REST API that allows all types of integrations for developers to use. The extensibility and ease of use of the API are unprecedented on Unified CCX. Agents and supervisors use the Cisco Finesse desktop APIs to communicate between the Finesse desktop and Finesse server, and they use Unified Contact Center Enterprise (Unified CCE) or Unified Contact Center Express (Unified CCX) to send and receive information.

The Finesse APIs can be broadly classified into the following categories:

- User
- Dialog

- Queue
- Team
- ClientLog
- Task Routing APIs
- Single Sign-On
- TeamMessage

Cisco Finesse supports both HTTP and HTTP Secure (HTTPS) requests from clients. Cisco Finesse desktop operations can be performed using one of the many available REST-like HTTP/HTTPS requests. Operations on specific objects are performed using the ID of the object in the REST URL. For example, the URL to view a single object (HTTP) would be as follows:

```
http://<FQDN>:  
<port>/finesse/api/<object>/<objectID>
```

where FQDN is the fully qualified domain name of the Finesse server.

Finesse configuration APIs require the application user ID and password, which are established during installation, for authentication purposes.

Finesse APIs use the following HTTP methods to make requests:

- **GET:** Retrieves a single object or list of objects (for example, a single user or list of users).
- **PUT:** Replaces a value in an object (for example, to change the state of a user from NOT_READY to READY).
- **POST:** Creates a new entry in a collection (for example, to create a new reason code or wrap-up reason).
- **DELETE:** Removes an entry from a collection (for example, to delete a reason code or wrap-up reason).

Finesse uses the standard HTTP status codes (for example, 200, 400, and 500) in the response to indicate the overall success or failure of a request.

API Authentication

All Finesse APIs use HTTP BASIC authentication, which requires the credentials to be sent in the authorization header. The credentials contain the username and password, separated by a single colon (:), within a Base64-encoded string. For example, the authorization header would contain the following string:

[Click here to view code image](#)

```
"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"
```

where ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk is the Base64-encoded string *devasc:strongpassword* (where *devasc* is the username, and *strongpassword* is the password). [Example 10-8](#) shows three lines of code that do Base64 encoding in order to plug the value in the authorization headers.

Example 10-8 *Python Code to Generate Base64 Encoding*

[Click here to view code image](#)

```
""" Generate Base64 Encoding """
import base64
ENCODED =
base64.b64encode('devasc:strongpassword'.encode('UTF-8'))
print(ENCODED.decode('utf-8'))
```

With Single Sign-On mode, the authorization header would contain the following string:

```
"Bearer <authtoken>"
```

Finesse User APIs

[Table 10-10](#) lists the various methods and User APIs to perform operations with the user, such as listing, logging in, and changing properties.

Table 10-10 Finesse User APIs

MethodAPIDescription		
GET	<p>http://<FQDN>/finesse/api/User/<id></p>	Get a copy of the user object
GET	<p>http://<FQDN>/finesse/api/User</p>	Get a list of all users
PUT	<p>http://<FQDN>/finesse/api/User/<id></p> <p>with XML body:</p> <pre><User> <state>LOGIN</state> <extension>5250001</extension> </User></pre>	Sign in to the CTI server
PUT	<p>http://<FQDN>/finesse/api/User/<id></p> <p>with XML body:</p> <pre><User> <state>READY</state> </User></pre>	Set the user's state: <ul style="list-style-type: none"> • READY • NOT_READY • LOGOUT
GET	<p>http://<FQDN>/finesse/api/User/<id>/PhoneBooks</p>	Get a list of phone books and the first 1500 associated contacts for that user

A full list of all User state change APIs with details can be found at <https://developer.cisco.com/docs/finesse/#!userchange-agent-state/userchange-agent-state>.

For example, the User—Sign in to Finesse API forces you to sign-in. Say that you use the following information with this API:

- **Finesse server FQDN:** <http://hq-uccx01.abc.inc>
- **Agent name:** Anthony Phyllis
- **Agent ID:** user001
- **Agent password:** cisco1234

[Example 10-9](#) shows a simple call using Python requests. The API call for user login uses the PUT request along with an XML body that sets the state to LOGIN.

Example 10-9 *Python Request: Finesse User Login*

[Click here to view code image](#)

```
""" Finesse - User Login"""
import requests

URL = "http://hq-uccx.abc.inc:8082/finesse/api/User/Agent001"
PAYLOAD = (
    "<User>" +
    "    <state>LOGIN</state>" +
    "    <extension>6001</extension>" +
    "</User>"
)

HEADERS = {
    'authorization': "Basic
QWdlbnQwMDE6Y2lzY29wc2R0",
    'content-type': "application/xml",
}

RESPONSE = requests.request("PUT", URL,
data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```


As another example, the User State Change API lets the users change their state. State changes could be any one as shown in [Table 10-9](#). Say that you use the following information with this API:

- **Finesse server FQDN:** <http://hq-uccx01.abc.inc>
- **Agent name:** Anthony Phyllis
- **Agent ID:** user001
- **Agent password:** cisco1234

The API changes the state to READY.

[Example 10-10](#) shows a simple call using Python requests. The API call for user login uses the PUT request along with an XML body that sets the state to READY.

Example 10-10 *Python Request for a Finesse User State Change*

[Click here to view code image](#)

```
""" Finesse - User State Change"""
import requests

URL = "http://hq-uccx.abc.inc:8082/finesse/api/User/Agent001"
PAYLOAD = (
    "<User>" +
    "  <state>READY</state>" +
    "</User>"
)

HEADERS = {
    'authorization': "Basic
QWdlbnQwMDE6Y2lzY29wc2R0",
    'content-type': "application/xml",
}

RESPONSE = requests.request("PUT", URL,
data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```

Finesse Team APIs

The Team object represents a team and contains the URI, the team name, and the users associated with the team. **Table 10-11** shows the Finesse Team APIs to access the Team object and list all team messages.

Table 10-11 Finesse Team APIs

Method	API	Description
GET	http://<FQDN>/finesse/api/Team/<id>?includeLoggedOutAgents=true	Allow a user to get a copy of the Team object
GET	http://<FQDN>/finesse/api/Team/<teamid>/TeamMessages	Get a list of all active team messages for a particular team

A full list of the Team APIs, with details, can be found at <https://developer.cisco.com/docs/finesse/#team-apis>.

The Python script in **Example 10-11** shows how to make an API call to get details about Team ID 2.

Example 10-11 *Python Request to Get Finesse Team Details*

[Click here to view code image](#)

```
import requests
url = "https://hq-uccx.abc.inc:8445/finesse/api/Team/2"
headers = {
    'authorization': "Basic QWdlbnQwMDE6Y2lzY29wc2R0",
    'cache-control': "no-cache",
}
response = requests.request("GET", url, headers=headers)
print(response.text)
```

Dialog APIs

The Dialog object represents a dialog (voice or non-voice) between two or more users. There are many flavors of dialog APIs. Table 10-12 shows the Finesse dialog API, which lets two users make a call with each other.

Table 10-12 Sample of Finesse Dialog APIs

MethodAPIDescription		
POST	<p>http://<FQDN>/finesse/api/User/<id>/Dialogs</p> <p>with XML Body:</p> <pre><Dialog> <requestedAction>MAKE_CALL</requestedAction> <fromAddress>6001</fromAddress> <toAddress>6002</toAddresses> </Dialog></pre>	Allow a user to make a call
PUT	<p>http://<FQDN>/finesse/api/Dialog/<dialogId></p> <p>with XML body:</p> <pre><Dialog> <requestedAction>START_RECORDING</requestedAction> <targetMediaAddress>6001</targetMediaAddress> </Dialog></pre>	Allow a user to start recording an active call

A full list of the Dialog APIs, with details, can be found at <https://developer.cisco.com/docs/finesse/#dialog-apis>.

The Python script in [Example 10-12](#) shows how to make an API call to make a call between extension 6001 and extension 6002.

Example 10-12 *Python Request to Initiate a Dialog Between Two Numbers*

[Click here to view code image](#)

```
""" Finesse - Initiate a dialog between two
numbers """
import requests

URL = "http://hq-
uccx.abc.inc:8082/finesse/api/User/Agent001/Dialogs"

PAYLOAD = (
    "<Dialog>" +
    "
    <requestedAction>MAKE_CALL</requestedAction>" +
    "    <fromAddress>6001</fromAddress>" +
    "    <toAddress>6002</toAddress>" +
    "</Dialog>"
)

HEADERS = {
    'authorization': "Basic
QWdlbnQwMDE6Y21zY29wc2R0",
    'content-type': "application/xml",
    'cache-control': "no-cache",
}

RESPONSE = requests.request("POST", URL,
data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```

Finesse Gadgets

As indicated earlier in this chapter, the Finesse desktop application is an OpenSocial gadget container. This means that an agent or anyone else can customize what is on the desktop. Gadgets are built using HTML, CSS,

and JavaScript. A gadget is a simple but powerful tool that allows an agent to quickly send one or more web pages (carefully curated by the client teams themselves) to their caller over Instant Messenger or via email. This feature automates the four or five manual steps usually associated with this task.

A gadget is defined using declarative XML syntax that is processed by a gadget server so that it can be embedded into the following contexts:

- Standalone web pages
- Web applications
- Other gadgets

The Finesse JavaScript library can be found on a Finesse server at `http(s)://<FQDN>:<port>/desktop/assets/js/doc/index.html`.

Several gadgets are available at <https://developer.cisco.com/docs/finesse/#!sample-gadgets/sample-gadgets>.

WEBEX MEETINGS APIS

The Webex Meetings APIs let users incorporate Cisco Webex meetings into their own applications:

- **URL API:** The URL API is a convenient lightweight, HTTP/HTTPS-based mechanism that provides browser-based external hooks into Webex Meetings services. The URL API is typically used in enterprise portal integrations to support basic interactions such as single sign-on (SSO), scheduling meetings, starting and joining simple meetings, and inviting attendees and presenters. [Figure 10-10](#) provides a high-level overview of where the URL API integration layer resides.

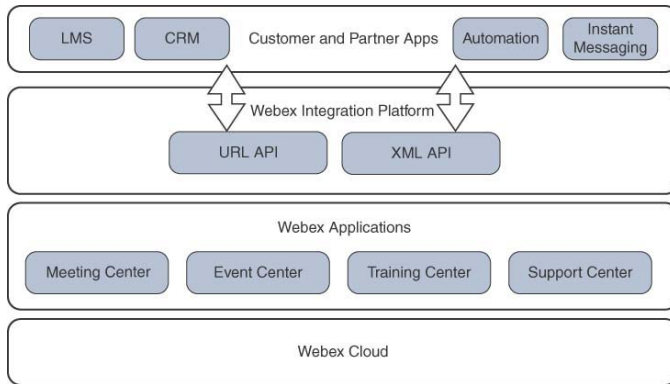


Figure 10-10 Webex Meetings API Architecture

- XML API:** If more advanced integration is needed than is possible with the URL API, Cisco strongly recommends using the Webex Meetings XML API. The XML API is a comprehensive set of services that supports most aspects of Webex Meetings services, including detailed user management, comprehensive scheduling features, and attendee management and reporting. The Webex XML API uses a service-oriented architecture (SOA) to provide comprehensive services to external applications wishing to interact with one or more Webex services.
- Teleconference Service Provider (TSP) API:** The TSP API provides full-featured XML-based integration with audio conferencing networks. The integration architecture supports redundancy, failover, and load balancing, and it provides robust features that tightly integrate audio conferencing capabilities with telephony management and provisioning in the Webex environment.

Table 10-13 summarizes the available URL API services and simple commands for integrating with a Webex Meetings–hosted website.



Table 10-13 Webex Meetings Services Supported via APIs

Service Name	Integration Usage
Managing User Accounts	Creating a new user account Editing an existing user account

	Activating and deactivating user accounts
Webex-hosted website login/logout	Using an authenticated server-to-server connection for logging in to and out of a Webex Meetings–hosted website
Using and Managing Meetings	<p>Scheduling a meeting</p> <p>Editing a meeting</p> <p>Starting or deleting a host’s scheduled meeting</p> <p>Listing all scheduled meetings</p> <p>Listing all open meetings</p> <p>Joining an open meeting</p>
Modifying My Webex Meetings page	<p>Modifying user information on the My Webex Meetings page</p> <p>Managing a user’s My Contacts list</p>
Using Attendee Registration forums	<p>Creating a registration form</p> <p>Determining the current required, optional, or do-not-display settings for a registration page</p> <p>Adding check boxes, buttons, and drop-down lists to a registration form</p>
Managing Attendee Lists	<p>Adding attendees to a list of invited users</p> <p>Removing attendees from a list of invited users</p>
Playing back a recorded event	Allowing an attendee to get a list of recorded events for playback
Querying for Questions and Answers	Viewing a list of custom questions created by the host

	<p>Viewing attendees' answers to custom questions</p> <p>Viewing a list of standard questions created by the host</p> <p>Viewing attendees' answers to standard questions</p>
<p>Making Recording Training Sessions available for viewing</p>	<p>Making all aspects of a previously recorded Training Center session available for later playback</p>
<p>Reporting</p>	<p>Sending email notifications with attendee information</p> <p>Displaying reporting information about training sessions you hosted</p> <p>Viewing a list of enrollees and attendees for events</p> <p>Viewing a list of all events a specific attendee has joined</p> <p>Viewing a list of people who have downloaded files</p>

Authentication

There are three methods by which a user can interact with the Webex Meetings via APIs:

- **Webex Meetings integration:** An application can authenticate XML API requests using OAuth 2.0 access tokens. This authentication method does not require end-user authorization and authentication and is best used when performing API functions on behalf of the active user.
- **Administrative account:** A system account can perform administrative functions on behalf of host users. This method requires the username and password to be passed via XML.
- **User accounts:** A system account can perform functions only for its own accounts. This method requires the username and password to be passed via XML.

Integration API Keys

In order to use the Webex Meetings APIs, you need to preregister an API key and then use the HM256 algorithm to generate a JWT token to access a Webex anonymous API (e.g., **GetSessionInfoAgg**, **GetAllSitesByEmailAgg**).

To apply for an API key, visit <https://api.Webex.com/gapi/registerapikey>. After you apply for the key, you get the API and the secret (see [Example 10-13](#)).

Example 10-13 *Webex Meeting Integration Key Obtained via the Portal*

[Click here to view code image](#)

```
Name: myname mylastname
Email: myname@devasc.com
New API key: deadbeaf-d1e3-4000-993f-0d9479ab4944
New Passcode: 2BEEF
New Secret key:
DEADBEEFNYH54RP8TH0JSXM80JKOUJVHXKGGK7QQZSL1KF2D967JY5XZIR64B3GY5N-
GRWMBVXG132XDGT5XJ62VQU4558I13J8IQ7TPEIPFB9MNG48WJM67F5C82P01VYTQTT4428T7RYQN0-
YMSHXAX8ARLVQHHPNCCH1RJZYRIUF3AI000A6BA3BQN54DJ9VN3V9XPKUZZM1E570X07IKTSHDVTWPW8CW-
MU5Y9XZG76Z9FFA5UN1DJ7N39RXQE99TRFA5HDBL2BJBPUOX6NMY6BDBZ4JP2IQ3RP3D6D7CUBP5U7W5EK-
BWVTRZQXBY0L4M98Y1BR9HX5FD9D9HWYA
Expiration time: 4/27/20 6:22 AM GMT
```

Webex XML APIs

The Webex XML model utilizes the exchange of well-formed XML documents to deploy messages. Each of these messages relates to a specific Webex operation. An XML request document specifies the desired action of the particular service's elements. The XML response document returned by the Webex XML server describes the revised state of that service, as determined by the Webex XML server.

Figure 10-11 provides a simplified representation of the architecture of the Webex model.

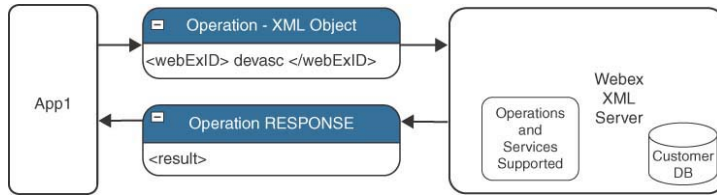


Figure 10-11 *Webex XML Model Architecture*



As shown in Figure 10-11, an application sends a request XML document to perform a specific request—such as creating a new meeting—to the Webex XML server. This request XML document describes the state of the elements associated with the operation for this request. The Webex XML server then returns to the original application the revised element values for this operation via a response XML document created by the Webex XML document processor.

The Webex XML server can be accessed at <https://api.Webex.com/WBXService/XMLService>.

Creating a New Meeting

The CreateMeeting API enables users to schedule a meeting. This API returns a unique meeting key for the session. Table 10-14 shows the API and the XML body to create a new meeting.

Table 10-14 Creating or Scheduling a Webex Meeting

MethodAPIDescription		
P	https://api.Webex.com/WBXService/XMLService	Create a new meeting with various attributes such as
O		
e		

S T	with XML body:	name, start time, password, and attendees
	<pre> <body> <bodyContent xsi:type="java.com.Web ex.service.binding.meeti ng.CreateMeeting"> <metaData> <confName>Br anding Meeting</confName> </metaData> <schedule> <startDate/> </schedule> </bodyContent> </body> </pre>	

Example 10-14 shows how to make an API call to create a new meeting with a particular subject line. The response is usually the new meeting ID.

Example 10-14 *Creating a New Meeting Using the CreateMeeting API*

[Click here to view code image](#)

```

curl -X POST \
  https://api.Webex.com/WBXService/XMLService \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/xml' \
  -d '<?xml version="1.0" encoding="UTF-8"?>
<serv:message
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <header>
    <securityContext>
      <WebexID>devasc</WebexID>

```

```

        <password>Kc5Ac4M1</password>
        <siteName>apidemoeu</siteName>
    </securityContext>
</header>
<body>
    <bodyContent
xsi:type="java:com.Webex.service.binding.meeting.

        CreateMeeting">
            <metaData>
                <confName>Branding
Meeting</confName>
            </metaData>
            <schedule>
                <startDate/>
            </schedule>
        </bodyContent>
    </body>
</serv:message>'

```

Listing All My Meetings Meeting

The LstsummaryMeeting API lists summary information for scheduled meetings. This API returns a unique meeting key for a session. [Table 10-15](#) shows the XML data that needs to be sent in order to list all meetings sorted by start time.

Table 10-15 Listing All Webex Meetings That Match Certain Criteria

MethodAPIDescription		
P O S T	https://api.Webex.com/WBXService/XMLService with XML body: <pre> <body> <bodyContent xsi:type="java:com.Webex.service.binding.meeting.LstsummaryMeeting"> <order> </pre>	List all meetings that meet certain criteria

```
<orderBy>STARTTIME</ord
erBy>

</order>

</bodyContent>

</body>
```

Example 10-15 shows how to make an API call to list meetings sorted by time.

Example 10-15 *Listing All Meetings for a User by Using the LstsummaryMeeting API*

[Click here to view code image](#)

```
curl -X POST \
  https://api.Webex.com/WBXService/XMLService \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/xml' \
  -d '<serv:message
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <header>
    <securityContext>
      <WebexID>devasc</WebexID>
      <password>Kc5Ac4Ml</password>
      <siteName>apidemoeu</siteName>
    </securityContext>
  </header>
  <body>
    <bodyContent

xsi:type="java:com.Webex.service.binding.meeting.LstsummaryMeeting">

    <order>
      <orderBy>STARTTIME</orderBy>
    </order>
  </bodyContent>
</body>
</serv:message>'
```

Setting or Modifying Meeting Attributes

The SetMeeting API enables hosts/users to update the information for a scheduled meeting that they are able to edit. Table 10-16 shows the XML data that needs to be sent in order to modify meeting attributes.

Table 10-16 Modify Meeting Attributes

MethodAPIDescription		
P O S T	<p>https://api.Webex.com/WBXService/XMLService with XML body:</p> <pre><bodyContent xsi:type="java:com.Webex.service.binding.meeting.SetMeeting"> <meetingkey>625579604</meetingkey> <participants> <attendees> <attendee> <person> <email>student@devasc.com</email> </person> </attendee> </attendees> </bodyContent></pre>	<p>Update meeting attributes such as attendees, record meeting, schedules, and media support</p>

Example 10-16 shows how to make an API call to add a new attendee (student@devasc.com) to the meeting and send an email invitation to that person.

Example 10-16 *Adding a New User to a Meeting by Using the SetMeeting API*

[Click here to view code image](#)

```
curl -X POST \  
  https://api.Webex.com/WBXService/XMLService \  
  -H 'cache-control: no-cache' \  
  -H 'content-type: application/xml' \  
  -d '<serv:message  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance">  
  <header>  
    <securityContext>  
      <WebexID>devasc</WebexID>  
      <password>Kc5Ac4Ml</password>  
      <siteName>apidemoeu</siteName>  
    </securityContext>  
  </header>  
<body>  
  <bodyContent  
  
xsi:type="java:com.Webex.service.binding.meeting.SetMeeting">  
  
    <meetingkey>625579604</meetingkey>  
    <participants>  
      <attendees>  
        <attendee>  
          <person>  
            <email>student@devasc.com</email>  
          </person>  
        </attendee>  
      </attendees>  
    </participants>  
    <attendeeOptions>  
  
    <emailInvitations>true</emailInvitations>  
    </attendeeOptions>  
    <schedule>  
      <openTime>300</openTime>  
    </schedule>  
  </bodyContent>  
</body>  
</serv:message>'
```

Deleting a Meeting

The DelMeeting API allows hosts to delete a meeting that is not currently in progress. The API continues to use the POST method, but the XML data contains the operation of deleting the meeting. [Table 10-17](#) shows the XML data that needs to be sent in order to delete a meeting.

Table 10-17 Deleting a Webex Meeting Using the Meeting Key

Method	API Description	
POST	<p>https://api.Webex.com/WBXService/XMLService with XML body:</p> <pre><body> <bodyContent xsi:type="java:com.Webex.service.binding.meeting.DelMeeting"> <meetingKey>625579604</meetingKey> </bodyContent> </body></pre>	Delete or cancel a meeting, given the meeting ID

[Example 10-17](#) shows how to make an API call to delete the meeting with ID =625579604.

Example 10-17 *Deleting a Meeting Using the DelMeeting API*

[Click here to view code image](#)

```
curl -X POST \
  https://api.Webex.com/WBXService/XMLService \
  -H 'cache-control: no-cache' \
```



```

-H 'content-type: application/xml' \
-d '<serv:message
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <header>
    <securityContext>
      <WebexID>devasc</WebexID>
      <password>Kc5Ac4M1</password>
      <siteName>apidemoeu</siteName>
    </securityContext>
  </header>
  <body>
    <bodyContent

xsi:type="java:com.Webex.service.binding.meeting.DeleteMeeting">

      <meetingKey>625579604</meetingKey>
    </bodyContent>
  </body>
</serv:message>'

```

The Meetings XML API provides a method to integrate Webex Meetings services with your custom web portal or application. For more details about the Webex Meetings APIs, see <https://developer.cisco.com/docs/Webex-meetings/#!Webex-meetings>.

WEBEX DEVICES

Webex Devices enables users to communicate and work with each other in real time. The collaboration devices can be placed in meeting rooms or at desks. Except for the high-quality audio and video support, these devices are fully programmable. Through embedded APIs (often referenced as xAPI), you can extend and leverage Webex Room and Desk device capabilities in several ways:

- Create user interface components to customize the meeting experience and control IoT devices such as lights, curtains, and projectors from the device controls interface
- Configure devices at scale via automation scripts
- Initiate or respond to calls automatically
- Deploy code onto the devices without needing to deploy external control systems

Webex Devices includes the following devices, as shown in [Figure 10-12](#):

- **Webex Board:** Cisco Webex Room Kit, Room Kit Plus, and Room Kit Pro
- **Room Devices:** Room 55, Room 70, Room 70 Dual, Board 55/55S, Board 70/70S, Board 85S, Cisco TelePresence MX200 G2, MX300 G2, MX700, MX800, MX800 Dual, SX10, SX20, and SX80
- **Webex Desk Device:** Cisco Webex DX80 and DX70 Collaboration Endpoint Software version 9



Figure 10-12 *Webex Boards, Room, and Desk Devices*

xAPI



xAPI is the API for collaboration endpoint software (Cisco CE and RoomOS) for both on-premises registered video conferencing devices (Unified CM and VCS) and devices registered to Cisco’s cloud service (Cisco Webex Devices). xAPI consists of four major groups:

- Commands
- Configurations
- Status
- Events

[Table 10-18](#) shows the high-level API categories that xAPI supports.

Table 10-18 xAPI Categories

Method	API	Description
--------	-----	-------------

G E T	http://<ip-address>/status.xml	Get the complete status of the device
G E T	http://<ip-address>/configuration.xml	Get the complete configuration of the device
G E T	http://<ip-address>/command.xml	Get the complete command set supported by the device
G E T	http://<ip-address>/valuespace.xml	Get an overview of all the value spaces used in the system settings, status information, and commands
P O S T	http://<ip-address>/put.xml	Configure any settings on the device

xAPI Authentication

Access to xAPI requires the user to authenticate using HTTP basic access authentication as a user with the ADMIN role. Unauthenticated requests prompt a 401 HTTP response containing a basic access authentication challenge.

If an application will be issuing multiple commands through xAPI, it is recommended that you use session authentication because the standard basic authentication does a full re-authentication per request, which may affect the performance of your application.

xAPI Session Authentication

Authenticating with your username and password combination for each API request might introduce too

much latency for some use cases. To mitigate this, the API supports a session-based authentication mechanism.

To open a session, issue a POST to `http://<ip-address>/xmlapi/session/begin` with basic access authentication. The response sets a `SessionId`-cookie that can be used with subsequent requests.

Note that when using API session authentication, it is important to explicitly close the session when you are done. Failing to do so may cause the device to run out of sessions, as a limited number of concurrent sessions are available, and they do not time out automatically.

Creating a Session

You create a session by sending an xAPI session request to the endpoint. [Example 10-18](#) shows a simple Python POST request, to which the server responds with the session cookie.

Example 10-18 *Python Script to Get a Session Cookie*

[Click here to view code image](#)

```
""" Webex Devices - Get Session Cookie """
import requests
URL =
"http://10.10.20.159/xmlapi/session/begin"
HEADERS = {
    'Authorization': "Basic
ZGV2YXNjOXMxc2NvITIZ"
}

RESPONSE = requests.request("POST", URL,
headers=HEADERS)
print(RESPONSE.headers["Set-Cookie"])
```

The corresponding response header contains the session cookie, as shown in [Example 10-19](#).

Example 10-19 *Response Header Containing the Session Cookie*

[Click here to view code image](#)

```
$ python3 sess.py
SessionId=031033bebe67130d4d94747b6b9d4e4f6bd29
65162ed542f22d32d75a9f238f9; Path=/;
HttpOnly
```

Getting the Current Device Status

After a session is established, the session cookie is used in each of the interactions. [Example 10-20](#) shows how to use the cookie to get the current status of the device by issuing a simple Python GET request to get the status.

Example 10-20 Python Script to Get Endpoint Status

[Click here to view code image](#)

```
""" Webex Device - Endpoint Status """

import requests
URL = "http://10.10.20.159/status.xml"
HEADERS = {
    'Cookie':
    "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b
7637b"
}

RESPONSE = requests.request("GET", URL,
headers=HEADERS)
print(RESPONSE.text)
```

Setting Device Attributes

When a session is established, the session cookie is used in each of the interactions. [Example 10-21](#) shows how to use the cookie to set the camera position with certain pan and tilt values.

Example 10-21 Python Script to Send a Command to Set the Camera Position

[Click here to view code image](#)

```

""" Webex Device - Set Camera Position """

import requests

URL = "http://10.10.20.159/put.xml"

PAYLOAD = (
    '<Command>' +
    ' <Camera>' +
    '   <PositionSet command="True">' +
    '   <CameraId>1</CameraId>' +
    '   <Pan>150</Pan>' +
    '   <Tilt>150</Tilt>' +
    ' </PositionSet>' +
    '</Camera>' +
    '</Command>'
)

HEADERS = {
    'Content-Type': "application/xml",
    'Cookie':
    "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b
    7637b"
}

RESPONSE = requests.request("POST", URL,
    data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)

```

Registering an Event Notification Webhook

You can get a device or an endpoint to post HTTP event notifications (via webhook) on changes to the API state (for example, statuses, events, configuration updates). These events are sent to the specified server URL. You can choose between events being posted in either XML or JSON format. You can subscribe to changes on multiple parts of the API by registering up to 15 different feedback expressions. The command for registering is **xCommand HttpFeedback**.

The HttpFeedback Register syntax is as follows:

[Click here to view code image](#)

```
FeedbackSlot: <1..4> ServerUrl(r): <S: 1, 2048>
Format: <XML/JSON>
Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255>
```

Example 10-22 shows a simple Python POST to register a webhook.

Example 10-22 *Python Script to Set a Webhook to Receive Event Notifications*

[Click here to view code image](#)

```
""" Webex Devices - Set Webhook """

import requests
URL = "http://10.10.20.159/put.xml"

PAYLOAD = (
    '<Command>' +
    ' <HttpFeedback>' +
    ' <Register command="True">' +
    ' <FeedbackSlot>1</FeedbackSlot>' +
    ' <ServerUrl>http://127.0.0.1/devasc-
webhook</ServerUrl>' +
    ' <Format>JSON</Format>' +
    ' <Expression
item="1">/Configuration</Expression>' +
    ' <Expression
item="2">/Event/CallDisconnect</Expression>' +
    ' <Expression
item="3">/Status/Call</Expression>' +
    ' </Register>' +
    ' </HttpFeedback>' +
    '</Command>'
)

HEADERS = {
    'Content-Type': "application/xml",
    'Cookie':
"SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4
085b7637b,SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef40
```

```

        85b7637b;
        SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7

        637b"
    }

    RESPONSE = requests.request("POST", URL,
    data=PAYLOAD, headers=HEADERS)
    print(RESPONSE.text)

```

Room Analytics People Presence Detector

People Presence shows day-to-day conference room usage analytics and helps you find available meeting rooms based on facial recognition and ultrasonic technology. xAPI allows you to turn on/off the detector via an API. [Example 10-23](#) shows a Python script that sets the People Presence detector on a device to ON.

Example 10-23 *Python Script to Set the People Presence Detector to ON*

[Click here to view code image](#)

```

""" Webex Devices - Set People Presence
detector ON """

import requests

URL = "http://10.10.20.159/put.xml"

PAYLOAD = (
    '<Configuration>' +
    ' <RoomAnalytics>'
    ,
    '<PeoplePresenceDetector>On</PeoplePresenceDetector>'
    +
    ' </RoomAnalytics>' +
    '</Configuration>'
)

HEADERS = {
    'Content-Type': "application/xml",
    'Cookie':
    "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b
7637b"
}

```



```
RESPONSE = requests.request("POST", URL,  
data=PAYLOAD, headers=HEADERS)  
  
print(RESPONSE.text)
```

CISCO UNIFIED COMMUNICATIONS MANAGER

Cisco Unified Communications Manager (Unified CM) is the powerful call-processing component of the Cisco Unified Communications solution. Unified CM supports the following interface types:

- Provisioning interfaces
- Device monitoring and call control interfaces
- Serviceability interfaces
- Routing rules interfaces

Administrative XML



The Administrative XML (AXL) API provides a mechanism for inserting, retrieving, updating, and removing data from the Unified CM configuration database using an Extensible Markup Language (XML) Simple Object Access Protocol (SOAP) interface. It allows a programmer to access Unified CM provisioning services using XML and exchange data in XML form. The AXL methods, referred to as *requests*, are performed using a combination of HTTP and SOAP. SOAP is an XML remote procedure call protocol. Users perform requests by sending XML data to the Unified CM publisher server, which then returns the AXL response, which is also a SOAP message. For more information, see <https://developer.cisco.com/docs/axl/#!axl-developer-guide/overview>.

Examples of Unified CM objects that can be provisioned with AXL include the following:

- Unified CM groups
- Call park DNs
- Call pickup groups
- Calling search spaces
- CTI route points
- Device pools
- Device profiles
- Dial plan tags
- Dial plans
- Digit discard instructions
- Directory numbers
- Gateways (analog, T1, PRI)
- Locations
- MGCP devices
- Phones
- Process nodes
- Process node services
- Regions
- Route filters
- Route groups
- Route lists
- Route partitions
- Service parameters
- Translation patterns
- Users
- Voicemail ports

Cisco AXL Toolkit

You can download the AXL Toolkit from Unified CM Administration page (<https://<CUCM-Address>/plugins/axlsqltoolkit.zip>), as shown in Figure 10-13.

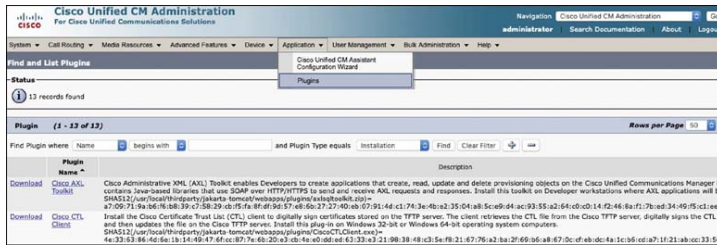


Figure 10-13 AXL Toolkit Zip File

When you download the AXL Toolkit and unzip the file, the schema folder contains AXL API schema files for supported AXL versions:

- **AXLAPI.wsdl:** WSDL file
- **AXLEnums.xsd:** Enum type definitions
- **AXLSoap.xsd:** Type definitions

These XML schema files contain full details about the AXL API format, including the request names, fields/elements, data types used, and field validation rules. One advantage of .xsd schema files is that they can be used to automatically/programmatically validate a particular XML document against the schema to ensure that it is well formatted and valid according to the schema specs.

Many XML editing applications and programmatic tools and components that can use .xsd schema files to validate XML documents and details about how to do so may vary. The main necessary detail, however, is to define the **xsi:schemaLocation** attribute with a URI pointing to the axlsoap.xsd schema file.

Accessing the AXL SOAP API

Now let's take a look at two different methods to access the AXL SOAP API.

Using the Zeep Client Library

The Python script in [Example 10-24](#) shows how to make an API call to update a phone device and get complete

information about the phone. It uses the AXLAPI.wsdl file downloaded with the AXL Toolkit.

Example 10-24 *Full Client Code to Update and Get Back Device Details*

[Click here to view code image](#)

```
""" Update and Retrieve Client Details """

from zeep import Client
from zeep.cache import SqliteCache
from zeep.transports import Transport
from requests import Session
from requests.auth import HTTPBasicAuth
import urllib3
from urllib3.exceptions import
InsecureRequestWarning

urllib3.disable_warnings(InsecureRequestWarning)

USERNAME = 'administrator'
PASSWORD = 'ciscopsdt'
IP_ADDRESS = "10.10.20.1"
WSDL = 'schema//12.0//AXLAPI.wsdl'
BINDING_NAME = "
{http://www.cisco.com/AXLAPIService/}AXLAPIBinding"

ADDRESS =
"https://{ip}:8443/axl/".format(ip=IP_ADDRESS)
def update_phone_by_name(client, name,
description):
    """ Update Phone by Name """
    return client.updatePhone(**{'name': name,
'description': description})

def get_phone_by_name(client, name):
    """ Get Phone by Name """
    return client.getPhone(name=name)

def main():
    """ Main """
    session = Session()
    session.verify = False
    session.auth = HTTPBasicAuth(USERNAME,
PASSWORD)
    transport = Transport(cache=SqliteCache(),
session=session, timeout=60)
    client = Client(wSDL=WSDL,
transport=transport)
    axl = client.create_service(BINDING_NAME,
```

```

ADDRESS)

    update_phone_by_name(axl,
"SEP001122334455", "DevAsc: adding new Desc")
    print(get_phone_by_name(axl,
"SEP001122334455"))

if __name__ == '__main__':
    main()

```

Using the CiscoAXL SDK

The Python SDK called CiscoAXL is pretty simple to use. As of this writing, there are other SDKs available, but we use CiscoAXL here as an example. To start with this SDK, you need to install it by using the **pip** or **pip3** command:

[Click here to view code image](#)

```
pip install ciscoaxl or pip3 install ciscoaxl
```

[Example 10-25](#) shows the simplest way to invoke the SDK and start working with it.

Example 10-25 *Using the CiscoAXL SDK to Get Phone Information*

[Click here to view code image](#)

```

""" Using CiscoAXL SDK to get phone info"""

from ciscoaxl import axl

CUCM = '10.10.20.1'
CUCM_USER = "administrator"
CUCM_PASSWORD = "ciscopsdt"
CUCM_VERSION = '12.0'
ucm =
axl(username=CUCM_USER,password=CUCM_PASSWORD,cucm=CUCM,cucm_version=CUCM_

VERSION)
print (ucm)
for phone in ucm.get_phones():
    print(phone.name)
for user in ucm.get_users():
    print(user.firstName)

```

For more examples, see Cisco DevNet’s Code exchange at <https://developer.cisco.com/codeexchange/github/repo/levensailor/ciscoaxl/>.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 10-19](#) lists these key topics and the page number on which each is found.



Table 10-19 Key Topics

Key Topic Element	Description	Page
Paragraph	Webex Teams APIs	261
Paragraph	Access Scope in Webex Teams	265
Table 10-8	Bot Types	271

<u>Example 10-7</u>	Python Code to Generate a JWT Token for a Guest Issuer	<u>2</u> <u>7</u> <u>3</u>
<u>Figure 10-9</u>	Finesse High-Level Flow	<u>2</u> <u>7</u> <u>4</u>
<u>Table 10-13</u>	Webex Meetings Services Supported via APIs	<u>2</u> <u>8</u> <u>2</u>
<u>Paragraph h</u>	The Webex XML API	<u>2</u> <u>8</u> <u>4</u>
<u>Paragraph h</u>	xAPI	<u>2</u> <u>9</u> <u>0</u>
<u>Paragraph h</u>	AXML for Unified CM	<u>2</u> <u>9</u> <u>4</u>

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Unified Communications

Finesse

JSON Web Token (JWT)

voice over IP (VoIP)

Extensible Messaging and Presence Protocol (XMPP)

Bidirectional-streams Over Synchronous HTTP (BOSH)

computer telephony integration (CTI)

fully qualified domain name (FQDN)

Chapter 11

Cisco Security Platforms and APIs

This chapter covers the following topics:

- **Cisco Security Portfolio:** This section introduces Cisco's security portfolio and describes the detect, segment, and protect mechanisms.
- **Cisco Umbrella:** This section introduces the Cisco Umbrella product and the relevant APIs.
- **Cisco Firepower:** This section provides an overview of Cisco Firepower Management Center and API categories.
- **Cisco Advanced Malware Protection (AMP):** This section provides information you need to understand Advanced Malware solution.
- **Cisco Identity Services Engine:** This section provides an overview of Cisco Identity Services Engine (ISE) and ISE APIs.
- **Cisco Threat Grid:** This section provides an overview of Cisco Threat Grid and Threat Grid APIs.

Cisco has been building the backbone of the Internet for nearly 35 years. In addition, Cisco has created networks, big and small, and gained vast amounts of knowledge about what happens on a network. Along the way, Cisco has built a robust security portfolio to keep networks and users safe. Cisco's security portfolio has three main pillars:

- Visibility
- Detection
- Mitigation

A lot of organizations don't have the visibility and the analytics to know what's going on across their networks. Visibility includes understanding who is on the network, including people and devices, who is accessing the various servers, who is communicating with whom, and what type of traffic is on the network.

Detecting all these activities is the second pillar. And as they say, one cannot discover what you can't see. The network now needs to observe, learn, and detect anomalies continuously. By staying ahead of continually evolving attacks, a network senses the critical threats by mitigating and responding with corrective actions.

This chapter introduces various Cisco security products as well as multiple aspects of integrating security products via APIs. It covers the following:

- Cisco Firepower
- Cisco Umbrella
- Cisco Advanced Malware Protection (AMP)
- Cisco Identity Services Engine (ISE)
- Cisco Threat Grid

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 11-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 11-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Cisco Umbrella	1, 2
Cisco Firepower	3, 4

Cisco Identity Services Engine	5, 6
Cisco Threat Grid	7, 8

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. The Umbrella _____ API provides enrichment of security events with intelligence to SIEM or other security visibility tool.
 1. Enforcement
 2. Investigate
 3. Management
 4. Reporting
2. The Umbrella Enforcement API involves an HTTP _____ request, which internally comprises an _____ API to check whether the domain is safe.
 1. POST, Reporting
 2. GET, Investigate
 3. POST, Investigate
 4. GET, Reporting
3. In the Firepower Management Center API, the token is generated and returned via _____.
 1. the HTTP header **X-auth-access-token**
 2. JSON object
 3. XML object
 4. JWT
4. A _____ object is a reusable configuration that associates a name with a value.
 1. simple
 2. named

3. objective
4. network
5. Which of the following allows bad actors to gain access to and control endpoint resources over an extended period to steal valuable data without being detected?
 1. Malware
 2. Medium persistent threat
 3. Advanced persistent threat
 4. Ransomware
6. Which of the following enables devices and users to be identified and provisioned and enables policies to be applied?
 1. Cisco AMP
 2. Cisco ISE
 3. Cisco Threat Grid
 4. Cisco Firepower Management Center
7. Which of the following describes Threat Grid?
 1. A VPN solution from Cisco
 2. A unified malware analysis and threat intelligence platform
 3. A threat intelligence organization
 4. An intelligence security framework
8. Which of the following are used to indicate that a system has been affected by some form of malware?
 1. IoTs
 2. IOCs
 3. Snort
 4. Reports

FOUNDATION TOPICS

CISCO'S SECURITY PORTFOLIO

Cisco's security portfolio is vast. This chapter provides an introduction to some of its critical security products. In addition, it covers the APIs that each of these products supports.

As shown in [Figure 11-1](#), the Cisco security portfolio focuses on three areas: visibility, segmentation, and

protection. Visibility is essentially the ability to “see” everything that is happening in the network. Segmentation is the ability of the network to reduce the surface attack to contain the spread. Protection is the ability to stop any breaches from happening.

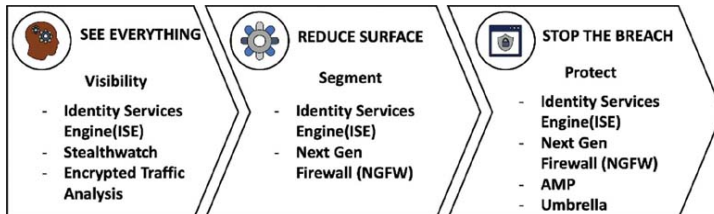


Figure 11-1 Cisco’s Security Portfolio: Visibility, Segmentation, and Protection

Potential Threats and Vulnerabilities

Anything we are trying to protect against—whether it is code or a person—is known as a threat. A threat tries to get access to an asset in order to control or damage the asset.

A vulnerability is a weakness or gap in protection efforts. It can be exploited by threats to gain unauthorized access to an asset.

Cybersecurity threats comprise a wide range of potentially illegal activities on the Internet. At a very high level, certain activities may be classified into two categories:

- **Malicious activities that target networks or devices directly:** Examples include malware, viruses, and denial-of-service attacks.
- **Malicious activities aided by computer networks or devices:** Examples include fraud, identity theft, phishing scams, information warfare, and cyberstalking.

Most Common Threats

Table 11-2 describes some of the most common threats that can be posed to an entity due to lack of security awareness.

Table 11-2 Common Threats

ThreatDescription	
Man-in-the-middle attack	Attackers insert themselves between two endpoints (such as a browser and a web server) and intercept or modify communications between the two. The attackers can then collect information as well as impersonate either of the two agents. In addition to websites, these attacks can target email communications, DNS lookups, and public Wi-Fi networks.
Denial-of-service (DoS) attack	An attacker sends multiple requests that flood the server or networks with traffic to exhaust resources and bandwidth. As the system continues with degraded performance, the system becomes more and more nonresponsive, and real requests are left unfulfilled. A DoS attack can be coordinated so that multiple devices launch the attack at the same time. This is known as a distributed denial-of-service (DDoS) attack.
Cross-site scripting (XSS)	Cross-site scripting is an exploit in which the attacker attaches code to a legitimate website that executes when the victim loads that website. Typically, a web page is loaded, and malicious code copies the user's cookies. The system then sends an HTTP request to an attacker's web server, with the stolen cookies in the body of the request. The attacker can then use cookies to access sensitive data.

Phishing	This type of exploit involves using emails or web pages to procure sensitive information, such as usernames and passwords.
Malware	Malware is a piece of malicious code, such as spyware, ransomware, a virus, or a worm. Malware is usually triggered when someone clicks a link or an email attachment, which then installs malicious software.
Structured Query Language (SQL) injection	SQL injection is a code injection technique used to modify or retrieve data from SQL databases. By inserting specialized SQL statements into an entry field, an attacker can execute commands that allow for the retrieval of data from the database.
Brute-force attack	Brute-force methods, which involve using trial and error to decode data, can be used to crack passwords and crack encryption keys. Other targets include API keys, SSH logins, and Wi-Fi passwords.

CISCO UMBRELLA



Cisco Umbrella is a cloud-based secure gateway that helps protect and defend against threats that arise on the Internet. Cisco Umbrella is the first line of defense for users who may be trying to connect to the Internet from anywhere. A device trying to connect to the Internet needs a DNS (Domain Name System) lookup to translate the name of the site or the service to the IP address that it needs to connect to.

Cisco Umbrella processes billions of DNS requests per day, analyzing and learning about various activities and blocking requests to unwanted and malicious destinations before a connection is even established. Cisco Umbrella incorporates the following security services in its offering, regardless of where the user is located:

- It blocks malware, ransomware, and phishing attempts from malicious or fraudulent sites.
- It can be integrated with Cisco AMP and other antivirus engines.
- It maintains content categories and custom-defined whitelists and blacklists to comply with any organization policy.

Understanding Umbrella

Cisco Umbrella processes DNS requests received from users or devices on the networks. It not only works on HTTP or HTTPS but supports other protocols as well.

Let's look at a simple flow using the network shown in [Figure 11-2](#). Say that a user wants to access a site and makes a request to the site. This results in a DNS request being sent to Cisco Umbrella. The following steps occur:

- Step 1.** Umbrella analyzes the DNS request to check whether the domain is malicious or safe.

Step 2. Umbrella checks to see if any of the policies are triggered, such as content filtering policies or blacklisted domains.

Step 3. If all is well, the IP address is sent to the requesting user. In the case of stanford.edu, Umbrella returns the correct IP address.

Step 4. When a DNS request is sent for domain.xyz, Umbrella checks whether any policies are triggered or whether this is a known malicious domain.

Step 5. Umbrella responds with a “blocked page” message, informing the user that the domain is either malicious or on the blocked list.

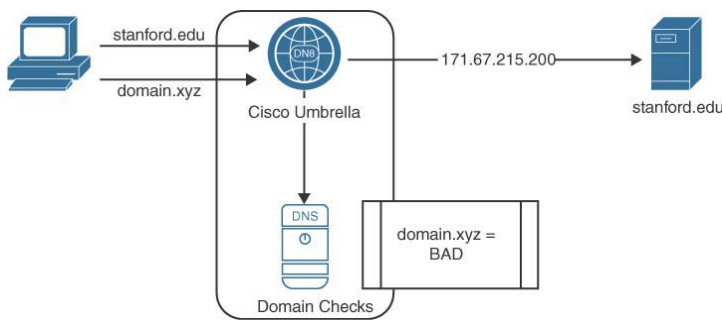


Figure 11-2 Cisco Umbrella: Blocking a Malicious Domain Lookup

Cisco Umbrella APIs

Cisco Umbrella supports various APIs for different functions. [Table 11-3](#) describes these APIs.

Table 11-3 Umbrella APIs

API	Description
Managing	This API directs customers to manage organizations, networks, network devices, users, and roaming computers and integrate actions in those areas into workflows.

e m e n t A P I	
R e p o r t i n g A P I	<p>This API for organizations consists of the following endpoints:</p> <ul style="list-style-type: none"> • Destinations: Most recent requests • Destinations: Top identities • Security activity report
C o n s o l e R e p o r t i n g A P I	<p>This API is for managed service providers and multiple-organization console administrators. It displays summary information that is available only in those consoles.</p> <p>The Console Reporting API has two endpoints:</p> <ul style="list-style-type: none"> • /security-summary: This is the security summary, which provides the total requests and the total requests blocked for all the child organizations in aggregate, as well as the same information for each child organization. • /detailed-summary: This is the deployment summary, which provides the overall deployment status for all customers of the console, as well as deployment details about each child organization of the console.
N e t w o	<p>A network device identity is any hardware device that can route DNS traffic to the Cisco Umbrella recursive DNS servers. The first step is registering the device with Cisco Umbrella. Once the traffic from a device reaches</p>

<p>r k D e v i c e M a n a g e m e n t A P I</p>	<p>the DNS servers, the organization with which the device is registered is identified, and policies for that organization can be applied to the traffic.</p>
<p>E n f o r c e m e n t A P I</p>	<p>This API enables organizations to manage security-related blocked domain lists.</p>
<p>I n v e s t i g a t e A P I</p>	<p>The RESTful API allows the querying of the Umbrella DNS database and to show security events and correlations related to the domain queried.</p>

Authentication

All Cisco Umbrella APIs use HTTP-basic authentication. The key and secret values need to be Base64 encoded and sent as part of a standard HTTP basic Authorization header. API requests are sent over HTTPS. APIs require the credentials to be sent in the Authorization header. The credentials are the username and password, separated by a colon (:), within a Base64-encoded string. For example, the Authorization header would contain the following string:

[Click here to view code image](#)

```
"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"
```

In this case, **ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk** is the Base64-encoded string *devasc:strongpassword* (where *devasc* is the username, and *strongpassword* is the password). [Example 11-1](#) shows three lines of code that do Base64 encoding to plug the value into the Authorization header.

Example 11-1 *Python Code to Generate Base64 Encoding*

[Click here to view code image](#)

```
""" Generate Base64 encoding using the base64
library """
import base64
encoded =
base64.b64encode('devasc:strongpassword'.encode('UTF-
8')).decode('ASCII')
print(encoded)
```

The Management API

The Umbrella Management API enables direct customers, service providers (SPs), managed service providers (MSPs), and managed security service providers (MSSPs) to manage organizations, networks, network devices, users, and roaming computers and integrate actions in their everyday workflows.

The Management API can use the ISP and MSSP endpoints passing the Base64-encoded authorization in the header (see [Example 11-2](#)).

Example 11-2 *Python Code to Get Customer Details Using the Management API*

[Click here to view code image](#)

```
""" Get Customer details given a customerID """
import requests
url =
"https://management.api.umbrella.com/v1/serviceproviders/serviceProviderId/

customers/customerId"
headers = {
    'accept': "application/json",
    'authorization': "Basic
ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"
}
response = requests.request("GET", url,
headers=headers)
print(response.text
)
```

The management API includes the following:

- **ISP and MSSP endpoints:** The API returns the service provider IDs and customer IDs. These endpoints are for Internet service providers (ISPs), managed service providers (MSPs) using the Master Service license, managed security service providers (MSSPs), and partner console users. To perform these queries, you must have your service provider ID (SPID) from your console's URL.
- **MSP and multiple-organization endpoints:** The API creates new MSP customers, returns MSP customer records, and updates and deletes MSP customers. This endpoint is for MSP and multiple-organization consoles, *not* for MSPs using MSLA.
- **Networks:** The API returns network records and deletes networks. Note that parent organizations do not have networks. To create and manage networks on behalf of child organizations, use the organizations/customerID/networks endpoints.
- **Roaming computers:** The API returns roaming computer records and updates or deletes roaming computers.
- **Internal networks:** The API creates, updates, and deletes internal networks and returns internal network records.
- **Internal domains:** The API creates, updates, and deletes internal domains and returns internal domain records.

- **Virtual appliances:** The API returns virtual appliance (VA) records and updates or deletes VAs. Note that you cannot create a virtual appliance through the API. A VA must be created within your hypervisor and must be registered as an identity within Umbrella before the API can manage it.
- **Umbrella sites:** The API creates, updates, and deletes sites and returns site records.
- **Users:** The API creates and deletes users and returns user records.
- **Roles:** The API returns a list of roles.
- **Destination lists:** The API creates, reads, updates, and deletes destination lists.

The Enforcement API



The Cisco Umbrella Enforcement API is designed to give technology partners the ability to send security events from their platform/service/appliance within a mutual customer's environment to the Umbrella cloud for enforcement. With this API, you can list domains or delete individual domains from the list.

The API is restricted to HTTPS and is hosted at <https://s-platform.api.opendns.com>. A fixed UUID-v4 customer key handles customer authentication to the API. A key must be supplied with each request to the API. To generate or get the customer key, you have to log in to the console and navigate to Policies > Policy Components > Integrations.

Now let's look at an example of enforcement. Here are the steps involved when the customer detects a malicious domain and wants to add it to Umbrella:

- Step 1.** The customer identifies malicious code or a malicious activity as users visit a particular URL or domain. The detection can occur with third-party software or Cisco AMP or any other mechanism that the customer already has in place.

Step 2. This event is sent to the Umbrella Enforcement API via a POST request (see [Example 11-3](#)).

Step 3. Cisco Umbrella follows the appropriate logic and algorithm before it adds the domain to the blocked list. It goes through these steps:

1. Umbrella checks whether the domain exists in the Umbrella global block list under one of the security categories.
2. It runs the Investigate API internally to decide if the domain is benign.
3. It checks on the status of the domain (that is, uncategorized or categorized).
4. It checks to see if the domain is already present on the customer's allow list within the organization.

Step 4. If all the checks are validated, Umbrella blocks domains in that list per that customer's Umbrella policy security settings.

Example 11-3 *Python POST Code to Add a Domain Using the Enforcement API*

[Click here to view code image](#)

```
""" Add domain using the Enforcement API """

import json
import requests

url = "https://s-
platform.api.opendns.com/1.0/events"

querystring = {"customerKey": "XXXXXXX-YYYY-
ZZZZ-YYYY-XXXXXXXXXXXX"}
payload = [
    {
        "alertTime": "2020-01-01T09:33:21.0Z",
        "deviceId": "deadbeaf-e692-4724-ba36-
c28132c761de",
        "deviceVersion": "13.7a",
        "dstDomain": "looksfake.com",
        "dstUrl":
"http://looksfake.com/badurl",
        "eventTime": "2020-01-01T09:33:21.0Z",
        "protocolVersion": "1.0a",
        "providerName": "Security Platform"
    }
]
```

```

headers = {
    'Content-Type': "text/plain",
    'Accept': "*/*",
    'Cache-Control': "no-cache",
    'Host': "s-platform.api.opendns.com",
    'Accept-Encoding': "gzip, deflate",
    'Connection': "keep-alive",
    'cache-control': "no-cache"
}

response = requests.request(
    "POST",
    url,
    data=json.loads(payload),
    headers=headers,
    params=querystring)

print(response.text)

```

Once domains are placed in the list, a customer can get the list of the domains by using the **GET** method for <https://s-platform.api.opendns.com/1.0/domains> endpoint. [Example 11-4](#) shows an example of a simple Python **requests** method.

Example 11-4 *Python GET Request to List Domains Using the Enforcement API*

[Click here to view code image](#)

```

""" List domains using the Enforcement API """

import requests
url = "https://s-
platform.api.opendns.com/1.0/domains"
querystring = {"customerKey": "XXXXXXX-YYYY-
ZZZ-YYYY-XXXXXXXXXXXX"}
response = requests.request("POST", url,
headers=headers, params=querystring)
print(response.text)

```

[Example 11-5](#) shows the response to the **GET** from [Example 11-4](#). The answer is a JSON response with a **meta** tag that describes the page number, the limit, and pointers to the previous and next pages.

Example 11-5 *Response to the GET Request to List Domains Using the Enforcement API*

[Click here to view code image](#)

```
{
  "meta":{
    "page":1,
    "limit":200,
    "prev":false,
    "next":"https://s-
platform.api.opendns.com/1.0/
domains?customerKey=XXXXXXX-YYYY-ZZZZ-
YYYY-XXXXXXXXXXXX&page=2&limit=200"
  },
  "data":[
    {
      "id":1,
      "name":"baddomain1.com"
    },
    {
      "id":2,
      "name":"looksfake.com"
    },
    {
      "id":3,
      "name":"malware.dom"
    }
  ]
}
```

The final part of the Enforcement API is the **DELETE** call to delete the domain API. The API helps in unblocking domains that were blocked because of previously injected events. [Example 11-6](#) shows an example of a simple Python **requests** command to delete a domain using the Enforcement API. The URL to do this is as follows:

[Click here to view code image](#)

```
https://s-
platform.api.opendns.com/1.0/domains/looksfake.
com?customerKey= XXXXXXX-YYYY-ZZZZ-YYYY-
XXXXXXXXXXXX
```


Example 11-6 *Deleting Domains Using the Enforcement API*

[Click here to view code image](#)

```
""" Delete domain using the Enforcement API """

import requests

url = "https://s-
platform.api.opendns.com/1.0/domains/looksfake.com"

querystring = {"customerKey": "XXXXXXX-YYYY-
ZZZZ-YYYY-XXXXXXXXXXXX"}

response = requests.request("DELETE", url,
headers=headers, params=querystring)
print(response.text)
```

The Investigate API



The Cisco Umbrella Investigate API is designed to give technology partners the ability to query security events from their platform/service/appliance within a mutual customer's environment to the Umbrella cloud for investigation purposes. The Umbrella Investigate API empowers users to query the Umbrella database to deem a domain safe or not safe. It goes beyond traditional DNS results to show security events and correlations.

The following are some of the tasks that can be done via the Umbrella Investigate REST API:

- Check the security status of a domain, IP address, or subset of domains that appears in the logs from your firewall, UTM, or other Internet egress points.
- Determine whether other related cooccurring domains were accessed at the same time as the domain or IP address you're looking up.
- Find a historical record for this domain or IP address in the DNS database to see what has changed.

- Query large numbers of domains quickly to find out whether they're scored as malicious and require further investigation.

The Investigate API can be accessed via an access token, and the access token can be generated via the Umbrella console.

Developers and customers can use and query Umbrella data via the Investigate API. Here are a few of the standard categories:

- **Categorization (shows the status and classification of the domain):** This category is often used by developers and customers as the primary classifier to determine whether a domain/IP address is good, bad, or unknown.
- **Scoring:** Several scores help rate the potential risk of the domain/IP address. For example:
 - **SecureRank2:** This score is designed to identify domains that are requested by known infected clients but never requested by clean clients—assuming that these domains are more likely to be bad. Scores range from -100 (suspicious) to +100 (benign).
 - **RIP Score:** This IP reputation score is designed to rate an IP address based on the amount of malicious activity hosted at that address. Scores range from -100 (very suspicious) to 0.
- **WHOIS record data:** This category includes the email address used to register the domain, the associated name server, historical information, and so on. It can be used to find out more about the history of the domain and the registrant, including whether the email address was used to register other malicious domains.
- **Cooccurrences:** This category depicts other domains that were queried right before or after a given domain and are likely related. The Investigate API is often used to uncover other domains that may be related to the same attack but are hosted on completely separate networks.
- **Passive DNS:** This category depicts the history of domain-to-IP address mappings. This information is used to see if anything suspicious happened with the domain or IP address. For example, you might find that the IP address is continually changing or find that the IP address has more domains than previously declared.
- **Malware file data:** Information is gathered in the form of malware file analysis and threat intelligence from Cisco AMP Threat Grid. This kind of information is used to find out if there are any specific malware files associated with a domain and also to query file hashes to see if they're malicious.

Now let's look at a couple of examples of the Investigate API. If you query the domain at

<https://investigate.umbrella.com/domain-view/name/cisco.com/view> via the user interface, you see a screen like the one shown in [Figure 11-3](#).



Figure 11-3 *Umbrella Investigate API via the UI for cisco.com*

As you can see, the domain cisco.com is classified as being Benign, and the SecureRank score is 2. Therefore, cisco.com is a safe domain. Now let's look at the same operation but via the Investigate API. The API to call is <https://investigate.api.umbrella.com/domains/categorization/cisco.com>. [Example 11-7](#) shows a Python `requests` command that makes an API call with the appropriate API key.

Example 11-7 *Getting Domain Categorization by Using the Investigate API*

[Click here to view code image](#)

```
""" Domains categorization using the
Investigate API """

import requests
url =
"https://investigate.api.umbrella.com/domains/categorization/cisco.com"

querystring = {"showLabels":""}
headers = {
    'authorization': "Bearer deadbeef-24d7-
40e1-a5ce-3b064606166f",
    'cache-control': "no-cache",
}
response = requests.request("GET", url,
headers=headers, params=querystring)
print(response.text)
```

There are three return values for the categorization:

- **Status:** The status is -1 if the domain is believed to be malicious, 1 if the domain is believed to be benign, or 0 if it hasn't been classified yet.
- **Security:** This field indicates whether there is an Umbrella domain match or whether this domain is associated with one.
- **Content:** This field indicates the type of domain (for example, Ecommerce/Shopping, Business Services).

Example 11-8 shows the response to the request in Example 11-7. Note that the query parameter **showLabels** in Example 11-7 gives the more human-readable information in the response.

Example 11-8 JSON Returned for Domain Categorization Using the Investigate API

[Click here to view code image](#)

```
{
  "cisco.com": {
    "status": 1,
    "security_categories": [],
    "content_categories": [
      "Software/Technology",
      "Business Services"
    ]
  }
}
```

Table 11-4 lists other Investigate API URLs for the cisco.com domain.

Table 11-4 Other Umbrella Investigate API Categories

API Category	API Endpoint URL
Classifiers for a domain	https://investigate.api.umbrella.com/domains/categories/cisco.com.json
Cooccurrences for a domain	https://investigate.api.umbrella.com/recommendations/name/cisco.com.json

Related domains for a domain	https://investigate.api.umbrella.com/links/name/cisco.com.json
Security information for a domain	https://investigate.api.umbrella.com/security/name/cisco.com
Domain volume	https://investigate.api.umbrella.com/domains/volume/cisco.com?start=-2days&stop=now&match=component
Threat Grid sample for a domain, an IP address, or a URL	https://investigate.api.umbrella.com/samples/cisco.com?limit=100&sortby=score

CISCO FIREPOWER



Cisco Firepower is a next-generation firewall (NGFW). NGFWs are part of Cisco's leading-edge firewall technology, which combines the traditional firewall functionality with an application firewall using an intrusion prevention system (IPS) with deep packet inspection (DPI) technology.

Firepower also employs other techniques, such as Transport Layer Security/Secure Sockets Layer (TLS/SSL) encrypted traffic inspection, website filtering, quality of service (QoS)/bandwidth management, and malware inspection. It also has built-in management integrations with Lightweight Directory Access Protocol (LDAP), RADIUS, and Active Directory.

In the past, stateful firewalls with simple packet filtering capabilities efficiently blocked unwanted applications

because most applications met the port/protocol expectations. NGFWs filter traffic based on the applications or traffic types traversing specific ports. For example, you could open up port 80 for only selected HTTP traffic or for particular applications, sites, or services that you allow. Firepower provides a combination of firewall and QoS functions in a single application-aware solution. Here are the characteristic features of most NGFWs:

- **Standard firewall features:** These include the traditional (first-generation) firewall functionalities such as stateful port/protocol inspection, Network Address Translation (NAT), and virtual private network (VPN) capabilities.
- **Application identification and filtering:** An NGFW identifies and filters traffic based on specific applications rather than just opening ports for all kinds of traffic. An NGFW prevents malicious apps and activities from using nonstandard ports in order to avoid the firewall.
- **SSL and SSH inspection:** NGFWs can inspect SSL- and SSH-encrypted traffic. An NGFW decrypts traffic, makes sure the applications are allowed, checks other policies, and then re-encrypts the traffic. This provides additional protection against malicious apps and activities that try to hide by using encryption to avoid the firewall.
- **Intrusion prevention:** An NGFW has intelligent capabilities to provide more in-depth traffic inspection to perform intrusion detection and prevention.
- **ISE integration:** NGFWs have the support of Cisco ISE. This integration allows authorized users and devices to use specific applications.
- **Malware filtering:** NGFWs can provide reputation-based screening to block applications that have bad reputations. This functionality can check for phishing, viruses, and other malware sites and apps.

Figure 11-4 shows the components of the Firepower solution:

- Firepower Management Center (a management console that has APIs to control and manage application control, URL filtering, AMP, and so on)
- Firepower Threat Defense

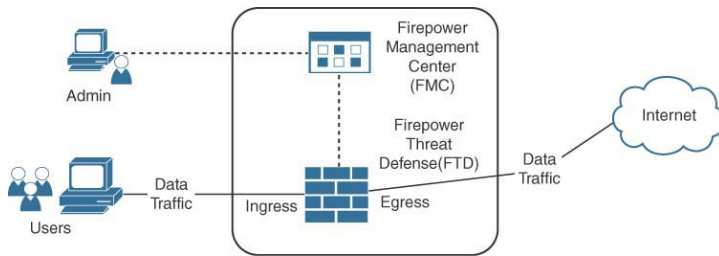


Figure 11-4 *Firepower Components*

Firepower Management Center provides complete and unified management of firewalls, application control, intrusion prevention, URL filtering, and advanced malware protection. It can help administrators smoothly go from managing firewalls to controlling applications to investigating and remediating malware attacks.

Firepower Management Center APIs

DevNet has a dedicated developer center for Firepower at <https://developer.cisco.com/firepower/>. This site provides links to various Firepower technologies as well as the DevNet sandboxes.

The Firepower Management Center REST APIs enable you to program Firepower devices in order to automatically provision devices, discover hosts, perform vulnerability analysis, automate and script firewall configurations, deploy policies and controls, and monitor device health. [Figure 11-5](#) provides an overview of how Firepower Management Center connects to various Firepower devices as well as how you can build your own applications by using REST APIs.

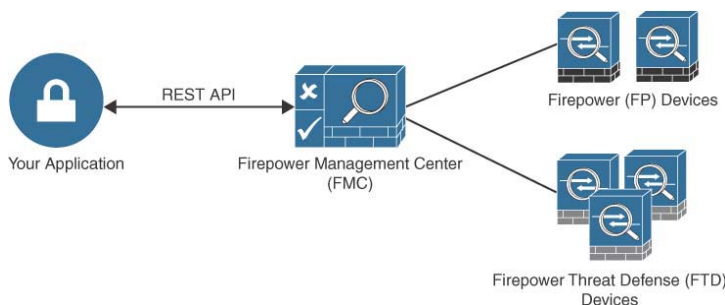


Figure 11-5 *Firepower Allowing Both Northbound and Southbound APIs*

Authentication

The Firepower APIs are already part of the FMC software by default, and the only thing that is required is to enable them via the UI. The Firepower APIs use token-based authentication for API users. Consider the simple example shown in [Example 11-9](#). It uses the Python **requests** command to make the REST call, the POST method, and the API https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/auth/generatetoken.

Example 11-9 *Python Code to Generate the Session Token for Firepower Management Center*

[Click here to view code image](#)

```
""" Generate the session token for FMC """
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

url =
"https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/auth/generatetoken"

headers = {
    'Content-Type': "application/xml",
    'Authorization': "Basic
YXNodXRvc2g6V0JVdkE5TXk=",
}
response = requests.request("POST", url,
headers=headers)
print(response.headers)
```

The response header contains '**X-auth-access-token**': **"03d91b3f-eeff-4056-a4a7-e121ddcf8910"**, which needs to be used in all subsequent API calls.

System Information

By using '**X-auth-access-token**' you can now make an API call to get the server version. [Example 11-10](#) uses the Python **requests** command to make the REST call, the GET method, and the API https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/info/serverversion.

Example 11-10 *Python Code to Get the Server Version via the Firepower Management Center AP*

[Click here to view code image](#)

```
""" Get Server Version """
import requests
url =
"https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/info/serverversion"

headers = {
    'X-auth-access-token': "2abd7bdc-16f8-477f-
8022-7f193e71c847",
}
response = requests.request("GET", url,
headers=headers, verify=False)
print(response.text)
```

Now that you know the basics of accessing the API, you can explore all the APIs that Firepower Management Center has to offer.

As indicated earlier, Cisco DevNet provides an instance of Firepower Management Center in the sandbox. The easiest way to figure out specific operations available with any version is by searching for “FMC API Explorer.” You can launch the API Explorer by using the URL https://fmc_url/api/api-explorer/, or if you have reserved the DevNet sandbox, you can simply use <https://fmcrestapisandbox.cisco.com/api/api-explorer/#>. [Figure 11-6](#) shows the API Explorer, which allows you to explore all possible FMC APIs.

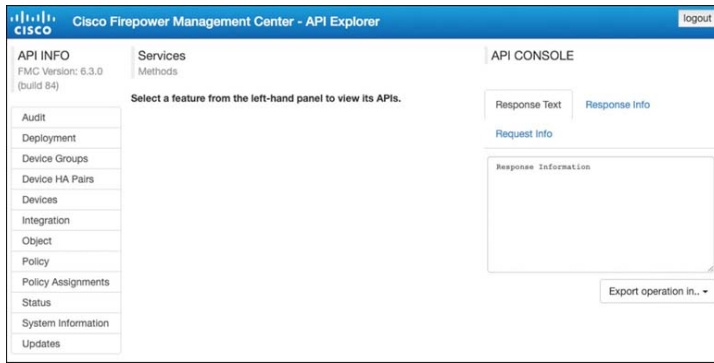


Figure 11-6 *The API Explorer, for Trying Out the Firepower Management Center APIs*

Firepower Management Center Objects

In Firepower Management Center, a named object is a reusable configuration that associates a name with a value. When you want to use that value, you can use the named object instead. Firepower Management Center provides many predefined objects that represent frequently used configurations. You can use objects in policies, rules, event searches, reports, and dashboards. The system offers many predefined objects that represent commonly used configurations. Group objects reference multiple objects with a single configuration. Certain predefined objects are groupable, as shown in [Table 11-5](#), which explains the various object types that are defined in Firepower Management Center.

Object Types

[Table 11-5](#) lists the objects you can create in the Firepower system and indicates which object types can be grouped.

Table 11-5 Firepower Management Center Object Types

Object Type	Groupable
Network	Y

	e s
Port	Y e s
Security zone	N o
Application filter	N o
VLAN tag	Y e s
URL	Y e s
Geolocation	N o
Variable set	N o
Security intelligence: Network, DNS, and URL lists and feeds	N o
Sinkhole	N o
File list	N o
Cipher suite list	N o
Distinguished name	Y e s

Public key infrastructure (PKI): Internal and trusted CA and internal and external certs	Y e s
Route map	N o
Community list	N o

Creating a Network

A network object represents one or more IP addresses. We use network objects and groups in other objects, including access control policies, network variables, identity rules, network discovery rules, event searches, and reports. [Example 11-11](#) shows how to generate a token, assign it in the header, and then create a network object.

Note that the Firepower Management Center instance in [Example 11-11](#) uses the DevNet Sandbox instance (<https://fmcrestapisandbox.cisco.com/>) of the FMC, which requires reservation. If you use this example, make sure that you modify the definition of the network object; otherwise, the object creation may fail because the object will be already present. [Example 11-11](#) uses the POST method and the

API `/api/fmc_config/v1/domain/" + uuid +
"/object/networks`.

Example 11-11 *Generating a Session Token and Creating a New Network Object*

[Click here to view code image](#)

```
""" generate a session token and create a new
network object """

import json
import requests
```

```

## Globals used in this file
url =
"https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/auth/generatetoken"

server = "https://fmcrestapisandbox.cisco.com"
username = "johnsmith"
password = "pwgDvQt3"
domain = "Global"
token = ""
headers = {
    'Content-Type': "application/json",
}

## Definition of Lab Network (10.10.10.0)

network_lab = {
    "name": "labnetwork-1",
    "value": "10.10.10.0/24",
    "overridable": False,
    "description": "Lab Network Object",
    "type": "Network"
}

def networkObject(network, uuid):
    """ Create a new Network object """

    netpath = "/api/fmc_config/v1/domain/" +
uuid + "/object/networks"
    url = server + netpath
    print("-----")
    print(headers)
    try:
        response = requests.post(url,
data=json.dumps(network), headers=headers,
verify=False)
        status_code = response.status_code
        resp = response.text
        json_response = json.loads(resp)
        print("status code is: " +
str(status_code))
        if status_code == 201 or status_code ==
202:
            print("Successfully network
created")
        else:
            response.raise_for_status()
            return json_response["name"],
json_response["id"]
    except requests.exceptions.HTTPError as
err:
        print("Reason Code: " + str(err))
    finally:
        if response:

```

```

        response.close()

def generateSessionToken():
    """ Generate a new session token using the
    username and password """
    global uuid
    global headers
    tokenurl =
"/api/fmc_platform/v1/auth/generatetoken"
    url = server + tokenurl
    response = requests.request(
        "POST",
        url,
        headers=headers,

auth=requests.auth.HTTPBasicAuth(username,
password),
        verify=False
    )
    print(response.headers)
    status_code = response.status_code
    if status_code == 201 or status_code ==
202:
        print("Successfully network created")
    else:
        response.raise_for_status()

    auth_headers = response.headers
    token = auth_headers.get('X-auth-access-
token', default=None)
    headers['X-auth-access-token'] = token
    domains = auth_headers.get('DOMAINS',
default=None)
    domains = json.loads("{\"domains\": " +
domains + "}")
    for item in domains["domains"]:
        if item["name"] == domain:
            uuid = item["uuid"]
        else:
            print("no UUID for the domain
found!")

    print(domains)
    print(uuid)
    print(headers)

## Main - Entry point - Invoke generate token
and create network object
if __name__ == "__main__":
    generateSessionToken()
    networkObject(network_lab, uuid)

```

CISCO ADVANCED MALWARE PROTECTION (AMP)



This section provides the information you need to understand Cisco's Advanced Malware Protection (AMP) solution.

Malware is a broad term used to define any malicious activity that aims to infect a network or a specific device. Often, the goal of malware is to steal valuable information, disrupt a user's ability to access data, or cause a device or network to crash. Ransomware is a common form of malware that has become particularly challenging for many companies. A threat actor uses ransomware to encrypt files on an endpoint and extort the owner of the information into paying a ransom to receive the decryption key.

Another common form of malware is an advanced persistent threat (APT). APTs allow threat actors to gain access to and control endpoint resources over an extended period in order to steal valuable data without being detected. In the past, the malware was deployed using malicious files to carry the payload. Today, malware is being delivered "file-lessly," by being embedded in endpoint memory or operating system functions. These new malware techniques can be difficult to detect with traditional defense mechanisms. AMP for Endpoints is useful with such techniques as it provides deep visibility to identify malware in a system, context to understand what is being affected, and control to protect against attack. AMP for Endpoints, which is Cisco's endpoint protection solution, is a cloud-managed tool delivered via the desktop client, mobile devices, and server-based endpoints. [Figure 11-7](#) shows how these various endpoints connect to AMP Private Cloud, which

has been integrated into the next-generation firewall platform and across network perimeter defense tools such as ESA and WSA. Cisco has also incorporated this tool into its industry-leading router platforms.

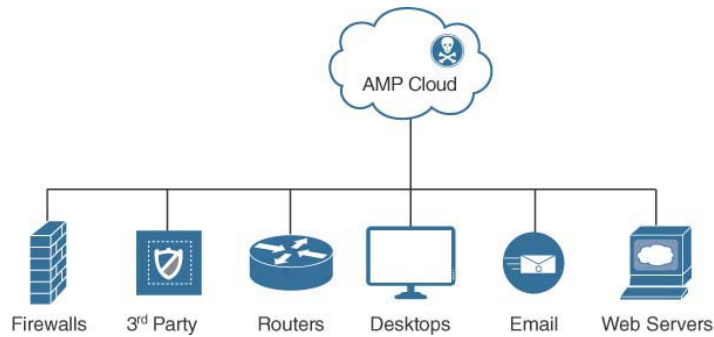


Figure 11-7 AMP Cloud and Endpoints

You can sign up for a free trial account at <https://www.cisco.com/c/en/us/products/security/amp-for-endpoints/index.html>. If you already have an account, you can log in into the AMP Private Cloud portal at <https://amp.cisco.com>.

With Cisco AMP for Endpoints, security teams leverage an integrated solution to proactively and efficiently combat threats in the following ways:

- **By streamlining incident response processes:** AMP for Endpoints allows security operations teams to have complete visibility into the location and trajectory of malware and automates remediation actions. It eliminates the arduous incident response processes that were necessary in the past. The incident response typically involves lengthy forensics to check all systems for compromise and to reimage those that have been affected. Due to resource requirements, many companies have to hire third-party specialists to manage incident response. AMP for Endpoints can reduce the time and effort required for incident response and can help companies avoid those engagements.
- **By consolidating endpoint security tools into a single solution that makes it possible to visualize the environment, patch vulnerabilities, and proactively hunt for malware:** AMP for Endpoints makes it possible to manage reputation filtering, behavior analytics, antivirus engine, exploit prevention, traffic analytics, and more from a single platform. The AMP for Endpoints management tool also integrates with AMP for Networks. This integration helps facilitate threat information sharing across network security appliances,

resulting in comprehensive malware protection and enabling a “see once, block everywhere” architecture.

- **By automating threat detection and remediation processes:** Customers can efficiently allocate time and resources and avoid the need to hire additional high-cost employees or sign expensive outsourcing contracts. This automation also promotes increased job satisfaction on security operations teams, as less time is spent on mundane research tasks.

Automation can be achieved by using the AMP for Endpoints API, which allows users to expedite their investigations by identifying which endpoints have seen a file, create custom file lists, and move endpoints in and out of triage groups. The API also makes it possible to collect and archive all events generated in an environment, which in turn makes possible extended historical data correlation. The AMP for Endpoints API enables developers and security teams to do the following:

- **Ingest events:** The API stores events in third-party tools, archives extended event histories, and correlates against other logs.
- **Search:** The API can find where a file has been, determine if a file has been executed, and capture command-line arguments.
- **Basic management:** The API allows you to create groups, move desktops or computers, and manage file lists.

AMP for Endpoints API Credentials and Authorization

The AMP for Endpoints API requires administrators to first set up an API credential. You can do this via the AMP Console by navigating to Accounts > API Credentials and then completing the dialog shown in [Figure 11-8](#).

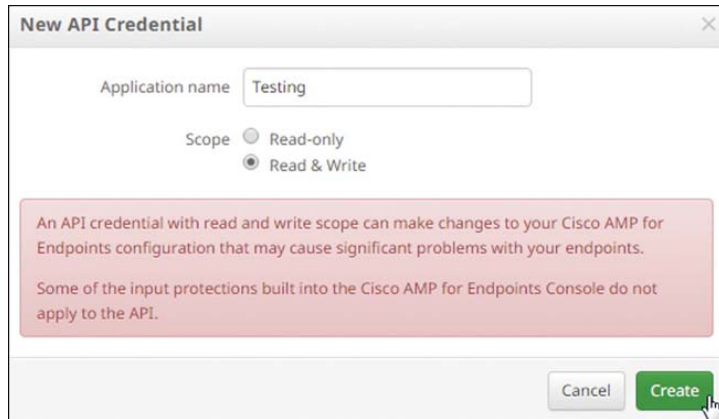


Figure 11-8 AMP Console: Creating API Credentials

Once this is done, an API client ID/key pair is generated. It looks something like this:

- **Client ID:** deadbeef123448cccc00d
- **Client key:** XXXXXXXX-YYYY-ZZZZ-0000-e384ef2dxxxx

Using the API client ID and key, you can now make the API calls as follows:

[Click here to view code image](#)

```
https://<clientID>:<clientKEY>@<api_endpoint>
```

Also, you can use basic HTTP authentication encoding for the client ID and key and the Authorization header. For the client ID and key generated, the credential is Base64 encoded as

"ZGVhZGJlZWYxMjMoNDhjY2MwMGQ6WFhYWfYWFgtWVlZWS1aWlpaLTAwMDAtZTM4NGVmMmR4eHh4", and the header looks as follows:

[Click here to view code image](#)

```
Authorization: Basic
ZGVhZGJlZWYxMjMoNDhjY2MwMGQ6WFhYWfYWFgt
WVlZWS1aWlpaLTAwMDAtZTM4NGVmMmR4eHh4
```

Now let's look at a couple of examples of the AMP for Endpoints API.

Listing All Computers

The API <https://api.amp.cisco.com/v1/computers> fetches the list of all computers. It requires basic authentication headers and uses the GET method.

[Example 11-12](#) shows a Python **requests** command that uses this API.

Example 11-12 *Python Code to Get a List of All Computers via API*

[Click here to view code image](#)

```
""" GET list of all computers via API """
import requests
url = "https://api.amp.cisco.com/v1/computers"
headers = {
    'authorization': "Basic
ZGVhZGJlZWYxMjMwMGQ6WFhYWfYWFgtWVlZWS1aWlpaL
    TAwMDAtZTM4NGVmMmR4eHh4",
    'cache-control': "no-cache",
}
response = requests.request("GET", url,
headers=headers)
print(response.text)
```

Listing All Vulnerabilities

The API <https://api.amp.cisco.com/v1/vulnerabilities> fetches a list of all vulnerabilities. The list can be filtered to show only the vulnerable programs detected for a specific time range. The **start_time** and **end_time** parameters accept the date and time expressed according to ISO 8601.

The list contains a summary of information such as the following on a vulnerability:

- Application name and version
- SHA-256 value for the executable file
- Connectors on which the vulnerable application was observed
- The most recent CVSS score

This API requires the basic authentication headers and uses the GET method. [Example 11-13](#) shows a Python **requests** command that uses this API.

Example 11-13 *Python Code to Get a List of All Vulnerabilities via API*

[Click here to view code image](#)

```
""" GET list of all vulnerabilities via API
"""
import requests
url =
"https://api.amp.cisco.com/v1/vulnerabilities"
querystring = {"offset":"0","limit":"1"}
headers = {
    'authorization': "Basic
ZGVhZGJlZWYxMjMwMGQ6WFhYWfYWFgtWVlZWS1awlpal-
TAWMDAtZTM4NGVmMmR4eHh4",
    'cache-control': "no-cache",
}
response = requests.request("GET", url,
headers=headers, params=querystring)
print(response.text)
```

[Example 11-14](#) shows a sample response to the request in [Example 11-13](#).

Example 11-14 *JSON Response Showing Vulnerabilities*

[Click here to view code image](#)

```
{
  "version": "v1.2.0",
  "metadata": {
    "links": {
      "self":
"https://api.amp.cisco.com/v1/vulnerabilities?
offset=0&limit=1"
    },
    "results": {
      "total": 1,
      "current_item_count": 1,
      "index": 0,
      "items_per_page": 1
    }
  }
}
```

```
},
"data": [
  {
    "application": "Adobe Flash Player",
    "version": "11.5.502.146",
    "file": {
      "filename": "FlashPlayerApp.exe",
      "identity": {
        "sha256":
"c1219f0799e60ff48a9705b63c14168684aed911610fec68548ea08f
        605cc42b"
      }
    },
  },
  "cves": [
    {
      "id": "CVE-2013-3333",
      "link":
"https://web.nvd.nist.gov/view/vuln/detail?
vulnId=CVE-2013-3333",
      "cvss": 10
    },
    {
      "id": "CVE-2014-0502",
      "link":
"https://web.nvd.nist.gov/view/vuln/detail?
vulnId=CVE-2014-0502",
      "cvss": 10
    }
  ],
  "latest_timestamp": 1574442349,
  "latest_date": "2019-11-
22T17:05:49+00:00",
  "groups": [
    {
      "name": "Triage",
      "description": "Triage Group for
FireAMP API Docs",
      "guid": "68665863-74d5-4bc1-ac7f-
5477b2b6406e"
    }
  ],
  "computers_total_count": 1,
  "computers": [
    {
      "connector_guid": "17d71471-805b-
4183-9121-3924b8982fac",
      "hostname": "Demo_ZAccess",
      "active": true,
      "links": {
        "computer":
"https://api.amp.cisco.com/v1/
computers/17d71471-805b-4183-9121-
3924b8982fac",
```

```

        "trajectory":
        "https://api.amp.cisco.com/v1/
        computers/17d71471-805b-4183-9121-
        3924b8982fac/trajectory",
        "group":
        "https://api.amp.cisco.com/v1/
        groups/68665863-74d5-4bc1-ac7f-
        5477b2b6406e"
    }
}
]
}

```

Table 11-6 shows all other APIs that AMP for Endpoints has to offer.

Table 11-6 AMP for Endpoints APIs

MethodAPIDescription		
G E T	https://api.amp.cisco.com/v1/audit_logs	Provide audit logs based on the filters specified in the query parameters
G E T	https://api.amp.cisco.com/v1/audit_log_types	Provide a list of all the audit log types supported by the API
G E T	https://api.amp.cisco.com/v1/computers	Fetch the list of computers
G E T	https://api.amp.cisco.com/v1/computers/user_activity	Fetch the list of computers that have observed activity by a given username
G E T	https://api.amp.cisco.com/v1/computers/activity	Search all computers across the organization for any events or activities associated with a file or network operation
G	https://api.amp.cisco.com/v1/computers/activity	Provide a general query interface

E T	cisco.com/v1/events	for events
G E T	https://api.amp.cisco.com/v1/event_types	Identify and filter events by a unique ID
P O S T	https://api.amp.cisco.com/v1/event_streams	Create a new Advanced Messaging Queue Protocol (AMQP) messaging resource for events information
G E T	https://api.amp.cisco.com/v1/file_lists/application_blocking	Return a list of application-blocking file lists
G E T / P O S T	https://api.amp.cisco.com/v1/groups	Provide basic information about groups in the organization
G E T	https://api.amp.cisco.com/v1/policies	Return a list of policies
G E T	https://api.amp.cisco.com/v1/versions	Fetch the list of versions
G E T	https://api.amp.cisco.com/v1/vulnerabilities	Provide a general query interface for vulnerabilities

CISCO IDENTITY SERVICES ENGINE (ISE)



Cisco Identity Services Engine (ISE) is a network access control and policy enforcement platform. Cisco ISE simplifies the delivery of secure access control across wired and wireless multivendor networks and remote VPN connections. With intelligent sensor and profiling capabilities, ISE penetrates deep to deliver visibility into who and what is accessing your networks and resources.

Cisco ISE provides the following benefits:

- It identifies every device and every user ID across the network.
- It enables simple provisioning for devices.
- It is a simple policy management engine that is centralized and can grant user access.
- It enables flexible integration with other solutions to speed threat detection, containment, and remediation.

Identification is required in order to access any network resources. Identification involves using credentials. Credentials are of the form passwords, certificates, tokens, or at the least the endpoint's MAC address.

Credentials reach Cisco ISE in a process called authentication. An enterprise can use various authentication protocols, depending on the type of network and the type of endpoints. With authentication, you basically tell Cisco ISE who you are.

Authentication typically results in authorization. After you reveal your identity to Cisco ISE, Cisco ISE determines your level of access. The moment an endpoint accesses the network access, the network devices generate a session ID and share it with Cisco ISE. Cisco ISE centrally knows what all the endpoints in the network are and where they are connected.

Today, enterprises already have some kind of identity services such as Microsoft Active Directory or LDAP; in addition, there could be other ODBC servers hosting some user and device accounts. A PKI infrastructure may already exist to manage certificates, and there might be

some mobile device managers and identity providers for single sign-on. ISE can seamlessly integrate with all such external identity stores and deliver network access control.

Figure 11-9 shows how Cisco ISE integrates with endpoints, networking devices, and external services.

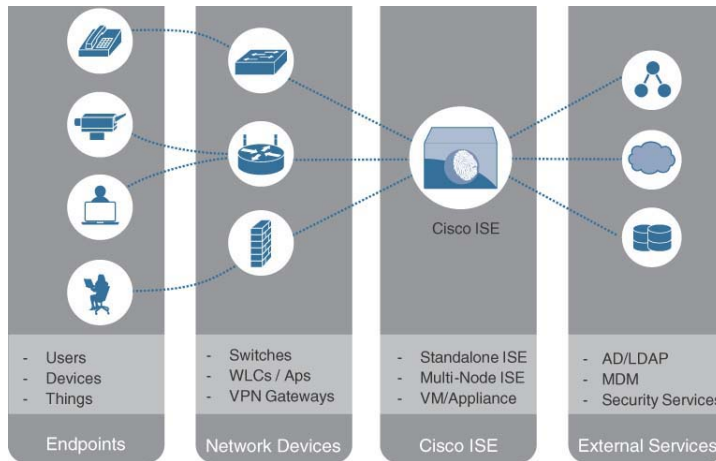


Figure 11-9 ISE: Components and Deployment

Once a profile gets associated, various policies can be enforced. These policies could be the following:

- **Time-based:** Policies can allow specific devices only at particular times.
- **Location-based:** Each network element has a piece of location information, and devices connected have specific policies attached.
- **Compliance based:** Policies can ensure that endpoints have all software patches before they are granted full access.

ISE REST APIs

The Cisco DevNet site

<https://developer.cisco.com/site/security/> provides details of all API docs located at

<https://developer.cisco.com/docs/identity-services-engine/>. In addition, you can find other resources within the DevNet Sandbox, and you can reserve and use Cisco ISE there.

Cisco ISE has two APIs:

- **Session API:** This API allows developers to gather session- and node-specific information by using Cisco ISE to monitor nodes.
- **External RESTful Services (ERS) API:** This API enables developers to perform operations on the following types of ISE resources:
 - Endpoints
 - Endpoint identity groups
 - Guest users and internal users
 - Identity groups
 - Portals
 - Profiler policies
 - Network devices
 - Network device groups
 - Security groups

The Cisco ISE administrator must assign special privileges to a user to perform operations using the ERS API. The Cisco ISE administrator can assign the following two roles to deliver services using the ERS API (see [Figure 11-10](#)):

- **External RESTful Services Admin:** For full access to all ERS methods (GET, POST, DELETE, PUT).
- **External RESTful Services Operator:** For read-only access (GET requests only).

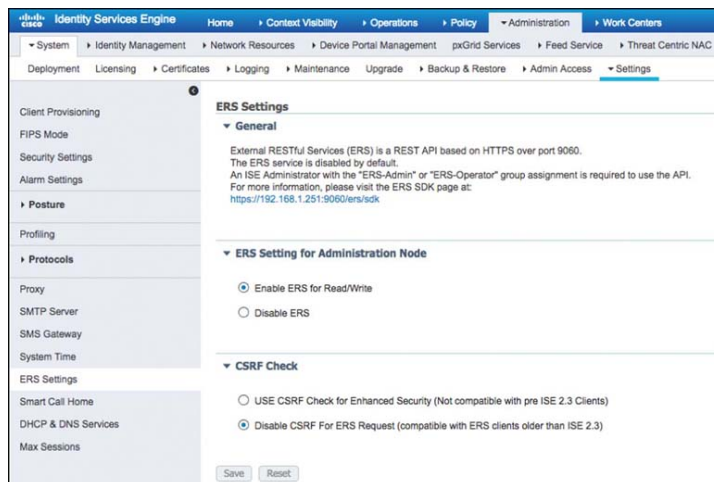


Figure 11-10 ISE ERS: Enabling API Access

ERS API Authentication

The ISE ERS API uses HTTP basic authentication, which requires the credentials to be sent in the Authorization header. The credentials are the username and password, separated by a colon (:), within a Base64-encoded string. For example, the Authorization header would contain the following string:

[Click here to view code image](#)

```
"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"
```

In this case, **ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk** is the Base64-encoded string *devasc:strongpassword* (where *devasc* is the username and *strongpassword* is the password). [Example 11-15](#) shows three lines of code that do Base64 encoding to plug the value into the Authorization headers.

Example 11-15 *Python Code to Generate Base64 Encoding*

[Click here to view code image](#)

```
import base64
encoded =
base64.b64encode('devasc:strongpassword'.encode('UTF-8')).decode('ASCII')
print(encoded)
```

All ERS API calls are made to the URL `https://<IP-of-ISE>:9060/`.

Now let's look at a couple examples of ISE ERS endpoint APIs.

Creating an Endpoint Group

The API posts the data to create a new endpoint group. It uses the POST method and requires basic authentication headers. The following shows the payload that is needed

to create an Endpoint Group called 'DevNet Associate Group':

[Click here to view code image](#)

```
Data - {  
  "EndPointGroup" : {  
    "name" : "DevNet Associate Group",  
    "description" : "DevNet Associate Group"  
  }  
}
```

Example 11-16 shows a Python **requests** command using this API.

Example 11-16 *Python POST Code to Create a New Endpoint Group*

[Click here to view code image](#)

```
""" create a new endpointgroup """  
import json  
import requests  
url =  
"https://ise.devnetsandbox.com/ers/config/endpointgroup"  
  
payload = {  
  "EndPointGroup": {  
    "name": "DevNet Associate Group",  
    "description": "DevNet Associate Group"  
  }  
}  
headers = {  
  'content-type': "application/json",  
  'accept': "application/json",  
  'authorization': "Basic  
ZGV2YXNjOnN0cm9uZ3Bhc3N3b3JkJw==",  
  'cache-control': "no-cache",  
}  
response = requests.request(  
  "POST",  
  url,  
  data=json.dumps(payload),  
  headers=headers  
)  
print(response.text)
```

The response header contains the newly created group ID:

[Click here to view code image](#)

```
Location:
https://ise.devnetsandbox.com:9060/ers/config/endpoint
group/00000000-1111-2222-3333-444444444444
```

Creating an Endpoint and Adding It to a Group

The API

<https://ise.devnetsandbox.com:9060/ers/config/endpoint> posts the data to create a new endpoint. It uses the POST method. The following shows the payload that is needed to create an Endpoint called 'DevNet Endpoint' with a specified groupId.

Method: POST

URL:

[Click here to view code image](#)

```
Data - {
  "ERSEndPoint" : {
    "name" : "DevNet_Endpoint",
    "description" : "DevNet Endpoint-1",
    "mac" : "FF:EE:DD:03:04:05",
    "groupId" : " 00000000-1111-2222-3333-444444444444",
    "staticGroupAssignment" : true
  }
}
```

This API uses the group ID from the header and requires basic authentication headers. [Example 11-17](#) shows a Python **requests** script.

Example 11-17 *Python POST Code to Create a New Endpoint*

[Click here to view code image](#)

```
""" create a new endpoint """
import json
import requests
url =
"https://ise.devnetsandbox.com/ers/config/endpoint"

payload = {
  "ERSEndPoint": {
    "name": "DevNet_Endpoint",
    "description": "DevNet Endpoint-1",
    "mac": "FF:EE:DD:03:04:05",
    "groupId": " 00000000-1111-2222-3333-
444444444444",
    "staticGroupAssignment": True
  }
}
headers = {
  'content-type': "application/json",
  'accept': "application/json",
  'authorization': "Basic
ZGV2YXNjOnN0cm9uZ3Bhc3N3b3JkJw==",
  'cache-control': "no-cache",
}
response = requests.request(
  "POST",
  url,
  data=json.dumps(payload),
  headers=headers
)
print(response.text)
```

The response header contains the newly created endpoint ID:

[Click here to view code image](#)

```
Location:
https://ise.devnetsandbox.com:9060/ers/config/endpoint/

deadbeef-1111-2222-3333-444444444444
```

Other ISE APIs

For a complete list of all Cisco ISE APIs, see <https://developer.cisco.com/docs/identity-services-engine/>.

CISCO THREAT GRID



Threat Grid is Cisco's unified malware analysis and threat intelligence platform. The idea behind Threat Grid is to present a combined analysis engine that can leverage and unify multiple capabilities and multiple infrastructures within an organization. It does this by performing static and dynamic analysis and producing reports and indicators that are human readable. File records are uploaded typically via the portal or API, and the output or results are usually consumed also via content-rich threat intelligence feeds. [Figure 11-11](#) shows the various functions that Cisco Threat Grid performs.

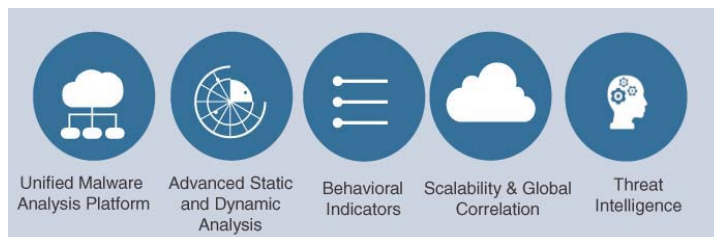


Figure 11-11 *Threat Grid in a Nutshell*

Threat Grid integrates real-time behavioral analysis and up-to-the-minute threat intelligence feeds with existing security technologies to protect a network from both known and unknown attacks. Threat Grid analyzes suspicious files against more than 1000 behavioral indicators and a malware knowledge base sourced from around the world to provide more accurate, context-rich threat analytics than ever before.

[Figure 11-12](#) shows the Cisco Threat Grid solution architecture.

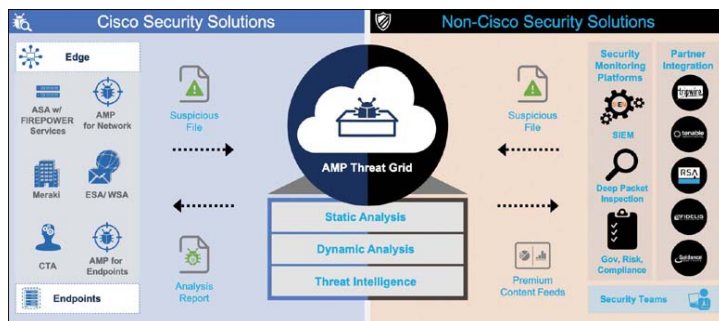


Figure 11-12 *Threat Grid Solution Architecture*

On the left side of [Figure 11-12](#) is the Cisco portfolio, and on the right are the non-Cisco or integration partners. The numbers in the figure correspond with the following details:

1. The solution can be integrated across the Cisco security portfolio, including AMP for Endpoints, AMP for Networks, ASA with Firepower, ESA, WSA, and Meraki.
2. Threat Grid can be deployed as either a cloud-based software-as-a-service product or an on-premises appliance.
3. A subscription to Threat Grid provides threat intelligence through the API.
4. Threat intelligence is automatically delivered to security-monitoring platforms.
5. Third-party integrations automatically submit samples and consume threat intelligence.
6. Context-rich analysis empowers junior analysts to make more accurate decisions more quickly.

Threat Grid APIs

The Threat Grid APIs offer a broad range of functionality, including user and organization account management, samples (file/malware/signature management), sample analysis data collection, and threat intelligence harvesting. The Cisco DevNet site <https://developer.cisco.com/threat-grid/> provides details, API documentation, and a lot of other information. You can sign up for a free trial account at <https://www.cisco.com/c/en/us/products/security/promotions-free-trials.html#~trials>. Once you have access, you can download all the APIs.

Threat Grid API Format

All Threat Grid API calls are made to the URL <https://panacea.threatgrid.com/api/>. The format of the API is as follows:

```
https://panacea.threatgrid.com/api/<ver>/<api-endpoint>?q=<query>&api_key=apikey
```

where <ver> could be “v2” or “v3”, <api-endpoint> is the actual API and the apikey is the key associated with the account.

API Keys

To get the API key from the Threat Grid portal UI, follow these steps:

- Step 1.** Go to the Threat Grid portal UI.
- Step 2.** From the Welcome menu in the upper-right corner of the navigation bar, select Manage Users.
- Step 3.** Navigate (use Search if necessary) to the User Details page for the integration’s user account and copy the API key.

This API key is used in every API call that is made to Threat Grid.

The following sections provide some examples of working with the Threat Grid APIs.

Who Am I

To see if the API key is working, you can use the GET method and the API <https://panacea.threatgrid.com/api/v3/session/whoami>. You need to pass the API key as a query parameter, as shown in [Example 11-18](#).

Example 11-18 *Threat Grid: Who Am I*

[Click here to view code image](#)

```

""" Threat Grid - who am I """
import requests
url =
"https://panacea.threatgrid.com/api/v3/session/whoami"

querystring =
{"api_key":"deadbeefelcpgib9ec0909"}
headers = {
    'cache-control': "no-cache",
}
response = requests.request(
    "GET",
    url,
    headers=headers,
    params=querystring
)
print(response.text)

```

Example 11-19 shows the JSON response to the request in Example 11-18.

Example 11-19 *Threat Grid Response*

[Click here to view code image](#)

```

{
  "api_version": 3,
  "id": 1234567,
  "data": {
    "role": "user",
    "properties": {},
    "integration_id": "z1ci",
    "email": "devasc@student.com",
    "organization_id": 666777,
    "name": "devasc",
    "login": "devasc",
    "title": "DevNet Associate",
    "api_key": " deadbeefelcpgib9ec0909",
    "device": false
  }
}

```

The Data, Sample, and IOC APIs

The Data API allows developers to search observables by specific criteria. You can do an entity search by using the `/search/` endpoint. You can pivot the Threat Grid data by

using the entity lookups `/domains/`, `/urls/`, `/paths/`, and so on.

The Sample API allows developers to submit and retrieve data for analysis. You can get the raw observable feeds by using the `/samples/feeds/` endpoint. The data is usually harvested from all sample activity, suspicious or not, and therefore has a very large footprint. You can use this API to query feeds to look at the sample for your organization only.

The Indicator of Compromise (IOC) API feeds can be accessed via the `/iocs/feeds` endpoint. With this API, you can see observables in conjunction with behavior indicators. Usually, if an item shows up in this feed, it means that there is at least some degree of suspicious behavior associated with the item. Also, filters can be applied to see only samples from your organization.

Let's look at an example. In this example, we will search for all records that have a sha1 value equal to **"8fbb3bd96b80e28ea41107728ce8c073cbadb0dd"**. To do so, we use the GET method and the API <https://panacea.threatgrid.com/api/v2/search/submissions>, and we need to pass the API key as a query parameter. **Example 11-20** shows the Python **requests** scripts to use in this case.

Example 11-20 *Threat Grid: Searching Submissions*

[Click here to view code image](#)

```
""" Threat Grid - search submissions API """
import requests
url =
"https://panacea.threatgrid.com/api/v2/search/submissions"

querystring = {
    "q":
    "8fbb3bd96b80e28ea41107728ce8c073cbadb0dd",
    "api_key": "deadbeefelcpgib9ec0909"
}
headers = {
    'cache-control': "no-cache",
```

```
}
response = requests.request(
    "GET",
    url,
    headers=headers,
    params=querystring
)
print(response.text)
```

Example 11-21 shows the JSON response to the request in Example 11-20.

Example 11-21 *Threat Grid Search Response*

[Click here to view code image](#)

```
{
  "api_version": 2,
  "id": 4482656,
  "data": {
    "index": 0,
    "total": 1,
    "took": 3956,
    "timed_out": false,
    "items_per_page": 100,
    "current_item_count": 1,
    "items": [
      {
        "item": {
          "properties": {
            "metadata": null
          },
          "tags": [],
          "vm_runtime": 300,
          "md5":
            "b5c26cab57c41208bd6bf92d495ee1f0",
          "state": "succ",
          "sha1":
            "8fbb3bd96b80e28ea41107728ce8c073cbadb0dd",
          "sample":
            "b7d3dd2a6d47ea640f719cc76c2122f8",
          "filename":
            "FlashPlayer.exe",
            ....cut
        }
      }
    ]
  }
}
```

Feeds

Threat Grid supports three different types of feeds:

- **Sample feeds:** These are all observables seen.
- **Indicator of Compromise (IOC) feeds:** These are observables seen via business intelligence. IOCs are used to indicate that the system has been affected by some form of malware.
- **Curated feeds:** These are highly curated and high-confidence feeds.

Table 11-7 shows the differences between these feeds.

Table 11-7 Threat Grid Feeds

	Sample Feeds	IOC Feeds	Curated Feeds
Version	/v2	/v2	/v3
Endpoint	/samples/feeds/	/iocs/feeds/	/feeds/
Content	All observables are seen	Observables are seen in all BIs	Observables are seen as part of a trusted high-confidence BI triggering
Pre-whitelisted	No	No	Yes
Filterable to only you/org?	Yes	Yes	No
Output Formats	JSON	JSON	JSON/CSV/Snort/STIX

Say that you want to retrieve all the curated feeds via API. The curated feed types are shown in Table 11-8.

Table 11-8 Curated Feed Types

Feed Name Description

autorun-registry	Registry entry data derived from querying registry changes known for persistence
banking-dns	Banking Trojan network communications
dga-dns	DGA domains with pseudo-randomly generated names
dll-hijacking-dns	Domains communicated to by samples leveraging DLL sideloading and hijacking techniques
doc-net-com-dns	Document (PDF, Office) network communications
downloaded-pe-dns	Samples downloading executables network communications
dynamic-dns	Samples leveraging dynamic DNS providers
irc-dns	Internet Relay Chat (IRC) network communications
modified-hosts-dns	Modified Windows hosts file network communications
parked-dns	Parked domains resolving to RFC 1918 localhost and broadcast addresses
public-ip-check-dns	Public IP address network communications
ransomware-dns	Samples communicating with ransomware servers

rat-dns	Remote Access Trojan (RAT) network communications
schedule-d-tasks	Scheduled task data observed during sample execution
sinkhole-d-ip-dns	DNS entries for samples communicating with a known DNS sinkhole
stolen-cert-dns	DNS entries observed from samples signed with a stolen certificate

Now let's look at an example of going through all the feed types and printing out the feed if any data exists. In this case, you can use the GET method and the API <https://panacea.threatgrid.com/api/v2/search/submissions>. The API key must be passed as a query parameter. **Example 11-22** show the Python **requests** script you use in this case.

Example 11-22 *Threat Grid: Listing Details for Each Curated Feed Type*

[Click here to view code image](#)

```

""" Threat Grid - List details for each curated
feed type """
import requests
FEED_URL =
"https://panacea.threatgrid.com/api/v3/feeds"
FEEDS_NAME = {
    "autorun-registry": "Autorun Registry
Malware",
    "banking-dns": "Banking Trojans",
    "dga-dns": "Domain Generation Algorithm
Destinations",
    "dll-hijacking-dns": "DLL Hijackers /
Sideloaders",
    "doc-net-com-dns": "Document File Network
Communication",
    "downloaded-pe-dns": "Dropper
Communication",
    "dynamic-dns": "Dynamic DNS Communication",
    "irc-dns": "IRC Communication",
    "modified-hosts-dns": "Modified HOSTS File

```

```

Communication",
    "public-ip-check-dns": "Public IP
Checkers",
    "ransomware-dns": "Ransomware
Communication",
    "rat-dns": "Remote Access Trojans",
    "scheduled-tasks": "Scheduled Task
Communication",
    "sinkholed-ip-dns": "Sinkholed IPs",
    "stolen-cert-dns": "Stolen Certificates",
}
for name, desc in FEEDS_NAME.items():
    url = "{}/{ }.json".format(FEED_URL, name)
    querystring =
{"api_key": "2kdn3muq7uafelcpgib9eccua7"}

    headers = {
        'cache-control': "no-cache",
        'Content-type': 'application/json',
        'Accept': 'application/json'
    }

    response = requests.request("GET", url,
headers=headers, params=querystring)

    print(response.text)

```

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 11-9](#) lists these key topics and the page number on which each is found.



Table 11-9 Key Topics

Key Topic Element	Description	Page
Paragraph	Cisco Umbrella	304
Paragraph	Enforcement APIs	308
Paragraph	Investigate APIs	311
Paragraph	Cisco Firepower	314
Paragraph	Cisco Advanced Malware Protection (AMP)	320
Paragraph	Cisco Identity Services Engine (ISE)	326
Paragraph	Cisco Threat Grid	331

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

threat

vulnerability

phishing

managed security service provider (MSSP)

next-generation firewall (NGFW)

advanced persistent threat (APT)

Chapter 12

Model-Driven Programmability

This chapter covers the following topics:

- **NETCONF:** This section introduces NETCONF—what it is, why it has been developed, and how to use it.
- **YANG:** This section covers YANG, YANG data models, and how these data models apply to networking.
- **RESTCONF:** This section covers RESTCONF, how it compares to NETCONF, and how to use it.
- **Model-Driven Telemetry:** This section provides a brief introduction to model-driven telemetry.

This chapter introduces the concepts of model-driven programmability. It discusses what led to the implementation of data models in networking and the advantages data models bring to network configuration and monitoring. This chapter begins by introducing the network programmability interfaces that network operating systems from Cisco are exposing. It then takes an in-depth look at NETCONF, why was it necessary to come up with yet another network configuration protocol, and the drawbacks and limitations that it addresses compared to similar protocols. It is difficult to talk about NETCONF without also talking about YANG, and this chapter therefore covers network data modeling. The chapter also includes sections on RESTCONF and model-driven telemetry. You will find peppered throughout this chapter examples of YANG data models and tools and libraries to interact with these models (for example, pyang and ncclient). This chapter gives special focus to Cisco IOS XE and Cisco NX-OS and their implementations of model-driven

programmability. It looks at using Postman to explore RESTCONF with a hands-on, real-life interaction with the protocol and the data models it supports.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 12-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 12-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
NETCONF	1-3
YANG	4-6
RESTCONF	7-9
Model-Driven Telemetry	10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for

an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

- 1.** What network protocols support model-driven programmability? (Choose three.)
 1. NETCONF
 2. RESTCONF
 3. SSH
 4. gRPC
- 2.** What is the default port on which the NETCONF protocol runs?
 1. TCP 22
 2. TCP 830
 3. UDP 161
 4. UDP 69
- 3.** What framework does NETCONF use to exchange messages between the client and the server?
 1. REST
 2. gRPC
 3. Apache Avro
 4. RPCs
- 4.** Which data type is not a YANG base data type?
 1. Binary
 2. Enumeration
 3. Percent
 4. Empty
- 5.** Which component of a YANG module header uniquely identifies a module?
 1. Prefix
 2. Notification
 3. Namespace
 4. Organization name
- 6.** Which of the following is a popular Python library used to interact with NETCONF servers?
 1. requests
 2. ncclient
 3. json
 4. pyang
- 7.** Which of the following are used for data encapsulation in RESTCONF messages? (Choose

two.)

1. XML
2. YANG
3. YAML
4. JSON

8. What NETCONF operation is the equivalent of the RESTCONF PATCH method?

1. <edit-config> (operation="merge")
2. <edit-config> (operation="create/replace")
3. <edit-config> (operation="patch")
4. <patch-config>

9. What RESTCONF resource allows for automatic discovery of the API root?

1. /discover/restconf
2. /.well-known/restconf
3. /.well-known/host-meta
4. /discover/root

10. What types of subscriptions are supported by model-driven telemetry? (Choose two.)

1. Static
2. Dynamic
3. Configured
4. Hard-coded

FOUNDATION TOPICS

Traditionally, network devices were managed almost exclusively through command-line interfaces (CLIs). For network monitoring, Simple Network Management Protocol (SNMP) is still widely used. While CLIs are extremely powerful, they are also highly proprietary—different from vendor to vendor—and human intervention is required to understand and interpret their output. Also, they do not scale for large network environments. SNMP also has limitations, as discussed later in this chapter. A common standard way of managing and monitoring network devices was needed. Data modeling is replacing manual configuration as it provides a standards-based, programmatic method of writing configuration data and gathering statistics and

operational data from devices. YANG data models have been developed specifically to address the need for standardization and commonality in network management. Model-driven programmability enables you to automate the configuration and control of network devices.

Cisco IOS XE, a network operating system for enterprises, Cisco NX-OS, a network operating system for data centers, and Cisco IOS XR, a network operating system for service providers, support three standards-based programmable interfaces for operating on the YANG data models: NETCONF, RESTCONF, and gRPC.



When a client request is received via NETCONF, RESTCONF, or gRPC, it is first converted into an abstract message object. Based on the namespace in the request, the message object is delivered to the underlying model infrastructure, where it is processed and executed on the device data store. The results are returned to the client through the agent on which the request was received. gRPC is an open-source project started by Google to provide a modern remote procedure call (RPC) framework that can be used in any environment—not just for network configuration. More details about it can be found at grpc.io. NETCONF and RESTCONF are discussed in detail in this chapter. [Figure 12-1](#) shows all the protocols that Cisco devices support for model-driven programmability.

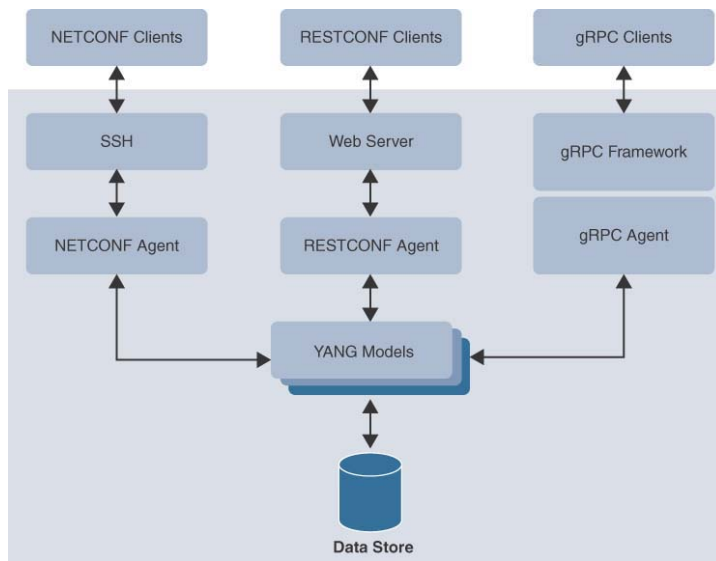


Figure 12-1 *Model-Driven Programmability on Cisco Devices*

NETCONF

In an effort to better address the concerns of network operators, the Internet Engineering Task Force (IETF) and the Internet Architecture Board (IAB) set up a workshop on network management in 2002. Several network operators were invited, and workshop participants had a frank discussion about the status of the network management industry as a whole. The conclusions and results of that workshop were captured in RFC 3535. Up to that point the industry had been extensively using Simple Network Management Protocol (SNMP) for network management. SNMP is discussed in more detail in [Chapter 18, “IP Services,”](#) but for the purposes of this chapter, you just need to know that SNMP is a network management protocol that was initially developed in the late 1980s. The intention with SNMP was to create a protocol to help with both configuration and monitoring of network devices. Several iterations of the protocol have been developed through the years, and version 3 is the latest one. SNMP has been extensively implemented by network vendors in their devices and used by network operators. By the

middle of 2002, when the IETF/IAB workshop took place, the advantages and shortcomings of SNMP were clear. SNMP had proved to work reasonably well for monitoring devices, especially when small amounts of information needed to be retrieved. For configuration purposes, however, the protocol had pretty much failed for a variety of reasons, including lack of writable MIBs, security concerns, and scaling constraints. By comparing SNMP with other network management options, such as CLIs, HTTP, and XML, network operators came up with a list of requirements for the network management technology of the future. Some of these requirements are summarized in the following list:

- This new network management technology should be easy to use.
- There should be a clear distinction between configuration data and operational data.
- It should be possible to configure extensive network services as a whole (such as IPTV and Layer 3 VPNs) instead of just configuring a network device by device.
- Configuration transactions and easy rollback in the event of failure should be supported.
- There should be a standard and consistent representation of the network configuration commands between different vendors.
- A distinction between candidate and active configurations should exist, meaning that devices should be able to hold multiple configurations and activate them as needed.



Based on these requirements, the IETF developed Network Configuration Protocol (NETCONF). The NETCONF protocol specifies the means of opening a secure management session with a network device, includes a set of operations to manipulate the configuration data and retrieve the operational state, and describes a mechanism to send out notifications. NETCONF was initially defined in 2006 in RFC 4741 and updated in 2011 with RFC 6241. While the NETCONF protocol specifies the mechanism to establish a

connection and exchange data between a network administrator and a device, it does not define the actual format of the data. This is the role of the YANG (Yet Another Next Generation) data modeling language, which was defined by the IETF in 2010 in RFC 6020 specifically to be used with NETCONF.

YANG proved to be so successful and powerful that it has evolved to be used as a general-purpose data modeling language in different programming environments. A data model is simply a well-understood and agreed upon method to describe something. As an example, let's consider a simple data model for a person. A person can be classified based on gender (male, female, other), height (feet/inches, meters/centimeters), weight (pounds, kilograms), hair color (brown, blond, black, bald, other), and eye color (brown, blue, green, other)—to name just a few characteristics. Using this generic data model for a person, you can describe an individual in a way that is easy for others to understand. If I tell you Amy is a woman who is 5'8" and 160 pounds with blond hair and blue eyes, you can very easily interpret and understand this data model information for a person.

The NETCONF protocol defines transport over SSH and includes operations and configuration data stores that allow for the management of network devices.

NETCONF authentication options are the same as for any SSH communication. Username and password as well as SSL certificates can be used, just as SSH is used with CLI commands.

NETCONF uses a client/server communication model. The server side is implemented on the network device through a NETCONF agent that acts as a northbound API. Clients connecting to the agent send partial or complete configuration data and receive state and operational data from the device. Any system that

implements the protocol can be a client; some common ones are Cisco NSO and the ncclient Python library.



Figure 12-2 shows the NETCONF architecture stack and the relationship with YANG.

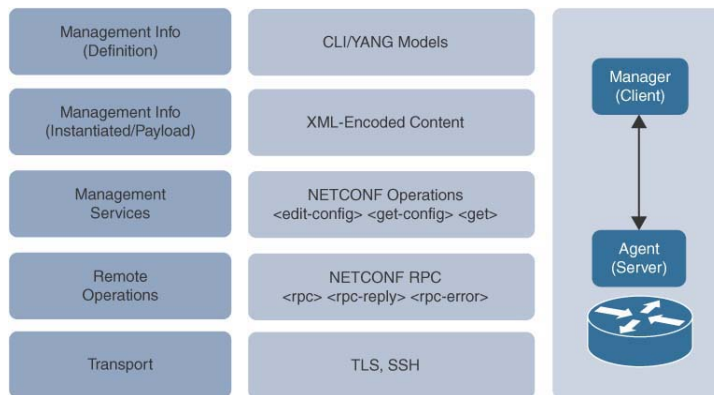


Figure 12-2 NETCONF

The main purpose of NETCONF is to transport data payloads between client and server. The data payloads can be configuration data, operational and status data, and notifications. NETCONF supports notifications, which are similar to SNMP traps.

Messages sent with NETCONF use remote procedure calls (RPCs), a standard framework for clients to send a request to a server to perform an action and return the results. The client or the manager application sends its XML-formatted message to the server, nesting the request within an `<rpc>` XML element, and the server returns results within an `<rpc-reply>` element.

As previously mentioned, the first version of NETCONF was defined in 2006, but YANG was not defined until 2010. So what was used as a payload with the first version of NETCONF? Configuration payloads in those days defaulted to CLI commands. This was an advantage

as new features exposed with new CLI commands were available over NETCONF right away; at the same time, this brought all the drawbacks of CLI to network automation, including unstructured data and vendor-specific and proprietary CLI commands. YANG was created as a standard way to define common data models for NETCONF.

NETCONF provides a set of operations to manage device configurations and retrieve status information. These operations include retrieving, configuring, copying, and deleting configuration data stores and retrieving operational data. Depending on the NETCONF capabilities that are advertised by a device, additional operations can be supported. [Table 12-2](#) shows a common set of NETCONF operations.



Table 12-2 NETCONF Operations

Operation Description	
<get>	Retrieve running configuration and device state information
<get-config>	Retrieve all or part of the specified configuration data store
<edit-config>	Load all or part of a configuration to the specified configuration data store
<copy-config>	Replace an entire configuration data store with another
<delete-config>	Delete a configuration data store

<commit>	Copy the candidate data store to the running data store
<lock> / <unlock>	Lock or unlock the entire configuration data store system
<close-session>	Gracefully terminate the NETCONF session
<kill-session>	Forcibly terminate the NETCONF session

The NETCONF <edit-config> operation supports several attributes:

- **merge:** When this attribute is specified, the configuration data is merged with the configuration at the corresponding level in the configuration data store. This is the default behavior.
- **replace:** The configuration data replaces any related configuration in the configuration data store. Only the configuration that is present in the config parameter is affected.
- **create:** The configuration data is added to the configuration only if the configuration data does not exist on the device. If the configuration already exists, an <rpc-error> message is returned with the <error-tag> value data-exists.
- **delete:** When this attribute is specified, the configuration data is deleted from the configuration data store.



NETCONF defines the existence of one or more configuration data stores and allows configuration operations on these data stores. A configuration data store is a set of configuration data that is needed to get the device from its default initial state into a desired operational state. The configuration data stores do not contain operational data. Three types of configuration data stores are defined in NETCONF:

- **running:** This data store holds the complete configuration currently active on the network device. Only one running data store can exist on a

device, and it is always present. NETCONF protocol operations refer to this data store with the <running> XML element.

- **candidate:** This data store acts as a workplace for creating and manipulating configuration data. A <commit> operation causes the configuration data contained in it to be applied to the running data store.
- **startup:** This data store contains the configuration data that is loaded when the device boots up and comes online. An explicit <copy-config> operation from the <running> data store into the <startup> data store is needed to update the startup configuration with the contents of the running configuration.

The existence of these data stores is advertised by the NETCONF device through capabilities. When opening a new session with a NETCONF server, the first message that is sent by the server contains a list of all the capabilities that the server supports.

The information that can be retrieved from a NETCONF server is separated into two classes: configuration data and state data or operational data. Configuration data is the set of writable data that starts the system operations; operational data, also known as the read-only status data, is the non-configuration data of the system (for example, interface traffic statistics, power supply status).

YANG

As mentioned in RFC 6020, YANG is “a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications.” The main motivation behind YANG was to provide a standard way to model configuration and operational data so that network devices can be configured and monitored in a standard and common way. While CLI configuration and monitoring of network devices is user friendly, it is not necessarily network automation friendly. Different network vendors have different CLIs with different features and capabilities. Trying to find a standard way to automate the configuration and monitoring of a

heterogeneous network with devices from different vendors was almost impossible before NETCONF and YANG data models were created.



YANG is a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF-based operations. It allows a complete description of all data sent between a NETCONF client and server. YANG models the hierarchical organization of data as a tree in which each node has a name and either a value or a set of child nodes. YANG provides clear and concise descriptions of the nodes, as well as the interaction between those nodes.

YANG strikes a balance between high-level data modeling and low-level bits-on-the-wire encoding. The reader of a YANG module can see the high-level view of the data model and understand how the data will be encoded in NETCONF operations.

YANG structures data models into modules and submodules. A module can import data from other external modules, and it can include data from submodules. The hierarchy can be augmented, allowing one module to add data nodes to the hierarchy defined in another module.

A YANG module contains three types of statements:

- Module-header statements describe the module and give information about it.
- Revision statements provide information about the history of the module.
- Definition statements are the body of the module, where the data model is defined.

A NETCONF server may implement a number of modules, which is the most common implementation for the Cisco devices that support NETCONF, or it might implement only one module that defines all the available data.

YANG is expressed in XML, meaning that an instance of a YANG model is an XML document.

YANG defines a set of built-in types and has a mechanism through which additional types can be defined. There are more than 20 base types to start with, including the ones in [Table 12-3](#).

Table 12-3 YANG Built-in Data Types

Data Type	Description
binary	Binary data
bits	Set of bits
boolean	True or false
decimal64	64-bit signed decimal number
empty	No value
enumeration	Enumerated strings
int8/16/32/64	Integer
uint8/16/32/64	Unsigned integer
string	Unicode string

The **typedef** YANG statement can be used to define derived types from base types. In the following example, a new data type called **percent** is created by limiting a 16-bit unsigned integer value to a range from 0 to 100:

[Click here to view code image](#)

```
typedef percent {  
    type uint16 {  
        range "0 .. 100";  
    }  
    Description "Percentage";  
}
```

This new type of data can then be used when building the YANG models. It is common practice to have definitions of new types of data contained in a YANG submodule that will then be imported in the main YANG module. IETF has also defined in RFC 6021 a large number of YANG types that are commonly used in networking. These data types are organized in the **inet-yang-types** module. The following are some of the many data types defined in this RFC:

- ipv4-address
- ipv6-address
- ip-prefix
- domain-name
- uri
- mac-address
- port-number
- ip-version
- phys-address
- timestamp
- date-and-time
- flow-label
- counter32/64
- gauge32/64

When building YANG models, the **import** statement can be used to make all these data types available in the new model:

```
import "ietf-yang-types" {  
    prefix yang;  
}
```

When importing an external YANG module, the **prefix** statement defines the prefix that will be used when accessing definitions inside the imported module. For example, in order to reference the IPv4 address data type from the newly imported **ietf-yang-types** module, you can use the following statement:

```
type yang:ipv4-address;
```



YANG defines four types of nodes for data modeling:

- Leaf nodes
- Leaf-list nodes
- Container nodes
- List nodes

The simplest component of a YANG module is the leaf node. A leaf node has one value of a specific type. The following definition of an interface name is an example of a leaf node:

[Click here to view code image](#)

```
leaf intf-name {  
    type string;  
    description "The name of the interface";  
}
```

The instantiation of this leaf node—also known as the wire representation—in a NETCONF message includes only the XML element and its value:

```
<intf-name>GigabitEthernet0/0</intf-name>
```

A leaf-list is a series of leaf nodes of a specific type. The following list of trunk interfaces is an example of a leaf-list:

[Click here to view code image](#)

```
leaf-list trunk-interfaces {  
    type string;  
    description "List of trunk interfaces";  
}
```

The NETCONF XML instantiation of this model on the wire would look like this:

[Click here to view code image](#)

```
<trunk-interfaces>TenGigabitEthernet0/1</trunk-  
interfaces>  
<trunk-interfaces>TenGigabitEthernet0/2</trunk-  
interfaces>  
<trunk-interfaces>TenGigabitEthernet0/3</trunk-  
interfaces>  
<trunk-interfaces>TenGigabitEthernet0/4</trunk-  
interfaces>
```

A container is used to group related nodes within a subtree. It has only child nodes and no value. A container can contain any number of child nodes of any type.

[Example 12-1](#) shows a container called **statistics** that contains four leaf nodes of different types:

Example 12-1 YANG Container Example

[Click here to view code image](#)

```

container statistics {
    description "A collection of interface
statistics.";
    leaf in-octets {
        type yang:counter64;
        description "The total number of
octets received on the interface.";
    }
    leaf in-errors {
        type yang:counter32;
        description "Number of inbound
packets that contained errors.";
    }
    leaf out-octets {
        type yang:counter64;
        description "The total number of
octets sent out on the interface.";
    }
    leaf out-errors {
        type yang:counter32;
        description "Number of outbound
packets that contained errors.";
    }
}

```

The XML instantiation of this model is the following:

[Click here to view code image](#)

```

<statistics>
  <in-octets>5983247896</in-octets>
  <in-errors>2578</in-errors>
  <out-octets>678845633</out-octets>
  <out-errors>0</out-errors>
</statistics>

```

The last type of node that is used for data modeling in YANG is the list. A list defines a sequence of list entries. Each entry is a record instance and is uniquely identified by the values of its key leaves. A list can be thought of as a table organized around the key leaf with all the leaves as rows in that table. For example, the user list in [Example 12-2](#) can be defined as having three leaves

called **name**, **uid**, and **full-name**, with the name **leaf** being defined as the key leaf.

Example 12-2 *YANG List Example*

```
list user {
  key name;
  leaf name {
    type string;
  }
  leaf uid {
    type uint32;
  }
  leaf full-name {
    type string;
  }
}
```

An instantiation of this data model on the wire might look in this case as shown in [Example 12-3](#).

Example 12-3 *Instantiation of the YANG List*

[Click here to view code image](#)

```
<user>
  <name>john</name>
  <uid>1000</uid>
  <full-name>John Doe</full-name>
</user>
<user>
  <name>jeanne</name>
  <uid>1001</uid>
  <full-name>Jeanne Doe</full-name>
</user>
```

We could combine all the nodes discussed so far to create a rudimentary YANG data model. Let's call this module **bogus-interfaces**. The data model defined in this module will be saved in a file that has to match the module name. In this case, the file that will contain the module will have to be called **bogus-interfaces.yang**.

There are several components that go into building the YANG data model, as shown in [Figure 12-3](#).

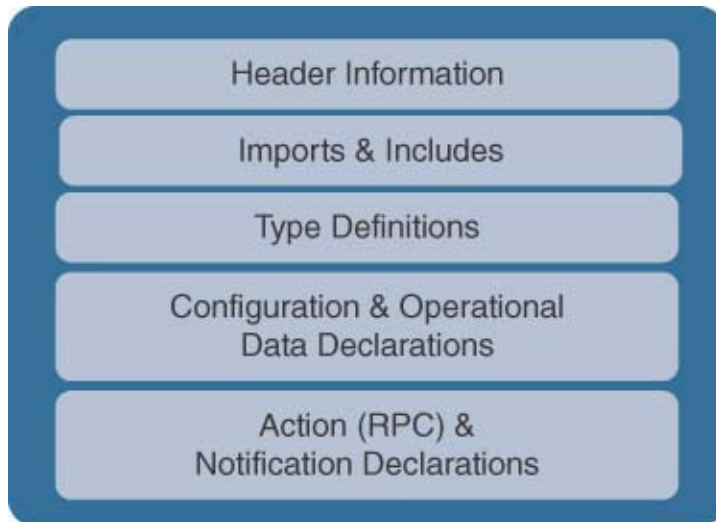


Figure 12-3 *YANG Module Components*

The YANG header contains the namespace for the module. The namespace is used to uniquely identify each module on a system that implements NETCONF. Next, the prefix is used as short notation for the module, so that it can be more easily referenced, if needed. The namespace and prefix statements in YANG have exactly the same role as in XML documents. (Keep in mind that YANG is an XML schema definition language.) The header also includes identifying information such as the name of the organization that created the module, contact information, a short description, and revision information. Revision information is particularly of interest; as the model evolves, the revision information will change. This way, a user can distinguish between different versions of the data model.

A section of imports and includes follows the header. In this section of the module, information contained in

additional modules and submodules is included, if needed. **import** is used to refer to definitions in another YANG module. It pulls references from the YANG module but does not actually pull in the body of the file that's being imported. It's very common to have definitions of data types and groupings defined in a module that are imported and used in several other modules. The **include** statement is used to pull a submodule into a main module. A module does not have to be contained in a single file. It can easily be split into several submodules for maintenance purposes and for easier separation of content. The **include** statement completely pulls the submodule into the main module.

After the **import** and **include** sections of the data model, an optional section of data type definitions follows. This is where any extra data types that are needed to build the module are defined. The next section is where the actual data model is built. All the configuration and operational data declarations are in this part. An optional section at the end is used to define custom RPCs and notifications.

A YANG data model can be defined by combining all the data nodes discussed earlier in this chapter. [Example 12-4](#) shows this in a YANG data model example.

Example 12-4 *YANG Data Model Example*

[Click here to view code image](#)

```
// Contents of "bogus-interfaces.yang"
module bogus-interfaces {
  namespace
  "http://bogus.example.com/interfaces";
  prefix "bogus";

  import "ietf-yang-types" {
    prefix yang;
  }

  organization "Bogus Inc.";
  contact "john@bogus.example.com";
  description
```

```
        "The module for entities implementing
the Bogus interfaces .";

    revision 2020-01-06 {
        description "Initial revision.";
    }
    container interfaces {
        leaf intf-name {
            type string;
            description "The name of the
interface";
        }

        leaf-list trunk-interfaces {
            type string;
            description "List of trunk
interfaces";
        }

        container statistics {
            description "A collection of
interface statistics.";
            leaf in-octets {
                type yang:counter64;
                description "Total
number of octets received on the
interface.";
            }

            leaf in-errors {
                type yang:counter32;
                description "Number of
inbound packets that contained
errors.";
            }

            leaf out-octets {
                type yang:counter64;
                description "Total
number of octets sent out on the
interface.";
            }

            leaf out-errors {
                type yang:counter32;
                description "Number of
outbound packets that contained
errors.";
            }
        }
    }
    list user {
        key name;
        leaf name {
            type string;
```

```
    }
    leaf uid {
        type uint32;
    }
    leaf full-name {
        type string;
    }
}
}
```

The IETF and other standards-setting groups have defined several YANG models. The following are some of the IETF YANG models:

- RFC 7224: *IANA Interface Type YANG Module*
- RFC 7317: *A YANG Data Model for System Management*
- RFC 7407: *A YANG Data Model for SNMP Configuration*
- RFC 8299: *YANG Data Model for L3VPN Service Delivery*
- RFC 8343: *A YANG Data Model for Interface Management*
- RFC 8344: *A YANG Data Model for IP Management*



For an up-to-date and complete collection of YANG modules, see <https://github.com/YangModels/yang>. Anyone can create YANG data models. In most cases, though, they are created by network equipment vendors, the IETF, and OpenConfig. OpenConfig is a group of network operators, including AT&T, Google, Microsoft, Facebook, and Apple, that was created to define standards intended to make networks more open and programmable. Because OpenConfig is not a formal standards body, the OpenConfig data models change rapidly. The data models released by the IETF are also called *open* data models. Open data models provide a common interface across multiple platforms, acting as the lowest common denominator for data models. Native data models are specific for each network vendor and each network operating system. Native models are

usually not interoperable with other platforms; they closely mirror the structure of the CLI and define extra features that are specific to each vendor.

YANG is an extensible language, allowing extension statements to be defined by standards bodies, vendors, and individuals. The statement syntax allows these extensions to coexist with standard YANG statements in a natural way. YANG allows a module to augment a data model by inserting additional YANG nodes into the model. This allows vendors to add vendor-specific parameters to standard data models in an interoperable way. The **augment** statement allows a module to add data to the schema tree. As an example, let's consider the **statistics** container as being part of the **interfaces-statistics** module. The statistics container looks as shown in [Example 12-5](#).

Example 12-5 *YANG Data Model Augmentation*

[Click here to view code image](#)

```
container statistics {
  description "A collection of interface
statistics.";
  leaf in-octets {
    type yang:counter64;
    description "The total number of
octets received on the interface.";
  }
  leaf in-errors {
    type yang:counter32;
    description "Number of inbound
packets that contained errors.";
  }
  leaf out-octets {
    type yang:counter64;
    description "The total number of
octets sent out on the interface.";
  }
  leaf out-errors {
    type yang:counter32;
    description "Number of outbound
packets that contained errors.";
  }
}
```

In order to augment the module, the "augment" statement is used as follows:

```
import interface-statistics {
    prefix "intf-stats";
}
augment "/intf-stats:statistics" {
    leaf in-unicast-pkts {
        type yang:counter64;
        description "Number of unicast
packets received on the interface.";
    }
    leaf out-unicast-pkts {
        type yang:counter64;
        description "Number of unicast
packets sent out the interface.";
    }
}
```

Notice in [Example 12-5](#) that the **augment** statement takes as input the relative path for the **statistics** object in the **intf-stats** module. Two new leaves are added to the model: **in-unicast-pkts** for inbound unicast data packets on an interface and **out-unicast-pkts** for outbound unicast data packets on the interface.

YANG supports the definition of NETCONF RPCs. This means that in addition to the predefined NETCONF operations (**get**, **get-config**, and **edit-config**), additional operations can be defined in YANG. This is done through the **rpc** declaration. It's a common occurrence to have several different operating system images stored in the flash memory of networking devices, including the current running image, the previous image, and an image for upgrade. Upgrading software packages and operating system images is a common operational requirement for network administrators. NETCONF does not support this common operational task by default. [Example 12-6](#) shows the YANG definition of a new NETCONF action to activate a specific software image on the device that implements this action.

The RPC definition contains inputs and outputs. In this case, the input leaf specifies the binary of the image that will be activated, and the output leaf contains the status of the RPC action. Keep in mind that the additional RPCs are defined through YANG models and available on the server side on the device. The client or the management system interacting with the server would also have to be able to interpret the output and execute the custom RPC action.

Example 12-6 *NETCONF RPC Action Definition*

[Click here to view code image](#)

```
rpc activate-software-image {
  input {
    leaf image {
      type binary;
    }
  }
  output {
    leaf status {
      type string;
    }
  }
}
```

NETCONF supports notification messages, which are similar to SNMP traps. The **notification** statement is used to define NETCONF notifications. It takes only one argument, which is the identifier, followed by a block of data statements that contain the detailed notification information. [Example 12-7](#) shows a simple NETCONF notification for a configuration change.

Example 12-7 *NETCONF Notification Definition*

[Click here to view code image](#)

```
notification config-change {
  description "The configuration change.";
  leaf operator-name {
    type string;
  }
}
```

```
leaf-list change {
    type instance-identifier;
}
}
```

The notification in [Example 12-7](#) has two nodes: one for the name of the operator that performed the change and one for the list of changes performed during the configuration session.

Reading a YANG data model straight from an RFC specification can be a daunting task, especially when you're new to data models. **pyang** is a Python program that has been developed to validate and convert YANG models to more user-friendly and readable formats. pyang can be downloaded from <https://github.com/mbj4668/pyang>. To see how pyang version 2.1.1 works, consider the **ietf-interfaces** YANG model defined in RFC 8343 and downloaded from <https://github.com/YangModels/yang>. This data model follows the structure discussed in this chapter, and [Example 12-8](#) shows what the **ietf-interfaces.yang** file looks like.

Example 12-8 *ietf-interfaces YANG Data Model Snippet*

[Click here to view code image](#)

```
module ietf-interfaces {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-
interfaces";
  prefix if;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF NETMOD (Network Modeling) Working
Group";

  contact
```

```

    "WG Web:
    <https://datatracker.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>

    Editor:  Martin Bjorklund
            <mailto:mbj@tail-f.com>";
    description
    "This module contains a collection of YANG
    definitions for
    managing network interfaces.+

    Copyright (c) 2018 IETF Trust and the
    persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and
    binary forms, with or
    without modification, is permitted
    pursuant to, and subject
    to the license terms contained in, the
    Simplified BSD License
    set forth in Section 4.c of the IETF
    Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part
    of RFC 8343; see
    the RFC itself for full legal notices.";

    revision 2018-02-20 {
    description
    "Updated to support NMDA.";
    ... omitted output

```

You can run this YANG model through `pyang` and specify the tree format by using the command **`pyang -f tree ietf-interfaces.yang`**. This results in the output shown in [Example 12-9](#).

Example 12-9 *pyang* Output for **ietf-interfaces.yang**

[Click here to view code image](#)

```

module: ietf-interfaces
  +--rw interfaces
  |   +--rw interface* [name]
  |       +--rw name

```

```

string
|   +--rw description?
string
|   +--rw type
identityref
|   +--rw enabled?
boolean
|   +--rw link-up-down-trap-enable?
enumeration {if-mib}?
|   +--ro admin-status
enumeration {if-mib}?
|   +--ro oper-status
enumeration
|   +--ro last-change?
yang:date-and-time
|   +--ro if-index                               int32
{if-mib}?
|   +--ro phys-address?
yang:phys-address
|   +--ro higher-layer-if*
interface-ref
|   +--ro lower-layer-if*
interface-ref
|   +--ro speed?
yang:gauge64
|   +--ro statistics
|   +--ro discontinuity-time
yang:date-and-time
|   +--ro in-octets?
yang:counter64
|   +--ro in-unicast-pkts?
yang:counter64
|   +--ro in-broadcast-pkts?
yang:counter64
|   +--ro in-multicast-pkts?
yang:counter64
|   +--ro in-discards?
yang:counter32
|   +--ro in-errors?
yang:counter32
|   +--ro in-unknown-protos?
yang:counter32
|   +--ro out-octets?
yang:counter64
|   +--ro out-unicast-pkts?
yang:counter64
|   +--ro out-broadcast-pkts?
yang:counter64
|   +--ro out-multicast-pkts?
yang:counter64
|   +--ro out-discards?
yang:counter32
|   +--ro out-errors?
yang:counter32

```

```

x--ro interfaces-state
  x--ro interface* [name]
    x--ro name          string
    x--ro type          identityref
    x--ro admin-status  enumeration
  {if-mib}?
    x--ro oper-status  enumeration
    x--ro last-change? yang:date-and-
time
    x--ro if-index    int32 {if-
mib}?
    x--ro phys-address? yang:phys-
address
    x--ro higher-layer-if* interface-
state-ref
    x--ro lower-layer-if* interface-
state-ref
    x--ro speed?      yang:gauge64
    x--ro statistics
      x--ro discontinuity-time
yang:date-and-time
      x--ro in-octets?
yang:counter64
      x--ro in-unicast-pkts?
yang:counter64
      x--ro in-broadcast-pkts?
yang:counter64
      x--ro in-multicast-pkts?
yang:counter64
      x--ro in-discards?
yang:counter32
      x--ro in-errors?
yang:counter32
      x--ro in-unknown-protos?
yang:counter32
      x--ro out-octets?
yang:counter64
      x--ro out-unicast-pkts?
yang:counter64
      x--ro out-broadcast-pkts?
yang:counter64
      x--ro out-multicast-pkts?
yang:counter64
      x--ro out-discards?
yang:counter32
      x--ro out-errors?
yang:counter32

```

Example 12-9 shows a compressed version of a YANG data model that is 1123 lines long, with only the critical information left. The clear separation between

configuration (**rw**, which means read/write) and state (**ro**, which means read-only) data can be observed right away. All the leafs and groupings in the model and their types are also displayed. pyang is extremely versatile and useful as it validates the correctness of YANG models and also converts them into several formats, including yin, dsdl, tree, jstree, uml, and jsonxsl.

Let's now explore the NETCONF interface that is provided by the Cisco IOS XE operating system. Cisco IOS XE is a network operating system for enterprise switching, routing, wired, and wireless access. The following examples use Cisco IOS XE 16.6.3 running on a Cisco CSR1000V device with NETCONF enabled. Enabling NETCONF on the Cisco CSR1000V running Cisco IOS XE version 16.6.3 is extremely simple: Just issue the **netconf-yang** command in the configuration mode of the device. By default, the NETCONF server on the device runs on TCP port 830 and uses the SSH process for transport. A NETCONF session can be established by using an SSH client and specifying port 830 when connecting to the device. When establishing a NETCONF session with a device, a complete list of all the capabilities and YANG modules supported by the device is exchanged with the client. An example of this exchange can be observed in [Example 12-10](#).

Example 12-10 *Capabilities Exchanged When Establishing a NETCONF Session*

[Click here to view code image](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>

    <capability>urn:ietf:params:netconf:base:1.1</capability>

    <capability>urn:ietf:params:netconf:capability:writable-
running:1.0</capability>
```



```
<capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
<capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
<capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
<capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
<capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
<capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-Ω
<capability>urn:ietf:params:netconf:capability:yang-library:1.0?revision=2016-06-21
&amp;module-set-id=0de3f66ee656759ede57b7f3b6cd310d</capability>
<capability>http://tailf.com/ns/netconf/actions/1.0</capability>
<capability>http://tailf.com/ns/netconf/extensions</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ip-deviation?module=cisco-xe-ietf-ip-deviation&amp;revision=2016-08-10</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ipv4-unicast-routing-deviation?module=cisco-xe-ietf-ipv4-unicast-routing-deviation&amp;revision=2015-09-11</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ipv6-unicast-routing-deviation?module=cisco-xe-ietf-ipv6-unicast-routing-deviation&amp;revision=2015-09-11</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-ospf-deviation?module=cisco-xe-ietf-ospf-deviation&amp;revision=2015-09-11</capability>
<capability>http://cisco.com/ns/cisco-xe-ietf-routing-deviation?module=cisco-xe-ietf-routing-deviation&amp;revision=2016-07-09</capability>
<capability>http://cisco.com/ns/cisco-xe-openconfig-acl-deviation?module=cisco-xe-openconfig-acl-deviation&amp;revision=2017-08-21</capability>
<capability>http://cisco.com/ns/mpls-static/devs?module=common-mpls-static-devs&amp;revision=2015-09-11</capability>
```

```
<capability>http://cisco.com/ns/nvo/devs?
module=nvo-devs&revision=2015-09-11</
capability>
<capability>http://cisco.com/ns/yang/Cisco-IOS-
XE-aaa?module=Cisco-IOS-XE-
aaa&revision=2017-09-05</capability>
<capability>http://cisco.com/ns/yang/Cisco-IOS-
XE-acl?module=Cisco-IOS-XE-
acl&revision=2017-08-01</capability>
<capability>http://cisco.com/ns/yang/Cisco-IOS-
XE-acl-
oper?module=Cisco-IOS-XE-acl-
oper&revision=2017-02-07</
capability>
...
</capabilities>
<session-id>5546</session-id></hello>]]>]]>
```

All the YANG models supported by the device are included in the HELLO message, and so is a session ID. This version of Cisco IOS XE supports both NETCONF 1.0 and 1.1 as well as the writable-running capability (which makes it possible to write directly to the running configuration of the device), the rollback-on-error 1.0 capability, and the notification 1.0 capability. These capabilities translate into YANG models defined by the IETF, as you can see in their XPath definition:

```
urn:ietf:params:netconf:
```

The delimiter string `]]>]]>` at the end of the response signifies the end of the message. Next, the client needs to respond to the hello message with a list of capabilities that it supports. The simplest response that a client can send is the following:

[Click here to view code image](#)

```
<?xml version="1.0" encoding="UTF-8"?>
<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
```

```
<capability>urn:ietf:params:netconf:base:1.0</capability>

</capabilities>

</hello>]]>]]>
```

There is no reply from the device to the client's initial hello message, which is expected behavior; however, an active NETCONF connection to the device is now established. In order to retrieve the running configuration of the device, the following XML message is sent next:

[Click here to view code image](#)

```
<?xml version="1.0" encoding="UTF-8"?>

<rpc message-id="101"
xmlns="urn:ietf:params:xml:ns:netconf:
base:1.0">

  <get-config>

    <source>

      <running/>

    </source>

  </get-config>

</rpc>]]>]]>
```

This example shows a NETCONF RPC message with an ID of 101 defined by the base IETF NETCONF 1.0 namespace. The message ID is used to keep track of the response received from the device. The operation within the message is **get-config**, with the target data source of the configuration that is stored in the **running** data store. The response contains the running configuration of the device enclosed in an <rpc-reply> XML element and is displayed in [Example 12-11](#).

Example 12-11 *Snippet of the NETCONF get-config Operation for the Running Configuration*

[Click here to view code image](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="101">
  <data>
    <native
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
native">
      <version>16.6</version>
      <boot-start-marker />
      <boot-end-marker />
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec />
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec />
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console
xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
platform">
          <output>serial</output>
        </console>
      </platform>
      <hostname>csr1kv</hostname>
      <enable>
        <password>
          <secret>cisco</secret>
        </password>
        <secret>
          <type>5</type>

        <secret>$1$A1eZ$Vp95LGe0XX3Cf0K05K5Nf1</secret>
        </secret>
      </enable>
      ... omitted output
    </data>
  </rpc-reply>

```

While interacting with a NETCONF server this way is possible, it is very cumbersome and error prone. Several

publicly available NETCONF tools and clients make it much easier to interact with a NETCONF server. One popular NETCONF client is the Python 3 **ncclient** library. Next, let's use ncclient to explore the Cisco NX-OS NETCONF interface.

Cisco NX-OS is the network operating system that powers Cisco Nexus switches in data centers around the world. Built on top of Linux, Cisco NX-OS, also known as Open NX-OS, exposes the full power of Linux, with rich programmable interfaces and a diverse set of automation tools. The following examples use Cisco NX-OS version 9.2.3, ncclient version 0.6.7, and Python 3.7.4. You enable NETCONF on Cisco NX-OS version 9.2.3 by issuing the **feature netconf** command in configuration mode. Much as with Cisco IOS XE, the NETCONF server is running by default on TCP port 830 and uses the SSH protocol as transport.

The **manager** module within the ncclient library handles all NETCONF RPC connections between the client and the server. By using the **connect** method available with the **manager** module, it is very easy to establish a connection to a NETCONF server. The **connect** method takes as input parameters the hostname or the IP address of the NETCONF server, the port on which the server is running, the username and password that are used to connect, and the **hostkey_verify** parameter, which specifies whether the script should look for hostkey verification information in the `~/.ssh/known_hosts` location on the server on which the script is run. **Example 12-12** shows a simple Python 3 script that retrieves the capabilities of the NETCONF server and displays them to the console.

Example 12-12 *Python Script to Retrieve and Print NETCONF Capabilities*

[Click here to view code image](#)

```

#!/usr/bin/env python
""" Get the capabilities of a remote device
with NETCONF """

from ncclient import manager

NXOS_HOST = "10.10.10.10"
NETCONF_PORT = "830"
USERNAME = "admin"
PASSWORD = "password"

# create a get_capabilities() method
def get_capabilities():
    """
    Method that prints NETCONF capabilities of
    remote device.
    """
    with manager.connect(
        host=NXOS_HOST,
        port=NETCONF_PORT,
        username=USERNAME,
        password=PASSWORD,
        hostkey_verify=False
    ) as device:

        # print all NETCONF capabilities
        print('\n***NETCONF Capabilities for
device {}***\n'.format(NXOS_HOST))
        for capability in
device.server_capabilities:
            print(capability)

if __name__ == '__main__':
    get_capabilities()

```

After importing the **manager** module from `ncclient`, the NETCONF server IP address, port number, username, and password are defined. The **get_capabilities()** function establishes a connection to the NETCONF server, retrieves the server capabilities with **device.server_capabilities()**, iterates with a **for** loop over each capability, and displays the capabilities to the console. [Example 12-13](#) shows the result of running this script on a Cisco Nexus 9000 switch running Cisco NX-OS 9.2.3 with NETCONF enabled.

Example 12-13 *Output of the Python Script from Example 12-12*

[Click here to view code image](#)

```
***NETCONF Capabilities for device
10.10.10.10***

urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-
running:1.0
urn:ietf:params:netconf:capability:rollback-on-
error:1.0
urn:ietf:params:netconf:capability:candidate:1.0

urn:ietf:params:netconf:capability:validate:1.1
urn:ietf:params:netconf:capability:confirmed-
commit:1.1
http://cisco.com/ns/yang/cisco-nx-os-device?
revision=2019-02-17&module=Cisco-NX-OS-
device&deviations=Cisco-NX-OS-device-deviations
```

At this writing, Cisco NX-OS by default implements the base NETCONF capabilities for version 1.0 and 1.1 as well as the YANG models for writable running configuration, rollback-on-error, candidate data store, and others. Additional YANG models are supported and can be installed manually.

In order to perform changes on the NETCONF server, the **edit_config** method available with the **manager** module can be used. In [Example 12-14](#), a new loopback interface is created, the administrative state is switched to up, and an IP address is configured on the newly created interface.

Example 12-14 *Python Script to Create a New Loopback Interface*

[Click here to view code image](#)

```
#!/usr/bin/env python
""" Add a loopback interface to a device with
NETCONF """
```

```

from ncclient import manager

NXOS_HOST = "10.10.10.10"
NETCONF_PORT = "830"
USERNAME = "admin"
PASSWORD = "password"
LOOPBACK_ID = "01"
LOOPBACK_IP = "1.1.1.1/32"

# create add_loopback() method
def add_loopback():
    """
    Method that adds loopback interface and
    configures IP address
    """

    add_loop_interface = """<config>
<System
xmlns="http://cisco.com/ns/yang/cisco-nx-os-
device">
    <intf-items>
        <lb-items>
            <LbRtdIf-list>
                <id>lo{id}</id>
                <adminSt>up</adminSt>
                <descr>Intf configured via
NETCONF</descr>
            </LbRtdIf-list>
        </lb-items>
    </intf-items>
    <ipv4-items>
        <inst-items>
            <dom-items>
                <Dom-list>
                    <name>default</name>
                    <if-items>
                        <If-list>
                            <id>lo{id}</id>
                            <addr-items>
                                <Addr-list>
                                    <addr>
{ip}</addr>
                                </Addr-
list>
                            </addr-items>
                        </If-list>
                    </if-items>
                </Dom-list>
            </dom-items>
        </inst-items>
    </ipv4-items>
</System>
</config>""".format(id=LOOPBACK_ID,
ip=LOOPBACK_IP)

```



```

with manager.connect(
    host=NXOS_HOST,
    port=NETCONF_PORT,
    username=USERNAME,
    password=PASSWORD,
    hostkey_verify=False
) as device:

    # Add loopback interface
    print("\n Adding Loopback {} with IP
address {} to device {}...\n".\
        format(LOOPBACK_ID, LOOPBACK_IP,
NXOS_HOST))
    netconf_response =
device.edit_config(target='running',
    config=add_loop_interface)
    # Print the XML response
    print(netconf_response)

if __name__ == '__main__':
    add_loopback()

```

After importing the **manager** module from the `ncclient` library, the connection details for the NETCONF server are specified. The loopback interface ID and IP address that will be created are also defined at this point in the script. The **add_loopback()** function contains the XML document that specifies the NETCONF configuration information in the **add_loop_interface** variable. The YANG model that is used in this case is specified through the XPath value, **xmlns="http://cisco.com/ns/yang/cisco-nx-os-device"**. Recall that this YANG data model is included by default in Cisco NX-OS. The actual path in the tree-like YANG model where the configuration changes are going to be performed is also included in the XML document. This XML document matches one to one the YANG data model **cisco-nx-os-device**. The script uses the **connect** method next to establish a connection to the NETCONF server and then send the XML configuration document to the **running** data store. The response received is displayed to the console using the **print**

function. The result of successfully running this script looks similar to the following one:

[Click here to view code image](#)

```
Adding Loopback 01 with IP address 1.1.1.1/32 to
device
10.10.10.10...

<?xml version="1.0" encoding="UTF-8"?>

<rpc-reply message-id="urn:uuid:6de4444b-9193-
4b74-837b-
e3994d75a319"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

    <ok/>

</rpc-reply>
```

RESTCONF

According to RFC 8040, RESTCONF is “an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the datastore concepts defined in the Network Configuration Protocol (NETCONF).” Basically, RESTCONF provides a REST-like interface to the NETCONF/YANG interface model.

Although NETCONF provides significant improvements over SNMP, it doesn’t provide network interfaces with a good REST API interface. Rather than developing an entirely new protocol and data model, the IETF extended NETCONF into RESTCONF. RESTCONF is not a replacement for NETCONF. Rather, RESTCONF provides an API that aligns with other web application APIs to provide an easy entry point for developers.



Like other REST APIs, RESTCONF uses the HTTPS protocol to encapsulate and send messages. Authentication is accomplished using typical HTTP authentication models, such as basic authentication,

where usernames and passwords are Base64 encoded and transmitted from the client to the server via an Authentication header. [Figure 12-4](#) shows the RESTCONF protocol stack.

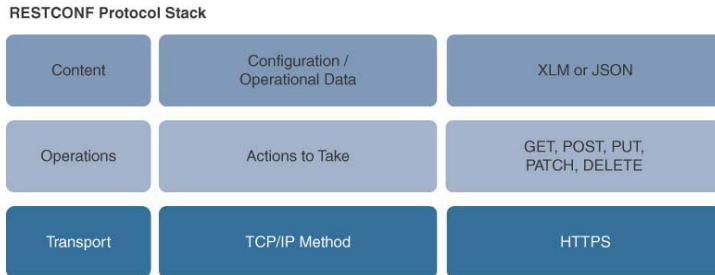


Figure 12-4 RESTCONF Protocol Stack

REST APIs typically implement CRUD (create, retrieve, update, and delete) operations, leveraging HTTP methods. RESTCONF maps the NETCONF operations into HTTP methods as shown in [Table 12-4](#).

Table 12-4 RESTCONF HTTP Methods and Their Corresponding NETCONF Operations

RESTCONF	NETCONF
GET	<get>, <get-config>
POST	<edit-config> (operation="create")
PUT	<edit-config> (operation="create/replace")
PATCH	<edit-config> (operation="merge")
DELETE	<edit-config> (operation="delete")

The available RESTCONF methods and their corresponding NETCONF operations are as follows:

- The HTTP GET method is sent in the RESTCONF request by the client to retrieve data and metadata for a specific resource. It translates into the NETCONF <get> and <get-config> operations. The GET method is supported for all resource types except operation resources.
- The HTTP POST method is used for NETCONF RPCs and to create a data resource. It represents the same semantics as the NETCONF <edit-config> operation with operation= “create”.
- The PUT method is used to create or replace the contents of the target resource. It is the equivalent of the NETCONF <edit-config> operation with operation=“create/replace”.
- The PATCH method provides the framework for resource patching mechanism. It is the equivalent of the NETCONF <edit-config> operation with operation=“merge”.
- The HTTP DELETE method is used to delete the target resource and is the equivalent of the NETCONF <edit-config> with operation=“delete”.

RESTCONF data is encoded with either XML or JSON. Compared with NETCONF, RESTCONF has added support for the JSON encoding. There are two new media types defined for RESTCONF:

- **application/yang.api+xml** for XML-encoded payloads
- **application/yang.api+json** for JSON-encoded data

With all REST APIs, including RESTCONF APIs, the URI is important in identifying the data being requested or configured. A unique characteristic of RESTCONF is that it lacks any true API documentation that a developer would use to learn how to use it. Rather, the YANG models themselves are the API documentation.



All RESTCONF URIs use the following format:

https://<ADDRESS>/<ROOT>/data/<[YANG_MODULE:]CONTAINER>/ <LEAF>[?<OPTIONS>]

where

- ADDRESS is the IP address or the hostname and port number where the RESTCONF agent is available.

- ROOT is the main entry point for RESTCONF requests. Before connecting to a RESTCONF server, the root must be determined. Per the RESTCONF standard, devices implementing the RESTCONF protocol should expose a resource called `/.well-known/host-meta` to enable discovery of ROOT programmatically.
- data is the RESTCONF API resource type for data. The operations resource type is also available for access to RPC operations.
- [YANG_MODULE:]CONTAINER is the base mode container being used.
- LEAF is an individual element from within the container.
- [?<OPTIONS>] are the options that some network devices may support that are sent as query parameters that impact the returned results. These options are optional and can be omitted. The following are some examples of possible options:
 - **depth = unbounded:** If nothing is specified, this option is the default. It indicates that the returned data should follow the nested models to the end. Integer values specifying the depth of the data to be returned are also supported.
 - **content = [all, config, nonconfig]:** This query option controls the type of data returned. If nothing is specified, the default value, **all**, is used.
 - **fields = expr:** This option limits what leaves are returned in the response.

Consider the **ietf-interfaces** YANG model defined in RFC 8343 and partially presented in [Figure 12-5](#). Using the data model details, the URIs for RESTCONF requests can be easily constructed. Generating the code to support RESTCONF APIs and the mapping of those API calls into NETCONF can be automated because the mapping from a YANG data model to a RESTCONF URI is known and well defined.

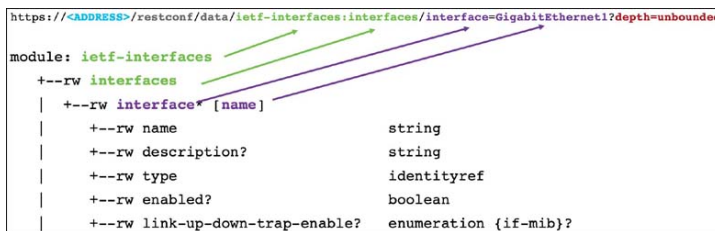


Figure 12-5 Mapping Between the YANG Model and RESTCONF URI

RESTCONF helps support a common, REST-based programming model for network automation in general.

Enabling RESTCONF on Cisco IOS XE is simple and straightforward. First, RESTCONF runs over HTTPS, so a secure HTTP server is enabled on the device by issuing **ip http secure-server** in configuration mode. Next, RESTCONF is enabled with the **restconf** command in configuration mode. Once these two commands are entered and a network administrator account exists on the device, RESTCONF is ready to be accessed. Cisco IOS XE 16.6.3 and Postman 7.13 are used in the following examples.

Devices that implement the RESTCONF protocol should expose a resource called `/.well-known/host-meta` in order to enable the discovery of the REST API root resource programmatically. Using Postman to perform a GET request on the `https://{{host}}:{{port}}/.well-known/host-meta` endpoint results in the output shown in [Figure 12-6](#).

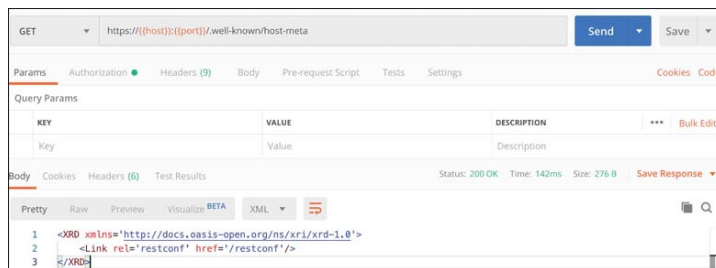


Figure 12-6 Getting the REST API Root Resource

The **href** attribute in the response contains the path to the REST API root resource: `/restconf`. Exploring the API further and performing a GET request on `https://{{host}}:{{port}}/restconf` will expose the top-level resources available in RESTCONF: **data** and **operations**, as shown in [Figure 12-7](#).

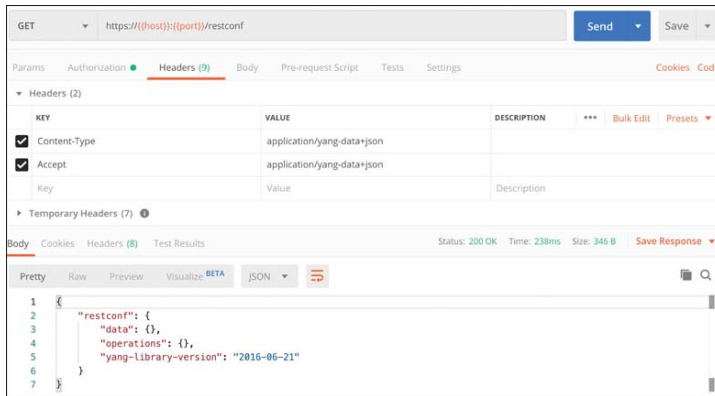


Figure 12-7 *Top-Level Resource Available in RESTCONF*

To explore further down the API tree and into the YANG model and retrieve a complete list of all the interfaces, their status, and traffic statistics through the RESTCONF interface, you can perform a GET request on the `https://{{host}}:{{port}}/restconf/data/ietf-interfaces:interfaces-state/` endpoint. The response received back from the API should look similar to the one in [Figure 12-8](#).

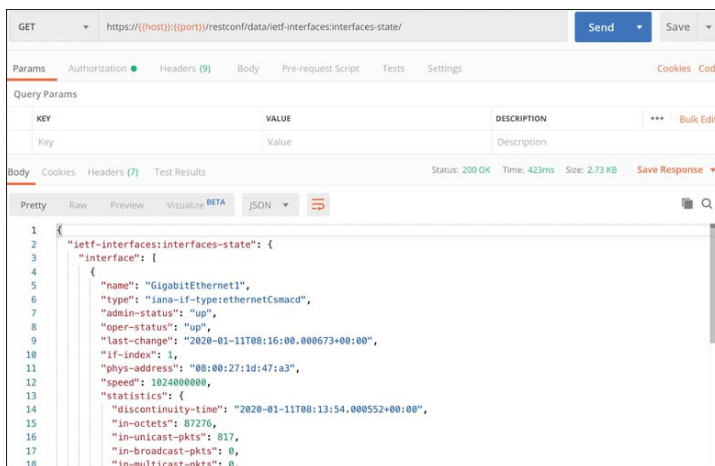


Figure 12-8 *Getting Interface Statistics with RESTCONF*

MODEL-DRIVEN TELEMETRY

Timely collection of network statistics is critical to ensuring that a network performs as expected and foreseeing and preventing any problems that could arise. Technologies such as SNMP, syslog, and the CLI have historically been used to gather this state information from the network. Using a pull model to gather the data, in which the request for network data originates from the client, does not scale and restricts automation efforts. With such a model, the network device sends data only when the client manually requests it. A push model continuously streams data from the network device to the client. Telemetry enables the push model, providing near instantaneous access to operational data. Clients can subscribe to specific data they need by using standard-based YANG data models delivered over NETCONF.



There are two types of telemetry subscriptions:

- **Dynamic:** The subscriber sends a request, usually via the **ietf-yangpush.yang** data model. The device approves the request, replies with a subscription ID, and begins streaming telemetry data. Dynamic subscriptions cannot be changed but can be terminated at any time, usually by closing the NETCONF session.
- **Configured:** The subscription is configured via the CLI, NETCONF, or RESTCONF and is persistent between reboots. Configured subscriptions can be modified and terminated at any point and can be used to stream data to more than one receiver.

A subscription can specify how notifications are sent to the receivers:

- **Periodic notifications:** These notifications are sent with a fixed rate defined in the telemetry subscription. This data is ideal for device counters or measures such as CPU utilization or interface statistics because of their dynamic, always-changing nature.
- **On-change notifications:** These notifications are sent only when the data changes. These types of notifications might be sent, for example, for faults, new neighbors being detected, and thresholds being crossed.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 12-5](#) lists these key topics and the page number on which each is found.



Table 12-5 Key Topics

Key Topic Element	Description	Page Number
Paragraph	Yang data models: NETCONF, RESTCONF, and gRPC	3
		4
		3
Paragraph	NETCONF protocol specification	3
		4
		4
Figure 12-2	NETCONF	3
		4
		5
Paragraph	NETCONF operations to manage device configurations and retrieve status information	3
		4
		6
Paragraph	NETCONF configuration data stores	3
		4
		6

Paragraph	YANG and NETCONF	3 4 7
Paragraph	YANG nodes for data modeling	3 4 9
Figure 12-3	YANG module components	3 5 2
Paragraph	YANG modules	3 5 4
Paragraph	RESTCONF	3 6 7
Paragraph	RESTCONF URIs	3 6 8
Paragraph	Telemetry subscription types	3 7 1

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Network Configuration Protocol (NETCONF)

Yet Another Next Generation (YANG)

remote procedure call (RPC)

gRPC

RESTCONF

Chapter 13

Deploying Applications

This chapter covers the following topics:

- **Application Deployment Models:** This section discusses the public, private, hybrid, and edge cloud application deployment models.
- **Application Deployment Methods:** This section discusses bare-metal virtual machines, containers, and serverless deployments.
- **DevOps:** This section provides an overview of the DevOps operational model.
- **Docker:** This section discusses Docker containers and the basic operation of Docker.

In the past, deployment options for getting an application up and running were fairly simple: You bought a server, fed it hundreds of CD-ROMs, and 12 hours later, you had your application ready to start configuring and inputting data. Thanks to the evolution of application architectures and microservice design patterns, new technologies support the new breed of applications. You now have numerous ways to deploy applications, including automated and flexible ways to get your applications up and providing value to your business. This chapter explores some of the core technologies and operating models that are in use today.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics,

read the entire chapter. [Table 13-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 13-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Application Deployment Models	1, 2
Application Deployment Methods	3, 4
DevOps	5, 6
Docker	7, 8

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which characteristic matches an SaaS deployment model?
 1. Provider deploys your software customizations.
 2. Any update to the software requires a new license.
 3. You can recommend tweaks to the underlying infrastructure.
 4. None of the above

2. Which is a good deployment model for real-time IoT sensors?

1. SaaS model
2. Edge computing model
3. Private cloud model
4. Hybrid cloud model

3. In which of the following ways are containers different from virtual machines? (Choose two.)

1. Containers have less storage requirements than VMs.
2. VMs can run any operating system, but containers run only on Linux.
3. Containers start in 500 ms, and VMs start in minutes.
4. VMs are better if you have a microservice architecture.

4. Which deployment method is best for processes that are run periodically?

1. Serverless
2. Containers
3. Virtual machines
4. All of the above

5. What is the second way of DevOps?

1. Automation
2. Continuous learning and experimentation
3. Culture
4. Feedback loop

6. What is continuous integration?

1. Automated software delivery and deployment
2. An Agile software development technique
3. The process of merging development work with the code base for automated testing
4. None of the above

7. A Docker images uses what type of file system?

1. Layered file system
2. NFS
3. XFS
4. Union file system

8. What command do you use to launch an nginx container on port 80 of the host file system?

1. `docker image build -p 80 nginx`
2. `docker start -it -d nginx -p 80|80`
3. `docker container run -p 80:80 -d nginx`
4. None of the above

FOUNDATION TOPICS

APPLICATION DEPLOYMENT MODELS

The concept of cloud has wormed its way into almost every facet of modern life. It has become central to how we interact with our friends, buy things, watch TV and movies, and run our businesses. So what is it really? Well, that is a very subjective question that has to take into account your own personal perspective. If you asked five people what cloud is, you would probably get eight different answers. Nearly half of all business meetings about cloud, when it was in its infancy, were spent on determining what it was and how it could be used. Is it a service? Is it a product? Is it something magical that will solve all business problems?

The biggest issue with this line of thinking is trying to call cloud a “thing” in the first place. The key value of cloud is that it gives you a new way of operating IT—allowing it to be fast, agile, and able to align better to the goals of the business. From this perspective, cloud can help transform manual and inefficient processes into something the business can use as a competitive advantage and a way to better serve customers. The strength of cloud is realized in the speed and agility it affords as an operational model to get applications deployed and providing value more quickly. Cloud has fundamentally changed how we think about building and operationalizing applications.

Today Amazon, Google, Microsoft, Cisco, and others offer cloud services. Many startups are building their applications and services completely in the cloud. Uber, for example, has completely transformed the transportation industry through a cloud-based application that allows us to easily find someone willing to pick us up and take us where we want to go. Square has transformed credit card processing by allowing

anyone with a smartphone and an inexpensive hardware add-on to take credit card payments. The growth of cloud services like these will continue at a steady pace, forcing businesses to rethink how they engage with their customers and operate business-critical services.

NIST DEFINITION

To define cloud, we need a common taxonomy, and for that we can turn to the National Institute of Standards and Technology, a U.S. government agency responsible for standardizing weights, measurements, and technology standards and practices. NIST Special Publication (SP) 800-145 was written to solve the “what is cloud” dilemma. While SP 800-145 is a rather short document, it hits on the major aspects of cloud and is therefore useful as a framework for understanding application deployment options and prepping for the 200-901 DevNet Associate DEVASC exam. In SP 800-145, NIST defines cloud through three primary lenses: essential characteristics, service models, and deployment models (see [Figure 13-1](#)).

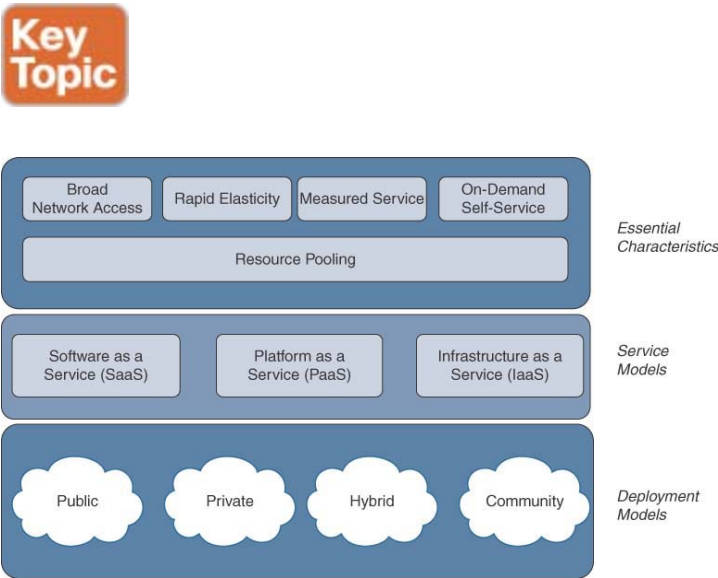


Figure 13-1 NIST Cloud Computing Definition

Essential Characteristics

According to SP 800-145, the essential characteristics describe the core requirements of any cloud and are the differentiators that determine whether a service can be classified as being a cloud offering. SP 800-145 defines the essential characteristics as follows:

- **Broad network access:** Services are available over the network and accessed via standard protocols and communications technologies on any type of client device (mobile phone, tablet, desktop, and so on).
- **Rapid elasticity:** Capacity can be automatically provisioned and decommissioned to scale the service to demand.
- **Measured service:** Cloud systems measure resource utilization (compute, storage, and network) and charge for those services accordingly. Utilization is monitored, controlled, and reported on, allowing transparency for the service provider and customer.
- **On-demand self-service:** The cloud consumer can provision compute, storage, and network as needed, without human interaction, through automation or self-service portals.
- **Resource pooling:** The infrastructure is a common pool of resources that can serve multiple customers at the same time. The customer does not interact with the underlying hardware, and workloads can be moved within the cloud environment based on demand without the end user knowing or needing to be involved.

Service Models

The cloud service models mainly differ in terms of how much control/administration the cloud customer has to perform. [Figure 13-2](#) shows the following service models:

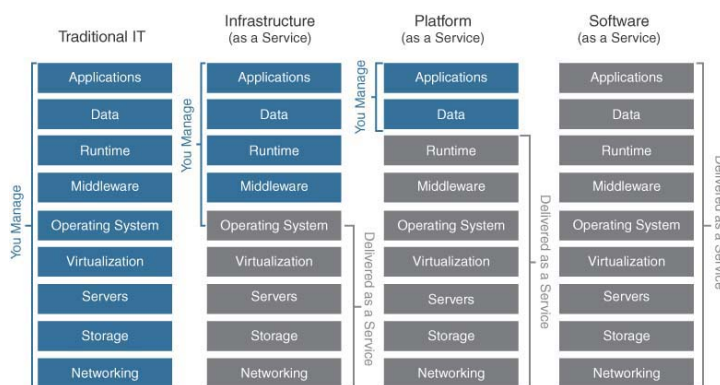


Figure 13-2 *Cloud Service Models*

- **Software as a service (SaaS):** A service provider hosts, manages, and controls an application and offers it to the customer to use. The customer does not interact at all with the underlying infrastructure,

operating systems, storage, or network. There may be some customization capabilities, but they are usually limited to application-specific configuration. What you see is what you get.

- **Platform as a service (PaaS):** A PaaS provider supplies a fully integrated service/software suite, and customers can deploy their applications on top of this suite with a predefined set of application programming interfaces, libraries, software development kits, and/or other tools and services. Customers can program the application in any way they choose and can customize it directly to their own workflow, as long as the customization is within the parameters of the service provider's offering. Customers do not have to worry about integration of the underlying infrastructure components, and in the case of a managed PaaS service, have no management responsibilities for the underlying infrastructure.
- **Infrastructure as a service (IaaS):** A provider enables the customer to provision compute, storage, and networking to run any combination of software and operating systems. While customers have no control of the underlying infrastructure hardware platform, they have full control over the software and services they deploy within the cloud service. The customer is also responsible for the maintenance of the software they deploy, including patching and upgrading.

APPLICATION DEPLOYMENT OPTIONS

Cloud can be deployed in a number of ways. The deployment model chosen depends on whether you want to own your own infrastructure, rent your infrastructure, or have a mixture of both. There are a multitude of reasons you might choose one option over the others. Factors such as protecting sensitive data, economics, and speed all need to be weighed in making choices between deployment models.



Private Cloud

A private cloud is provisioned for a single organization that may have multiple service consumers within its business units or groups (see [Figure 13-3](#)). The organization may own the private cloud or lease it from another entity. It also does not have to reside on the organization's premises and may be in another facility

owned and operated by a third party. The following are the key characteristics of a private cloud:

- A private cloud has dedicated resources for a single organization.
- Connectivity can occur through the Internet, fiber, or a private network.
- A private cloud may be on premises or off premises.
- Applications are deployed within a private cloud, and the organization has complete control and responsibility for their maintenance and upkeep.

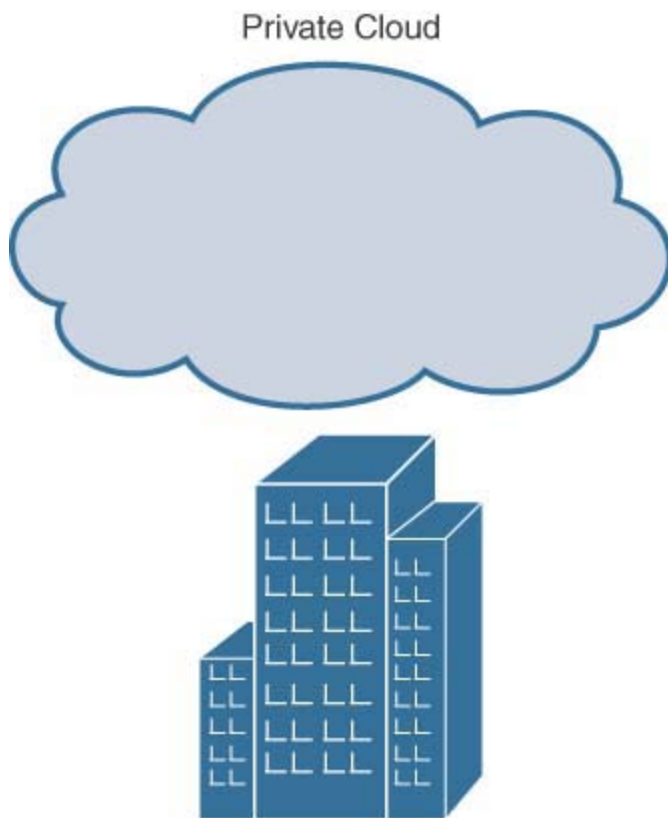


Figure 13-3 *Private Cloud*

Public Cloud

A public cloud is provisioned for open utilization by the public at large (see [Figure 13-4](#)). Anyone with a credit card can gain access to a public cloud offering. A public cloud exists solely on the premises of the cloud provider. The key characteristics of a public cloud are as follows:

- Resources are publicly shared.

- A public cloud supports multiple customers.
- Connectivity occurs through the Internet or virtual private connections.
- Use of a public cloud is billed based on usage.
- Applications are deployed on the infrastructure hardware and services offered by the cloud provider. The user is responsible for security and application operations but not responsible for hardware maintenance.

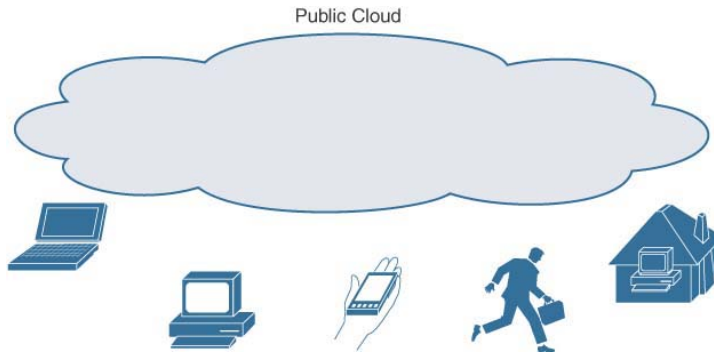


Figure 13-4 *Public Cloud*

Hybrid Cloud

A hybrid cloud is composed of one or more cloud deployment models (private and public, for example) and is used to extend capabilities or reach and/or to augment capacity during peak demand periods (see [Figure 13-5](#)). The key characteristics of a hybrid cloud are as follows:

- A hybrid cloud is a combination of public and private models.
- A hybrid cloud has on-premises and off-premise resources.
- Orchestration between the parts of a hybrid cloud is the responsibility of the application owner.
- Applications are deployed in a distributed fashion, with some services and components residing on premises and others (such as client access) in the cloud provider's environment.

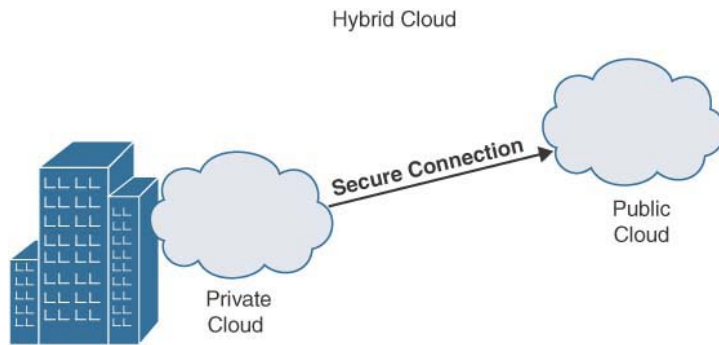


Figure 13-5 *Hybrid Cloud*

Community Cloud

A community cloud, as shown in [Figure 13-6](#), is unique in that it is provisioned for the sole utilization of a specific community of customers, such as a school district or multiple government agencies. Basically any group of entities that have a common policy, security, compliance, or mission can join together and implement a community cloud. The key characteristics are as follows:

- A community cloud is a collaborative cloud effort that shares infrastructure between several similar organizations with common needs.
- A community cloud can be owned, managed, and operated by one or more members of the community or a third party.
- A community cloud may reside on premises or off premises.
- Applications are deployed in a shared fashion and operated by the community for the use of all community members. This can be similar to a hybrid model but with a focus on resource sharing.

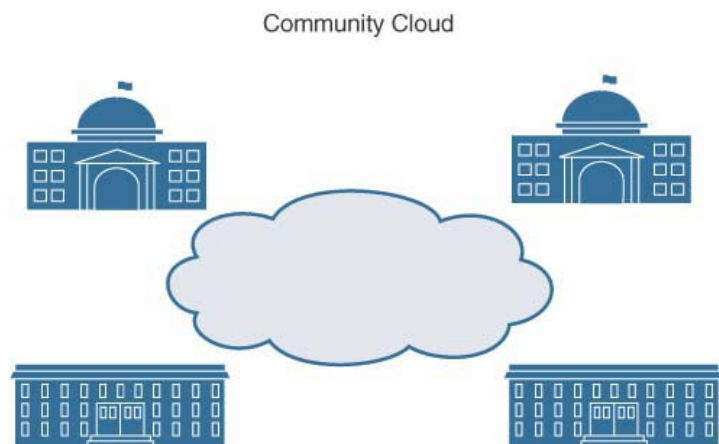


Figure 13-6 *Community Cloud*

Edge and Fog Computing

With the increase in IoT applications, devices, and sensors, the various deployment models covered so far are just not up to the task of handling the sheer volume of data being created and needing to be processed. Self-driving cars, robotics, computer video processing, and many other use cases can produce gigabytes of data in real time. This sea of data cannot be consumed and processed by a centralized cloud application, given the cost of transporting the data and the latency involved in receiving information back to do anything with it. Your self-driving car would run into a ditch before a cloud service could detect and respond.



These constraints required engineers to think differently about use cases requiring intelligence at the edge. A new class of application deployment was created, called fog computing (because fog is basically a cloud on the ground). The idea behind fog is to leverage the scale and capabilities of cloud models to handle the control and deployment of applications at the edge. With edge computing, you let local processing of data occur closer to the actual sensors and devices where it can be acted on more quickly. Think of fog as a framework and structure that enables edge computing. [Figure 13-7](#) shows how edge and fog work together.

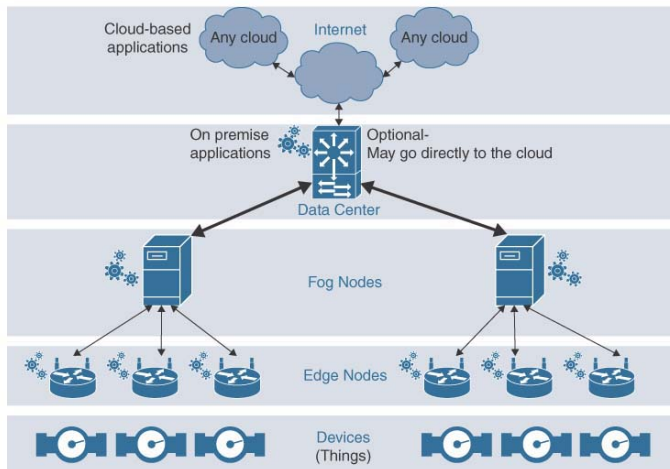


Figure 13-7 *Edge and Fog Computing*

APPLICATION DEPLOYMENT METHODS

What is IT's value to the business if you boil it down to the simplest aspect? IT is charged with supporting and maintaining applications that the business relies on. No one builds a network first and then looks for applications to stick on it. Instead, a network is built to connect applications to employees and customers. The importance of the application and its evolution have driven many of the advances we have seen in cloud.



BARE-METAL APPLICATION DEPLOYMENT

The traditional application stack, or bare-metal, deployment is fairly well known. It's how the vast majority of applications have been deployed over the past 40 years. As [Figure 13-8](#) shows, one server is devoted to the task of running a single application. This one-to-one relationship means the application has access to all of the resources the server has available to it. If you

need more memory or CPU, however, you have to physically add new memory or a new processor, or you can transfer the application to a different server.

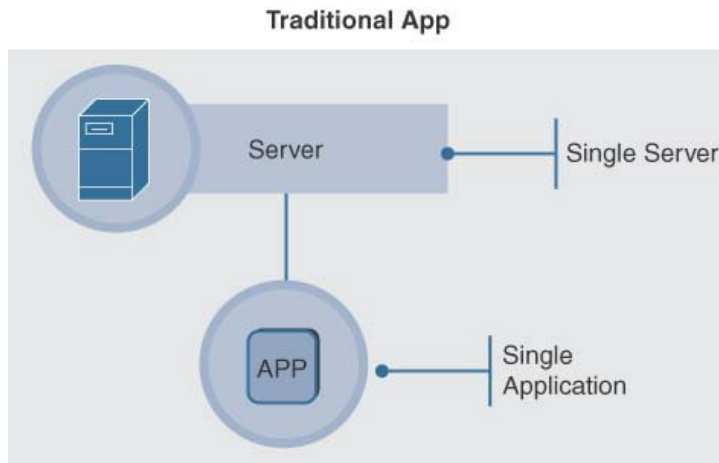


Figure 13-8 *Bare-Metal Application Stack*

While bare-metal application deployment is an effective way to isolate applications and get consistent performance characteristics, it's very inefficient. In fact, many of these traditional server deployment models result in enormous amounts of waste in regard to power and cooling of servers when they are not under load. It's not uncommon to find a server during non-peak hours using less than 10% of its capacity, which is not very cost-effective.

Some applications, however, really need a dedicated server. Big data applications benefit greatly from a dedicated server environment; most Hadoop clusters consist of numerous one-rack-unit servers all running in parallel. These workloads consume 100% of the resources of the server, making them poor candidates for virtualization.

VIRTUALIZED APPLICATIONS

Virtualization was created to address the problems of traditional bare-metal server deployments where the

server capacity was poorly utilized. The idea with virtualization is to build one large server and run more than one application on it. Sounds simple, right? Many of the techniques mentioned earlier, from the days of the mainframe, were leveraged to create an environment where the underlying server hardware could be virtualized. The hypervisor was created to handle all of the time slicing and hardware simulation. This made it possible to run various applications and operating systems at the same time and reap the benefit of better utilization of resources. [Figure 13-9](#) shows this concept.

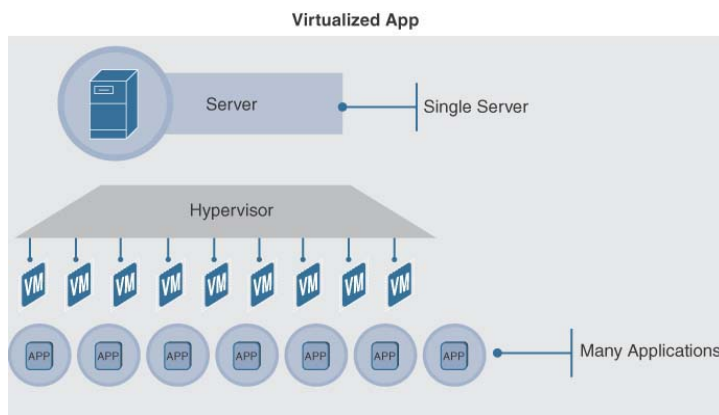


Figure 13-9 *Virtualized Application Stack*

Virtualizing hardware had other benefits, too. The ability to make applications portable and not tied to a single server allows for mobility in the data center. If a server needs to be worked on, you simply move the running virtual machine to another server and never need to take down the application. You can also create common deployment packages for new virtual machines and applications; this gives administrators the ability to quickly provision a new server in a consistent and secure manner.

The virtualized application is a key aspect of cloud, but *virtualization* and *cloud* are not synonymous. A cloud environment could easily be configured to deploy a physical server as well as a virtual machine. Conversely,

just because you have a virtualized infrastructure doesn't mean you have a cloud; the big difference is in the operational model.

CLOUD-NATIVE APPLICATIONS

Cloud-native, or microservice, applications represent the further intersection and evolution of virtualization and a cloud environment. As virtualization became popular, application developers started to ask themselves why they needed a heavy operating system to run their applications. All the patching, drivers, and configuration requirements don't go away just because you load a virtual machine. All that work still needs to be done. What if applications could be decoupled and written as simpler packages that do not need a hypervisor or full-blown operating system in order to function? That's exactly what happened. As [Figure 13-10](#) shows, cloud-native applications run on multiple servers in parallel with each other.

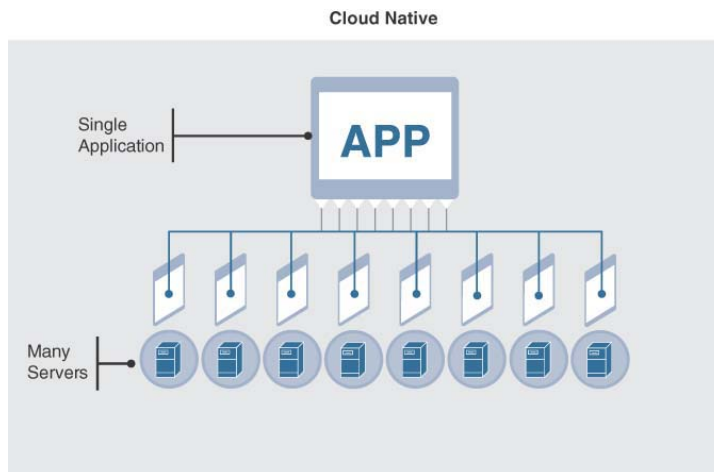


Figure 13-10 *Cloud-Native Applications*

If a server goes bad in a cloud-native application environment, no one cares because the application is distributed across many servers in the data center, and the application was written from the beginning to handle failures and redirect work to working nodes. If you need

more capacity, just plug in a new server, and it is added to the server pool automatically. The infrastructure deploys the application onto the new server, and capacity has instantly increased. If load decreases, the application can shut down unused nodes automatically to save money. Large-scale cloud applications such as Facebook and Twitter operate in this fashion, which is why they appear to have infinite capacity from a user perspective.

CONTAINERIZED APPLICATIONS

The evolution of applications to leverage microservices architectures was a very important change that provided developers with new ways to build massively scaled applications with tremendous improvements in terms of availability and fault tolerance. Microservices made available small custom-built components, but they didn't address the need for an efficient infrastructure that could support them and their dynamic nature. Containers became a very popular way to provide an easier way to get application components deployed in a consistent way. Imagine being able to have all of the components and dependencies of your application loaded in an interchangeable format that you can simply hand off to operations to deploy. This was what containers offer. Google created Kubernetes and made it open source to provide an orchestration and automation framework that can be used to operationalize this new application deployment model.

The benefits of containers are as follows:

- Consistency for deployment automation
- Simplified lightweight image files measured in megabytes (whereas VM files are measured in gigabytes)
- Providing only what the app needs and nothing else
- Ability to work the same in production as on a developer's laptop
- Open community-built best-of-breed containers for faster innovation
- Ability to deploy applications in seconds

Deploying applications in a container decouples parts of the application. Consider a traditional application stack like LAMP (Linux, Apache, MySQL, Perl/Python/PHP). This was the foundation for many websites over the years. To deploy a LAMP stack in a virtual environment, you load an operating system and application on top of a hypervisor and then repeat this process for however many virtual machines you need to support the number of users you expect. Include a load balancer in front of these virtual machines, and you can add capacity as needed. The problem with this design is that the virtual machines run full operating systems, replicated a number of times. You also have to maintain the virtual machines the same as you would as if they were physical servers. Capacity is handled manually, and you can easily run out of resources if your website receives more visitors than you planned for. [Figure 13-11](#) shows a LAMP deployment on virtual machines.

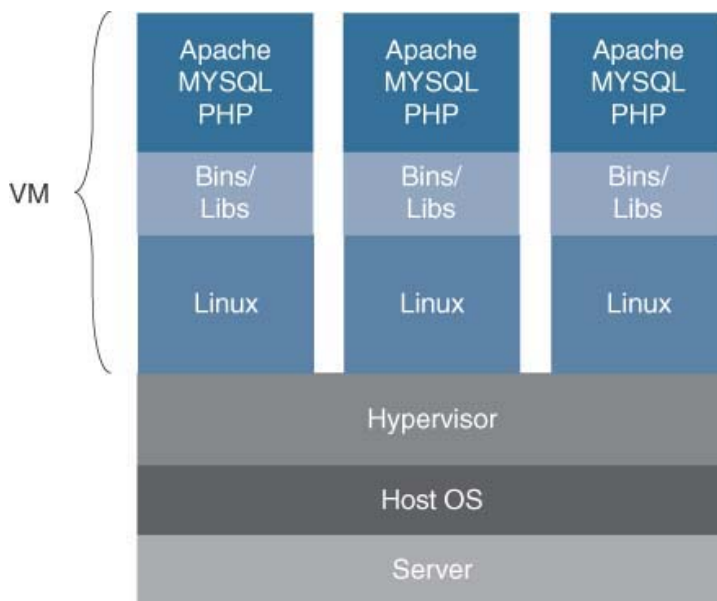


Figure 13-11 *LAMP Stack on a Virtual Machine Infrastructure*

Now consider what happens if you deploy that same application stack in a container infrastructure. You decouple the components of the stack and deploy them

separately. So instead of everything being loaded on a single virtual machine, you break off the Apache web server and PHP code from the MySQL database. By themselves, they are much smaller in size and resource utilization and can be scaled independently of each other. Instead of having a gigabyte or more being taken up for an operating system and hardware virtualization of memory and other parts of virtualization, the container simply shares the kernel of the underlying operating system of the server it is running on. The container is treated like an application and is run in an isolated space within the operating system. Any binaries and libraries needed to run the container are prepackaged, which means you don't have to touch the container host server. [Figure 13-12](#) shows a containerized LAMP stack.

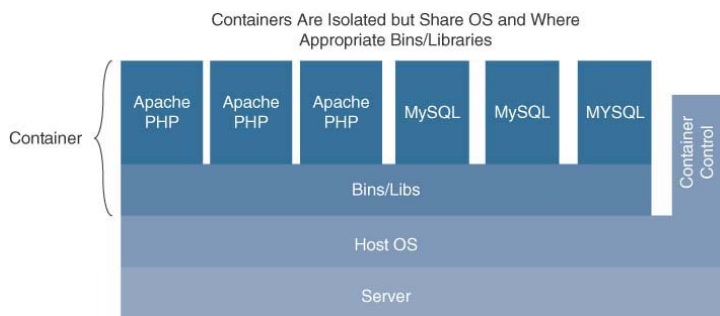


Figure 13-12 *Containerized LAMP Stack*

SERVERLESS

Serverless is one of the silliest names in the industry. Of course, there is a server involved! However, this type of application deployment mechanism is intended to make it even easier to get an application up and running, and the term *serverless* conveys this idea. You simply copy and paste your code into a serverless instance, and the infrastructure runs your code without any further configuration. This type of deployment is also referred to as function as a service, and it works best for applications that run periodically or that are part of batch processes.

When writing code, you create a function that you call repeatedly at different places in the code to make it more efficient. Serverless deployment uses the same write once/use many times concept: You write some code and then call it remotely through your application.

Serverless applications typically execute some type of application logic but do not store data. Your calling application is expected to handle that part of it. For example, say you want to create a function that converts images you receive to a universal size and shape. You can then load this function into a serverless offering from AWS, and any time you receive a new picture, it is sent to the function and returned in the correct format (see [Figure 13-13](#)).

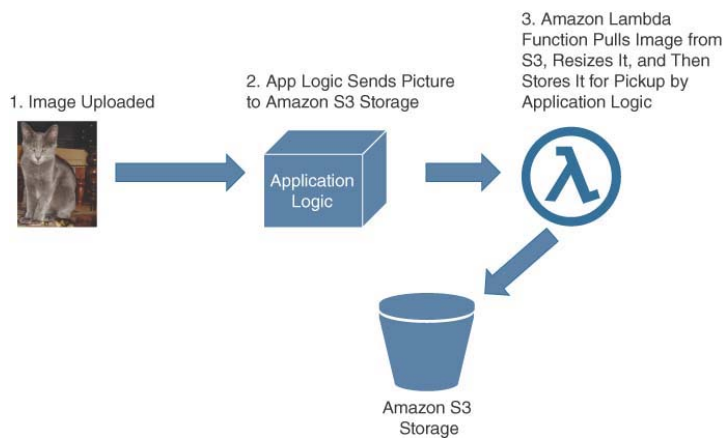


Figure 13-13 *Serverless Image Resizing Example*

The whole idea behind serverless and why a customer may want to use it is that it doesn't require dedicated hardware or resources to be set aside for these periodic processes. The customer pays for the resources used, and then the services are shut down until needed again; this is a better cost model. The following are some of the advantages of serverless deployment:

- **Cost:** Serverless can be significantly cheaper than prepaying for infrastructure that is underutilized. The pay-as-you-go computing model means you are not charged for time your service is not being used.

- **Scalability:** Developers don't need to build in their own autoscaling policies or technologies. The provider is responsible for adding more capacity on demand.
- **Easier to use and write code for:** Small teams of developers can use serverless deployment without needing to involve infrastructure teams or acquire advanced skills.

The following are some of the disadvantages of serverless deployment:

- **Latency:** Spin-up time from idle for the function can cause significant delays that must be accounted for in application design.
- **Resource constraints:** Some workloads need more resources than the service may typically be allocated for, causing heavy charges, which may make it cheaper to dedicate infrastructure.
- **Monitoring and debugging:** There is limited visibility into the underlying infrastructure, which makes it difficult to troubleshoot performance issues with the application. You are often limited to the tools provided by the service provider, which are likely to be proprietary.
- **Security and privacy:** Most providers are built on proprietary offerings and use shared resources. Misconfiguration can result in compromises and data loss, just as with any other cloud service. There are on-premises options for serverless that can give businesses more control over their data and security posture.
- **Vendor lock-in:** This is a big one. Each provider has its own tools and frameworks, which makes it very difficult to switch providers if costs go up or if services are not working as planned. Migration between providers is not trivial.

DEVOPS

Agile has dramatically changed the software development landscape by introducing a more efficient and faster way of delivering software and value to the business. Much of the improvement in the development process has been focused on delivery speed. You can have as much Agile development as you want, but if you can't get the software deployed in a timely manner, you can't take advantage of its capabilities. Software was traditionally built and tested by developers, and if it worked on their laptops or test systems, it was pitched over the fence to IT operations for deployment. When things did not go as smoothly as planned, a significant amount of finger-pointing would ensue. Developers

Figure 13-14 Dev and Ops Calendars

Traditional IT service delivery is slow, manual, and often prone to errors (see Figure 13-15). The infrastructure is a shared resource, and one change can have a ripple effect that could break other unrelated systems. The fragile nature of the infrastructure makes it very hard to anticipate issues that can arise. To combat this, many IT organizations create layers upon layers of approval processes. While this sounds like a rational way to protect the business from downtime, the net effect is to slow new deployments to a crawl. For these reasons and many others, DevOps was created.

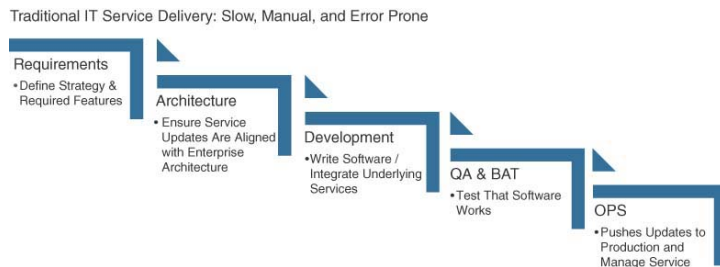


Figure 13-15 *Traditional Sequential Approach to Operations*

WHAT IS DEVOPS?

In 2009 two employees from Flickr (an image sharing site), John Allspaw and Paul Hammond, presented a talk titled “10+ Deploys per Day: Dev and Ops Cooperation at Flickr” to a bunch of developers at the O’Reilly Velocity conference. In this talk, Allspaw and Hammond said that the only way to build, test, and deploy software is for development and operations to be integrated together. The sheer audacity of being able to deploy new software so quickly was the carrot that fueled the launch of the DevOps concept. Over the years since then, DevOps has moved from being revered by a small group of zealots and counterculture types to being a very real and quantifiable way to operate the machinery of software creation and release. The vast majority of companies that

do software development are looking for ways to capture the efficiencies of this model to gain a competitive edge and be better able to adapt to change from customer and industry perspectives.

In a nutshell, DevOps is a culture of sharing in which developers and operations folks are one unit that can rise and fall together. It is a practical application of Lean and Agile. The goal of DevOps is to be a real-time business enabler by removing wasted effort and bureaucracy from getting in the way of better addressing the needs of the customer. DevOps has five guiding principles:

- **Culture:** For DevOps to work, organizational culture must change. This is by far one of the most difficult aspects to embrace, but it is the single most important factor for success. DevOps requires a culture of sharing.
- **Automation:** While DevOps is more than just a set of software tools, automation is the most easily identifiable benefit. Automation techniques greatly speed up the deployment process, enable defects to be caught and corrected earlier, and eliminate the need for human intervention in repetitive tasks.
- **Lean:** Reducing wasted efforts and streamlining the process are the goals of Lean. It's a management philosophy of continuous improvement and learning.
- **Measurement:** Unless you measure your results, you can never improve. Success with DevOps requires the measurement of performance, process, and people metrics as often as is feasible.
- **Sharing:** DevOps requires a culture of feedback and sharing. Breaking down silos and creating an inclusive shared fate environment is the ultimate goal.

Figure 13-16 shows the core components of DevOps and how they are interrelated.

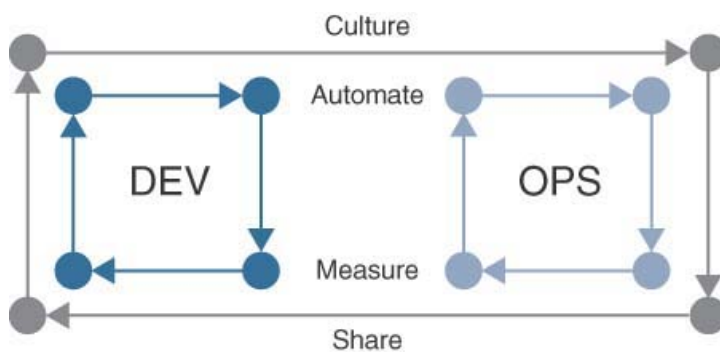


Figure 13-16 *DevOps*

PUTTING DEVOPS INTO PRACTICE: THE THREE WAYS

When discussing DevOps, it's hard not to mention two very important books on the subject. The first of these books, *The Phoenix Project*, by Gene Kim, Kevin Behr, and George Spafford, set the stage for understanding the benefits of DevOps practices through a story that builds on the experiences of the authors and contributors and shows how a fictional company moves from being almost out of business to a rising star outpacing its competition. The interesting thing about the book is that you can find many examples of your everyday life and practices outlined in it. While it probably won't be made into a blockbuster movie anytime soon, it is a quick read that easily shows the benefits of DevOps practices. The second important DevOps book is *DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*, written by Gene Kim, Jez Humble, Patrick Debois, and John Willis. This book is a more practical view of the subject and provides insight into the "Three Ways" of DevOps (discussed in the following sections), offering a framework for implementation in your own organization. If you want to know more about DevOps, you should certainly pick up these two books.



First Way: Systems and Flow

A highway system at full capacity is basically a parking lot, where no one is moving. We have all had to spend time staring at brake lights in traffic. For a technologist, an equivalent work scenario featuring overtime and missed family dinners occurs all too commonly. Many

problems occur because of the ripple effect of not knowing how everything flows and not being able to visualize all work moving through the system. Having a bird's-eye view of how work flows from development to production is essential to figuring out optimal ways to get to the end goal of a working application that provides value. Many teams use kanban boards to visualize work, whether through an application like Jira or by simply putting sticky notes with the workflow on a whiteboard so that everyone can see what is being done and what needs to be done. Using a kanban board can help a team be more rational in what it expects to accomplish in a given period of time. [Figure 13-17](#) shows the first way of DevOps, which is focused on systems and flow.



Figure 13-17 *First Way: Systems and Flow*

To accomplish work goals, you have to reduce work units, or batches of work, so that they are more streamlined to prevent workload traffic jams. In an Agile sprint, it is common to work on a small set of features during a two-week period to reduce how much is being worked in a given interval of time. This allows developers time to build the capability without being overwhelmed or pulled in multiple directions. This also applies to the operations side, as both sides are integral to success. Developers can't just rain app updates on their operations teams without having a solid view of how each update gets deployed and the operations teams have to build a system that is able to handle the speed and agility developers need. By having both developers and operations work together to understand where constraints are, you can optimize the system and move faster.

Quality is not a bolt-on at the end of the development and deployment process. It has to be integrated into the whole system. Waiting for a problem to be found in Q&A will create compounded issues that can make things much worse. The earlier you address a challenging situation, the more your overall work effort will be reduced. This equates to saving money and time. The whole point is to reduce wasted time on rework.

You have to continually optimize the system to make it more efficient. Nothing is ever perfect with DevOps; you are in a constant state of removing inefficiencies. You must be able to adapt to business needs and your customers' ever-changing requirements.

The following are the key characteristics of the first way:

- Make work visible
- Reduce batch sizes
- Reduce intervals of work
- Build in quality by preventing defects from being passed downstream
- Constantly optimize for business goals

Second Way: Feedback Loop

One of the things you will see repeatedly in DevOps is analogies to manufacturing. Since so much of this management philosophy is derived from Lean and the Toyota Production System, concepts like defect prevention are front and center in the approach. The idea of a feedback loop is to provide guidance and direction on what is working and what is not working. If something isn't working, you have to make sure that it doesn't happen again so you don't end up on a hamster wheel of pain. DevOps requires that you take feedback as the gift that it is and make corrections. [Figure 13-18](#) shows the second way of DevOps, which is focused on the feedback loop.

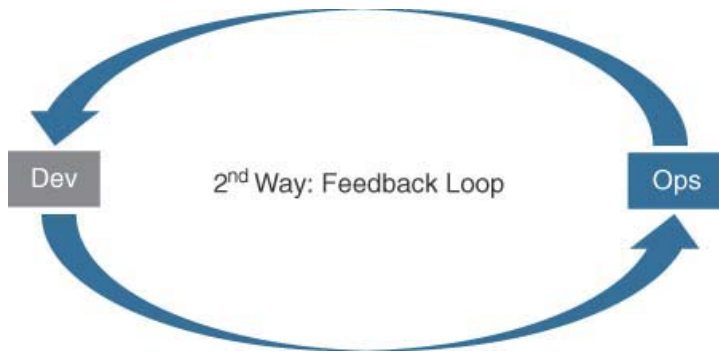


Figure 13-18 *Second Way: Feedback Loop*

The more you sample feedback, the faster you can detect issues and recover from them. In the Toyota Production System, the manufacturing line has something called an Andon Cord, which can be pulled by anyone at any time to halt the production line. This form of feedback is immediate and highly visible. It also kicks off the process of swarming a problem until it is fixed, where everyone in the area focuses on solving the issue before the line is restarted. In DevOps, this type of empowerment means that when problems with software are detected, the whole team pitches in to fix the underlying cause.

Once a problem has been overcome, it is not just time to have a party; it is time to document and improve the processes that led up to the issue at hand: Learn from your mistakes so you can move on.

The following are the key characteristics of the second way:

- Amplify feedback to prevent problems from happening again
- Enable faster detection and recovery
- See problems as they occur and swarm them until they are fixed
- Maximize opportunities to learn and improve

Third Way: Continuous Experimentation and Learning

How do you foster innovation? Create a culture of trust where your people are allowed to try new things and fail.

That doesn't mean that you are just rolling the dice on crackpot ideas; rather, it means having a dynamic and disciplined approach to experimentation and risk taking. It's very rare that success happens on the first try. When problems occur, avoid pointing fingers and blaming people; instead, figure out what went wrong and what could be improved to make it better. [Figure 13-19](#) shows the third way of DevOps, which focuses on continuous experimentation and learning.

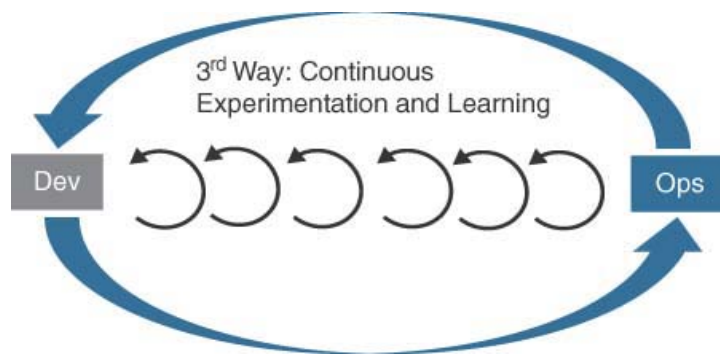


Figure 13-19 *Third Way: Continuous Experimentation and Learning*

DevOps talks a lot about continuous improvement and fixing issues to make the system perform better. It's a noble goal, but when do you find time? The simple answer is that you need to make time. In other words, you schedule time to get people focused on improving the system and try new methods and technologies.

Finally, you must build a culture of sharing and learning. A simple method to accomplish this is to create shared code repositories and get those treasure troves of information off individual laptops and available to the rest of the organization.

The following are the key characteristics of the third way:

- Conduct dynamic, disciplined experimentation and risk taking
- Define time to fix issues and make the system better
- When things go wrong, don't point fingers

- Create shared code repositories

DEVOPS IMPLEMENTATION

Implementing DevOps technical practices within an organization typically involves three stages. These stages follow a natural progression that leads to a fully automated code-to-deployment operational model:



- **Continuous integration:** This stage involves merging development work with the code base constantly so that automated testing can catch problems early.
- **Continuous delivery:** This stage involves a software package delivery mechanism for releasing code to staging for review and inspection.
- **Continuous deployment:** This stage relies on continuous integration and continuous delivery to automatically release code into production as soon as it is ready.

A vast number of technical tools can be used to automate a DevOps environment, from commercial applications to bleeding-edge open-source projects. You need to be aware of this DevOps tool chain if you are asked to administer a DevOps environment. Xebialabs is a company that specializes in DevOps and continuous delivery, and it is a great resource for understanding the various tools in a DevOps pipeline. The company's DevOps periodic chart provides an interactive view of these tools and provides links and descriptions of the tools and technologies you might encounter (see [Figure 13-20](#)). You can go to <https://xebialabs.com/periodic-table-of-devops-tools/> to get access and download your own copy.

The Periodic Table of DevOps Tools (V4.1)

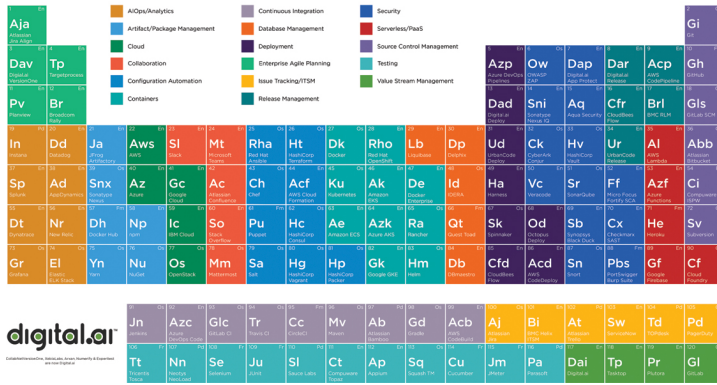


Figure 13-20 Xebialabs Periodic Table of DevOps Tools (Source: <https://xebialabs.com/periodic-table-of-devops-tools/>)

If you are not interested in the do-it-yourself nature of connecting and configuring open-source tools, you might be better off looking to a vendor that offers a platform as a service or one that will build your DevOps pipeline for you and also offer support with maintenance and upgrades. Regardless of the DevOps platforms or tools you use or whether you build your own or buy them, there are some commonalities for all DevOps pipelines. It's easiest to understand the workings of a DevOps pipeline by seeing it in action. [Figure 13-21](#) is a graphical representation of what happens at each step of a sample DevOps pipeline.

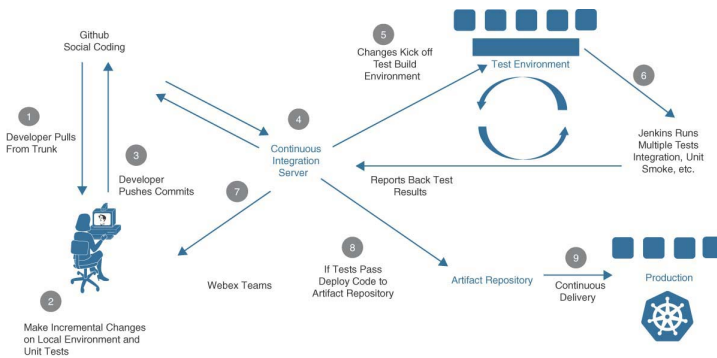


Figure 13-21 DevOps Pipeline in Action

Here is how the pipeline works:

- Step 1.** The developer pulls the latest code from the version control system with Git. This ensures that the developer has the most recent changes and is not working with an old version.
- Step 2.** The developer makes changes to the code, adding new features or fixing bugs. The developer also writes test cases that will be used to automatically test the new code to software functional requirements. The developer eventually stages the changes in Git for submission.
- Step 3.** The developer uses Git to push the code and tests to the version control system (for example, GitHub), synchronizing the local version with the remote code repository stored in the version control system.
- Step 4.** The continuous integration server, such as Jenkins, has a login that monitors GitHub for new code submissions. When Jenkins sees a new commit, it is able to pull the latest version and starts an automated build process using the test cases.
- Step 5.** Jenkins kicks off the automated build of a testing environment for the new software build. It is possible to use Python scripts, Ansible, or other infrastructure automation tools to build the testing environment as close to production as possible.
- Step 6.** Jenkins executes various automated testing, including unit testing, integration testing, smoke testing (stress testing), and security testing. When all the tests are finished, the results are sent back to Jenkins for compilation.
- Step 7.** Jenkins sends the test results to the developer for review. They can be sent via email, but a more modern way of alerting the developer is through a collaboration tool such as Webex

Teams. Jenkins has plug-ins for all major team management tools and allows for easy integration. If the build fails, the developer can use links to the information on what failed and why, make changes to the code, and restart the process.

Step 8. If the build process is successful and all of the tests pass, Jenkins can deploy the new code to an artifact repository (just a fancy name for the place finished software is stored). The software is not able to be deployed in production.

Step 9. Jenkins signals the infrastructure that an updated version of software is ready. For example, there may be a new container ready to be deployed into a Kubernetes platform. The updated container replaces the existing containers with the new code fully tested and ready to go. Now the application is updated with minimal (or no) disruption.

The nuts and bolts of building a DevOps pipeline are beyond of the scope of the 200-901 DevNet Associate DEVASC exam. There are simply more tools than grains of sand at the beach, and luckily you are not going to be tested on all of them. For technologists, it's easy to focus on the tools and technology, but honestly that's the easy part of DevOps. Changing culture and implementing Lean methodologies are where many companies have struggled. Focus on streamlining your processes and implement technologies that help accelerate your efforts.

DOCKER

Some of the best innovations come from reusing older technologies and bringing them forward in a new way. That's exactly what Solomon Hykes and Sebastien Pahl did back in 2010 when they started a small platform as a service (PaaS) company called dotCloud. They were looking for a way to make it easier to create applications

and deploy them in an automated fashion into their service. They started an internal project to explore the use of some interesting UNIX technologies that were initially developed in the 1970s to enable process isolation at the kernel level. What would eventually become Docker started as a project to use these capabilities to grow the PaaS business. In 2013, the world was introduced to Docker, which represented a new paradigm in application packaging and deployment that took off exponentially. While dotCloud eventually went away, Docker has grown into the leading container platform. Luckily for us, it was open source from the very beginning, and the container runtime itself is hosted by the Cloud Native Computing Foundation. While there are other container runtimes, such as like Rocket and Linux Containers, none of them are as popular or as widely deployed as Docker.

UNDERSTANDING DOCKER

Docker containers use two capabilities in the Linux kernel: namespaces, which provide isolation for running processes, and cgroups, which make it possible to place resource limits on what a process can access. These features allow you to run a Linux system within another Linux system but without needing to use virtualization technologies to make it work. From the host operating system's perspective, you are just running another application, but the application thinks it is the only application that is running. Instead of needing to virtualize hardware, you just share the kernel; you don't need to load a full operating system, drivers, and memory management processes each time you want to run an application.

Namespaces

Namespaces are essential for providing isolation for containers. Six namespaces are used for this purpose:

- **mnt (mountpoints):** This namespace is used for mapping access to host operating system storage resources to the container process.
- **pid (processes):** This namespace is used to create a new process ID for an application.
- **net (networks):** This namespace is responsible for network access and mapping communication ports.
- **ipc (System V IPC):** Inter-process communication controls how the application can access shared memory locations between applications within containers.
- **uts (hostname):** This namespace controls host and domain names, allowing unique values per process.
- **user (UIDs):** This namespace is used to map unique user rights to processes.

Figure 13-22 shows a Linux host using namespaces to isolate three different containers.

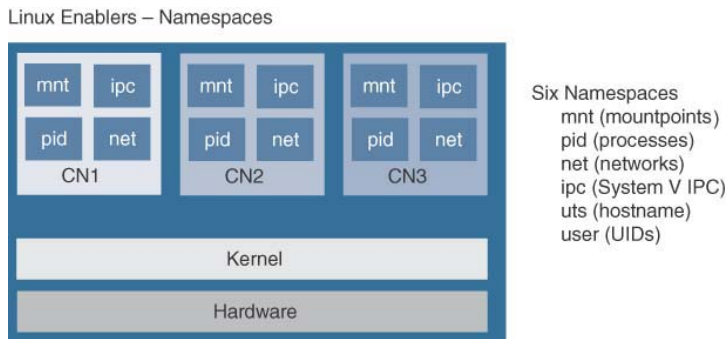


Figure 13-22 *Linux Namespace Isolation for Containers*

Cgroups

Cgroups, or control groups, are used to manage the resource consumption of each container process. You can set how much CPU and RAM are allocated as well as network and storage I/O. Each parameter can be managed to tweak what the container sees and uses. These limits are enforced by cgroups through the native Linux scheduler and function to restrict hardware-level resources. Figure 13-23 shows an example of a cgroup that allocates a maximum of 25% of the various hardware resources of the container host.

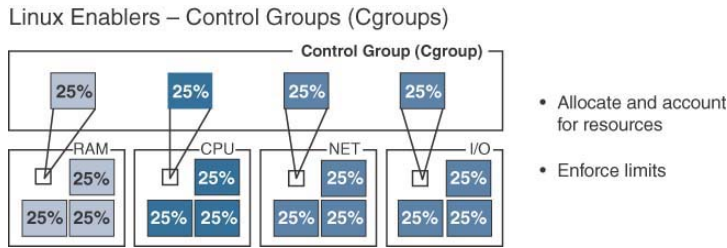


Figure 13-23 *Control Group in Action*

Union File System

The Union File System is a foundational building block for a container. It is a file system service that was developed for Linux to allow different file systems to be layered on top of each other to create a combination, or union, of the various files to create a single merged representation of the contents. If you have ever worked with photo editing software, this will probably make sense to you. Say you want to remove someone or something from a picture (like that ex you never speak of). You take the base picture and then add layers on top of it, and you add or remove pixels until you have a whole new picture. If you want to remove layers you created, you simply peel them back to expose the previous image. Every layer you add on top takes precedence over all the layers below it. In essence, this is how a union file system works, and it is one of the main reasons such a file system is so efficient at storage. Each layer includes only what is new, and nothing is duplicated from a previous layer. The layers are read-only, which means they are locked in place. There is a read/write layer that sits on top that you can use to interact with the file system, but nothing you do will be maintained unless you save your changes. If you want to make your changes permanent, you have to put another layer on top of the union file system. This is called a copy-on-write operation. [Figure 13-24](#) shows an example of a container image and its layers.

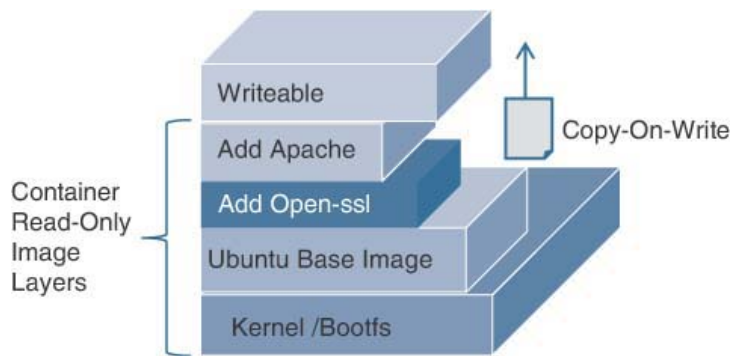


Figure 13-24 *Container Image Layers Using a Union File System*

This read-only aspect of a container is crucial to understand. You may also hear the term *immutable* used to describe the container file system; it simply means unchanging over time. In other words, a container doesn't get patched; rather, it gets re-created with new layers added. In [Figure 13-24](#) you can see a Ubuntu Linux base image layer that has OpenSSL added on top of it via another layer. Above that is the Apache web server. Think of these layers as simply running the **apt-get** command and adding a new software package. What happens if you need to update the OpenSSL layer because of a vulnerability? You simply rebuild the container with the latest versions of software. There is no patching with a container; you destroy the old one and re-create a new one with updated code. Later on in this section, you will learn how this works through a Dockerfile, which is how Docker builds container images.

DOCKER ARCHITECTURE

Just like your engine in a car, the Docker Engine is the central component of the Docker architecture. Docker is a client/server application that installs on top of a Linux, Mac, or Windows operating system and provides all the tools to manage a container environment. It consists of the Docker daemon and the Docker client. Docker allows you to package up an application with all of its

dependent parts (binaries, libraries, and code) into a standardized package for software development.



The Docker architecture consists of three primary parts: the client, the Docker host, and the docker registry (see [Figure 13-25](#)).

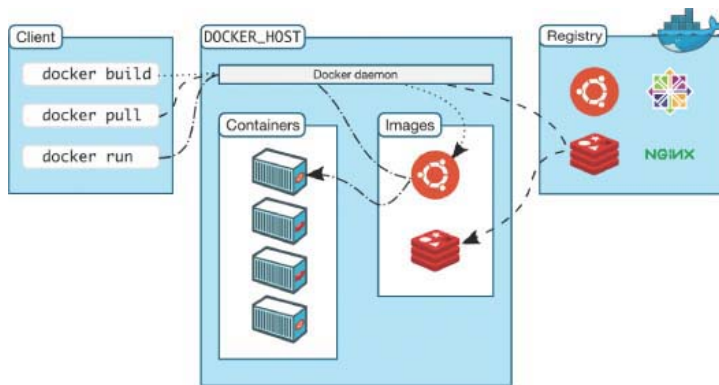


Figure 13-25 *Docker Architecture*

The Docker client is a command-line utility (run with the **docker** command) that talks to the REST API of the Docker daemon so that the administrator can issue commands for the operations and management of Docker containers. The client can communicate to a local or remote Docker instance.

The Docker host is where the Docker daemon resides. The Docker daemon (**dockerd**) is a service that runs on the host operating system and interfaces with the operating system kernel to allow containers to function. The client communicates with the Docker daemon through the REST API. The Docker host houses running containers and also interfaces with the registry to pull container images housed there. When containers are launched, the daemon looks in its local images, and if the appropriate image is there, it launches the container; if

the image is not there, the daemon asks the registry for the image file and then stores it locally.

The registry is a place to store container images; it is also known as the repository for the container infrastructure. You can pull images from a registry to run or push new images you create yourself to the registry for storage. Then other hosts in your container environment can make use of your new container image. A registry can be private, for the sole use of a single organization, or public, as in the case of Docker Hub.

USING DOCKER

The best way to get familiar with Docker is to use it. Download the appropriate version of Docker for your computer and launch its containers. It is available for both macOS, Windows, and Linux allowing you to install the Docker engine and command-line tools directly on your local machine.



Note

As of version 1.13, Docker has changed the command line to include a more logical grouping for people just getting started. If you are familiar with the old-style command line, fear not, those commands still work. The additions of the management command syntax just add a hierarchy that will differentiate what you are working on instead of everything being lumped together. This book uses the new Docker command structure. If you were to get a question on the exam that uses the old command line, you could simply omit the first command after **docker**. For example, the command **docker container ps** would be shortened to **docker ps**.



The command **docker** runs the client process, which communicates to the Docker daemon. When you type **docker** and press Enter, you see a long list of available commands. [Example 13-1](#) displays a shortened version of this list.

Example 13-1 Docker Command List

[Click here to view code image](#)

```
$ docker

Usage: docker [OPTIONS] COMMAND

A self-sufficient runtime for containers
<cut for brevity>
Management Commands:
  builder      Manage builds
  checkpoint   Manage checkpoints
  config       Manage Docker configs
  container    Manage containers
  context      Manage contexts
  image        Manage images
  network      Manage networks
  node         Manage Swarm nodes
  plugin       Manage plugins
  secret       Manage Docker secrets
  service      Manage services
  stack        Manage Docker stacks
  swarm        Manage Swarm
  system       Manage Docker
  trust        Manage trust on Docker images
  volume       Manage volumes
<cut for brevity>

Run 'docker COMMAND --help' for more
information on a command.
```

In [Example 13-1](#), notice the list of management commands. These represent the new hierarchy into which Docker groups subcommands. For commands that operate on containers, you can enter **docker container** and then the operation you wish to perform. At any time,

you can type **--help** to get more details on what commands are available, as shown in [Example 13-2](#).

Example 13-2 *Docker Container Help*

[Click here to view code image](#)

```
$ docker container --help

Usage: docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input,
              output, and error streams to a running
              container
  commit      Create a new image from a
              container's changes
  cp          Copy files/folders between a
              container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or
              directories on a container's filesystem
  exec        Run a command in a running
              container
  export      Export a container's filesystem
              as a tar archive
  inspect     Display detailed information on
              one or more containers
  kill        Kill one or more running
              containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or
              more containers
  port        List port mappings or a specific
              mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped
              containers
  stats       Display a live stream of
              container(s) resource usage statistics
  stop        Stop one or more running
              containers
  top         Display the running processes of
              a container
  unpause     Unpause all processes within one
```

```
or more containers
  update      Update configuration of one or
more containers
  wait        Block until one or more
containers stop, then print their exit codes
```

You can run the command **docker container** *COMMAND* **--help** for more information on a command.

For the purposes of the 200-901 DevNet Associate DEVASC exam, you are expected to know how to use Docker images in a local developer environment. This means you are not expected to be an expert on all things Docker, but you do need to know how to build, launch, and manage containers on your local machine. The management commands you will use the most for launching and working with Docker are within the **container** and **image** categories. If you are working with a container image, you use the commands under **image**, and if you want to run or stop a container, you use the commands under **container**. As long as you know what you want to work on, mapping it to the command is fairly straightforward.

Working with Containers

When working with containers, the key commands are as follows:

- **create:** Create a container from an image.
- **start:** Start an existing container.
- **run:** Create a new container and start it.
- **ls:** List running containers.
- **inspect:** Get detailed information regarding the container.
- **logs:** Print run logs from the container's execution.
- **stop:** Gracefully stop running the container.
- **kill:** Stop the main process in the container abruptly.
- **rm:** Delete a stopped container.

There are a few ways to launch a container. You may have noticed that the **create**, **start**, and **run** commands seem very similar. The **create** command is used to instantiate a container from an image but does not start the container. Think of it as preloading a container that you can run later. The **start** command starts up the loaded container and allows it to execute. The **run** command, on the other hand, creates and starts the container as a single command. [Example 13-3](#) provides a quick example of how to launch a sample container with the **run** command.

Example 13-3 *Docker Hello World*

[Click here to view code image](#)

```
$ docker run hello-world
Unable to find image 'hello-world:latest'
locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest:
sha256:4df8ca8a7e309c256d60d7971ea14c27672fc0d10c5f803856d7bc48f8cc17ff

Status: Downloaded newer image for hello-
world:latest

Hello from Docker!
This message shows that your installation
appears to be working correctly.

To generate this message, Docker took the
following steps:
 1. The Docker client contacted the Docker
daemon.
 2. The Docker daemon pulled the "hello-world"
image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container
from that image which runs the
    executable that produces the output you are
currently reading.
 4. The Docker daemon streamed that output to
the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an
Ubuntu container with:
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

In [Example 13-3](#), Docker looks for the container image `hello-world` and can't find it in the local image store. Since it has never downloaded it before, it defaults to the Docker Hub registry, does a lookup, finds the image, and then downloads (or pulls, in Docker speak) the latest one. It then starts the container and prints to the console a message saying that the Docker installation is working as expected. Since you now have a container that has been executed, you can use the subcommand `ls` to see what is running:

[Click here to view code image](#)

```
$ docker container ls

CONTAINER ID  IMAGE          COMMAND        CREATED
STATUS       PORTS         NAMES
```

Unfortunately, you don't see any active containers. That is because the one that was launched ran, printed the message, and then ended. To see containers that have been run but are now stopped, you have to add the `-a` flag to `ls`:

[Click here to view code image](#)

```
$ docker container ls -a

CONTAINER ID  IMAGE          COMMAND        CREATED
STATUS       PORTS         NAMES

bac0c2a2cca8  hello-world   "/hello"      43 minutes
ago         Exited(0)    43 minutes ago
```

Now you can see the container ID, the image name, the command that was issued, and its current status, which

in this case is exited. You also see any ports or names that were assigned to the container.

Note

The older syntax for Docker used the **ps** command instead of **ls** (for example, **docker ps -a**). You can still type **docker container ps -a** and get exactly the same output as the previous example, even though the **ps** command doesn't appear in the menu. This is for legacy compatibility.

Launching a container and interacting with it requires a few more commands. [Example 13-4](#) runs a Ubuntu container image (**ubuntu**) in interactive mode and connects the local terminal to the containers terminal and launches the bash shell (**bash**). (The options **-i** and **-t** can be presented separately or together, as **-it**.)

Example 13-4 docker run output *Command*

[Click here to view code image](#)

```
$ docker container run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
7ddbc47eeb70: Pull complete
c1bbdc448b72: Pull complete
8c3b70e39044: Pull complete
45d437916d57: Pull complete
Digest:
sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d

Status: Downloaded newer image for
ubuntu:latest
root@a583eac3cadb:/#
```

Since this container hasn't been loaded locally before, Docker pulls the image from Docker Hub, layer by layer. It executes a Dockerfile hosted on the repository and pulls the very latest version of a base Ubuntu container. Once it is pulled down, the container is started, and the bash shell is executed. The terminal now shows that you

are connected to the container as root (in the container's namespace). If you execute the bash **ps** command, you can see that there are only two things: the bash shell and the **ps** command that you just ran. This highlights the container's isolation from the host OS:

```
root@a583eac3cadb:/# ps
PID TTY          TIME CMD
  1 pts/0        00:00:00 bash
 11 pts/0        00:00:00 ps
```

To disconnect from the container, you can type **exit**, but that would stop the container. If you want to leave it running, you can hold down Ctrl and press P and then while still holding Ctrl press Q. This sequence drops you back to the host OS and leaves the container running. From the host OS, you can type **docker container ls** to see that your container is actively running:

[Click here to view code image](#)

```
$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED
STATUS        PORTS    NAMES
a583eac3cadb  ubuntu   "bash"    About a minute ago
Up About a minute           musing_bartik
```

If you want to reconnect to the container, you can use the container ID or the assigned name to attach to the container and interact with it again:

[Click here to view code image](#)

```
$ docker container attach a583eac3cadb
root@a583eac3cadb:/#
```

The **inspect** and **logs** commands are two very useful commands for troubleshooting a container. **inspect** generates a ton of information—pretty much everything

having to do with the creation of your container, such as network and runtime environment, mappings, and so on (as shown in [Example 13-5](#)). Be warned: Reading the output is like reading a container's DNA.

Example 13-5 docker container inspect Command

[Click here to view code image](#)

```
$ docker container inspect a583eac3cadb
[
  {
    "Id":
      "a583eac3cadbafca855bec9b57901e1325659f76b37705922db67ebf22fdd925",
    "Created": "2019-11-
      27T23:35:12.537810374Z",
    "Path": "bash",
    "Args": [],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 127,
      "Error": "",
      "StartedAt": "2019-11-
        27T23:35:13.185535918Z",
      "FinishedAt": "2019-11-
        27T23:53:15.898516767Z"
    },
    "Image":
      "sha256:775349758637aff77bf85e2ff0597e86e3e859183ef0baba8b3e8fc8d3c
        ba51c",
    "ResolvConfPath":
      "/var/lib/docker/containers/a583eac3cadbafca
        855bec9b57901e1325659f76b37705922db67ebf22fdd925/resolv.conf"
  }
]
<cut for brevity>
```

The **logs** command details everything the container recorded as output and is super helpful for catching error messages:

[Click here to view code image](#)

```
$ docker container logs a583eac3cadb
root@a583eac3cadb:/# ps
  PID TTY          TIME CMD
   1 pts/0    00:00:00 bash
  11 pts/0    00:00:00 ps
root@a583eac3cadb:/# uname
Linux
```

Now that you know the basics on launching a container, the next step is to make it accessible to the rest of the world. To do this, you can start by pulling an nginx web server container and assigning it ports to use with the **-p** flag. This flag requires that you map the container port to a free port on the Docker host. If you don't assign one to your local host, Docker will pick a random port above 32700. In addition, you want to make your nginx container stay running in the background instead of just quitting after you launch it. By using the **-d** flag, you can detach the container and leave it running in the background. The last thing you can try is to give the container a name with the **--name** flag. This way, you can refer to it by name instead of by using the random one Docker assigns or the container ID:

[Click here to view code image](#)

```
$ docker container run --name test-nginx -p 80:80
-d nginx
dfe3a47945d2aa1cdc170ebf0220fe8e4784c9287eb84ab0bab7048307b602b9
```

After the container is launched, you can see it running by using the **ls** command, get information on what port it is using (port 80), and see that the container is mapped to the local host IP address (referenced by 0.0.0.0):

[Click here to view code image](#)

```
$ docker container ls
```

```

CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS NAMES

dfe3a47945d2 nginx "nginx -g 'daemon of..." 21
seconds ago Up 20 seconds 0.0.0.0: test-nginx
80->80/tcp

```

You can also use **docker container port test-nginx** to list just the port mapping.

If you open a web browser and point it to `http://0.0.0.0`, your nginx server should show you a default HTML page (see [Figure 13-26](#)).

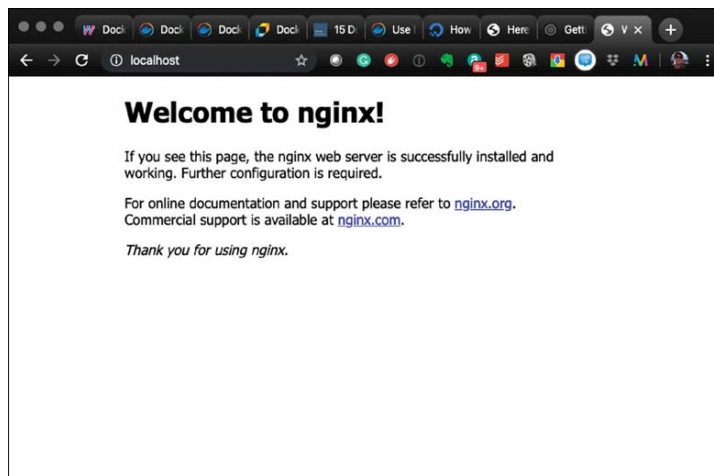


Figure 13-26 *nginx in a Docker Container*

A web server with default content is not very useful. Containers are meant to be read-only, but there are a couple ways to add your own content. There is the **cp** command, which allows you to copy files from your local host to the container when it is up and running; however, in order to save those changes, you would have to store another layer on top of your container and then push that new container to your repository. This would have to be done every time a change is made. A better way is to mount the content directly to the container when it boots up so that it can share the content. This way, if you need to update anything, you just change the files in one

directory, and those changes are automatically shared to any container using them. A quick example of this using the nginx server should make this process clearer.

We can start with some HTML files that nginx can display. I have a simple HTML file in a directory called `html` under Documents on my local laptop running Docker; in macOS, the path is `~/Documents/html`. There is a default directory where nginx looks for HTML files to display to users connecting to the service: `/usr/share/nginx/html`. You have to supply this mapping with the `-v` or `--volume` flag when you launch your container using the previous example. Here is what it looks like:

[Click here to view code image](#)

```
$ docker container run --name test-nginx -p 80:80
-d -v
~/Documents/html:/usr/share/nginx/html nginx
d0d5c5ac86a2994ea1037bd9005cc8d6bb3970bf998e5867fe392c2f35d8bc1a
```

After you enter this long command, when you connect to `http://0.0.0.0`, you see your own HTML content. The `-v` flag maps your local directory to the shared directory on the container, and from the container's perspective, those files are in `/usr/share/nginx/html`. [Figure 13-27](#) shows the results of this container mapping to the container host's file system.

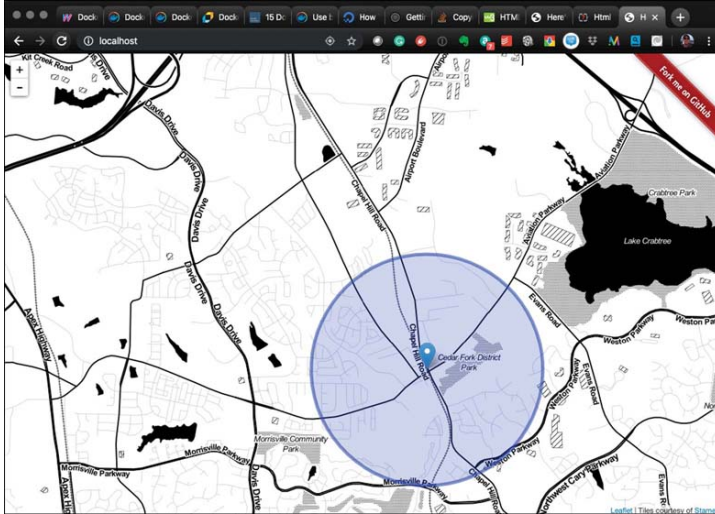


Figure 13-27 Custom Content Shared from the Local Host

To stop running a container, you use the **stop** or **kill** command. The difference between the two is pretty obvious. **stop** tries to allow the container time to finish any final work in a graceful way. Using **kill**, on the other hand, is like pulling the plug. For most situations, you want to use **stop**:

[Click here to view code image](#)

```
$ docker container stop test-nginx
test-nginx
```

stop and **kill** both halt the container but don't remove it from memory. If you type **docker container ls -a**, you will still see it listed. When you want to remove the container from memory, you can use the **rm** command. You can also use the very handy command **prune** to remove all halted containers from memory. This is useful on a laptop when you're testing containers and want to free up memory without having to individually remove multiple containers. Just be careful not to get rid of any container you might actually want to start back up because **prune** will get rid of every inactive container. Here is an example of **prune** in action:

[Click here to view code image](#)

```
$ docker container rm test-nginx
test-nginx

$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
a583eac3cadbafca855bec9b57901e1325659f76b37705922db67ebf22fdd925
```

Dockerfiles

A Dockerfile is a script that is used to create a container image to your specifications. It is an enormous time saver and provides a ready-made system for replicating and automating container deployments. Once you have built a Dockerfile, you can create an infinite number of the same images without having to manually edit or install software. A Dockerfile is simply a text file with a structured set of commands that Docker executes while building the image. Some of the most common commands are as follows:



- **FROM:** Selects the base image used to start the build process or can be set to **scratch** to build a totally new image.
- **MAINTAINER:** Lets you select a name and email address for the image creator.
- **RUN:** Creates image layers and executes commands within a container.
- **CMD:** Executes a single command within a container. Only one can exist in a Dockerfile.
- **WORKDIR:** Sets the path where the command defined with **CMD** is to be executed.
- **ENTRYPOINT:** Executes a default application every time a container is created with the image.
- **ADD:** Copies the files from the local host or remotely via a URL into the container's file system.

- **ENV:** Sets environment variables within the container.
- **EXPOSE:** Associates a specific port for networking binding.
- **USER:** Sets the UID (or username) of the user that is to run the container.
- **VOLUME:** Sets up a sharable directory that can be mapped to a local host directory.
- **LABEL:** Provides a label to identify the created Docker image.

Creating a Dockerfile starts with creating a text file. The file must be named Dockerfile, and you place it in the working directory where you want to create an image. Once you have opened your favorite editor, the syntax is straightforward. Docker reads the file and operates on it line by line, so the order in which you build your Dockerfile should follow the steps you would perform manually to load your software and configure it. First, you need to select your foundation. This can be any container, but in this instance, you can use Ubuntu as the foundation. If you don't select a specific version, Docker defaults to the latest version. You should also provide details on who created the image with a full name and email address. Here is an example:

[Click here to view code image](#)

```
FROM ubuntu:16.04
MAINTAINER Cisco Champion (user@domain.com)
```

Next, you need to specify the software you want to load and what components should be added. Just as when loading software on a Linux server, you first need to run **apt-get update**. For this container, you install nginx and any dependencies:

[Click here to view code image](#)

```
RUN apt-get update && apt-get upgrade -y
RUN apt-get install nginx -y
```

Now you need to tell Docker what port to expose for networking. In this case, you would expose ports 80 and 443:

```
EXPOSE 80 443
```

Next, you create a sharepoint for HTML files in nginx so that you can give it access to your web files:

```
VOLUME /usr/share/nginx/html
```

Finally, set the container to run nginx on launch:

[Click here to view code image](#)

```
CMD ["nginx", "-g", "daemon off;"]

FROM ubuntu:latest
MAINTAINER Cisco Champion (user@domain.com)
RUN apt-get update && apt-get upgrade -y
RUN apt-get install nginx -y
EXPOSE 80 443
VOLUME /usr/share/nginx/html
CMD ["nginx", "-g", "daemon off;"]
```

Now that you have a Dockerfile, it's time to build an image.

Docker Images

When working with Docker images, you primarily use the following commands:



- **build:** Builds an image from a Dockerfile.
- **push:** Pushes a local image to a remote registry for storage and sharing.
- **ls:** List images stored locally.

- **history:** Shows the creation and build process for an image.
- **inspect:** Provides detailed information on all aspects of an image, including layer detail.
- **rm:** Deletes an image from local storage.

In the previous example, you built a Dockerfile using the latest Ubuntu base and then updated and upgraded the Ubuntu base so that its packages are current. You then installed nginx, shared a mount point, and exposed ports to access the new web server. The next step is to kick off the build process with Docker. While in the directory where you created your Dockerfile, execute the build process as shown in [Example 13-6](#).

Example 13-6 *Building a Container from a Dockerfile*

[Click here to view code image](#)

```
$ docker build -t myimage:latest .
Sending build context to Docker daemon 2.048kB
Step 1/8 : FROM ubuntu:latest
----> 775349758637
Step 2/8 : MAINTAINER Cisco Champion
(user@domain.com)
----> Using cache
----> f83e0f07db18
Step 3/8 : RUN apt-get update
----> Using cache
----> 646cc0e9f256
Step 4/8 : RUN apt-get upgrade -y
----> Using cache
----> c2701f555b0f
Step 5/8 : RUN apt-get install nginx -y
----> Using cache
----> 4abf50fd4a02
Step 6/8 : EXPOSE 80 443
----> Using cache
----> a9a1533064b2
Step 7/8 : VOLUME /usr/share/nginx/html
----> Using cache
----> 2ecd13c5af2b
Step 8/8 : CMD ["nginx", "-g", "daemon off;"]
----> Running in e03bd2387319
Removing intermediate container e03bd2387319
----> 04ae8c714993
Successfully built 04ae8c714993
Successfully tagged myimage:latest
```


The build is successful. You can see that with the **-t** flag, you can set a name that can be used to call the image more easily. The **.** at the end of the command tells Docker to look for the Dockerfile in the local directory. You could also use the **-f** flag with a path to the location of Dockerfile file, if it is not in your current directory. Docker also accepts a URL to the Dockerfile, such as at GitHub. As shown in this example, the name **Dockerfile** must be capitalized and one word.

When an image is built, you can use **docker image ls** to see it in local storage:

[Click here to view code image](#)

```
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED
SIZE
myimage latest 04ae8c714993 48 minutes
ago 154MB
```

You can use a number of commands to get information from your images. The first is **inspect**. Entering **docker image inspect myimage**, as shown in [Example 13-7](#), gets all the details about the image, including its layer structure.

Example 13-7 docker image inspect Command

[Click here to view code image](#)

```
$ docker image inspect myimage
[
  {
    "Id":
      "sha256:04ae8c7149937b2ce8b8963d6c34810d8dc53607c9a97064e1f3c85cdc3
        abb46",
    "RepoTags": [
      "myimage:latest"
    ],
    "RepoDigests": [],
    "Parent":
      "sha256:2ecd13c5af2b41349b58f2397cbbbc70f5c1c604262f13bae443f1f6fc
        3758cc",
    "Comment": ""
```

```

    "Created": "2019-11-
28T05:53:37.794817007Z",
    "Container":
    "e03bd238731932ca2cab6c8b7fa1346105b839ce2a8604c0c7d
34352b907a4af",
    "ContainerConfig": {
      "Hostname": "e03bd2387319",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {
        "443/tcp": {},
        "80/tcp": {}
      },
      "Tty": false,
      "OpenStdin": false,
      "StdinOnce": false,
      "Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

<Cut for brevity>

```

The **image history** command shows you the creation of the image and all the steps that were taken to build the image, as shown in [Example 13-8](#). This can be very useful in troubleshooting an image build.

Example 13-8 docker image history Command

[Click here to view code image](#)

```

$ docker image history myimage
IMAGE          CREATED          CREATED BY
SIZE  COMMENT
04ae8c714993  56 minutes ago  /bin/sh -c #
(nop) CMD ["nginx" "-g" "daemon... 0B
2ecd13c5af2b  57 minutes ago  /bin/sh -c #
(nop) VOLUME [/usr/share/nginx/... 0B
a9a1533064b2  57 minutes ago  /bin/sh -c #
(nop) EXPOSE 443 80
4abf50fd4a02  57 minutes ago  /bin/sh -c apt-
get install nginx -y
60.2MB
c2701f555b0f  58 minutes ago  /bin/sh -c apt-
get upgrade -y
1.55MB
646cc0e9f256  58 minutes ago  /bin/sh -c apt-
get update
27.6MB

```

```

f83e0f07db18 58 minutes ago /bin/sh -c #
(nop) MAINTAINER Cisco Champion... 0B
775349758637 3 weeks ago /bin/sh -c #
(nop) CMD ["/bin/bash"] 0B
<missing> 3 weeks ago /bin/sh -c mkdir
-p /run/systemd && echo 'do... 7B
<missing> 3 weeks ago /bin/sh -c set -
xe && echo '#!/bin/sh' > /... 745B
<missing> 3 weeks ago /bin/sh -c [ -z
"$(apt-get indextargets)" ] 987kB
<missing> 3 weeks ago /bin/sh -c #
(nop) ADD file:a48a5dc1b9dbfc632... 63.2MB

```

You probably noticed the missing image IDs at the bottom of the history. The build process uses something called *intermediate layers* to execute commands. These layers are like temporary storage that gets rolled up into the next layer when each stage is finished, and then the intermediate layer is deleted.

When the image is built and ready to go, you can run it by using the following command:

[Click here to view code image](#)

```

$ docker container run -p 80:80 -p 443:443 -d
myimage

bf0889f6b27b034427211f105e86cc1bfeae8c3b5ab279ccaf08c114e6794d94

$ docker ps

CONTAINER ID   IMAGE     COMMAND                  CREATED
STATUS        PORTS    NAMES
bf0889f6b27b  myimage  "nginx -g               7 seconds
ago Up 6 seconds 0.0.0.0:80->80/tcp,
elastic_maxwell
                  'daemon of...'"
0.0.0.0:443->443/tcp

```

You can use the **docker image rm** command to remove an image from storage. This command needs the container ID or a name to select the appropriate container.

DOCKER HUB

Docker Hub is a free service that Docker offers to the community to house and share prebuilt container images. It is the world's largest repository for containers, with more than 100,000 prebuilt container applications. Chances are that if you are looking to build a container, someone has already put together one that would work for you. Many software vendors also publish official images that have been validated by the vendor and Docker; when you use such an image, you can be confident that the container you are using has been tested and is malware free. While anyone can sign up for a free account, Docker makes money by offering, for a fee, private container repositories suitable for individuals and businesses. Docker Hub is a cloud-only service, so if you want to run a private container registry in your data center, you have to deploy Docker Enterprise software or another container registry system on premises.



Docker Hub has a slick web interface that makes it easy to set up a repository and secure it. It also integrates with GitHub and Bitbucket to enable automated container creation from source code stored in your version control system. It's a great way to start building a CI/CD pipeline as it is free to get started. If you set up a free account, you can take advantage of Docker Hub's capabilities immediately. [Figure 13-28](#) shows Docker Hub signup page.

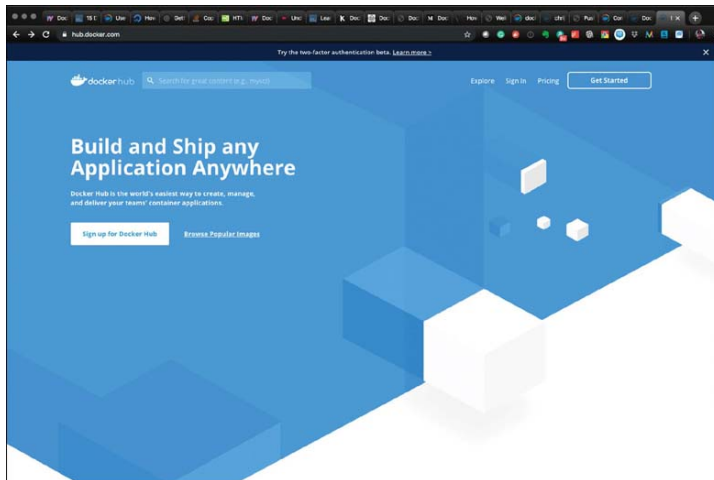


Figure 13-28 *Signing Up for Docker Hub*

Once you have an account, you can search through the many containers available and can filter on operating system, category, and whether or not the image is an official Docker certified image, verified by the publisher, or an official image published by Docker itself. [Figure 13-29](#) shows the image search function.

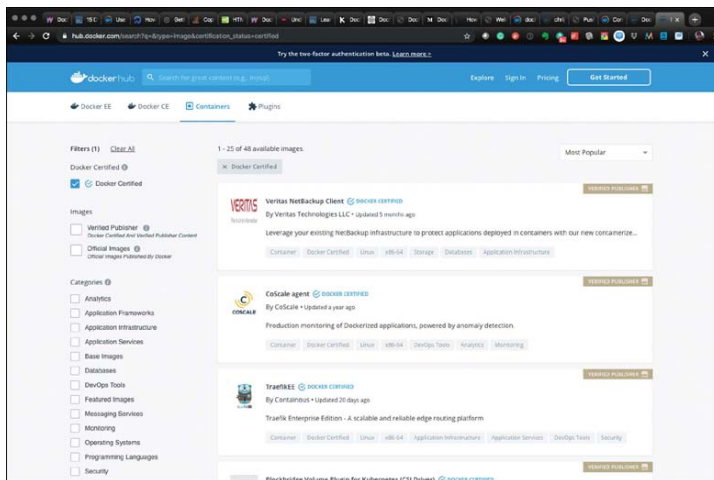


Figure 13-29 *Docker Hub Image Search*

Docker ships a GUI application called Kitematic for interacting with containers on your local host as well as Docker Hub. This free app can download and launch containers with a simple point and click. It's great for people who are struggling with all of the command-line

options that the typical Docker client requires. But be warned: With Kitematic you don't have as much direct control as you do with the Docker client. Download it and give it a try. [Figure 13-30](#) shows Kitematic in action.

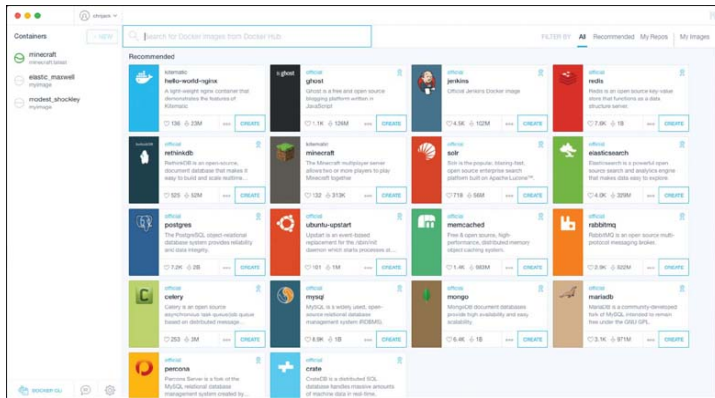


Figure 13-30 Kitematic

After you build a container, as you did in the previous example, you need to push it to a repository. Since you can use Docker Hub for free, you can set up a private repository to store this container. If you wanted to share with the world the container you created, you could do so by simply making the repository public. A private repository is always a good bet for code that is just for your organization. [Figure 13-31](#) shows how to set up a private repo in Docker Hub named newrepo that you can push your new container to.

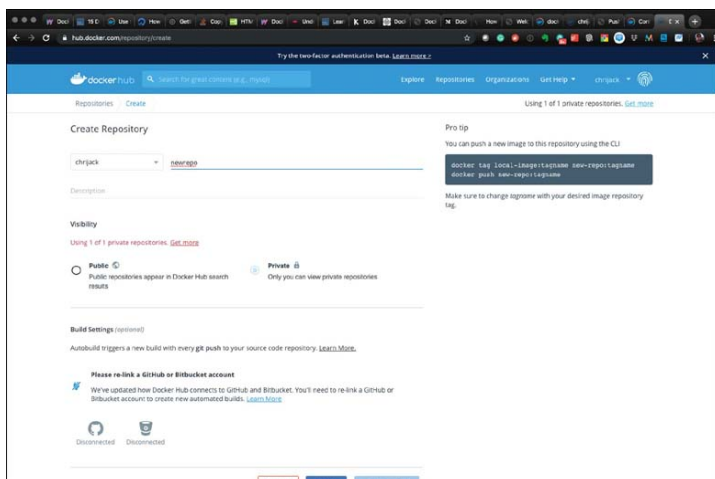


Figure 13-31 Docker Hub Repository Setup

Once your repository is set up, you need to tag your image to be pushed to Docker Hub. Use **docker image ls** to list your images and take note of the image ID:

[Click here to view code image](#)

```
$ docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED
SIZE
myimage       latest    04ae8c714993   About an
hour ago     154MB
```

Next, you need to issue the **docker image tag** command with the image ID of your image and assign it to *username/newrepo:firsttry*. *newrepo* simply identifies the repo you want to place the image in, and the tag after the **:** allows you to differentiate this particular image from any other in the repository (in this instance, it is named *firsttry*):

[Click here to view code image](#)

```
$ docker image tag 04ae8c714993
chrijack/newrepo:firsttry
```

Now you are ready to issue the **push** command, as shown in [Example 13-9](#), and you see Docker start to push the layers of your container image to the repository.

Example 13-9 docker image push Command

[Click here to view code image](#)

```
$ docker image push chrijack/newrepo:firsttry
The push refers to repository
[docker.io/chrijack/newrepo]
3e92d80e0ac4: Pushed
a87bf84680fc: Pushed
02d0765ebf97: Pushed
e0b3afb09dc3: Pushed
6c01b5a53aac: Pushed
2c6ac8e5063e: Pushed
```

```
cc967c529ced: Pushed
firsttry: digest:
sha256:01cf95854003cd1312fb09286d41bc6bfd9fe3fb82f68e66e5060c7fd5a

6230a size: 1786
```

You can now check back with Docker Hub and see that the image is now hosted in your private repository (see [Figure 13-32](#)).

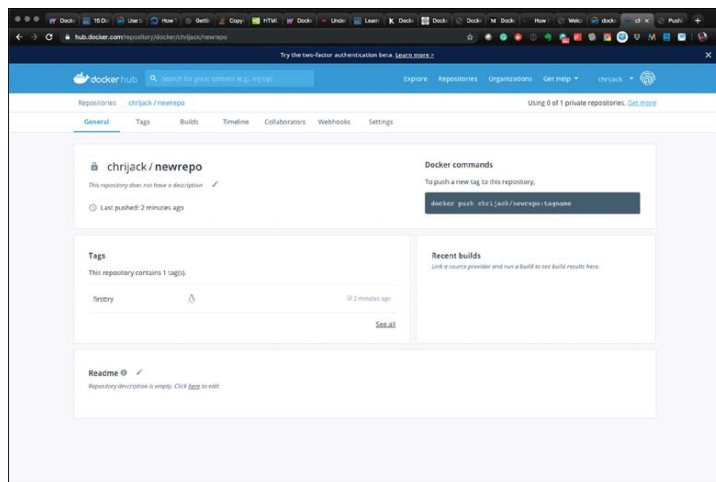


Figure 13-32 Docker Hub Pushed Image

After this, any time you want to retrieve the image, you can type **docker image pull username/newrepo:firsttry**, and Docker will load it in your local image storage.

There is quite a bit more that you can do with Docker, but this section should get you started and focused on what you need to know for the 200-901 DevNet Associate DEVASC exam. Make sure that to review the Docker documentation and try these examples on your own computer. Nothing beats practical experience.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “Final Preparation,” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 13-2](#) lists these key topics and the page number on which each is found.

Table 13-2 Key Topics

Key Topic Element	Description	Page Number
Paragraph	NIST definitions of cloud	376
Paragraph	Cloud deployment models	379
Paragraph	Edge computing model	381
Paragraph	Application deployment methods	382
Paragraph	The three ways of DevOps	391
Paragraph	Implementing DevOps	394
Paragraph	Docker architecture	400
Note	Command line change	401
Paragraph	Dockerfiles	410
List	Docker images	411
Paragraph	Docker Hub	414



DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

software as a service (SaaS)

private cloud

public cloud

hybrid cloud

edge computing

container

DevOps

registry

artifact repository

Docker

Docker daemon

continuous integration/continuous deployment (CI/CD)

Docker Hub

ADDITIONAL RESOURCES

Periodic Chart of DevOps:

<https://xebialabs.com/periodic-table-of-devops-tools/>

15 Docker Commands You Should Know:

<https://towardsdatascience.com/15-docker-commands-you-should-know-970ea5203421>

Docker Basics: How to Use Dockerfiles:

<https://thenewstack.io/docker-basics-how-to-use-dockerfiles/>

Chapter 14

Application Security

This chapter covers the following topics:

- **Identifying Potential Risks:** This section introduces some of the concepts involved in application security and shows how to identify potential risks in applications.
- **Protecting Applications:** This section shows how to protect an application from various vulnerabilities using various frameworks and also by leveraging firewalls, IDSs, and IPSs.

Application security involves making applications more secure by proactively finding, fixing, and enhancing the safety of applications. Much of what happens during the development phase includes tools and methods to protect apps once they are deployed. Today, application security is paramount. Through the years, many tools have emerged to secure networks, storage, and even code. Various coding tools and methodologies can be used to access inadvertent code threats.

In this chapter, you will learn about application security issues and how applications can be secured using modern networking components. This chapter also provides an overview of the Open Web Application Security Project (OWASP) and what it brings to the table for application developers.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or

your own assessment of your knowledge of the topics, read the entire chapter. [Table 14-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 14-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Identifying Potential Risks	1-4
Protecting Applications	5-8

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

- 1.** A vulnerability is a _____ in protection efforts. It can be exploited by threats to gain unauthorized access to an asset.
 1. strength
 2. weakness
 3. neutral condition
 4. side effect
- 2.** Which of the following are threats? (Choose two.)
 1. Phishing
 2. Man-in-the-middle

3. JWT token
 4. Brute force
3. What type of test is used to identify the possible weak spots in applications, servers, or networks?
1. Pen
 2. White box
 3. Eraser
 4. Unit
4. Which of the following is a tool that can be used for network discovery and security auditing?
1. Nslookup
 2. Nmap
 3. ps
 4. curl
5. What is the minimum number of identity components MFA uses to authenticate a user's identity?
1. One
 2. Two
 3. Three
 4. Four
6. Which of the following is used for fingerprinting of data to detect whether the data has been modified?
1. Private key
 2. Public key
 3. One-way hash
 4. Certificate
7. Data needs to be secured in multiple locations. Which of the following are the correct locations?
1. Memory, storage, and network
 2. Flash, wire, and optics
 3. Hard disk, database, and analytics
 4. AWS, Google Cloud, and Azure
8. Which modes best describe IDSs and IPSs, respectively?
1. Passive, passive
 2. Passive, active
 3. Active, passive
 4. Active, active

IDENTIFYING POTENTIAL RISKS

The National Institute of Standards and Technology (NIST) defines a framework called the Cybersecurity Framework that is shown in [Figure 14-1](#). The framework organizes necessary cybersecurity activities into the following functions:

- **Identify:** An organization needs to understand what kinds of cybersecurity risks can affect daily modes of operation. Areas such as data theft, network breaches, and employee information are some of the risks that need to be identified.
- **Protect:** An organization needs to understand what it can do to prevent attacks. Protection could include deploying proper networking elements such as firewalls and tools for better software development. Protection helps minimize the impact of any attack.
- **Detect:** It is important to install tools that detect any data breaches or attacks in a time-sensitive manner or while attacks are happening.
- **Respond:** An organization needs to have a plan in place to deal with an attack. It needs to know what procedures need to be followed and what can be done to minimize the impact of an attack.
- **Recover:** An organization needs to be able to quickly resolve any services or systems that have been affected.

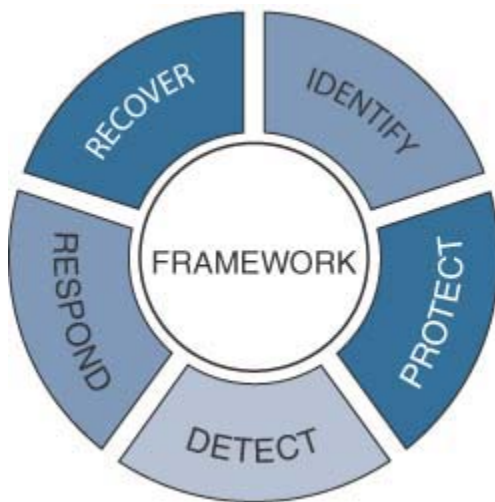


Figure 14-1 *NIST Cybersecurity Framework*

For more details about the Cybersecurity Framework, visit <https://www.nist.gov/cyberframework/online-learning/components-framework>.

Application security, as explained in this chapter, uses the Cybersecurity Framework as a substratum to understand some of the key concepts. To ensure application security, it is important to identify threats and what they affect.



Before we talk about the potential risks, it is essential to understand some key terms and their relationships. The following are some commonly misunderstood terms:

- **Asset:** An asset is something you're trying to protect. It could be information, data, people, or physical property. Assets are of two types: tangible and intangible. Information and data may include databases, software code, and critical company records. People include employees and customers. Intangible assets include reputation and proprietary information.
- **Threat:** Anything you are trying to protect against is known as a threat. A threat is any code or person that can exploit a vulnerability when it has access to an asset and can control or damage the asset.
- **Vulnerability:** A vulnerability is a weakness or gap in protection efforts. It can be exploited by threats to gain unauthorized access to an asset.
- **Risk:** Risk is the intersection of assets, threats, and vulnerabilities. Certain risks can potentially compensate for loss, damage, or destruction of an asset as a result of a threat exploiting a vulnerability. Threats can always exist, but if vulnerabilities don't exist, then there is little or no risk. Similarly, you can have a weakness, but if you have no threat, then you have little or no risk.

Common Threats and Mitigations

In this section, we examine the most common threats. [Table 14-2](#) describes some of the dangers to application and some mitigation tips for each one of them.

Table 14-2 Threats and Sample Mitigation Options



--

Threat What It Does Mitigation Options

Buffer overflow

An attacker uses a program to try to cause an application to store that input in a buffer that isn't large enough. The attacker's data overwrites portions of memory connected to the buffer space. Attackers can use a buffer overflow to modify a computer's memory and undermine or take control of program execution.

- Separate executable memory from non-executable memory.

- Randomize address spaces for data.

- Use the built-in protection options in

		<p>wer soft wa re OS s an d lan gua ges .</p>
<p>M a n - i n - t h e - m i d d l e</p>	<p>Attackers insert themselves between two endpoints (such as a browser and a web server) and intercept or modify communications between the two. The attackers can then collect information as well as impersonate either of the two agents. In addition to targeting websites, these attacks can target email communications, DNS lookups, and public Wi-Fi networks.</p>	<ul style="list-style-type: none"> • Ad opt a Sec ure Soc ket s Lay er (SS L)/ Tra nsp ort Lay er Sec urit y (TL S) str ate gy for bot h we b an d em ail.

		<ul style="list-style-type: none"> • Avoid sensitive data in public Wi-Fi or computers.
Denial-of-service attack	<p>A threat actor sends multiple requests that flood the server or networks with traffic to exhaust resources and bandwidth. As the system performance degrades, the system becomes more and more nonresponsive, and legitimate requests are left unfulfilled. These kinds of attacks can be coordinated, with multiple devices launching attacks at the same time. Such an attack is known as a distributed denial-of-service (DDoS) attack.</p>	<ul style="list-style-type: none"> • Use specially designed protection services (cloud or network).
Crorts	<p>An attacker attaches to a legitimate website code that will execute when the victim loads that website. This is the typical process:</p>	<ul style="list-style-type: none"> • Validate

<p>s - s i t e s c r i p t i n g (X S S)</p>	<ol style="list-style-type: none"> 1. The webpage is loaded, and the malicious code copies the user's cookies. 2. The system sends an HTTP request to an attacker's web server with the stolen cookies in the body of the request. 3. The attacker can then use cookies to access sensitive data. 	<p>an d san itiz e inp ut dat a.</p> <ul style="list-style-type: none"> • Em plo y coo kie sec urit y, suc h as tim eou ts, enc odi ng the clie nt IP ad dre ss, an d so on.
<p>P h i s h i n g</p>	<p>A threat actor procures sensitive information—typically usernames, passwords, and so on—from emails or web pages.</p>	<ul style="list-style-type: none"> • Ed uca te use rs to avo id fall ing

		<p>for the bait.</p> <ul style="list-style-type: none"> • Detect and mark emails and sites as spam.
<p>Malware</p>	<p>Malware is a piece of malicious code such as spyware, ransomware, a virus, or a worm. Malware is usually triggered when someone clicks a link or an email attachment and inadvertently installs malicious software.</p>	<ul style="list-style-type: none"> • Deploy technologies that continually monitor and detect malware that has evaded

		perimeter defenses.
SQL injection technique	Structured Query Language (SQL) injection is a code injection technique used to modify or retrieve data from SQL databases. By inserting specialized SQL statements into an entry field, an attacker can execute commands that allow for the retrieval of data from the database.	<ul style="list-style-type: none"> Use character escaping. Use stored procedures as opposed to queries. Enforce privileges.
Brute force	A threat actor may use trial and error to decode data. Brute-force methods can be used to crack passwords and crack encryption keys. Other targets include API keys, SSH logins, and Wi-Fi passwords.	<ul style="list-style-type: none"> Lock the system

c e		aft er a spe cifi ed nu mb er of att em pts. Us e two - fact or aut hor izat ion .
--------	--	--

Open Web Application Security Project

The goal of a penetration (pen) test is to identify the possible weak links in applications, servers, or networks that could be used to gain sensitive information or privileged access for an attacker. It is important to detect such vulnerabilities not only to know that they exist and calculate the risk attached to them but also to make an effort to mitigate them or reduce them to the minimum risk level.

The Open Web Application Security Project (OWASP) is a community that produces and articulates various reports, tools, methodologies, and technologies for web application security. OWASP classifies and defines many application security vulnerabilities.



OWASP has defined a list of security risks called the OWASP Top 10, which can be found at <https://owasp.org/www-project-top-ten/>.

This is the current OWASP Top 10 list:

1. Injection
2. Broken authentication
3. Sensitive data exposure
4. XML external entities
5. Broken access control
6. Security misconfiguration
7. Cross-site scripting
8. Insecure deserialization
9. Using components with known vulnerabilities
10. Insufficient logging and monitoring

The CVE (which stands for Common Vulnerabilities and Exposures) is a list of publicly disclosed computer security vulnerabilities. When someone refers to a CVE, he or she usually means the CVE ID number assigned to a security defect. Security advisories issued by vendors and researchers almost always mention at least one CVE ID. CVEs help IT professionals coordinate their efforts to prioritize and address these vulnerabilities to make computer systems more secure. CVEs are supervised by the MITRE Corporation, with funding from the Cybersecurity and Infrastructure Security Agency, part of the U.S. Department of Homeland Security. A CVE record usually provides a short one-line description. Details are usually available on sites such as the U.S. National Vulnerability Database NVD; (<https://nvd.nist.gov/>) and the CERT/CC Vulnerability Notes Database (<https://www.kb.cert.org/vuls/>), as well as the sites of prominent vendors in the industry.

A CVE record consists of the following:

- ID
- Description

- Impact (low/moderate/important/critical)
- Date published

A CVE record would look something like this:

- ID: CVE-2020-5313
- Description: An out-of-bounds read was discovered in python-pillow in the way it decodes FLI images. An application that uses python-pillow to load untrusted images may be vulnerable to this flaw, which can allow attackers to read the memory of the application they should be not allowed to read.
- Impact: Moderate
- Date: January 3, 2020

Using Nmap for Vulnerability Scanning

Nmap, which is short for Network Mapper, is a free open-source utility for network discovery and security auditing. Nmap is commonly used for network scanning or vulnerability scanning. Target users for this tool are pen testers, security professionals, and system administrators. Nmap provides detailed and real-time snapshot information of the devices or hosts on a network. Nmap primarily provides three functionalities.

- It gives detailed information on every IP active on a network, and each IP address can then be scanned for more details, if needed.
- It provides a list of live hosts and open ports and identifies the OS of every connected device. This makes Nmap an excellent system-monitoring and pen-testing tool.
- It helps identify security vulnerabilities to protect against attacks.

The best way to get familiar with Nmap is to use it. Nmap is available with macOS and Linux by default. Example 14-1 shows some of the command-line options available.



Example 14-1 *Using Nmap for Network and Vulnerability Scanning*

[Click here to view code image](#)


```
$ nmap --help
Nmap 7.80 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target
specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks,
  etc.
  Ex: scanme.nmap.org, microsoft.com/24,
  192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of
  hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>:
  Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list
  from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host
  discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or
  SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask
  request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve
  [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify
  custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host

<cut for brevity>
```

Basic Nmap Scan Against an IP Address or a Host

Many switch options can be used with Nmap, and here we focus on a practical one. To run Nmap against an IP address or a host, you can scan the hostname with the **nmap** *hostname* command, as shown in [Example 14-2](#). (In this example, *hostname* is www.google.com, but you can replace it with any IP address or hostname, including localhost.) Use the **-vv** option as shown in this example to see a more verbose output.

Example 14-2 Using Nmap to Get Details About a Host or an IP Address

[Click here to view code image](#)

```
$ nmap -vv www.google.com
Starting Nmap 7.80 ( https://nmap.org ) at
2019-12-08 22:10 PST
Warning: Hostname www.google.com resolves to 2
IPs. Using 216.58.194.196.
Initiating Ping Scan at 22:10
Scanning www.google.com (216.58.194.196) [2
ports]
Completed Ping Scan at 22:10, 0.02s elapsed (1
total hosts)
Initiating Parallel DNS resolution of 1 host.
at 22:10
Completed Parallel DNS resolution of 1 host. at
22:10, 0.06s elapsed
Initiating Connect Scan at 22:10
Scanning www.google.com (216.58.194.196) [1000
ports]
Discovered open port 443/tcp on 216.58.194.196
Discovered open port 80/tcp on 216.58.194.196
Completed Connect Scan at 22:10, 8.49s elapsed
(1000 total ports)
Nmap scan report for www.google.com
(216.58.194.196)
Host is up, received syn-ack (0.025s latency).
Other addresses for www.google.com (not
scanned): 2607:f8b0:4005:804::2004
rDNS record for 216.58.194.196: sfo03s01-in-
f196.1e100.net
Scanned at 2019-12-08 22:10:05 PST for 9s
Not shown: 998 filtered ports
Reason: 998 no-responses
PORT      STATE SERVICE REASON
80/tcp    open  http   syn-ack
443/tcp   open  https  syn-ack
```

CVE Detection Using Nmap

One of Nmap's most magnificent features for finding vulnerabilities is called the Nmap Scripting Engine (NSE). The NSE allows you to use a predefined script or even write your own by using Lua programming language.

Using NSE is a crucial part of automating system and vulnerability scans. It requires the following syntax:

```
nmap -Pn --script vuln hostname
```

Example 14-3 shows an example of a vulnerability scan for a device on my home network.

Example 14-3 *Using the Nmap Scripting Engine*

[Click here to view code image](#)

```
$ nmap -Pn --script vuln 10.168.243.179
Starting Nmap 7.80 ( https://nmap.org ) at
2019-12-08 22:18 PST
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-
1002).
|_ Hosts are all up (not vulnerable).
Illegal character(s) in hostname -- replacing
with '*'
Nmap scan report for RX-
V677*B9772F.hsd1.ca.domain.net (10.168.243.179)
Host is up (0.029s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE
80/tcp    open  http
|_clamav-exec: ERROR: Script execution failed
(use -d to debug)
| http-csrf:
| Spidering limited to: maxdepth=3;
maxpagecount=20; withinhost=RX-
V677*B9772F.hsd1.ca.domain.net
|   Found the following possible CSRF
vulnerabilities:
|
|   Path: http://RX-
V677*B9772F.hsd1.ca.domain.net:80/
|   Form id: recoveryform
|_   Form action: /Config/avr_recovery.cgi
|_http-dombased-xss: Couldn't find any DOM
based XSS.
| http-fileupload-exploiter:
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
```

```
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Failed to upload and execute a payload.
|
|   Failed to upload and execute a payload.
|
|   Failed to upload and execute a payload.
|
|   Failed to upload and execute a payload.
|
|_  Failed to upload and execute a payload.
|_http-stored-xss: Couldn't find any stored XSS
vulnerabilities.
1029/tcp open  ms-lsa
|_clamav-exec: ERROR: Script execution failed
(use -d to debug)
1900/tcp open  upnp
|_clamav-exec: ERROR: Script execution failed
(use -d to debug)
8080/tcp open  http-proxy
|_clamav-exec: ERROR: Script execution failed
(use -d to debug)
|_http-aspnet-debug: ERROR: Script execution
failed (use -d to debug)
| http-enum:
|_ /test.html: Test page
| http-slowloris-check:
|   VULNERABLE:
|   Slowloris DOS attack
|     State: LIKELY VULNERABLE
|     IDs:  CVE:CVE-2007-6750
|     Slowloris tries to keep many
connections to the target web server open and
hold
|     them open as long as possible.  It
accomplishes this by opening connections
to
|     the target web server and sending a
partial request.  By doing so, it starves
|     the http server's resources causing
Denial Of Service.
|
|   Disclosure date: 2009-09-17
|   References:
|     https://cve.mitre.org/cgi-
```

```
bin/cvname.cgi?name=CVE-2007-6750
|_      http://ha.ckers.org/slowloris/
50000/tcp open  ibm-db2
|_clamav-exec: ERROR: Script execution failed
(use -d to debug)

Nmap done: 1 IP address (1 host up) scanned in
130.29 seconds
```

PROTECTING APPLICATIONS

An important step in protecting applications is to recognize the risks. Before we talk about the potential risks, it is essential to understand some key terms and their relationships:

- **Hacker or attacker:** These terms are applied to the people who seek to exploit weaknesses in software and computer systems for gain. The majority of the time, hackers have benign intentions and are simply motivated by curiosity; however, their actions may violate the intended use of the systems they are exploiting. The results can range from mere mischief (such as creating a virus with no intentionally negative impact) to malicious activity (such as stealing or altering information).
- **Malicious code:** Malicious code is unwanted files or programs that can cause harm to a computer or compromise data stored on a computer. Malicious code includes viruses, worms, and Trojan horses.

Tiers of Securing and Protecting

Enterprises depend on applications to run their businesses. Apps not only help customers and partners connect with the enterprise, but a lot of times they enable employees to get their jobs done as well. Unfortunately, applications remain one of the most commonly exploited threat vectors. An enterprise needs to secure and protect web, mobile, and API applications from being compromised and preventing data breaches. As mentioned earlier, application security is a framework that involves making applications more secure and, therefore, an end-to-end approach is needed.

The multilayered software architecture is one of the most famous architectural patterns. [Figure 14-2](#) shows a

simple three-tier application architecture that has the following foundations:

- **Tier 1 (Presentation):** This tier presents content to the end user through a web user interface or a mobile app or via APIs. To present the content, it is essential for this tier to interact with the other tiers. From a security standpoint, it is very important that access be authorized, timed, and encrypted and that the attack surface be minimized.
- **Tier 2 (Application):** This is the middle tier of the architecture, and it is where the business logic of the application runs. The components of this tier typically run on one or more application servers; hence, from a security standpoint, load balancing, limiting access, and proxying help.
- **Tier 3 (Data):** This is the lowest tier of this architecture, and it is mainly concerned with the storage and retrieval of application data. The application data is typically stored in a database server, a file server, or any other device or media that supports data access logic and provides the necessary steps to ensure that only the data is exposed, without providing any access to the data storage and retrieval mechanisms.

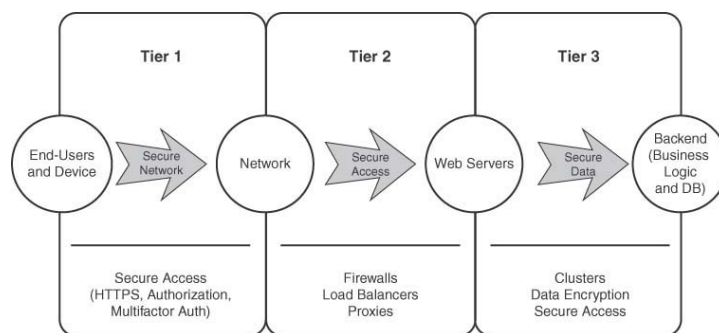


Figure 14-2 *Three-Tier Approach to Application Security*

To minimize risks, the following are some of the best practices in the industry:

- **Keep software up-to-date:** Install software patches so that attackers cannot take advantage of known problems or vulnerabilities. Many operating systems offer automatic updates.
- **Install end-user or device security:** Install endpoint security software on an end user's device so that viruses and malware are kept as far away as possible.
- **Use strong passwords:** Using a strong password ensures that only authorized users can access resources. With strong passwords, it becomes hard to guess and hence decreases security risk.

- **Implement multifactor authentication (MFA):** Authentication is a process used to validate a user's identity. Attackers commonly exploit weak authentication processes. MFA uses at least two identity components to authenticate a user's identity, minimizing the risk of a cyberattacker gaining access to an account by knowing the username and password.
- **Install a firewall:** Firewalls may be able to prevent some types of attack vectors by blocking malicious traffic before it can enter a computer system and by restricting unnecessary outbound communications.
- **Encrypt data:** Ensure that data cannot be accessed even if storage can be reached.

Encryption Fundamentals

Cryptography is the science of transmitting information securely against potential third-party adversaries. The main objectives of cryptography are the following:

- **Confidentiality:** Guaranteeing that the information exchanged between two parties is confidential between them and is not visible to anyone else
- **Integrity:** Ensuring that the integrity of a message is not changed while the message is in transit
- **Availability:** Ensuring that systems are available to fulfill requests all the time

Encryption is an operation that involves applying an encryption key to plaintext by using an encryption algorithm. Encryption turns the plaintext into ciphertext. Decryption is the inverse operation: The decryption key is applied to the ciphertext, and the result is the original plaintext. Encryption and decryption both involve symmetric keys or public/private key pairs.

Public Key Encryption

Public key encryption is a method of encrypting data that involves a pair of keys known as a public key and a private key (or a public/private key pair). The public key is usually published, and the corresponding private key is kept secret. Data that is encrypted with the public key can be decrypted only with the corresponding private key. [Figure 14-3](#) illustrates a very simplified way to understand public key encryption.

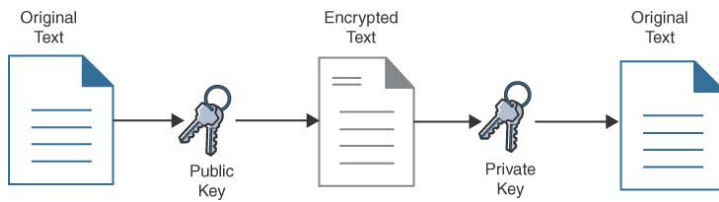


Figure 14-3 *Simple Public Key Encryption*

Public key encryption is used to establish secure communications over the Internet (via HTTPS). A website's SSL/TLS certificate, which is shared publicly, contains the public key, and the private key is installed on the web server.

A TLS handshake uses a public key to authenticate the identity of the origin server and to exchange data that is used for generating the session keys. A key exchange algorithm, such as Rivest–Shamir–Adleman (RSA), uses the public/private key pair to agree on session keys, which are used for symmetric encryption when the handshake is complete. Clients and servers can agree on new session keys for each communication session, so that bad actors are unable to decrypt communications even if they identify or steal one of the session keys.

Data Integrity (One-Way Hash)

As data and documents move across networks or storage devices, it is crucial to ensure that this data is identically maintained during any operation.

In the case of data transfer, data flowing between applications in a public network environment can flow across many network elements, each of which can “see” the data. Encryption ensures that even though these elements can see the data, they cannot understand the data. But because the data can flow across nodes that you do not control, there is a risk that a node in the network could alter the data before it reaches the destination.

You cannot prevent the data from being altered by someone on the network, so the receiving element (destination) must be able to detect whether data has been modified and, if so, not pass the corrupted data to the application. A one-way hash is used for this purpose. The message hash is a fingerprint of the data. If the data changes, the fingerprint (that is, the message digest or hash) changes as well. Someone who alters the data would have no idea what the corresponding digest will be for the modified data. The content of the hashed data cannot be determined from the hash. This is why it is called a *one-way* hash.

Digital Signatures

You can use a private key for encryption and your public key for decryption. Rather than encrypting the data itself, you can create a one-way hash of the data and then use the private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a *digital signature*. Figure 14-4 illustrates the use of a digital signature to validate the integrity of signed data. The data and the digital signature are sent across the network. On the receiving end, two hashes are calculated: one from the received data and the other by decrypting the digital signature using the public key. If the two hashes match, you know that the private and public keys match, and the identity of the sender is verified.

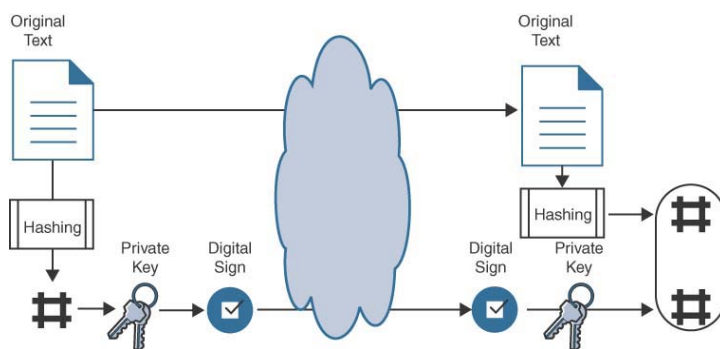


Figure 14-4 Digitally Signed Data

Data Security

Today, all our interactions with the world involve online systems in some way, and all these systems deal with data. Data is typically an organization's most valuable asset, and it is very often sensitive. Data security is the process of securing data and protecting it from unauthorized access. As data traverses various points of interaction, it needs to be secured at all these various places. Data needs to be protected in three places, as shown in [Figure 14-5](#):

- Network (data in motion)
- Storage (data at rest)
- Memory (data in use)

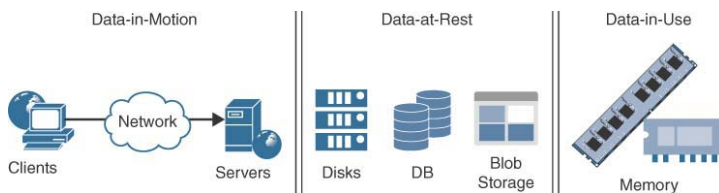


Figure 14-5 *Data Security*



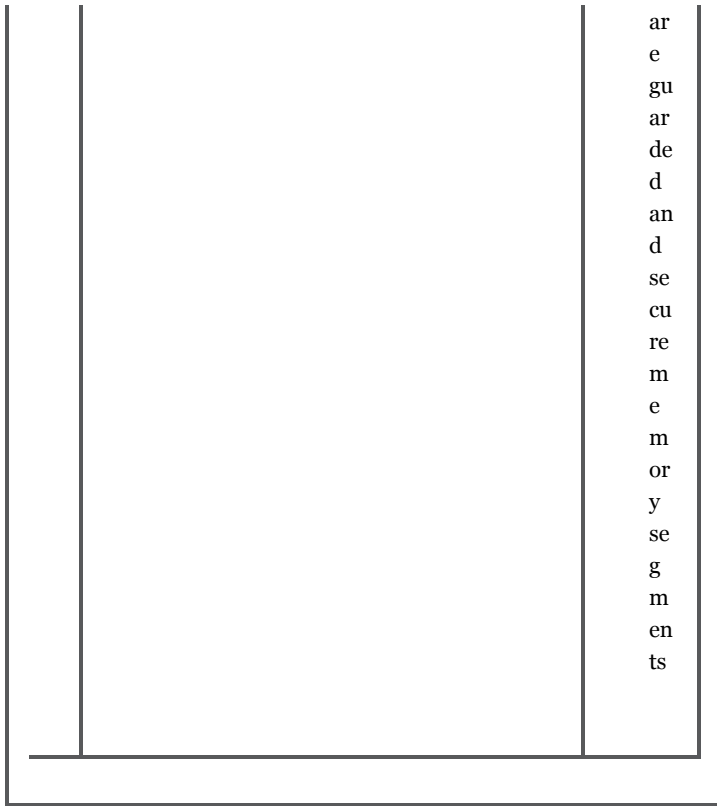
[Table 14-3](#) shows each of these types of security and describes some ways to secure or protect each type.

Table 14-3 Data Security Types

Data Type	Description	Examples
Data	Data travels across various networks and entities. It is prone to be inspected, and data can be stolen.	<ul style="list-style-type: none"> • H T T PS
in	Transport Layer Security (TLS) is a widely adopted security protocol designed to facilitate privacy and data security for communication over the Internet. A primary	

<p>m o t i o n (n e t w o r k)</p>	<p>use case of TLS is encrypting the communication between web applications and servers, such as web browsers loading a website.</p>	<ul style="list-style-type: none"> • Firewalls
<p>D a t a a t r e s t (s t o r a g e)</p>	<p>Any data that is stored can be accessed digitally or physically.</p> <p>Encrypting data at rest protects it against physical theft of the file system storage devices, protects against unauthorized access to data, and satisfies information security or regulatory requirements.</p>	<ul style="list-style-type: none"> • Full disk encryption • File system encryption • Database encryption

<p>D a t a i n u s e (m e m o r y)</p>	<p>When a running application is processing data, the data is in memory (or in use). All open files are considered data in use by operating systems. When an encrypted file is unencrypted in memory, it is vulnerable to being stolen, or “scraped.”</p>	<ul style="list-style-type: none"> • Full memory encryption • CPU-Base ded key storage (keys stored in CPU registers) • Encaves, which
---	---	---



Secure Development Methods

As mentioned earlier in this chapter, the application security process starts during the development phase. Instead of trying to bring in security at the end of the development process, secure development needs to be baked in from the start. Addressing security issues from the very beginning saves a company time and money in the long run.

It is a common practice for corporations to use some type of software development lifecycle (SDLC). Best practices in secure software development suggest integrating security aspects into each phase of the SDLC. Figure 14-6 show the various aspects of the SDLC.



Figure 14-6 *Secure Development*

As you can see in the figure, the SDLC includes these steps:

- **Training:** Training helps get everyone on the project teams into a security frame of mind. Teach and train developers on the team to analyze the business application attack surface as well as the associated potential threats. Not just developers but all team members should understand the exposure points of their applications (user inputs, front-facing code, exposed function calls, and so on) and take steps to design more secure systems wherever possible.
- **Threat modeling:** For every component and module in a system, ask the “what-how-what” questions: What can go wrong? How can someone try to hack into it? What can we do to prevent this from happening? Various frameworks for threat modeling are available, including the following:
 - STRIDE (Spoofing, Tampering, Repudiation, Information Leak, DoS, Elevation of Privilege)
 - PASTA (Process for Attack Simulation and Threat Analysis)
 - VAST (Visual, Agile, and Simple Threat Modeling)
- **Secure coding:** Build and use code libraries that are already secured or approved by an official committee. The industry has several guidelines for secure coding, and we have listed some of the standard ones here:
 - Validating inputs

- Encoding output
- Ensuring authentication and credential management
- Managing sessions
- Using access control lists
- Monitoring error handling and logging
- Protecting data, including files, databases, and memory
- **Code review:** Code review is one of the most essential steps in securing an application. Usually, the rule that you have to keep in mind is that pen testing and other forms of testing should not be discovering new vulnerabilities. Code review has to be a way to make sure that an application is self-defending. Also, it should be conducted using a combination of tools and human effort. It is important to designate a security lead who can help review code from a security point of view.
- **Secure tooling:** Static analysis helps catch vulnerabilities. Static analysis tools detect errors or potential errors in the structure of a program and can be useful for documentation or understanding a program. Static analysis is a very cost-effective way of discovering errors. Data flow analysis is a form of static analysis that concentrates on the use of data by programs and detects some data flow anomalies.

Dlint (see <https://github.com/duo-labs/dlint>) is a tool from Duo Labs (Cisco) that defines and checks for common best practices when it comes to writing secure Python. To evaluate a variety of rules over a code base, Dlint leverages Flake8. Flake8 does the heavy lifting of parsing Python's AST, allowing you to focus on writing robust rule sets.

- **Testing:** Testing includes penetration (pen) testing and system testing, black box testing, and white box testing. Black box testing is a method used to test software without knowing the internal structure of the code or program. Testing teams usually do this type of testing, and programming knowledge is not required. Black box testing includes functional testing, behavior testing, and closed box testing. White box testing, on the other hand, is a software testing method in which internal structure is known to the tester who is testing the software. Generally, this type of testing is carried out by software developers. Programming knowledge is usually required for white box testing. White box testing includes structural testing, logic testing, path testing, loop testing, code coverage testing, and open box testing.

Testing involves the following steps:

- **Intelligence gathering:** Define the goals, understand what's included in the application, and identify potential areas of vulnerabilities.
- **Scanning:** Understand both running and non-running behaviors. Static analysis tools enable developers and testers to see faults without actually running an application. These tools can save a lot of time and effort in the long run, and the more errors and defects found here, the better. Dynamic analysis, on the other hand, involves actually running the application in a real or virtual environment. Usually, a lot of external services and interactions are exercised here.

- **Access:** Use various methods to try to hack the application, such as testing the app for SQL injection, back doors, traffic interception, and so on. Long-term access testing looks at the kinds of vulnerabilities exposed when a system is exploited for a long time.
- **Reporting:** Include all details on the vulnerabilities and sensitive data exposed, as well as the amount of time the system remained unhacked.

Securing Network Devices

Network devices are the components of a network that transport communications needed for data, applications, services, and multimedia. These devices include routers, firewalls, switches, servers, load balancers, intrusion detection systems, Domain Name System servers, and storage area networks. These devices are ideal targets for malicious actors because most or all organizational and customer traffic must pass through them.

Firewalls

A firewall is a barrier that's put in place to limit damage. A firewall can be either hardware or software that's installed to limit damage from external and internal cyberattacks. It monitors and controls network traffic based on a set of security rules. [Figure 14-7](#) shows how firewalls can monitor and control network traffic as it flows between a local-area network (LAN) and the Internet (WAN).



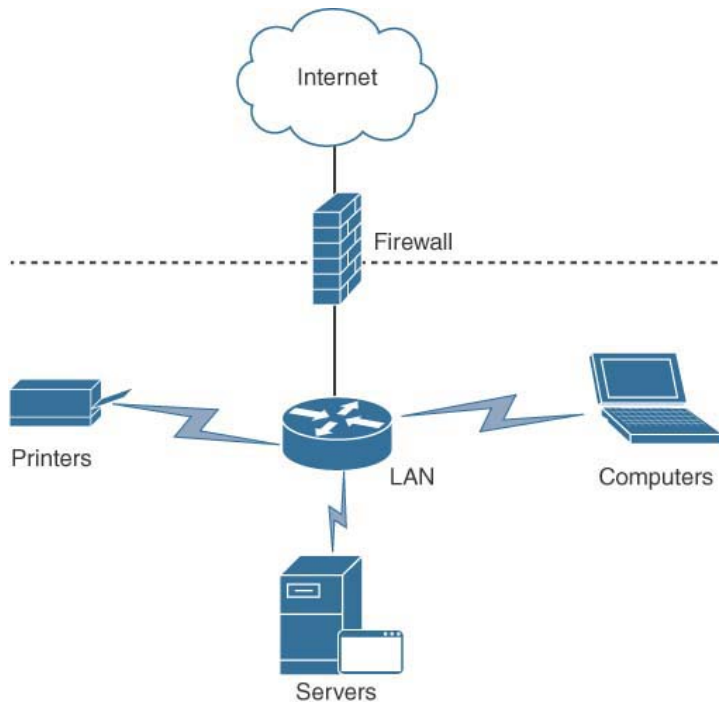


Figure 14-7 Firewall

The primary function of all firewalls is to screen network traffic and prevent unauthorized access between two network entities. There are several types of firewalls:

- **Packet filtering firewalls:** Individual packets are examined, although the firewall does not know the contents of a packet. These firewalls provide this security by filtering the packets of incoming traffic and distinguishing between TCP/UDP traffic and port numbers. The packets are either allowed entry onto the network or denied access, based on either their source or destination address or some other static information, such as the traffic type. [Figure 14-8](#) shows an example of a stateless firewall that has a very simple rules engine and shows which traffic is allowed and which is denied.

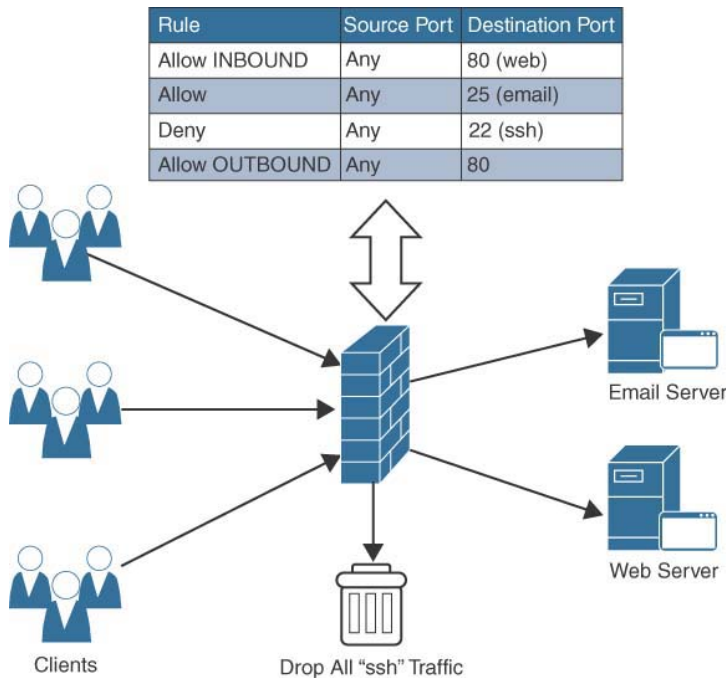


Figure 14-8 *Stateless Firewall (Packet Filtering)*

- Stateful inspection firewalls:** Packets are examined with other packets in the flow. Such firewalls monitor the state of active connections and use this information to determine which network packets to allow. Stateful firewalls are advanced compared to stateless packet filtering firewalls. They continuously keep track of the state of the network and the active connections it has, such as TCP streams or User Datagram Protocol (UDP) communication. The ability to acknowledge and use the contents of incoming traffic and data packets is one of the principal advantages of stateful firewalls, as it enables these firewalls to tell the difference between legitimate and malicious traffic or packets. This ultimately makes stateful firewalls one of the most powerful security tools in modern policies that protect network connections through the implementation of additional security procedures for new or ongoing/active links. Figure 14-9 shows an example of a stateful firewall that keeps track of the data from User1 and allows it to flow to the email and web server.

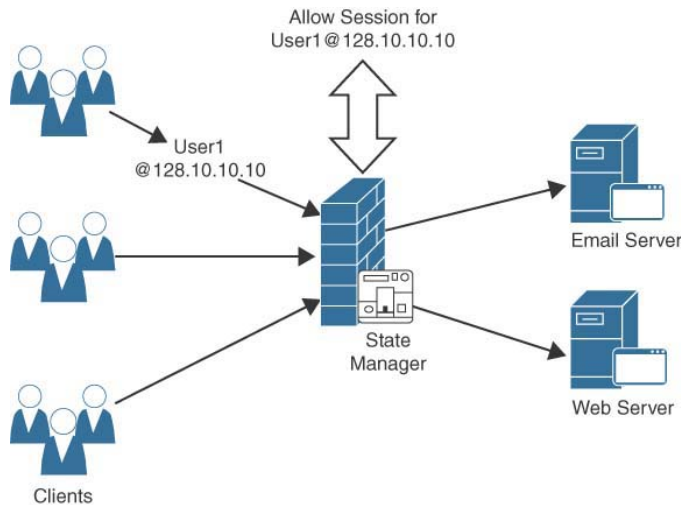


Figure 14-9 *Stateful Firewall (Context Aware)*

- Application-level or proxy firewall:** This type of firewall protects network resources by filtering messages at the application layer. In addition to determining which traffic is allowed and which is denied, a proxy firewall uses stateful inspection and deep packet inspection to analyze incoming traffic for signs of attack. The key benefit of application-layer filtering is the ability to block specific content, such as known malware or content from individual websites. A proxy firewall can recognize when particular applications and protocols, such as Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Domain Name System (DNS), are being misused. As the firewall sees incoming packets, it inspects each protocol in the stack, noting the various states. At Layer 7 (the application layer), the firewall looks up the rules and applies them to incoming packets. Based on the system rules, it may perform other functions, such as URL filtering, data modification, logging, and object caching.
- Next-generation firewall:** Some standard features of next-generation firewall architectures include deep-packet inspection (checking the actual contents of the data packet), TCP handshake checks, and surface-level packet inspection. Next-generation firewalls may include other technologies as well, such as intrusion prevention to stop attacks against a network automatically.

Intrusion Detection Systems (IDSs)

An intrusion detection system (IDS), as shown in [Figure 14-10](#), is a passive system that monitors network traffic. The traffic in this case is a copy of the traffic as the network sees it. In a typical network scenario, packets or flows are replicated in hardware and sent to the IDS. The IDS processes the packets and detects malicious signatures. In this case, the IDS cannot detect any

suspicious activity in real time; it has to let a few packets pass before it can initiate any action. An IDS usually has to rely on other networking elements, such as routers or switches, to take any corrective measures to stop suspicious traffic from passing through. One of the most significant disadvantages of an IDS is that a single packet attack is almost always successful, which means it is hard to prevent these attacks.

Key Topic

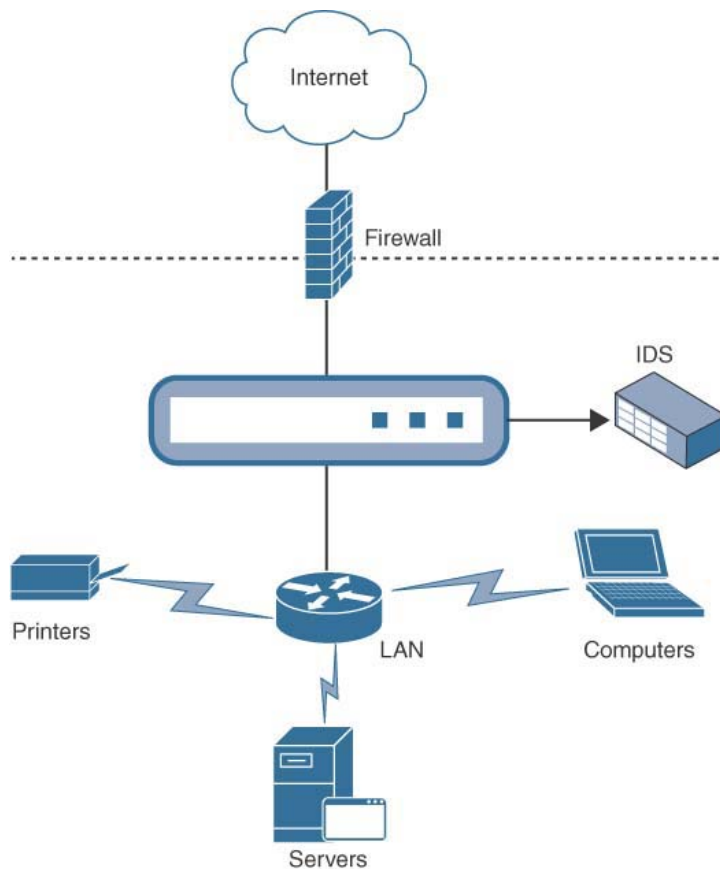


Figure 14-10 *IDS Receiving a Copy of the Packets*

Intrusion Prevention Systems (IPSs)

An intrusion prevention system (IPS), as shown in [Figure 14-11](#), works in real time and in an inline manner. Usually, an IPS is collocated with a network element

such as a router or switch. Like an IDS, an IPS does deep-packet inspection of packets; however, an IPS takes action immediately if it determines that a packet is vulnerable. Therefore, only trusted packets are allowed into the enterprise network.

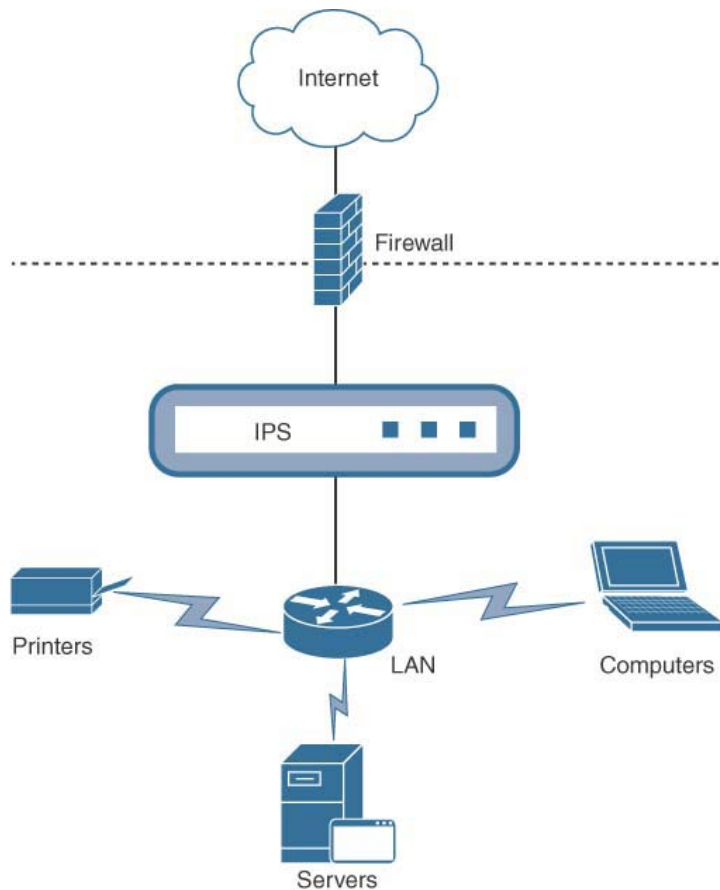


Figure 14-11 *IPS Inline with Traffic*

Domain Name System (DNS)

You probably don't remember a lot of phone numbers but instead record them in your contacts so that you can look up a name of a person you want to call. Domain Name System works quite similarly: It keeps a mapping of domain names and the IP addresses of the servers where the domains can be reached. Each device on the Internet has a unique IP address, and DNS eliminates the need to memorize any IP addresses that it uses.

Figure 14-12 shows how a client request gets to the DNS resolver, which in turn gets to the primary DNS server.

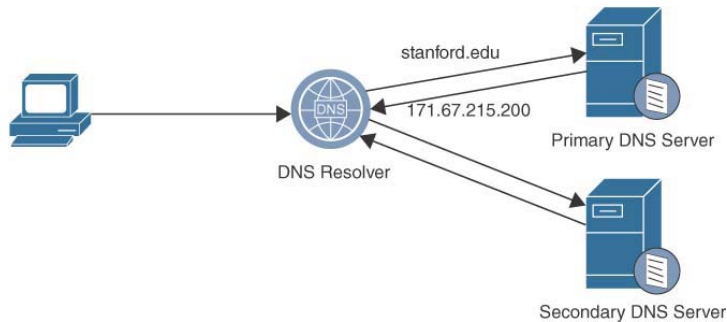


Figure 14-12 *DNS Operation*

To understand how DNS works, the following steps walk through how a client, such as a web browser, asks for a new website:

- Step 1.** The user clicks on a link or types a URL for a site to visit (for example, <https://developer.cisco.com>).
- Step 2.** The browser looks in the local cache to see if the name can be resolved locally on the client.
- Step 3.** The browser sends a query to the DNS recursive resolver in an attempt to resolve the name. The DNS recursive resolver usually resides with the Internet service provider, and it can reply if it has the IP address or pass on the query to the next resolver in the chain. (There are public name resolvers, such as Cisco Umbrella and Google.) Eventually, one of the resolvers may respond with the IP address.
- Step 4.** If the DNS resolvers cannot find the IP address, the browser tries the root name servers, which in turn forward the query to top-

level domain (TLD) servers and eventually authoritative servers, until results are achieved.

Step 5. When the client has the IP address for the domain, it gives the IP address to the browser.

Nslookup (Name Server Lookup) is an excellent utility that most operating systems can use to get details about a particular host or domain from a DNS server. The syntax for the command is as follows:

[Click here to view code image](#)

```
nslookup [-option] [name | -] [server]
```

[Examples 14-4](#) and [14-5](#) show examples of using Nslookup.

Example 14-4 *Using Nslookup for a Simple Host Lookup*

[Click here to view code image](#)

```
nslookup stanford.edu :
nslookup with a simple domain name will return
the IP address of the stanford.edu

$ nslookup standford.edu
Server: 2601:647:5500:1ea:9610:3eff:fe18:22a5
Address:
2601:647:5500:1ea:9610:3eff:fe18:22a5#53

** server can't find standford.edu: NXDOMAIN

$ nslookup stanford.edu
Server: 2601:647:5500:1ea:9610:3eff:fe18:22a5
Address:
2601:647:5500:1ea:9610:3eff:fe18:22a5#53

Non-authoritative answer:
Name: stanford.edu
Address: 171.67.215.200
```

Example 14-5 *Using Nslookup to Get Information About a Domain*

[Click here to view code image](#)

```
nslookup -type=any stanford.edu :  
The -type=any parameter allows us to get all  
the DNS records for that domain,  
including the mail servers, etc.
```

```
$ nslookup -type=any stanford.edu  
;; Truncated, retrying in TCP mode.  
Server: 10.168.243.90  
Address: 10.168.243.90#53
```

```
Non-authoritative answer:  
stanford.edu mail exchanger = 10 mxa-  
00000d03.gslb.pphosted.com.  
stanford.edu mail exchanger = 10 mxb-  
00000d03.gslb.pphosted.com.  
stanford.edu nameserver = ns5.dnsmadeeasy.com.  
stanford.edu nameserver = ns6.dnsmadeeasy.com.  
stanford.edu nameserver = ns7.dnsmadeeasy.com.  
stanford.edu nameserver = argus.stanford.edu.  
stanford.edu nameserver =  
atalante.stanford.edu.  
stanford.edu nameserver =  
avallone.stanford.edu.  
stanford.edu  
origin = argus.stanford.edu  
mail addr = hostmaster.stanford.edu  
serial = 2019138199  
refresh = 1200  
retry = 600  
expire = 1296000  
minimum = 1800
```

Load Balancing

Load balancing is the process of distributing user requests across multiple servers. The goal is to ensure that no single server bears too much demand. In essence, load balancers are like virtual servers, receiving all user requests and forwarding these requests based on the load balancer's policy to one of the servers that host the website. The load balancer attempts to ensure that each server receives a similar number of requests. Many load balancers are capable of monitoring servers and compensating for any servers that become unavailable. By spreading the work evenly, load balancing improves application responsiveness.

Figure 14-13 shows how requests coming in from various clients can be load balanced (or distributed) to various servers.

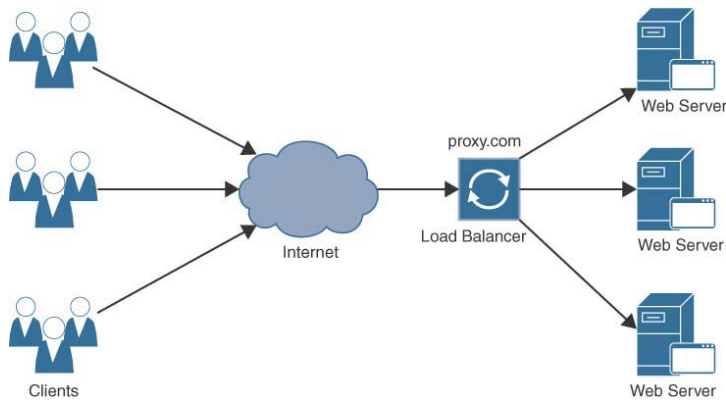


Figure 14-13 *Load Balancing*

Load balancing uses algorithms such as the following:

- **Round-robin:** Selects servers in turn
- **Least connected:** Selects the server with the lowest number of connections; this is recommended for more extended sessions
- **Source/IP-hash:** Chooses a server based on a hash of the source IP
- **Cookie marking:** Adds a field in the HTTP cookies, which could be used for decision making
- **Consistent IP-hash:** Adds and removes servers without alarming cached items or session persistence

A reverse proxy accepts a request from a user, forwards it to a server that can fulfill it, and returns the server's response to the client. A reverse proxy can include some or all of the following functionality:



- **Security:** The web servers or application servers are not visible from the external network, so malicious clients cannot access them directly to exploit any vulnerabilities. Many reverse proxy servers include features that help protect backend servers from distributed denial-of-service (DDoS) attacks—for example, by rejecting traffic from particular client IP addresses (blacklisting) or limiting the number of connections accepted from each client.

- **Scalability and flexibility:** Clients see only the reverse proxy's IP address. This is particularly useful in a load-balanced environment, where you can scale the number of servers up and down to match fluctuations in traffic volume.
- **Web acceleration:** Acceleration in this case means reducing the time it takes to generate a response and return it to the client. Some of the techniques for web acceleration include the following:
 - **Compression:** Compressing server responses before returning them to the client (for instance, with gzip) reduces the amount of bandwidth they require, which speeds their transit over the network.
 - **SSL termination:** Encrypting the traffic between clients and servers protects it as it crosses a public network such as the Internet. However, decryption and encryption can be computationally expensive. By decrypting incoming requests and encrypting server responses, the reverse proxy frees up resources on backend servers, which the servers can then devote to their primary purpose—serving content.
 - **Caching:** Before returning the backend server's response to the client, the reverse proxy stores a copy of it locally. When the client (or any other client) makes the same request, the reverse proxy can provide the response itself from the cache instead of forwarding the request to the backend server. This both decreases response time to the client and reduces the load on the backend server. This works great for “static” content, but there are new techniques that can be used for “dynamic” content as well.
- **Content filtering:** This involves monitoring traffic to and from the web server for potentially sensitive or inappropriate data and taking action as necessary.
- **Authentication:** The reverse proxy authenticates users via a variety of mechanisms and controls access to URLs hosted on the web server.

Figure 14-14 shows how requests come in from various clients. The reverse proxy can terminate an SSL connection or even inspect incoming traffic.

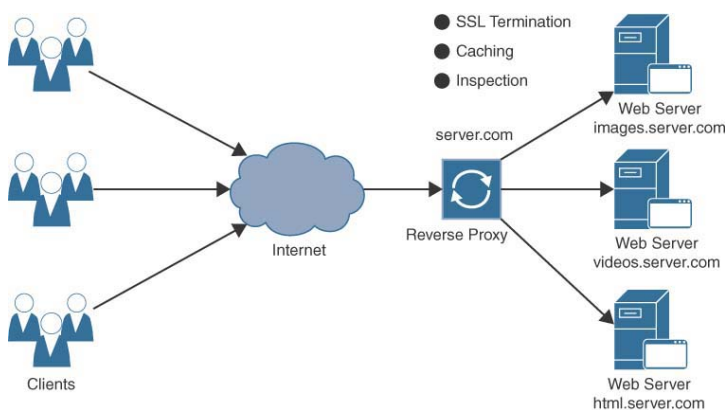


Figure 14-14 *Reverse Proxy*

Figure 14-15 illustrates the concept of a reverse proxy with a user requesting a web page from <https://server.com>. In this case, server.com is the reverse proxy, which first terminates SSL and then translates the request and issues a new request to the images.server.com server. The images server then responds to the request, gathering and translating the response and sending it back to the client. The translation could mean applying compression (gzip) to the response.

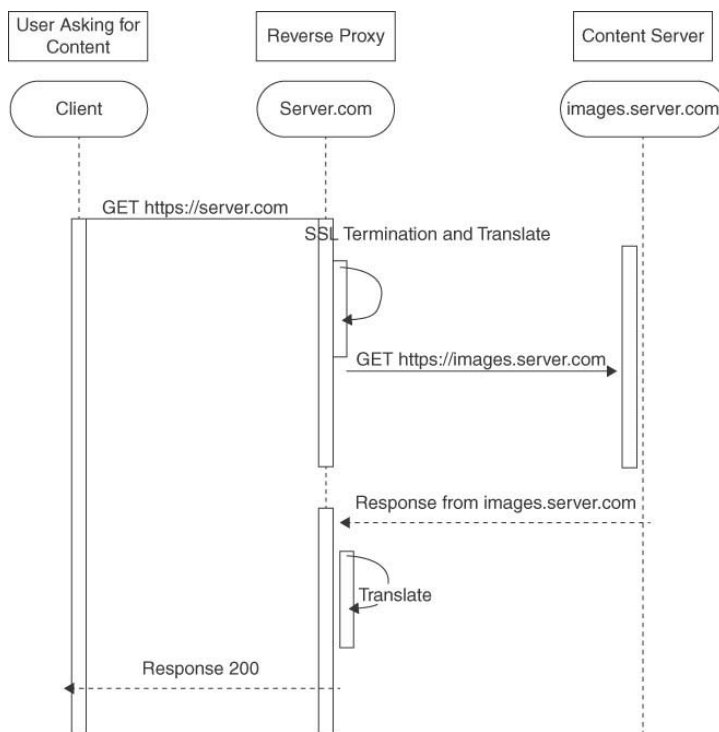


Figure 14-15 *Reverse Proxy Flow Diagram*

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “Final

Preparation," and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 14-4](#) lists these key topics and the page number on which each is found.

Table 14-4 Key Topics



Key Topic	Element	Description	Page
List		Assets, threats, vulnerabilities, and risks	<u>42</u> 3
Table 14-2		Threats and mitigation	<u>42</u> 3
Paragraph		Open Web Application Security Project (OWASP)	<u>42</u> 5
Paragraph		Nmap	<u>42</u> 6
Figure 14-5		Data security	<u>43</u> 3
Paragraph		Firewalls	<u>43</u> 7
Paragraph		Intrusion detection system	<u>43</u> 9
Paragraph		Domain Name System	<u>44</u> 0

List	Reverse proxy	44 4
------	---------------	---------

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Secure Sockets Layer (SSL)

Transport Layer Security (TLS)

Open Web Application Security Project (OWASP)

OWASP Top 10

Nmap

multifactor authentication (MFA)

cryptography

Rivest–Shamir–Adleman (RSA)

software development lifecycle (SDLC)

Dlint

Domain Name System (DNS)

intrusion detection system (IDS)

intrusion prevention system (IPS)

Domain Name System (DNS)

Chapter 15

Infrastructure Automation

This chapter covers the following topics:

- **Controller Versus Device-Level Management:** This section compares and contrasts the two network device management options currently available: controller-based and device-level management.
- **Infrastructure as Code:** This section introduces the concepts of infrastructure as code.
- **Continuous Integration/Continuous Delivery Pipelines:** This section examines CI/CD pipelines, why they are used, the problems they address, and how they can be applied to infrastructure automation.
- **Automation Tools:** This section covers popular automation tools that are used for configuration management and network automation.
- **Cisco Network Services Orchestrator (NSO):** This section provides an introduction to Cisco NSO.
- **Cisco Modeling Labs/Cisco Virtual Internet Routing Laboratory (CML/VIRL):** Cisco Modeling Labs (CML) is the new name for VIRL and we will use it interchangeably (CML/VIRL). This section explains what Cisco CML/VIRL is and how it can be used to build network simulations.
- **Python Automated Test System (pyATS):** This section examines pyATS, including the components that make up the testing solution and how it can be used to automate network testing.

This chapter covers infrastructure automation concepts. It starts with a comparison of management of infrastructure on a device-by-device basis or using a central controller. Thanks to virtual devices and public cloud technologies, infrastructure can be created, modified, and decommissioned dynamically by using the concept of infrastructure as code. This chapter introduces infrastructure as code and the problems it solves. This chapter also covers continuous integration/continuous delivery (CI/CD) pipelines and their importance in the software development lifecycle.

The chapter also covers automation tools and introduces the popular configuration management tools Ansible, Puppet, and Chef. This chapter discusses Ansible concepts such as control nodes, playbooks, and modules; Puppet concepts such as Puppet Master, manifests, and Puppet modules; and Chef concepts such as Chef Infra Server and cookbooks. This chapter also covers Cisco NSO, including the architecture of the platform, common use cases, the network simulation feature, and using tools such as **curl** and Postman. Finally, the chapter provides overviews of Cisco CML/VIRL and pyATS.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 15-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 15-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Controller Versus Device-Level Management	1, 2
Infrastructure as Code	3, 4
Continuous Integration/Continuous Delivery Pipelines	5, 6

Automation Tools	7 – 9
Cisco Network Services Orchestrator (NSO)	10
Cisco Modeling Labs/Virtual Internet Routing Laboratory (CML/VIRL)	11
Python Automated Test System (pyATS)	12

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

- 1.** What protocols were historically used to manage network devices? (Choose two.)
 1. SMTP
 2. SSH
 3. TFTP
 4. Telnet
 5. SNMP

- 2.** Which of the following is an example of a central network controller?
 1. Nagios
 2. Grafana
 3. Cisco DNA Center
 4. Prometheus

- 3.** What are two common approaches to infrastructure as code?
 1. Subjective
 2. Declarative

3. Imperative
4. Objective
4. Which of the following are issues with deploying infrastructure that are being addressed with infrastructure as code? (Choose two.)
 1. The amount of time needed to deploy new infrastructure
 2. Maintaining code across different versions
 3. Troubleshooting issues
 4. Lack of code documentation
5. What is a key component of a continuous integration pipeline?
 1. Integration server
 2. Delivery server
 3. File server
 4. Build server
6. What are the four steps in a CI/CD pipeline?
 1. Code, test, build, deploy
 2. Source, build, test, deploy
 3. Source, test, deploy, monetize
 4. Build, test, deploy, monetize
7. What command is used to run Ansible playbooks?
 1. **ansible**
 2. **ansible-playbook**
 3. **ansible-playbooks**
 4. **ansible-plays**
8. What language are Puppet manifests written in?
 1. Puppet Domain Specific Language
 2. Puppet Python
 3. YAML
 4. Java
9. What language are Chef recipes written in?
 1. Python
 2. Java
 3. Ruby
 4. Go
10. What Cisco NSO component manages the YANG data models that get sent to a device?
 1. Service Manager
 2. Device Manager
 3. Core Engine
 4. CDB

11. What language are Cisco CML/VIRL topology files written in?

1. JSON
2. CSV
3. YAML
4. XML

12. What are the two main components of pyATS?

1. pyATS test framework
2. pyATS repo
3. pyATS testbed
4. pyATS library

FOUNDATION TOPICS

CONTROLLER VERSUS DEVICE-LEVEL MANAGEMENT

Historically, network devices were managed through command-line interfaces (CLIs) by using protocols such as Telnet and Secure Shell (SSH). The network administrator would connect to the CLI of a network device using a software client that implements these protocols and perform the configuration changes needed for the device to function as expected. On a small- to medium-sized network, this approach might still work, but as networks keep growing bigger and more and more devices become connected, managing networks in a device-by-device manner becomes time-consuming and prone to errors. Combine this with the needs for automation, network programmability, and reductions in operational costs, and you can see that a new way of managing networks is needed.

Network controllers are a possible solution to the challenges of managing networks in a device-by-device manner. A network controller is a centralized software platform dedicated to managing the configuration and operational data of network devices. Controllers take over the management of network devices, meaning that

all interactions with the devices have to go through the controller. They provide an abstraction layer between the administrators of the network and network devices. Network controllers usually expose a northbound REST API interface for management and integration with third-party systems and one or several southbound interfaces through which they connect and manage the network devices. Typical southbound interfaces are CLI-based interfaces that use Telnet and SSH, OpenFlow, SNMP, NETCONF, RESTCONF, and so on. In most situations, the network controller manages different network devices running different network operating systems with different versions and different features; by abstracting the underlying network capabilities, the controller exposes only a subset of the network functions. This minimal loss of network functionality is an assumed risk when choosing to go the network controller route for managing a network.



On one side is the controller-based management of networks supporting a subset of network functions, and on the other side is direct access to the devices with full access to all network features. Network administrators have to choose between these two options, keeping in mind the requirements of the network and which option works best for each situation.

There are several other considerations to keep in mind when integrating controllers into a network. Support for atomic network configuration is one of them. Atomicity in the case of network configuration means that either the intended configuration has been applied to all network elements without error or, to deal with potential errors, there is a rollback mechanism in place to ensure that the partial configuration is removed and the devices are brought back to the initial state before there is any

attempt to change the configuration. Atomicity is very important to ensure a holistic and uniform view of the network and to make sure that partial and incomplete configurations do not negatively affect the functionality of the network.

Another consideration with network controllers involves auditing and managing the configuration drift of the network elements. Configuration drift happens when the configuration of a network device has changed from its intended state, which means the configuration of the device in the network controller is different from the configuration of the device itself. This situation can easily lead to inconsistencies and unexpected network behavior. A good controller has built-in mechanisms to address configuration drift; in some instances these mechanisms go as far as limiting or even blocking device configuration changes.

The main advantage provided by network controllers is that the abstraction and central administration of individual network devices eliminate the need for device-by-device configuration, management, and monitoring, as shown in [Figure 15-1](#).

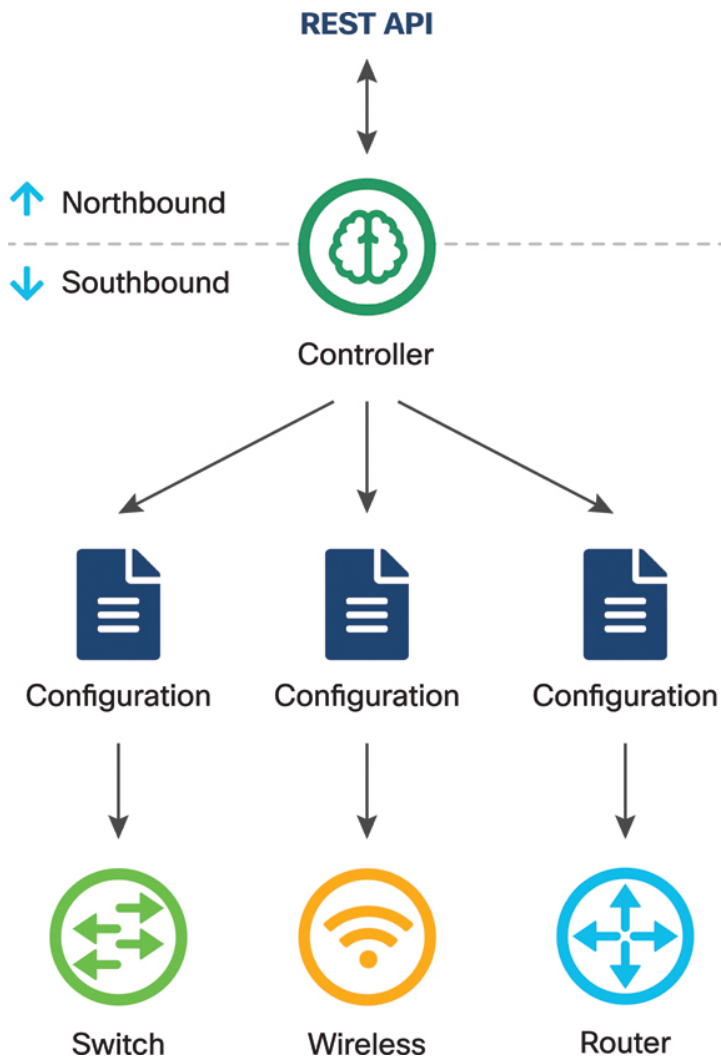


Figure 15-1 *Network Managed Through a Network Controller*

Direct device configuration allows access to the full feature set of a device, but it can be time-consuming and prone to errors—especially in large networks. The scale of a network is certainly a limiting factor when performing configuration and management tasks on a device-by-device basis. As you’ve seen in [Chapter 12](#), “[Model-Driven Programmability](#),” historically, there has been an evolution from custom CLIs to standard data models and network configuration protocols such as NETCONF and RESTCONF. Having custom-built protocols for network configuration as well as standard

data models is definitely helping with the network automation efforts and the scale limitations. Configuration changes made on a single device can be replicated across an entire network. Device-by-device configuration is illustrated in [Figure 15-2](#).

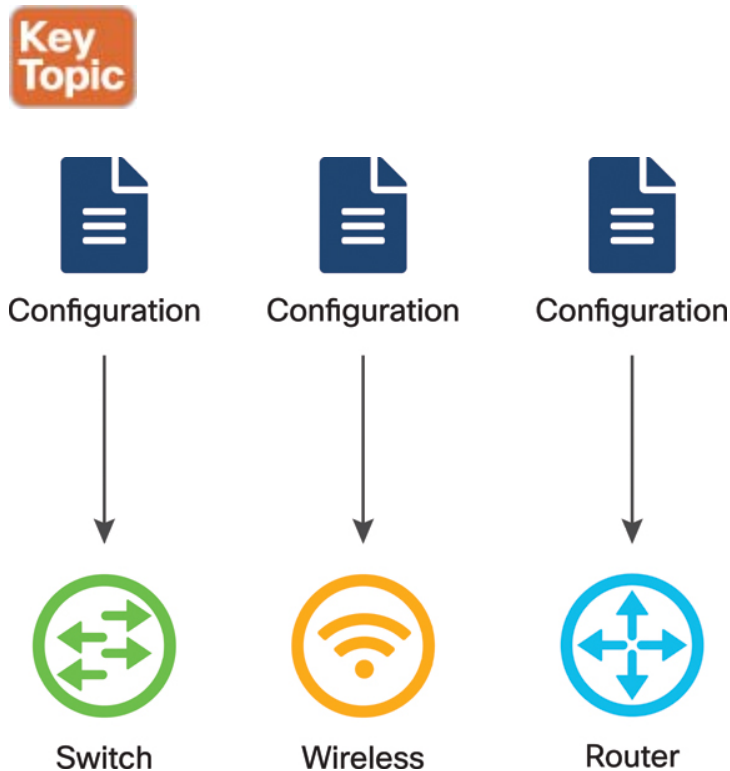


Figure 15-2 *Network Managed on a Device-by-Device Basis*

Using a controller for network management and device-by-device configuration are not mutually exclusive. A controller can be used to globally configure devices, while direct connection to devices can be helpful for monitoring and ensuring that the changes made by the controller do not create undesired behavior.

[Chapter 8, “Cisco Enterprise Networking Management Platforms and APIs,”](#) provides an example of a network controller with Cisco DNA Center. Cisco DNA Center can be used to completely configure, manage, and monitor networks. It can also be used to deploy SD-Access

fabrics. It is possible to deploy SD-Access by using automation tools and connecting to each device using NETCONF, the CLI, or even SNMP. This gives the network administrator more granular control but at the same time requires more work and time to define configuration parameters and data models and to ensure that the correct configuration gets applied to all the devices in a unified manner. Cisco DNA Center performs all these tasks automatically, and automation scripts can be used to collect operational data and ensure that the network performs within optimal parameters.

INFRASTRUCTURE AS CODE

Inspired by software development practices, infrastructure as code is a new approach to infrastructure automation that focuses on consistent, repeatable steps for provisioning, configuring, and managing infrastructure. For a long time, infrastructure has been provisioned and configured in a manual fashion. Deploying a new application on a network used to take anywhere from weeks to months to even years in some situations. During that time, the IT team would try to size the hardware requirements for the new application, go through the approval process to purchase the new hardware, order the hardware, wait for it to be delivered, rack it up, and start the provisioning and configuration steps. The provisioning and configuration of the new hardware was in most cases a manual process: installing the operating system one step at a time following guided instructions and then finally installing and configuring the new application. At best, technologies like PXE booting and installation scripts would be used to try to automate the arduous process of provisioning new hardware and installing new applications. This resulted in one-of-a-kind, or “snowflake,” environments in which each application ran on different hardware, with different options and features enabled; supporting and maintaining such

environments was incredibly challenging. A new paradigm for deploying and managing infrastructure was needed.

The IT infrastructure of the past had a number of limitations, including high deployment and maintenance costs, long waiting times for new infrastructure to be ready for production (slowing down new application deployments and hampering innovation and fast prototyping), and difficulty in troubleshooting since each environment was slightly different. Several technologies have evolved to address these limitations. The one that has had the biggest impact is public cloud computing. Besides making infrastructure available almost instantaneously and reducing the costs associated with purchasing hardware, cloud computing has also changed the way IT infrastructure is being provisioned, configured, and consumed. With the advent of APIs, public cloud infrastructure can more easily be automated, and cookie-cutter templates can be created for easy and fast deployment of new infrastructure.

Infrastructure as code is all about the representation of infrastructure through machine-readable files that can be reproduced for an unlimited amount of time. It is a common practice to store these files in a version control system like Git and then use them to instantiate new infrastructure—whether servers, virtual machines, or network devices. In this way, infrastructure can be treated as source code, with versioning, history, and easy rollback to previous versions, if needed. Being able to spin up new infrastructure in a repeatable, consistent fashion becomes extremely useful, for example, in cases in which there is a spike in traffic and in the number of application requests for a certain period of time. In such cases, new servers, load balancers, firewalls, switches, and so on can be dynamically provisioned within seconds to address the additional load. Once the traffic subsides and the need for higher capacity subsides, the

infrastructure can be scaled down by dynamically decommissioning servers, load balancers, and so on to save on costs. This *elasticity* of the infrastructure is another of the benefits of defining all infrastructure as code.



By defining infrastructure through code, the infrastructure becomes transparent in the sense that it is enough to read the code in the document to know all the characteristics and features that will be instantiated when it gets provisioned. Other people can review the code, make improvements, have the changes tracked through version control, and make sure the code is in compliance with the security requirements of the company.

While change was frowned upon in the infrastructure of the past, and a limited number of strict change windows were provided through the year, with infrastructure as code, change is welcomed and even desired. With infrastructure as code, change is a catalyst to improve the reliability and the performance of the infrastructure as a whole. Just as source code goes through multiple versions and becomes better with each new release, infrastructure becomes more resilient and reliable with each new version.

By using infrastructure as code, it is possible to have identical environments for testing, integration, and production. This leads to significantly fewer troubleshooting steps needed to repair and maintain the infrastructure and the applications running on top of it.

There are usually two types of approaches to infrastructure as code:



- **Declarative:** With the declarative approach, the desired state of the system is defined and then the system executes all the steps that need to happen in order to attain the desired state.
- **Imperative:** The imperative approach defines a set of commands that have to be executed in a certain order for the system to achieve the desired state.

A popular infrastructure as code solution is Terraform from HashiCorp. Infrastructure as code integrates very well in the continuous integration/continuous delivery pipelines discussed next.

CONTINUOUS INTEGRATION/CONTINUOUS DELIVERY PIPELINES

Software is quickly becoming pervasive in all aspects of our lives. From smartphones to smart cars and smart homes, we interact with software hundreds or even thousands of times each day. Under the DevOps umbrella, there have been several efforts to improve software development processes in order to increase the speed, reliability, and accuracy of software development. *Continuous integration/continuous delivery (CI/CD)* pipelines address all these requirements and more. All the software development processes—from writing code to building, testing, and deploying—were manually performed for years, but CI/CD pipelines can be used to automate these steps. Based on specific requirements for each company, different tools and different solutions are used to implement these pipelines. Some companies implement only continuous integration solutions, and others take advantage of the whole CI/CD pipeline, automating their entire software development process.

CI/CD is a series of automated steps that code goes through from the IDE of the individual developer, through building, testing, and finally deployment to

staging and production environments. Although usually used together, continuous integration and continuous delivery are two separate concepts that can be addressed separately.



Continuous integration is a software development practice in which developers commit their code to a central repository that is part of a version control system on an hourly, daily, or weekly basis or at some other frequency that is agreed upon within the team. As developers commit their code to the central repository, it is important to verify that the code integrates well with the full code base.

As new code is written, it is critically important to run code base tests against the new changes in the code base in a fast, automated fashion. Continuous integration tools accomplish exactly this task. As the code gets committed to a central repository—in most cases, to a Git-style version control system—a webhook gets triggered that starts the whole automated testing phase of the pipeline. Common triggers at this stage include automatic integrations with tools such as Cisco Webex Teams for immediate notifications on the status of the tests. In the event that a test fails, the developer is notified and can correct the issues that caused the test to fail very early in the development process. If all the tests pass, the code moves on to the next stage of the pipeline.

A key component of any CI pipeline is the build server. The role of the build server is to react to developers committing their code to the central repository and to start the initial tests on the new code features. Most version control systems support webhook mechanisms to automatically notify the build server when pull requests are opened and code is committed. Popular build servers

include Jenkins, Travis CI, and Drone CI. [Table 15-2](#) compares development environments that have implemented CI pipelines and the ones that have not.



Table 15-2 Development Environments With and Without CI

Development Without CI	Development With CI
Increased number of bugs	Fewer bugs
Insufficient testing	Automated builds, tests, documentation, and reports
Few releases per year	Multiple releases per day
Lack of integration testing	Dedicated build and test servers
Project delays	Frequent commits
Fewer features	More features
Instability	Stability

Continuous delivery adds on top of continuous integration all the remaining steps needed to automate the entire software release cycle, from building to testing to deployment (see [Figure 15-3](#)).

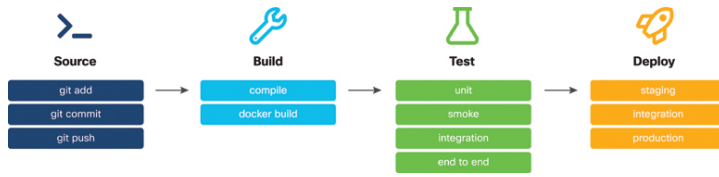


Figure 15-3 Complete CI/CD Pipeline

In the building phase, the code is compiled for languages such as Java, C/C++, or Go; for code written in interpreted languages such as Ruby or Python, the compilation step can be ignored. Regardless of the language the code is written in, in most cases, the output of the building phase is a Docker image that contains all the code, all required dependencies, and startup scripts to ensure that the application runs consistently, whatever the destination environment might be.

During the testing phase, automated tests are run to ensure correct behavior of the product. In an effort to catch software bugs as soon as possible, tests are developed at this stage to make sure the new code follows the expected requirements. Several types of tests are run at this stage, from unit and smoke tests that perform quick sanity checks, to integration, code coverage, code standards, and end-to-end tests that try to mimic the way users interact with the software as closely as possible. Developers follow the results of the tests, and if any of the tests fail, the developers can immediately address the problems and fix the code and restart the pipeline until all tests pass.

The next phase in the CI/CD pipeline is the deployment phase. By this stage, the software has been packaged and tested and is ready to be deployed for consumption. Before containers, in most cases, this meant installing the new binaries in new servers, either physical or virtual, and reconfiguring the load balancers to point to the new servers and monitor the environment to ensure that the new version worked as expected. With the advent of Docker and similar container technologies, the

deployment phase has become much easier to implement. Containers can be destroyed and restarted with the new version of code within seconds. Container orchestrator solutions, such as Kubernetes, DC/OS, and Docker Datacenter, can be used to simplify this process even further. By using the container orchestrator APIs, a call can be made to roll out the new Docker containers with the new features—and there is zero downtime. Several strategies can be used to ensure zero downtime while deploying containers; changing only one container at a time from a pool of many instances is one of them.

In most situations, there are several environments available at the deployment stage of the pipeline, but at least two are present in all cases: staging and production. In the staging environment, final preparations and manual testing by the product team takes places to ensure that everything functions as expected. Approved changes in the staging environment automatically get deployed into the production environment.

The CI/CD pipelines described so far can be adapted for network management and configuration. Taking advantage of infrastructure as code, network configurations can be stored in version control systems, and build servers can be used to integrate with solutions such as Cisco CML/VIRL (Virtual Internet Routing Lab) to dynamically instantiate virtual networks, test the configuration changes in the virtual network, and, if the tests pass, use automation solutions such as Ansible to perform the desired changes in the production network. A working network management CI/CD pipeline could include the following:



- Github.com as a cloud-based version control system or Gogs as a self-hosted Git option could be used to store the configurations.

- Travis CI as a cloud-based build server or Drone as a self-hosted build server could be used to orchestrate the whole CI/CD pipeline.
- Cisco CML/VIRL could be used to create a test network and verify the impact of the configuration changes in the network.
- Ansible could be used to automate the configuration of all the elements in the network.

CI/CD pipelines are used by more and more companies to speed up and increase the reliability and accuracy of their software development processes. Companies that have adopted CI/CD pipelines as part of their development processes release hundreds of software features every week.

AUTOMATION TOOLS

It is common practice for network administrators to perform configuration, monitoring, and maintenance activities every day on all the devices in a network or at least a subset of them. These changes were traditionally deployed manually: The network administrator would connect to each network device individually and perform the changes. The changes could be anything, from adding a new access VLAN throughout the network, to updating entries in an access control list, to changing SNMP communities' names. Where there is a manual task that is being performed, there is an opportunity to improve the process by automating that task. Open-source tools such as Ansible, Puppet, and Chef can dramatically reduce the number of manual interactions with a network. These tools enable automation at scale for application deployment, infrastructure management, and network configurations. The following sections discuss each of them in turn.

Ansible

Ansible is a configuration management and orchestration tool that can be used for a variety of purposes. It can be used to configure and monitor servers and network devices, install software, and

perform more advanced tasks such as continuous deployments and zero-downtime upgrades. It was created in 2012 and acquired by RedHat in 2015. Ansible is appropriate for both small and large environments and can be used for managing a handful of servers and network devices or for managing thousands of devices. It is agentless, meaning there is no software or service that needs to be installed on the managed device. Ansible connects to managed devices just as a regular system or network administrator would connect for management purposes—in most cases over SSH, but NETCONF and REST API interfaces are also supported. Ansible is open source and was developed using Python. There is also a commercial offering available, called Ansible Tower, which includes a web user interface, a northbound REST API, role-based access control, statistics, and much more.

Several Ansible concepts need to be discussed before we go further:



- **Control node:** The control node is any machine that has Ansible installed. All flavors of Linux and BSD operating systems are supported for the control node. Any computer, laptop, virtual machine, or server with Python installed can be an Ansible control node. The exception to this rule is that Microsoft Windows machines currently cannot be used as control nodes. Multiple control nodes can run at the same time in the same environment. It is a best practice to place the control nodes close to the systems that are being managed by Ansible to avoid network delays.
- **Managed node:** The managed nodes in the Ansible taxonomy are the network devices or servers that are being managed by Ansible. They are also called hosts and do not need to have Ansible installed on them or even Python, as discussed later in this chapter.
- **Task:** Ansible tasks are the units of action. You can run them as ad hoc commands by invoking them as follows:

[Click here to view code image](#)

```
$ ansible [pattern] -m [module] -a "[module options]"
```


- **Playbook:** An Ansible playbook is a file that contains an ordered set of tasks that will be run in the order in which they are defined and as many times as needed. Playbooks are written in YAML, which makes them easy to read, write, and share. It is common (but not mandatory) to name the main playbook `site.yml`.
- **Inventory file:** The inventory file is a list of all the managed nodes. Also called the hostfile, it contains a list of IP addresses or hostnames for all the managed nodes as well as credentials and variables that can be referenced in playbooks. Managed devices can be grouped together by function or location or based on some other feature. As the hostfile grows larger in a bigger environment, it is a best practice to move the variables to dedicated files in **group_vars/** and **host_vars/** folders. The `group_vars` folder would contain files with definitions for variables related to groups of devices, and the `host_vars` folder would contain files with definitions of variables related to individual hosts. Variables can define anything of interest for the specific environment: TCP/UDP port numbers, custom proxy configurations, timer values, and so on. Inventory files are created in either INI or YAML format. INI is a file format commonly used for configuration files of software platforms. It was extensively used in older versions of Microsoft Windows to configure the operating system. INI files are simple text files composed of basic structures such as sections, properties, and values.
- **Module:** Ansible modules are units of parameterized Python code that get executed by Ansible. Every module in Ansible has a specific use. There are modules, for example, for installing server packages, for managing users, for configuring NTP servers on Cisco NX-OS switches, and for performing **show** commands on Cisco IOS devices. An individual module is invoked through an Ansible task, or multiple modules can be invoked through an Ansible playbook.

Ansible can be installed in several different ways:

- **Using the operating system package manager:** For example, on RedHat and CentOS Linux, it can be installed with the following command:

```
sudo yum install ansible
```

- **Using the Python package manager:** The command for installing using the Python package manager is **pip install ansible**. As always, it is recommended to use a virtual environment when working with Python. Some dependencies might need to be installed before you can install Ansible. Always check the latest release notes at <https://docs.ansible.com>.
- **Using the development version:** You can clone the developer repository and issue the **source** command from a bash terminal.

When automating server configurations, Ansible uses SSH to log in to the server, copies the Python code, and runs that code on the server. There is no need to have

any Ansible component installed on the managed servers, as the implementation is agentless. For network devices, because there is limited Python support on the devices themselves, the Python code that represents the Ansible jobs runs locally on the control host. The control host still uses SSH or NETCONF, RESTCONF, SNMP, and other interfaces to connect to the network devices and perform the desired configuration changes. Ansible has a large number of modules that support several Cisco operating systems, including IOS, IOS XE, IOS XR, and NX-OS. Some of the most popular Ansible modules that are used with Cisco devices are the following:

- **ios_command** to send mostly **show** commands to devices running IOS and IOS XE operating systems
- **ios_config** to send configuration commands to IOS and IOS XE devices
- **nxos_command** and **nxos_config** to interact with devices running the NX-OS operating system

Next, let's explore a sample Ansible project. Two files are required in order to get started with Ansible playbook projects: an inventory file and a playbook. In the following Ansible project, the inventory file is called **hosts**, and the playbook file is called **site.yml**:



```
$ tree.  
├── hosts  
└── site.yml  
  
0 directories, 2 files
```

The hosts file contains an inventory of all the devices that will be managed by Ansible. For example, the hosts file can look like this:

[Click here to view code image](#)

```
$ cat hosts
[iosxe]
10.10.30.171

[iosxe:vars]
ansible_network_os=ios
ansible_connection=network_cli
```

The brackets are used to define group names. Groups are used to classify hosts that share a common characteristic, such as function, operating system, or location. In this case, the [iosxe] group of devices contains only one entry: the management IP address of a Cisco CSR1000v router. The **vars** keyword is used to define variables. In this example, two variables are defined for the iosxe group. The **ansible_network_os** variable specifies that the type of operating system for this group of devices is IOS, and the **ansible_connection** variable specifies that Ansible should connect to the devices in this group by using network_cli, which means SSH. Variables can be referenced in playbooks; as mentioned previously, as the inventory files become larger, it is a good idea to separate the variables' definitions into separate files.

The site.yml file in this example uses the **ios_command** Ansible module to send two commands to the iosxe group of devices: **show version** and **show ip interface brief**. The YAML definition of the playbook looks as shown in [Example 15-1](#).

Example 15-1 *Ansible Playbook Example*

[Click here to view code image](#)

```
$ cat site.yml
---
- name: Test Ansible ios_command on Cisco IOS
  XE
  hosts: iosxe
```

```
tasks:
  - name: show version and ip interface brief
    ios_command:
      commands:
        - show version
        - show ip interface brief
```

YAML files start with `---` (and so do Ansible playbooks). A playbook contains one or multiple plays. The plays are logical groupings of tasks and modules. The playbook in this example contains only one play that is marked by the hyphen (-) character at the leftmost position. The name of the play is optional, but it should contain an arbitrary string describing the actions that the play will perform. The name is displayed on the screen when the playbook is executed. The **hosts** keyword specifies which hosts or machines the play will be executed against. In [Example 15-1](#), the play is executed against the `iosxe` group of devices. The value for the **hosts** parameter in the playbook must match the names of the groups, as defined in the inventory file. Tasks are executed on the hosts that are defined in the play definition. For each task, a name value should contain an arbitrary description of the task. Like the name of the play, the name of the task is optional, but it should contain pertinent description text, as it is displayed to the screen when the playbook is executed. **ios_command** is the name of the Ansible module that is part of the task in this example. The **ios_command** module takes in multiple parameters, but only one of them is mandatory: the actual commands that will be sent to the IOS device. The two commands that will be sent to the `[iosxe]` device are **show version** and **show ip interface brief**.

Some of the other optional parameters for the **ios_command** module are the following:

- **interval:** Specifies the interval of time, in seconds, to wait between retries of the command.

- **retries:** Configures the number of retries for a command before it is considered failed.
- **wait_for:** Specifies a list of conditions that have to be evaluated against the output of the command. In this example, it could be defined as **wait_for: result[o] contains IOS-XE**, which verifies that the output of the **show version** command contains the value **IOS-XE** before going further with the execution of the playbook.

The playbook can be run from a terminal with the following command:



[Click here to view code image](#)

```
$ ansible-playbook -i hosts site.yml -u admin -k
```

The **-i** option specifies the name of the inventory file, the **-u** option specifies the username that will be used to connect to the device, and the **-k** option specifies that the user should be asked for the connection password when the playbook is executed. Connection credentials can also be included in the inventory file but in this example they are passed in as parameters of the `ansible-playbook` command. The output of this command looks as shown in [Example 15-2](#).

Example 15-2 *Output of ansible-playbook Command*

[Click here to view code image](#)

```
$ ansible-playbook -i hosts site.yml -u admin -k
SSH password:

PLAY [Test Ansible ios_command on Cisco IOS XE]
*****
*****

TASK [show version and ip interface brief]
*****
*****
ok: [10.10.30.171]
```

```
PLAY RECAP
*****

****
10.10.30.171      : ok=1    changed=0
unreachable=0    failed=0
                                skipped=0
rescued=0        ignored=0
```

The names of the play and the task are displayed to the screen, as are a play recap and color-coded output based on the status of the playbook execution. In this case, the playbook ran successfully, as indicated by the value **1** for the **ok** status. You can reuse the output of the two **show** commands in the playbook to build custom automation logic, or you can display it to the screen in JSON format by using the **-v** option with the **ansible-playbook** command.

Puppet

Puppet is a configuration management tool used to automate configuration of servers and network devices. Puppet was founded in 2005, making it one of the most venerable automation tools on the market today. It was created as an open-source project—and it still is today, but it is also available as a commercial offering called Puppet Enterprise that was created by Puppet Labs in 2011. It is written in Ruby and defines its automation instructions in files called *Puppet manifests*. Whereas Ansible is agentless, Puppet is agent based. This means that a software agent needs to be installed on each device that is to be managed with Puppet. This is a drawback for Puppet as there are instances of network devices in which third-party software agents cannot be easily installed. Proxy devices can be used in these situations, but the process is less than ideal and means Puppet has a greater barrier to entry than other automation tools. Puppet is architected in a client/server manner, with the client being the software agent running on the managed devices and the server being the main Puppet server,

referred to as the *Puppet Master*. By default, agents check their configuration every 30 minutes and ensure that a match exists between the expected configuration and the local configuration. There are software agents for Linux and Windows hosts, as well as for various network devices, including Cisco NX-OS and IOS XR.

Puppet manages systems in a declarative manner, meaning that the administrator defines the state the target system should be in without worrying about how the system gets to that state. Puppet models the desired system state, enforces that state, and reports any differences between the desired state and the current state of the system for tracking purposes. In order to model the system states, Puppet uses a declarative resource-based language called Puppet Domain Specific Language (DSL). The desired state of the system is defined in this language, and Puppet converges the infrastructure to the desired state. The communication between the Puppet Master and the agents is over an encrypted SSL connection.



Several components make up the Puppet architecture, as shown in [Figure 15-4](#).

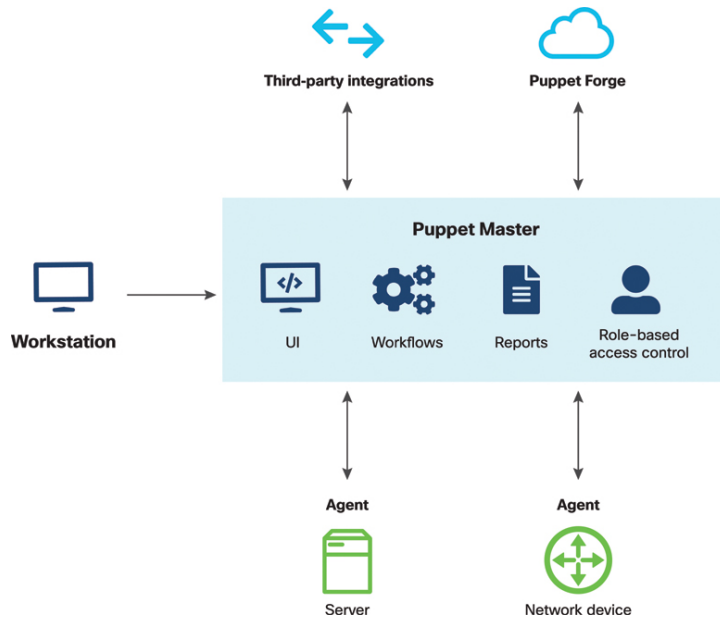


Figure 15-4 *Puppet Architecture*

The central control server, the Puppet Master, provides features such as reporting, web user interface, security, and more. The software agents run on the target node that will be monitored by Puppet. The agents connect to the Master and retrieve the correct configuration for the device they are running on. Puppet Forge is a community-based central repository where people can share their Puppet manifests and modules.

There are currently two separate versions of Puppet:

- **Open source:** This community-driven version is a collection of smaller projects and binaries, including the main Puppet agent binaries, `mcollective` and `factor`. `mcollective` provides orchestration capabilities, and `factor` is a separate binary that resides on the monitored devices and gathers facts about them.
- **Enterprise:** This version streamlines the installation process and the use of the various software packages required. For example, a single agent package that contains both the agent software and `factor` is available with the Puppet Enterprise offering. Enterprise Console, a single-pane-of-glass solution that presents a simplified view of all the facts collected by the agents, is also available with the Enterprise offering.

For a system administrator or network administrator using Puppet as a configuration management solution,

the first step is to define—using Puppet DSL—what the desired state of the infrastructure needs to be. These definitions are captured in text files called *Puppet manifests*. Some examples of desired states at this stage could be the existence of a certain VLAN on all switches in the network or maybe just a subset of devices, how and which interfaces should be configured on border routers, and which NTP servers to use for time synchronization.

Once the desired state is defined, Puppet offers the option to simulate the changes needed to reach that state and see what would happen if the changes were applied. This gives the administrator a chance to dry run the Puppet manifests and take note of what changes would actually get applied to the devices.

If the changes to be performed to reach the desired state are in line with expectations, the manifests can be executed, and the changes can be enforced. The Puppet agents report back to the Master with the status of the tasks that are being executed.

The Puppet software agent running on the managed devices is the enforcing element of the solution. The other component that is usually installed on the managed device is *facter*, which has the role of gathering facts about the managed device and reporting them back to the Master control server. *facter* reports to the control node facts such as operating system, hardware configuration, and number and type of interfaces.

The Puppet Master analyzes the facts, compares them to its database, and builds a catalog that describes how the managed device should be configured. The device-specific catalog is then pushed back to the device. The device receives the catalog and compares the policy with its current state, and if they are different, it applies the changes specified in the catalog. If the new policy and the

current state of the device match, no change is made. A report is sent back to the Master after the agent performs these functions.

The Puppet manifests are standard text files that contain Puppet DSL code and have the .pp extension. They contain declarative configuration and are descriptive and easy to read. The following is an example of a manifest used to ensure that the interface Ethernet 1/3 is in Layer 2, or switching, mode:



[Click here to view code image](#)

```
# Configuring the interface using Puppet
cisco_interface { "Ethernet1/3" :
    switchport_mode => enabled,
}
```

Puppet Forge is a cloud-based repository that contain manifests and modules contributed by the community. Puppet modules are collections of files and directories that contain Puppet manifests. When you download modules from Puppet Forge, with each module you get a group of subdirectories that have in them all the components that are needed to specify the desired state.

Chef

Chef is another popular open-source configuration management solution that is similar to Puppet. It is written in Ruby, uses a declarative model, is agent based, and refers to its automation instructions as *recipes* and *cookbooks*. Several components are part of the Chef Infra offering, as shown in [Figure 15-5](#).

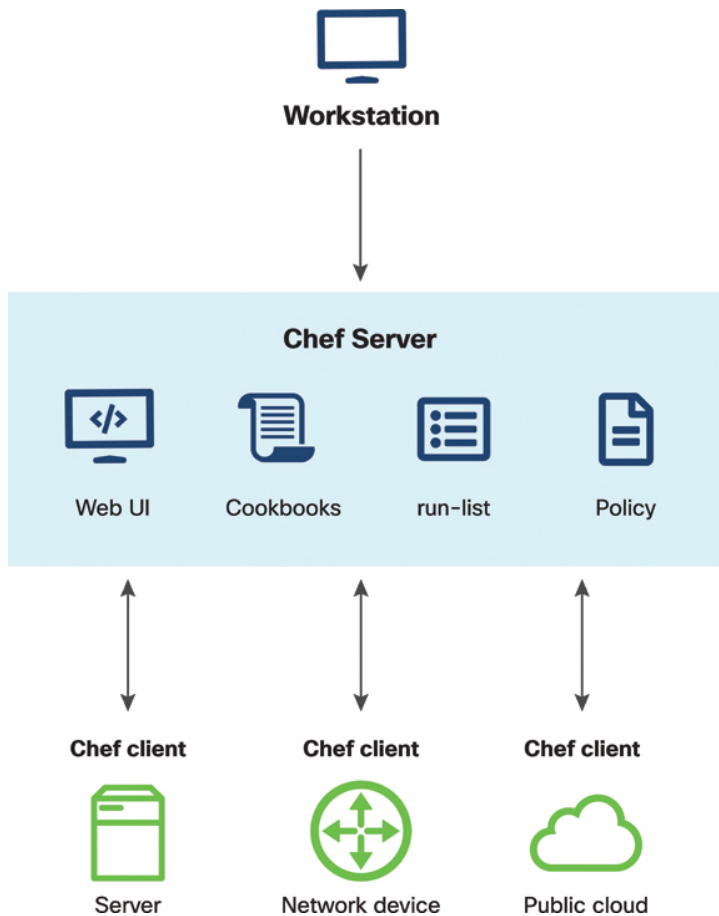


Figure 15-5 *Chef Architecture*

A Chef workstation is an administrator computer used to manage the network. One or more workstations can exist in an environment, depending on the size and complexity. A Chef workstation contains all the tools necessary for developing and testing the infrastructure automation tasks that are captured in documents called *recipes*. A Chef workstation has an instance of Chef Infra Client and can run command-line tools such as **chef** and **knife** as well as testing tools such as **Test Kitchen**, **ChefSpec**, and **Cookstyle**. A workstation computer also includes **chef-repo**, the central repository in which recipes and cookbooks are created, tested, and maintained. Chef cookbooks contain recipes, attributes, libraries, files, templates, tests, custom resources, and metadata. **chef-repo** should be managed with a version control system such as Git. Recipes are authored in

Ruby, and most of them contain simple configuration patterns that get enforced through the Chef client. Cookbooks are uploaded from the workstation to Chef Infra Server.



Chef Infra Server acts as the main hub for all the configuration information. Chef Infra Client, which is installed on each managed device, connects to Chef Infra Server to retrieve the configuration data that will be enforced on the managed client device. After each Chef Infra Client run finishes, the run data is uploaded to Chef Infra Server for troubleshooting and historical purposes. The actual managed device configuration work is done as much as possible through Chef Infra Client on the managed device; offloading these tasks from Infra Server makes the Chef solution more scalable. Infra Server also indexes all the infrastructure data, including environments, nodes, and roles, making them available for searching. The Chef management console is a web-based interface through which users can manage nodes, cookbooks and recipes, policies, roles, and so on.

Cookbooks are the fundamental building blocks for configuration and policy distribution with Chef. A cookbook defines a scenario and contains everything needed to support that scenario: recipes that specify the resources to use, templates, attribute values, tests, metadata, file distributions, and custom resources and libraries. A large set of resources to support the most common infrastructure automation requirements comes built in with Chef Infra Client. As Chef is written in Ruby, additional resources and capabilities can easily be created, if necessary.

A Chef node is any device that has the Chef Infra Client software installed, which means it is managed by Chef

Infra. A large variety of nodes are supported by Chef Infra, including virtual and physical servers; cloud-based nodes running in public and private clouds; network devices from vendors such as Cisco, Arista, F5, and others; and container environments. The main roles of Chef Infra Client are to register the node and authenticate to Chef Infra Server using RSA public key pairs, synchronize cookbooks, configure the node to match the desired state specified in the cookbooks, and report back to Infra Server with status reports. Chef has a tool built in to the Infra Client called **ohai** that is used to collect system information such as the operating system, network, memory, disk, CPU, and other data; ohai is similar to `facter` in Puppet.

Much like the Puppet Forge, Chef Supermarket is a community-maintained central location where cookbooks are created and shared between the members of the community.

The following is a sample Chef cookbook used for configuring an interface on a Cisco Nexus switch:



[Click here to view code image](#)

```
cisco_interface 'Ethernet1/3' do
  action :create
  ipv4_address '10.1.1.1'
  ipv4_netmask_length 24
  ipv4_proxy_arp true
  ipv4_redirects true
  shutdown false
  switchport_mode 'disabled'
end
```

CISCO NETWORK SERVICES ORCHESTRATOR (NSO)

Cisco Network Services Orchestrator (NSO) is a network services orchestration platform that enables end-to-end service deployment across multivendor physical and virtual infrastructure. NSO can also be considered a multivendor service-layer SDN controller for data center, enterprise, and service provider networks. It takes full advantage of NETCONF and YANG data models to provide a single API and a single user interface to the network that it manages. Cisco NSO is the single source of truth in a network, constantly maintaining the current state of all the devices in its database. It ensures that the configuration database is synchronized with all the network devices at all times.

The main components of Cisco NSO are as follows (see [Figure 15-6](#)):



- Service manager
- Device manager
- Mapping logic
- Configuration database

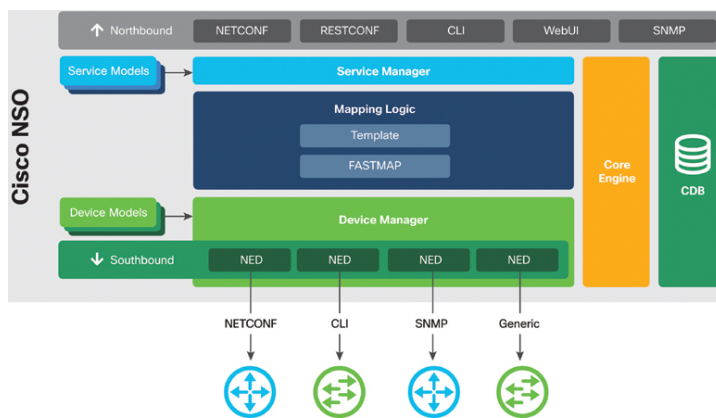


Figure 15-6 Cisco NSO Architecture

The two primary technologies that are being used with Cisco NSO are the following:

- **NETCONF:** Used for standard and efficient automation of configuration
- **YANG:** Used for services and device configuration data modeling

One of the main advantages of Cisco NSO is the model-to-model mapping capability. A network administrator can define the data models for the end-to-end services that need to be implemented in the network (for example, Layer 3 MPLS VPNs, IPTV), and NSO maps those service models into device models for various vendor devices. Because a large number of devices do not fully support NETCONF and YANG data models, NSO uses Network Element Drivers (NEDs) to model native device CLIs into YANG models. NSO offers a level of abstraction that makes network devices transparent to the service management of the network. This way, complex services can be described and implemented in simple ways and pushed to the devices no matter the device vendor, configuration semantics, and so on.

NSO provides several northbound interfaces for service and platform management, including NETCONF, RESTCONF, JSON/RPC, a CLI, a web user interface, and SNMP. The NSO CLI and web user interface are usually used for human interaction with the platform, SNMP and NETCONF are used for monitoring, and the NETCONF, RESTCONF, and JSON/RPC interfaces are used for automation integration. The NSO CLI has two modes of operation: operational mode, used primarily to monitor and maintain the platform, and configuration mode, used to configure the services, the devices, and the NSO server. Two CLI styles are offered to NSO operators: Cisco style, which is identical to the Cisco CLI, and Juniper style, which is identical to the Juniper CLI. CLI users can easily switch between them by using the **switch cli** command.

The southbound communication with the network devices is done through components called Network Element Drivers (NEDs), which implement a large set of southbound interfaces, including NETCONF, the CLI, SNMP, and OpenFlow.

The logical architecture of Cisco NSO is a dual-layered approach: a device manager component that handles the device configuration scenarios and a service manager component that provides an interface for the administrator to define the services that need to be implemented in the network. The configuration data store, called the Configuration Database (CDB), is in sync with all the devices in the network and contains all the devices and services configurations. A dedicated mapping layer manages the correspondence between the service models and the device models.



The Cisco NSO Core Engine manages critical operational functions such as transactions, high-availability replication, upgrades and downgrades, role-based access control, and rollback management. All operations in NSO are handled by the transaction manager. The Core Engine connects all the other NSO components together; it is the communication backbone for all other components and is the process that reads the initial configuration defined in the `ncs.conf` file.

The CDB stores all the platform data in a RAM database, including the NSO configuration, the configurations of all services and all managed devices, and all NSO operational data. The CDB is stored in RAM for increased speed and quick response, and a persistent version of it is stored on the disk drive of the server. The CDB is a hierarchical database organized in a tree-like structure, and although other database solutions can be

used with NSO, it is recommended to use CDB as it has been optimized for NSO and the transactional nature of the solution.

With the Cisco NSO service manager, users develop YANG models of the services they want deployed in the network, such as IPTV or MPLS VPNs. At this stage, the service model also has to be mapped to the corresponding device configuration model. This is done either through configuration templates that map service parameters into device configuration parameters or programmatically by building the mapping logic with Java or Python code. The lifecycle of the services defined in NSO is managed by a FASTMAP algorithm. As services get created, modified, and deleted, FASTMAP dynamically reacts and adjusts the configuration of the devices in the network to reflect the status of the service models. In addition, FASTMAP enables the following functions:

- **Service dry-run:** NSO calculates what the device changes would be if the service model were to be pushed on the devices in the network.
- **Service check-sync:** The service configuration is verified to be in sync with the actual device configuration. Device configuration drift can be detected this way.
- **Service re-deploy:** Device configurations can be redeployed to restore a service on the network.

The NSO device manager manages the network devices by using YANG data models and NETCONF. For devices that natively implement NETCONF and YANG models, the device manager is automatic; devices that do not support NETCONF are integrated in the platform with NEDs. NEDs that support different vendor CLIs as well as SNMP configuration and hundreds of other southbound management options come with Cisco NSO by default. If needed, custom NEDs can be developed to address any device management protocol that is not already included with the default installation.

Distributed atomic transactions are used for all configuration changes on all the devices in the network.

If applying the configuration changes fails on any of the devices in the service path, a rollback mechanism can be used to revert all the changes and return the network devices to their initial status before the service deployment was attempted. Configuration templates can be used at this stage to easily and quickly provision new devices.

As of this writing, an educational version of Cisco NSO can be downloaded from Cisco DevNet at <https://developer.cisco.com/site/nso/>. Installation steps can also be found there. The following examples use Cisco NSO version 5.1 running locally on macOS Catalina.

Cisco NSO provides a network simulation tool called **ncs-netsim**. This tool makes it very easy to test NSO packages against simulated devices and to learn how to use the platform without any hardware needed. The same YANG models are used for both real and NetSim simulated devices. Several options are available with ncs-netsim, as shown in [Example 15-3](#).

Example 15-3 *ncs-netsim Options*

[Click here to view code image](#)

```
$ ncs-netsim --help
Usage ncs-netsim [--dir <NetsimDir>]
                create-network <NcsPackage>
<NumDevices> <Prefix> |
                create-device <NcsPackage>
<DeviceName>      |
                add-to-network <NcsPackage>
<NumDevices> <Prefix> |
                add-device <NcsPackage>
<DeviceName> |
                delete-network
|
                [-a | --async] start
[devname]      |
                [-a | --async ] stop
[devname]      |
                [-a | --async ] reset
[devname]      |
```

```

    [-a | --async ] restart
[devname] |
    list |
    is-alive [devname] |
    status [devname] |
    whichdir |
    ncs-xml-init [devname] |
    ncs-xml-init-remote
<RemoteNodeName> [devname] |
    [--force-generic]
|
    packages |
    netconf-console devname
[XpathFilter] |
    [-w | --window] [cli | cli-c
| cli-i] devname |
    get-port devname [ipc |
netconf | cli | snmp]
See manpage for ncs-netsim for more info.
NetsimDir is optional and defaults to
./netsim, any netsim directory above in the
path, or $NETSIM_DIR if set.

```

In order to create a new network simulation, the **create-network** option is used with the following parameters:

[Click here to view code image](#)

```

ncs-netsim create-network <NCS package> <#N
devices> <PrefixY>

```

<NCS package> contains the path to where the NCS packages were installed. By default, this is `$NCS_DIR/packages/neds`, and it contains NCS packages for Cisco IOS, Cisco IOS XR, Cisco NX-OS, Juniper JunOS, and so on. The *<#N devices>* parameter specifies the number of devices that will be created, and *<PrefixY>* defines the prefix that will be used to name the devices. For example, the following command would be used to create a network simulation with three Cisco IOS devices named `ios0`, `ios1`, and `ios2`:

[Click here to view code image](#)

```

$ ncs-netsim create-network
$NCS_DIR/packages/neds/cisco-

```

```
ios-cli-3.8 3 ios
DEVICE ios0 CREATED
DEVICE ios1 CREATED
DEVICE ios2 CREATED
```

After the virtual devices are created, the simulation can be started with the **start** command:

```
$ ncs-netsim start
DEVICE ios0 OK STARTED
DEVICE ios1 OK STARTED
DEVICE ios2 OK STARTED
```

An ordered list of all the devices that are running at any point on the NSO server can be obtained by using the **list** option for ncs-netsim, as shown in the following output:

[Click here to view code image](#)

```
$ ncs-netsim list
ncs-netsim list for $NCS_DIR/nso-run/netsim

name=ios0 netconf=12022 snmp=11022 ipc=5010
cli=10022 dir=$NCS_DIR/
nso-run/netsim/ios/ios0

name=ios1 netconf=12023 snmp=11023 ipc=5011
cli=10023 dir=$NCS_DIR
/nso-run/netsim/ios/ios1

name=ios2 netconf=12024 snmp=11024 ipc=5012
cli=10024 dir=$NCS_DIR
/nso-run/netsim/ios/ios2
```

Access to the console ports of all the simulated devices is possible by using the **cli-i** option followed by the name of the device, as in [Example 15-4](#), which demonstrates console access to the ios0 device.

Example 15-4 CLI Access to a Simulated Device

[Click here to view code image](#)

```
$ ncs-netsim cli-i ios0
```

```
admin connected from 127.0.0.1 using console on
*
ios0> en
ios0# show running-config
no service pad
no ip domain-lookup
no ip http server
no ip http secure-server
ip routing
ip source-route
ip vrf my-forward
  bgp next-hop Loopback 1
!
```

The simulated devices implement all control plane functions of the original operating system, but they do not have the data plane implemented, so they do not forward data traffic. Even with this limitation, ncs-netsim network simulations are used extensively for testing and learning.

As mentioned previously, Cisco NSO exposes a RESTCONF northbound interface for automation and integration purposes. Before simulated devices are accessible over this interface, they have to be onboarded within Cisco NSO. Devices can be onboarded through any of the northbound interfaces that Cisco NSO exposes, but in this example, the NSO CLI was chosen. First, access to the NSO Cisco version (**-C** parameter) of the CLI for the admin user is obtained by issuing the **ncs_cli -C -u admin** command. The default password is also **admin**. Once access to the NSO CLI is granted, all the simulated Cisco IOS devices need to be added in the configuration of the Cisco NSO server. [Example 15-5](#) shows the configuration for ios0.

Example 15-5 *Onboarding Simulated Devices in Cisco NSO*

[Click here to view code image](#)

```
$ ncs_cli -C -u admin
```

```
admin connected from 127.0.0.1 using console on
*
admin@ncs# config term
Entering configuration mode terminal
admin@ncs(config)# devices device ios0
admin@ncs(config-device-ios0)# address
127.0.0.1
admin@ncs(config-device-ios0)# port 10022
admin@ncs(config-device-ios0)# authgroup
default
admin@ncs(config-device-ios0)# device-type cli
ned-id cisco-ios-cli-3.8
admin@ncs(config-device-ios0)# state admin-
state unlocked
admin@ncs(config-device-ios0)# exit
```

When all three simulated IOS devices are onboarded and the configuration changes are committed, the devices can be displayed through a GET call over the RESTCONF interface. Cisco NSO exposes its full functionality over this RESTCONF interface that is available by default starting at

http://<NSO_Server_IP>:8080/restconf/ root. A list of all the devices that have been onboarded can be obtained by performing a GET call on the following RESTCONF URI resource:

http://<NSO_Server_IP>:8080/restconf/data/tailf-ncs:devices/device. The **curl** command to test this GET call looks as follows:



[Click here to view code image](#)

```
$ curl --request GET
'http://localhost:8080/restconf/data/tailf-
ncs:devices/device' \

--header 'Content-Type: application/yang-
data+json' \

--header 'Authorization: Basic YWRtaW46YWRtaW4='
```

The Content-Type header needs to be **application/yang-data+json** because JSON-formatted YANG data models are exchanged over the RESTCONF interface. The authorization header contains the basic authentication Base64 encoded for the **admin** username and the **admin** password. A snippet of the response is shown in [Example 15-6](#).

Example 15-6 *Output of the RESTCONF GET Devices API Call*

[Click here to view code image](#)

```
{
  "tailf-ncs:device": [
    {
      "name": "ios0",
      "address": "127.0.0.1",
      "port": 10022,
      "authgroup": "default",
      "device-type": {
        "cli": {
          "ned-id": "cisco-ios-cli-3.8:cisco-
ios-cli-3.8"
        }
      },
      "commit-queue": {
        "queue-length": 0
      },
      "active-settings": {
        "connect-timeout": 20,
        "read-timeout": 20,
        "write-timeout": 20,
        "ssh-keep-alive": {
          "interval": 20,
          "count": 3
        },
        "ned-keep-alive": {
          "count": 3
        }
      },
      ... omitted output
    }
  ]
}
```

The Cisco NSO RESTCONF interface can also be explored by using Postman. For example, we can get the configuration of the ios1 simulated device from the NSO RESTCONF interface by using Postman instead of **curl**.

The URI that returns this information is very similar to the one explored previously, with one difference: `http://localhost:8080/restconf/data/tailf-ncs:devices/device=ios1`. `device=ios1` is used to indicate that the API should return only data specific to the device `ios1`. The same headers for Content-Type, `application/yang-data+json`, Authorization, and Basic Auth with username **admin** and password **admin** need to be included. [Figure 15-7](#) shows the response from the NSO server after the Send button is pressed.

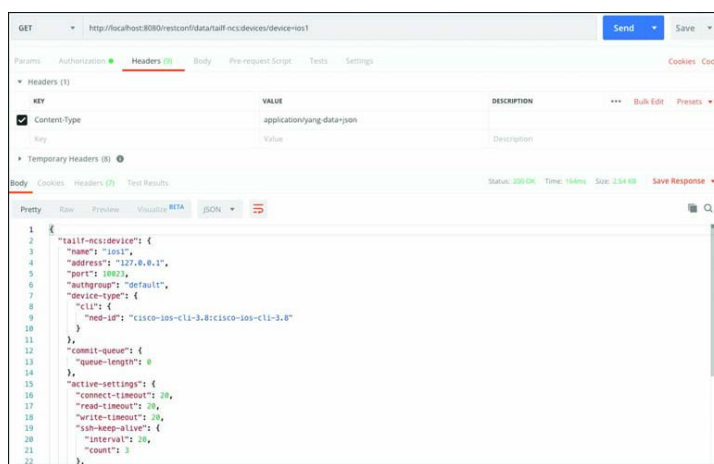


Figure 15-7 RESTCONF GET Device API Call in Postman

Cisco Modeling Labs/Cisco Virtual Internet Routing Laboratory (CML/VIRL)

For a long time, building and maintaining infrastructure laboratories for testing and learning was time-consuming and required a lot of hardware, which needed space, power, and cooling; these labs were therefore very expensive and in reach of only a few companies. With the advent of Network Function Virtualization (NFV) and Cisco Modeling Labs/Cisco Virtual Internet Routing Laboratory (CML/VIRL), it has finally become easy and cost-effective to build infrastructure laboratories for all types of purposes.

NFV is meant to decouple the hardware and software requirements that are typical for network devices and enable the software component or the network operating system to run on commodity hardware as a virtual machine, in a container, or on bare-metal servers. While in most cases this leads to a loss of performance as custom hardware components such as Application Specific Integrated Circuits (ASICs) are not used with NFV offerings, it is usually considered an acceptable trade-off and offers a cost-effective way to take advantage of the software features that come with the network operating system. If maximum performance is required, then a combination of dedicated custom hardware and software can be used to ensure guaranteed performance. If the performance requirement is not as stringent, then an NFV alternative that includes just the software network operating system running on off-the-shelf hardware can be considered. For example, Cisco offers the enterprise-grade network operating system Cisco IOS XE as a downloadable virtual machine in the form of a Cisco CSR1000v router for an NFV offering covering enterprise routing and switching capabilities, and it also offers IOS XE as a combination of hardware and software with the Cisco Catalyst 9000 Series of switches that offer 100Gbps interfaces and guaranteed hardware routing and switching forwarding performance. There is a virtual option for almost all Cisco operating systems currently available, including Cisco IOS XE with Cisco CSR1000v, Cisco IOS XR with Cisco IOS XRv, and Cisco NX-OS with Cisco NX-OSv 9000. These virtual instances can be run on x86 commodity hardware on premises or in private and public cloud environments.

Cisco CML/VIRL is a powerful network simulation platform that is meant to be a flexible, all-in-one virtual networking lab. With Cisco CML/VIRL and virtual instances of network operating systems from Cisco, it is possible to build entire network simulations. The

CML/VIRL version 2 web interface looks as shown in [Figure 15-8](#).

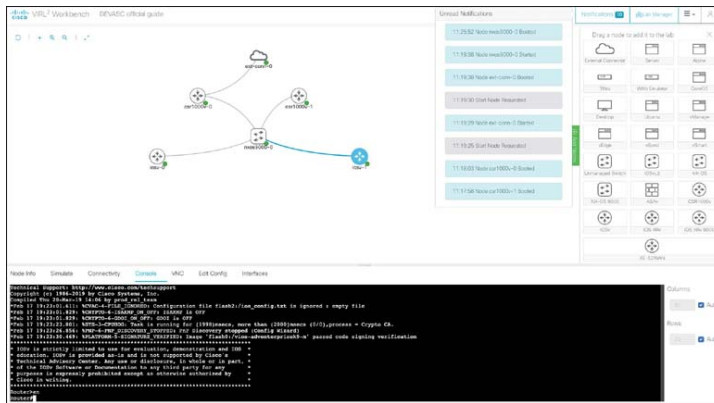


Figure 15-8 CML/VIRL 2 Web Interface



Cisco CML/VIRL provides several interfaces: a CLI interface, a powerful web user interface (refer to [Figure 15-8](#)), and a REST API for automation and integration. Multiple virtual instances of operating systems and solutions—including Cisco SD-WAN (Cisco vManage, Cisco vSmart, Cisco vBond, Cisco vEdge), Cisco CSR1000v, Cisco NX-OS, Cisco IOS XRv, Ubuntu, CoreOS, and TRex—are included with CML/VIRL by default, and it is possible to add other third-party virtual instances as well. While the software features in the virtual instances of the Cisco network operating systems are in most cases on parity with the binary files installed on Cisco routers and switches, there are limitations in regard to the data plane implementation. There are no switching backplanes and no ASIC modules with the virtual nodes in CML/VIRL, so throughput performance cannot be tested accurately in most cases.

Using an external connector, a simulation running in Cisco CML/VIRL can be connected to the external world and the Internet. This is done through the network

interface on the server that Cisco CML/VIRL is installed on. Cisco CML/VIRL comes as an OVA file that can be deployed on virtual machine hypervisors such as VMware ESXi or on bare-metal servers. For details on where to download CML/VIRL, installation steps, and licensing, see <http://virl.cisco.com>.

Several lab simulations can be run at the same time. Going back to the web user interface from [Figure 15-8](#), under the Lab Manager tab, new labs can be created, modified, or deleted. A lab in CML/VIRL is a logical construct used to organize and separate network topologies into separate environments. Once a lab is created, a network topology can be built with the nodes and virtual instances that are available in CML/VIRL. Network interfaces can be dynamically added and removed from CML/VIRL nodes and can be easily interconnected by simply hovering the mouse over the devices, selecting the network connection option, and then dragging the virtual network cable between the devices that need to be connected. Direct access to the console of the nodes is also managed through CML/VIRL. In the case of servers, VNC connections are also supported. Startup configurations for network devices can be specified prior to starting a simulation. Network topologies are stored as YAML files and can be easily modified and shared. [Example 15-7](#) shows a snippet of a CML/VIRL network topology file.

Example 15-7 CML/VIRL Topology File Example

[Click here to view code image](#)

```
lab:
  description: ''
  notes: ''
  timestamp: 1581966586.8872395
  title: DEVASC official guide
  version: 0.0.3
nodes:
- id: n0
  label: iosv-0
  node_definition: iosv
```

```
x: -500
y: 50
configuration: ''
image_definition: iosv-158-3
tags: []
interfaces:
  - id: i0
    label: Loopback0
    type: loopback
  - id: i1
    slot: 0
    label: GigabitEthernet0/0
    type: physical
  - id: n1
    label: csr1000v-0
    node_definition: csr1000v
...omitted output
```

Downloading and uploading CML/VIRL network topology YAML files is easy and straightforward, and CML/VIRL users can easily share topologies and also store them in version control systems such as Git. CML/VIRL can be part of a CI/CD pipeline for network configuration changes. An example of such a pipeline could contain the following components and processes:

- CML/VIRL topology files could be stored together with automation scripts, device configuration files, and possibly Ansible playbooks in a version control system.
- Whenever a change is made to any of these files, a build automation tool such as Jenkins or Drone can be used to monitor and detect the change in version and initiate the automation processes.
- A complete automated process can create a test environment using the CML/VIRL REST APIs, build application servers, and deploy them in the test environment.
- The network configuration changes can be tested in this CML/VIRL virtual environment and a pass or fail criterion can be set based on the expectations of the administrator.
- If the tests pass, a configuration automation solution like Ansible could use playbooks and automation scripts to apply the tested configuration changes in the production network.

Python Automated Test System (pyATS)

pyATS and the pyATS library together form the Cisco test and automation solution, a vibrant ecosystem that aims

to standardize how automated network tests are set up and run. pyATS was developed by Cisco in 2014 and is used extensively internally by thousands of engineers to run unit tests, regression tests, and end-to-end and integration tests for a large number of Cisco products. The solution is completely developed in Python3, making it easy to work with, scalable, and extensible. Millions of pyATS tests are run every month internally at Cisco. pyATS tests provide sanity, feature, solution, system, and scale checks for any type of physical or virtual device, including routers, switches, access points, and firewalls.

The solution has two main components: the pyATS test framework and the pyATS library, which used to be called Genie but was renamed in an effort to simplify the nomenclature of the product. The pyATS test framework provides ways to define how the network topologies are created and modeled, how to connect to devices through connection libraries, and how to actually perform the tests and generate reports. The pyATS library builds on this infrastructure framework and provides easy-to-use libraries that implement pyATS features, parsers to interpret the data received from the devices, a mechanism for modeling network device configurations for both Cisco and third-party vendors, reusable test cases in the form of triggers and verifications, and the ability to build test suites through YAML-based text files. In the rest of this chapter, when pyATS is mentioned, assume that we are referring to the product as a whole, unless otherwise specified. [Figure 15-9](#) shows the components of the pyATS solution.



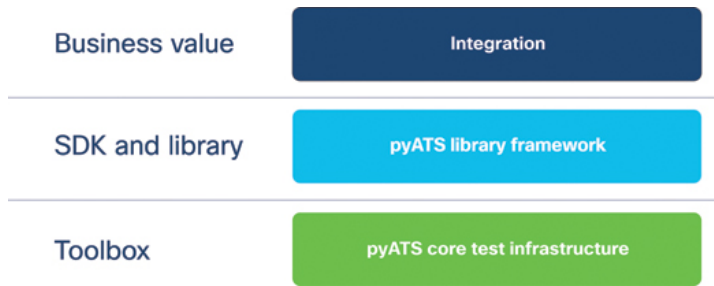


Figure 15-9 *pyATS Architecture*

pyATS fits very well in infrastructure automation use cases, especially around configuration change validation. As discussed earlier in this chapter, tools like Ansible, Puppet, and Chef are great at infrastructure configuration automation, but they do not have great change validation options. pyATS can be easily integrated in day-to-day DevOps activities and CI/CD automation pipelines. Some of the most common use cases for pyATS are as follows:

- **Profiling the current status of a network and taking a snapshot of both the configuration status as well as the operational data of the network:** This can be done before and after a configuration change or a software upgrade/downgrade is performed to ensure that the network still performs within desired parameters. For example, a snapshot of the network is taken with pyATS before a software upgrade is performed, and key metrics are noted, such as number of BGP sessions in the established state or the number and type of routing table entries or any other metric that is considered critical for that environment. The software upgrade is completed, and then a new pyATS snapshot is taken to ensure that those critical metrics have values within expected parameters. pyATS offers all the tooling to be able to automatically perform the network snapshots, compare key metric values, set pass/fail criteria, and even generate reports.
- **Automating configuration changes and monitoring of devices:** By using the Conf module within the pyATS library, configuration changes can be performed in a uniform fashion across different devices from different vendors. Together with the profiling capabilities, pyATS offers a full set of capabilities for infrastructure automation, including creating a snapshot of the network pre-change, performing the changes, taking another snapshot post-change, and continuously monitoring for any performance outliers.

pyATS offers an abstraction layer and programmatically stores device configuration and operational data in Python objects such as dictionaries and lists. It is not

necessary to screen scrape and parse different proprietary CLI outputs, as pyATS automatically accomplishes this through its extensive collection of parsers and outputs the data into standard formats that can be easily consumed. The platform is easily extensible and customizable with scripts and test cases that pertain to various environments. A pyATS CLI is also available for network testers who are not comfortable with Python. pyATS supports several options to connect to devices: Telnet, SSH, REST, NETCONF, and RESTCONF.

pyATS can be installed through **pip** by issuing the following command:

```
pip install pyats[full]
```

Different options are available:

- **pyats[full]** installs all pyATS components, the pyATS library framework, and optional extras.
- **pyats[library]** installs all the pyATS components without the optional extras.
- **pyats** without any options installs just the pyATS test infrastructure framework.

As of this writing, pyATS version 20.1 is the current version, and it is only available for Linux and macOS. Microsoft Windows is not directly supported, but Windows Subsystem for Linux (WSL) can be used to run pyATS on Windows 10. As usual, it is recommended to install pyATS in its own virtual environment. A prebuilt Docker container with pyATS already installed can be found at <https://hub.docker.com>. More up-to-date information on pyATS versions, documentation, and installation steps can be found online at <https://developer.cisco.com/pyats/>.

In order to start working with pyATS, a testbed YAML file needs to be created. This file contains the connectivity details for all the devices in the testbed.

Example 15-8 shows an example of a YAML file with only one device.



Example 15-8 *pyATS Testbed File*

[Click here to view code image](#)

```
---
devices:
  csr1000v-1:
    type: 'router'
    os: 'iosxe'
    platform: asr1k
    alias: 'uut'
    credentials:
      default:
        username: vagrant
        password: vagrant
    connections:
      cli:
        protocol: ssh
        port: 2222
        ip: "127.0.0.1"
```

Most of the options, while defining devices in a testbed, should be self-explanatory. There are just a few options that need to be explained in more detail. First, the hostname—which is `csr1000v-1` in Example 15-8—has to match the configured hostname of the device. Second, the alias is used to identify the device during script execution. The test script can be reused on a different testbed as long as the aliases stay the same. A complete list of values for the type, operating system, and platform is available with the pyATS documentation. Say that you save this file and call it `testbed.yaml`.

Example 15-9 shows a simple Python script that uses the pyATS library and loads the `testbed.yaml` file, searches in that file for a device with alias **uut**, connects to that device, runs the **show version** command, and extracts

the Cisco IOS XE version number and prints it to the screen.

Example 15-9 *pyATS Test*

[Click here to view code image](#)

```
#!/usr/bin/env python
from genie.testbed import load

# Load the testbed
tb = load('testbed.yaml')

# Find the device with alias uut
dev = tb.devices['uut']

# Connect to the device
dev.connect()

# Parse the output of the show version command
output = dev.parse('show version')

# Extract the version number and print it to
the screen
print('IOS-XE version number: ' +
output['version']['version'])
```

The **output** variable is a Python dictionary that contains the JSON standardized data returned by the **show version** command. This makes it very easy to parse and extract the needed data from the output of the **show** command.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. Table 15-3 lists these key topics and the page number on which each is found.

Table 15-3 Key Topics



Key Topic Element Description Page Number		
Parag raph	Network controllers	4 5 1
Parag raph	Direct device configuration	4 5 3
Parag raph	Infrastructure as code	4 5 4
List	Two approaches to infrastructure as code: declarative and imperative	4 5 5
Parag raph	Continuous integration/continuous delivery	4 5 5
Parag raph	Build server	4 5 6
Parag raph	CI/CD pipelines	4 5 7
List	Ansible concepts	4 5

		<u>8</u>
Parag raph	Sample Ansible project	<u>4</u> <u>6</u> <u>0</u>
Parag raph	Ansible playbooks	<u>4</u> <u>6</u> <u>1</u>
Parag raph	Puppet	<u>4</u> <u>6</u> <u>2</u>
Parag raph	Puppet manifests	<u>4</u> <u>6</u> <u>4</u>
Parag raph	Chef workstation	<u>4</u> <u>6</u> <u>6</u>
Parag raph	Chef cookbook	<u>4</u> <u>6</u> <u>6</u>
List	Main components of Cisco NSO	<u>4</u> <u>6</u> <u>7</u>
Parag raph	Logical architecture of Cisco NSO	<u>4</u> <u>6</u> <u>8</u>
Parag raph	GET call over the RESTCONF interface	<u>4</u> <u>7</u> <u>2</u>
Parag raph	Cisco CML/VIRL	<u>4</u> <u>7</u> <u>4</u>
Parag raph	pyATS test framework and pyATS library	<u>4</u> <u>7</u> <u>7</u>

Parag raph	Testbed YAML file for pyATS	4 7 <u>8</u>
---------------	-----------------------------	--------------------

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

continuous integration/continuous delivery (CI/CD)
pipeline

Network Services Orchestrator (NSO)

Configuration Database (CDB)

Cisco Modeling Labs/Cisco Virtual Internet Routing
Laboratory (CML/VIRL)

Python Automated Test System (pyATS)

Chapter 16

Network Fundamentals

This chapter covers the following topics:

Network Reference Models: This section covers the two most popular networking reference models: the OSI model and the TCP/IP model.

Switching Concepts: This section introduces Layer 2 switching concepts such as Ethernet, MAC addresses, and VLANs.

Routing Concepts: This section delves into Layer 3 routing concepts and introduces IP addressing.

This chapter covers network fundamentals. It starts with a short introduction to what networking is and how it has evolved over the years. It also takes an in-depth look at the two most popular networking reference models: the OSI model and the TCP/IP model. This chapter also explores critical Layer 2 technologies, including Ethernet, which has become the de facto protocol for local-area networks (LANs), MAC addresses, virtual local-area network (VLANs), switching, and switching tables. Internet Protocol (IP) is the Layer 3 protocol that the Internet is built on.

This chapter explores both IP version 4 and version 6, as well as what routing is, how routing tables are built, and how data packets are forwarded over networks.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly

or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 16-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 16-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Network Reference Models	1–4
Switching Concepts	5–7
Routing Concepts	8–10

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Which OSI layer is responsible for establishing an end-to-end connection between the sender and the receiver?
 1. Network layer
 2. Transport layer
 3. Session layer

4. Presentation layer
2. What layer of the TCP/IP reference model is the equivalent of the network layer in the OSI model?
 1. Physical layer
 2. Data link layer
 3. Internet layer
 4. Transport layer
3. Which of the following are examples of application layer protocols in the TCP/IP reference model? (Choose two.)
 1. TCP
 2. HTTP
 3. BGP
 4. FTP
4. What is the transport layer PDU called in the TCP/IP reference model?
 1. Data
 2. Frame
 3. Packet
 4. Segment
5. What is the role of the Preamble field in the Ethernet header?
 1. It is used as padding data to ensure that the frame has at least the minimum number of bytes for transmission.
 2. It is used as padding data to ensure that the frame has 1500 bytes for transmission.
 3. It is used to ensure that the frame was transmitted without data corruption.
 4. It is used to synchronize the signal between the sender and receiver.
6. How many bits are in a MAC address?
 1. 24 bits
 2. 48 bits
 3. 32 bits
 4. 64 bits
7. What happens to a data frame for which a switch doesn't have the destination MAC address in its switching table?
 1. It gets discarded.
 2. It gets transformed into a broadcast data frame.
 3. It gets sent back to the sender as it cannot be switched.
 4. It gets flooded out all the ports except the port on which it was received.

8. What is the bit pattern in the first byte for Class C IPv4 addresses?

1. 110XXXXX
2. 11110XXX
3. 10XXXXXX
4. 1110XXXX

9. What is the broadcast address for the 192.168.0.96/27 network?

1. 192.168.0.191
2. 192.168.0.159
3. 192.168.0.127
4. 192.168.0.255

10. What are some of the characteristics of IPv6 addresses? (Choose three.)

1. They are 128 bits long.
2. Colons separate 16-bit hexadecimal fields.
3. Hexadecimal characters are case sensitive.
4. Successive fields of zero can be represented as ::.

FOUNDATION TOPICS

Networks of devices have been built for more than 50 years now. The term *device* is used here to mean any piece of electronic equipment that has a network interface card of some sort. End-user devices or consumer devices such as personal computers, laptops, smartphones, tablets, and printers as well as infrastructure devices such as switches, routers, firewalls, and load balancers have been interconnected in both private and public networks for many years. What started as islands of disparate devices connected on university campuses in the 1960s and then connected directly with each other on ARPANET has evolved to become the Internet, where everyone and everything is interconnected. This chapter discusses the fundamentals of networking.

NETWORK REFERENCE MODELS

Several network reference models have been developed through the years, some of them proprietary and some open. Two reference models stand out from the crowd for their importance and longevity: the OSI model and the TCP/IP reference model.

The OSI Model

In the late 1970s, as computing devices became more and more popular, a need to connect them in a manner in which they would be able to exchange data between them arose. The old sneakernet—which involved physically copying data that needed to be transferred between devices on a floppy disk and delivering it on foot—was no longer appropriate for the amounts of data that needed to be exchanged. Disparate proprietary solutions such as System Network Architecture (SNA) from IBM and DECnet from Digital Equipment Corporation (DEC) existed for this purpose, but there was a need to have a standard, interoperable, and common way to do networking. The International Organization for Standardization (ISO) took on this challenge and came up with the Open Systems Interconnection (OSI) model. Although it has not been implemented in production systems, the OSI model is still extensively used especially from an educational perspective. The simpler TCP/IP (Transport Control Protocol/Internet Protocol) model has had much more success in real life and is covered in more detail later in this chapter.

The main idea with developing networking models based on different layers that interact with each other was inspired by the computer science mantra of splitting a difficult problem into smaller more manageable problems. Following this philosophy, the OSI model splits the data transmission processes needed to transfer data between two or more devices into seven layers. An advantage with a layered approach is that there can be different vendors at different layers implementing the specific functionality required by each layer. There are

network equipment vendors that focus on developing devices that operate at Layer 2 of the OSI model, called *switches*, and devices that operate at Layer 3 of the model, called *routers*, and so on. But probably the biggest advantage of layering is the level of innovation that this model allows at each layer. As long as the functionality provided by each layer stays consistent with the expectations of the layer above, protocols, hardware, and software can be developed and improved. Take, for example, the way the Ethernet protocol and specifications have evolved over the years from transfer rates of 10 Mbps in the early days of the protocol to 400 Gbps today. This improvement of several orders of magnitude has been possible because of the layering philosophy. As long as the data transmission layer provides a set of primitives to the layer above, it can freely innovate and improve.



In the layered approach, each layer provides a set of functionality and features to the layer above it and consumes the features of the layer below it. The OSI model defines seven layers for exchanging data between networked devices (see [Figure 16-1](#)). Two or more devices that would exchange data with each other using the OSI model need to implement all seven layers.



Figure 16-1 *OSI Model Layers*

Applications that need to transmit data over the network start at the application layer, gathering the data that needs to be transmitted. This data makes its way down the transmission model and gets processed at each layer by being split into smaller chunks, with header and footer information added, and then being encoded on the physical medium. As the bits arrive at the destination, the reverse process takes place: As the data moves up the layers at the receiver, headers and footers are removed, and data is recombined into bigger chunks and made available to the receiving application. This is similar in a way to how postal mail works. When sending out heartwarming holiday wishes via snail mail, first you need to put your message on a piece of paper. This is the equivalent of the application layer at the sender. Then you put the paper with the message in an envelope, seal the envelope, write down the sender and receiver information, and drop the envelope in a mailbox or at the post office. The postal service plays the role of the physical layer in this case, as it sorts through the mail and delivers each piece to its destination. The person receiving the mail makes sure that the envelope is meant

for him or her and then goes ahead and reverses part of the process you went through, removing the paper from the envelope and reading the message. The message is successfully delivered from source to destination in both cases—on a network and via snail mail—but it takes a fraction of a second for data to get anywhere on planet Earth on a network, whereas it might take days or weeks for snail mail to reach a recipient, depending on how far the recipient is from the sender.

Data transmission protocols have been developed at each layer of the OSI model to perform the functionality required by that layer. Each layer is supposed to perform specific tasks and provide specific functionality to the layers above it.

The **physical layer** is tasked with the transmission and reception of raw bit streams. At this layer, specifications about the voltage levels, pinouts for the connectors, physical transmission data rates, modulation schemes, and so on are defined. This is where the proverbial rubber meets the road, and data to be transmitted is converted into electrical, optical, or radio signals. Protocols such as Ethernet and Bluetooth have specifications at the physical layer.

The **data link layer** provides device-to-device data transfer on a local-area network. It defines how data is formatted for transmission and how access to the physical medium is controlled. Errors that might have happened in transit on the physical layer are corrected at the data link layer. Bridges and switches are devices that operate at the data link layer. The IEEE has several protocols defined for the data link layer, organized into the IEEE 802 family of protocols. According to the specifications in these protocols, the data link layer is further subdivided into two sublayers:

- The Media Access Control (MAC) layer provides specifications on how devices gain access to the transmission medium.

- The Logical Link Control (LLC) layer is responsible for encapsulating network layer data and frame synchronization.

The **network layer** provides connectivity between devices that are connected to separate networks. Whereas the data link layer ensures communication between directly connected devices on a local network, the network layer enables communication between devices that are in geographically separate locations. Logical addressing of all the devices on the network and routing of data packets on the best path between a sender and a receiver are also specified at the network layer. Routers are devices that operate at the network layer. If the data that needs to be transmitted over the data link layer is too large to be sent at once, the network layer has the capability to fragment the data into smaller pieces. As the name indicates, the process of *fragmentation* consists of splitting the data into smaller pieces, sending those pieces on the network, and then reassembling them when they arrive at another device on the network. Fragmentation should be avoided as much as possible as it slows down the transmission of data considerably. This slowdown occurs because of how modern routers forward data packets and the mechanism they implement to accomplish this as fast as possible. Most modern routers perform data routing in hardware using specialized integrated circuits called ASICs. When data becomes fragmented, instead of being forwarded in hardware using the ASICs, the packets get process switched, which means they are forwarded based on an interrupt system using the regular CPU on the router. Data delivery at the network layer is best effort, as reliable transmission is not required at this layer. As we'll see shortly, reliable transmission is handled at the next higher layer, the transport layer.

The **transport layer**, as the name suggests, is responsible for end-to-end transport of data from the source to the destination of the data traffic. It segments data that it receives from the layer above and

reassembles it into a stream of data on the receiving device. This layer specifies several mechanisms so that data is reliably transmitted between devices. Error detection and recovery mechanisms and information flow control ensure reliable service. This is also where the delineation between the application and the network is usually done. The transport layer and the network, data link, and physical layers make up the data transport layers. The main purpose of these layers is to reliably transfer the data from the sending device to the receiving device. The layers on top of the transport layer—the session, presentation, and application layers—as you will soon see, are concerned more with application issues and making sure data is formatted the right way. Two types of protocols are defined at this layer: connection-oriented and connectionless protocols.

A connection-oriented protocol establishes an end-to-end connection between the sender and the receiver, keeps track of all the segments that are being transmitted, and has a retransmission mechanism in place. Retransmissions happen in cases in which some of the segments that are sent are lost on the way and don't get to their destination or become corrupted and cannot be recovered at the receiver. A system of acknowledgments is in place to signal successfully receiving the data and availability to receive additional data. A sliding window mechanism dynamically adjusts the amount of data that is transmitted on the wire before the sender is waiting for an acknowledgment from the receiver. Large data transfers, SSH terminal sessions, and World Wide Web and email sessions use connection-oriented protocols at the transport layer. Although not defined with the OSI specification, Transmission Control Protocol (TCP) is the most common connection-oriented protocol at the transport layer.

A connectionless protocol does not establish an end-to-end connection between the sender and the receiver.

This type of data transmission is mostly used for real-time traffic such as voice and video, in which it is better to ignore the data segments that are lost or corrupted on the way than to stop the transfer of data and ask for retransmission. For example, in the case of a voice over IP (VoIP) call, the end-user experience is much better if the network just ignores a few milliseconds of lost data that the human ear can compensate for anyway than if the network stops the flow of data, queues the data already received at the receiver, retransmits the missing few milliseconds, replays the message, and then dequeues the data that was waiting for playback at the receiver. User Datagram Protocol (UDP) is the most common connectionless protocol used at the transport layer.

The **session layer** establishes, manages, and terminates sessions between two communicating devices. On top of the end-to-end communication channel that the transport layer establishes, the session layer has the role of keeping track of whose turn is to transmit data, and it ensures that the same operation is not performed at the same time by different clients or even by resuming failed transmissions. Applications that implement the Remote Procedure Call (RPC) framework use the session layer extensively.

The **presentation layer** ensures that the information from the application layer of the sending device is readable and interpretable by the application layer of the receiving device. The presentation layer is not focused on the moving of bits between devices as much as it is focused on how the data being transferred is organized and the syntax and semantics used for this data. For example, if different encodings for text-based exchange of information are used on the sending and receiving devices (for example, ASCII versus EBCDIC), the presentation layer uses a common encoding format for translating between the two encodings and makes sure

data at the application layer is displayed correctly on both systems. Also at this layer, the data is differentiated and encoded based on the type of information it contains, such as text files, video files, binary files, and so on.

The **application layer** is the OSI layer that is closest to the user and contains all the applications and protocols that the users are interacting with, such as email, file transfer, and video streaming. One application layer protocol that is extensively used is Hypertext Transfer Protocol (HTTP), which is at the heart of the World Wide Web. Whenever a website is accessed on a web server, the data between the client browser and the web server is carried over HTTP.

The OSI model didn't find much success when it was initially developed, and it was not adopted by most network equipment vendors at the time, but it is still used for education purposes and to emphasize the importance of separating data transmission in computer networks into layers with specific functions.

The TCP/IP Model

A much more successful implementation of the layered architecture to data transmission and device networking is the TCP/IP (Transmission Control Protocol/Internet Protocol) reference model, also referred to as the *Internet Protocol suite*. Created around the same time as the OSI model, the TCP/IP reference model was adopted and implemented by most networking vendors back in those days and has over time become the reference model for the Internet.

The Internet Protocol suite and the main protocols that make up the model, TCP and IP, started as a research project by the U.S. Department of Defense through a program called Defense Advanced Research Projects Agency (DARPA) in the 1960s. It was the middle of the

Cold War, and the main purpose of the project was to build a telecommunications network that would be able to withstand a nuclear attack and still be able to function if any of the devices making up the network were destroyed or disabled.



Much like the OSI model, the TCP/IP reference model uses a layered approach (see [Figure 16-2](#)). Each layer provides the functionality and features specified in the reference model and serves the layer above it.

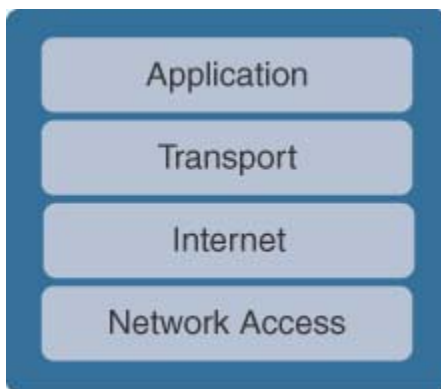


Figure 16-2 *TCP/IP Reference Model Layers*

The network access layer in the TCP/IP model corresponds to the physical and data link layers in the OSI model, and it provides the same services to the layer above. It deals with moving data packets between the Internet layer interfaces of two devices connected on the same link. It specifies the physical characteristics of serial lines, Ethernet, and wireless links, such as voltage levels, data transmission rates, maximum transmission distance, and physical connectors. The network layer defines how data is formatted for transmission on the physical link, how the data packet is received from the Internet layer for transmission by adding a header and footer, how the data frames are transmitted over the physical medium, how access to the network is

controlled, and which device can transmit at which point. Switching and switches, as described later in this chapter, operate at the network access layer.

The Internet layer of the TCP/IP reference model corresponds to the network layer of the OSI model in terms of functions and characteristics. It is responsible for transmitting data across disparate and distant networks in a process called *data routing*. *Routers* are devices that operate at the Internet layer and perform the routing function. As you might have guessed from the name of the reference model, Internet Protocol (IP) is the main protocol operating at the Internet layer. The IP protocol performs two basic functions:

- Device addressing and identification
- Data packet routing

Devices connected to a TCP/IP network are identified by their IP addresses. Two addressing solutions are currently supported on the Internet: IP version 4 (IPv4) and IP version 6 (IPv6). IPv4 uses addressing based on 32 bits, which means that approximately 4 billion devices can be uniquely identified on the network. As you can imagine, there are more than 4 billion endpoints connected to the Internet today. Temporary solutions to address the lack of IPv4 addresses have been developed over the years, including private IP address subnets and Network Address Translation (NAT), both of which are discussed later in this chapter. The permanent fix to this problem was the development and standardization in 1998 of IPv6, which uses 128-bit addresses and is able to provide unique addresses for a gigantic number of endpoints and devices. Both IPv4- and IPv6-addressed devices are currently supported and connected to the Internet, but IPv4 is slowly being phased out.

The second function of the Internet layer is data packet routing. Routing packets means forwarding data from its source toward the destination through a network of

interconnected routers. Through this functionality, the Internet layer makes possible internetworking and the connection of different IP networks, essentially establishing the Internet. The packet forwarding at the Internet layer is best effort and unreliable, and any error correction and retransmission mechanisms required have to be implemented at higher layers. The Internet layer is agnostic to the data structures and the operations that the layer above it is implementing. If we go back to the postal mail analogy, the Internet layer acts as the postal service infrastructure and the mail carriers who deliver the letters and packages without knowing what they actually contain. In the same way, the Internet layer receives data segments from the transport layer and carries them all the way to the destination without knowing what the actual data is.

The transport layer is at the core of the TCP/IP reference model. It establishes end-to-end data channels over which applications exchange data, regardless of the structure of the user data and the underlying network. Since the Internet layer is best effort, the protocols at the transport layer provide flow control, congestion control, segmentation, and error control. Differentiating between different applications running on the same host occurs at the transport layer through port numbers. The transport layer port is a 16-bit logical construct allocated for each of the communications channels an application requires. The ports in the range 0 to 1023, called *well-known ports* or *system ports*, are used for system processes that provide widely used types of network services. For example, web servers that implement HTTP are by default listening for connections on port 80 for regular HTTP traffic or on port 443 for secure HTTP traffic via HTTP Secure (HTTPS). End-to-end connections at the transport layer can be categorized as either connection oriented (usually implemented using TCP) or connectionless (using UDP).

TCP uses several different technologies to ensure reliable bit stream delivery. First of all, data is delivered in the order in which it was sent, and TCP has mechanisms of keeping track of each data segment sent. A dynamic windowing system controls traffic congestion and ensures that the sender doesn't overwhelm the receiver with too much data. If packets are lost on the way to the destination or become corrupted in transit, TCP handles the retransmission of this data. If the data becomes duplicated in transmission, the duplicate data is discarded at the receiver.

UDP is a connectionless datagram protocol that is used in situations where timely data delivery is more important than reliability. It is a best-effort, unreliable protocol that implements a basic checksum mechanism for error detection. UDP is typically used for applications such as streaming media, voice, and video.

The TCP/IP model transport layer corresponds roughly to the OSI model transport layer described earlier in this chapter.

The application layer, as the name suggests, contains all the end-user applications that need to exchange data over the network. The application layer protocols exchange data on top of the end-to-end, sender-to-receiver connection that is established by the lower-layer protocols. There are a large number of application layer protocols, including the following:

- Hypertext Transfer Protocol (HTTP) is used for transferring web pages between web browsers and web servers.
- File Transfer Protocol (FTP) is used for transferring files between a client and a server.
- Dynamic Host Configuration Protocol (DHCP) is used to dynamically assign IP addresses to devices on a network.

The application layer in the TCP/IP reference model corresponds to the session, presentation, and application layers of the OSI model.

Key Topic

In order to be able to communicate between devices using a layered model, the protocols running at each layer on the source host must be able to communicate with their counterparts at the destination host. The data that is being exchanged at each layer is called a protocol data unit (PDU). Based on the layer processing the data, the PDUs have different names:

- At the application layer, the general term for the PDU is *data*.
- At the transport layer, the PDU is called a *segment*.
- At the Internet layer, the PDU is called a *packet*.
- At the data link layer, the PDU is called a *frame*.

Figure 16-3 shows the PDUs at each layer and the flow of data from sender to receiver.

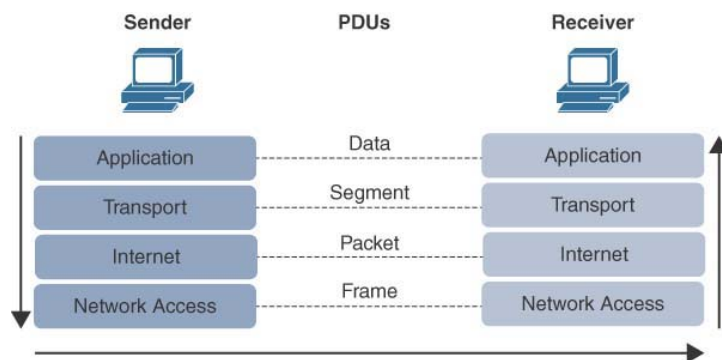


Figure 16-3 *TCP/IP Reference Model PDUs at Each Layer*

As data moves down the layers at the sending host, it gets processed at each layer. This process is called *encapsulation*, and there is a corresponding process at the destination host called *de-encapsulation*. The encapsulation process consists in adding extra protocol information at each layer so that layer-to-layer communication can take place. Starting at the top, the user data generated in the application layer gets passed down to the transport layer for transmission. The

protocol running at the transport layer takes the data it received from the application layer, adds its header information, and passes the datagram PDU to the Internet layer. Similarly, the Internet layer adds its own header and passes the packet PDU to the data link layer. The data link layer takes the data from the Internet layer and builds a frame PDU that is ready to be sent on the physical medium by adding its own protocol header and a frame check sequence (FCS) footer. The role of the FCS footer is to ensure that the data is transmitted without errors. At each hop along the way to the destination, a checksum of the whole frame is computed and added to the FCS footer. If the computed checksum is different from the checksum in the FCS footer, the receiving device knows an error occurred on the transmission path. At the data link layer, the user information with all the header and footer information is finally transmitted on the physical medium.

As the data is received at the destination host, it goes through a similar process but in the reverse order. This process, called *de-encapsulation*, consists of removing the header and footer information at each layer and passing the data to the layer above until it gets to the application layer, where it is interpreted by the application protocol. Figure 16-4 shows the processes of encapsulation at the sender and de-encapsulation at the receiver.

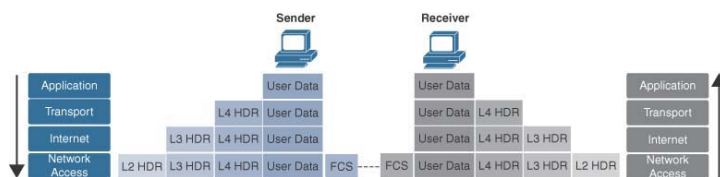


Figure 16-4 Encapsulation and De-encapsulation

SWITCHING CONCEPTS

Data frame switching is a critical process in moving data traffic from a source network endpoint to a destination

endpoint within a local-area network (LAN). This section introduces several networking concepts: Ethernet, MAC addresses, VLANs, and switching.

Ethernet

A LAN is a network that is confined within a building or campus. These types of networks start on the smaller side with home networks, where end-user devices such as personal computers, smartphones, laptops, and printers connect—usually through a wireless connection—to a local home network. The main characteristic of these networks is that the data traffic between the devices connected to the LAN stays local and can be transferred between these devices by a *switch*, which is a network device that operates at the data link layer and enables direct communication between devices connected to a LAN.

The most common LAN technology is Ethernet. Ethernet is a data link layer protocol defined by the IEEE. The Ethernet protocol encompasses guidelines and standards that specify cabling and signaling formats at the physical and data link layers. For example, Ethernet standards specify characteristics for different types of cables and network interface ports, and they also specify voltage levels and the maximum distance the signal can travel before it becomes too attenuated to be successfully received. Over time, the Ethernet protocol has evolved from transfer rates of 10 Mbps to 100 Gbps and even 400 Gbps. The transmission medium has also evolved from coaxial cable to twisted-pair cable, fiber optics, and wireless.

As mentioned previously, the PDU at the data link layer is called a *frame*. The Ethernet protocol specifies the format shown in [Figure 16-5](#) for the Ethernet frame, which includes the following fields:

Field Length (Bytes)	8	6	6	2	46-1500	4
Typical Ethernet Frame	Preamble	Destination Address	Source Address	Type	Data	FCS

Figure 16-5 *Ethernet Frame Format*

- **Preamble:** This field consists of 8 bytes of alternating 1s and 0s that are used to synchronize the signals of the sender and receiver.
- **Destination Address:** This field contains the address of the receiver.
- **Source Address:** This field contains the address of the source device.
- **Type:** This field contains a code that identifies the network layer protocol.
- **Data:** This field contains the data that was received from the network layer and that needs to be transmitted to the receiver.
- **Frame Checksum Sequence (FCS):** This 4-byte field includes a checksum mechanism to ensure that the frame has been transmitted without corruption.

MAC Addresses

Media Access Control (MAC) addresses are used to enable communication between devices connected to a local network. Several types of MAC addresses are used to accommodate the different types of network communications. There are three major types of network communications:

- **Unicast:** In this type of communication, data frames are sent between one specific source and addressed to one specific destination. This type of transmission has one sender and one receiver, and it is the most common type of traffic on any network.
- **Broadcast:** In this type of communication, data frames are sent from one source address to all other addresses connected on the same LAN. There is one sender, but the information is sent to all devices connected to the network.
- **Multicast:** In this type of communication, information is sent from one device to a group of devices or clients. In order for the clients to receive the multicast data, they have to be members of a multicast group. Whereas broadcast is used to transmit data to all the clients on the network, multicast is used to transmit data to just a subset of those clients.

In order to distinguish between these three types of network communications, the IEEE has defined MAC addresses for each type:

- When the least significant bit of a MAC address's first byte is 0, it means the frame is meant to reach only one receiving NIC, which means unicast traffic.
- When the least significant bit of the first octet of a MAC address is 1, it means the frame is a multicast frame, and the receiving NICs will process it if they were configured to accept multicast MAC addresses. An example of a multicast MAC address that is used by Cisco Discovery Protocol (CDP) is 01-00-0C-CC-CC-CC.
- When all the bits are set to 1, it means the frame is a broadcast frame, and it is being received by all the NICs on that network segment.



Each client that needs to exchange data on an Ethernet network needs to have a unique MAC address so that the data is directed to the proper destination device. The MAC address is burned into the network interface card (NIC) and is also called the burned-in address (BIA) or the physical address of the device. A MAC address has 48 bits organized as 12 hexadecimal numbers. There are several different ways of representing MAC addresses, all of them containing the same information. The following representations of a MAC address are all equivalent:

- 0000.0c59.beef
- 00:00:0c:59:be:ef
- 00-00-0C-59-BE-EF

A MAC address has two components (see [Figure 16-6](#)):

- **Organizationally unique identifier (OUI):** This 24-bit number identifies the manufacturer of the NIC. The IEEE assigns OUIs to NIC manufacturers.
- **Vendor-assigned address:** This 24-bit number uniquely identifies the Ethernet hardware.

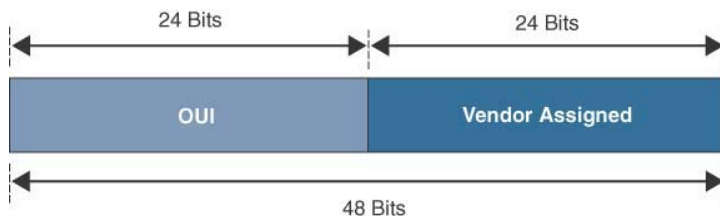


Figure 16-6 MAC Address Components

Virtual Local-Area Networks (VLANs)

A virtual local-area network (VLAN) is a group of devices on an Ethernet network that are logically segmented by function, team, or application. Imagine a university campus network and all the devices that need access to it. It is a common practice to combine into the same logical construct devices that need the same type of access to the network or that perform the same function. For example, the student devices will probably be grouped into the student VLAN, the faculty devices into the faculty VLAN, the finance department devices into the finance VLAN, and so on. On top of the physical network that connects these devices, a software layer is added to separate them into different silos. This is done following the IEEE 802.1q standard, which specifies an additional field in the data link layer frame: the 802.1q header. This header is 4 bytes long and is wedged between the source MAC address field and the Type field, as shown in [Figure 16-7](#).

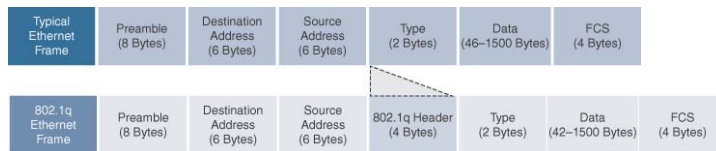


Figure 16-7 Typical Ethernet Frame Versus 802.1q Ethernet Frame



In the 802.1q header, 12 bits are dedicated to the VLAN identifier, so a maximum of 4094 VLANs can be defined on an Ethernet network; the VLANs with IDs 0 and 4095 are reserved. Each VLAN defines its own broadcast domain. A data link layer broadcast domain is defined as the subset of devices on a network that receive Layer 2 broadcast traffic. Layer 2 broadcast traffic stops and is not forwarded further by Layer 3 devices or routers. It is

a best practice to keep the Layer 2 broadcast domain as small as possible. Going back to the university campus example, imagine the thousands of devices connected to the network as being part of the same broadcast domain. In addition, because there is no Time to Live (TTL) field in the Ethernet frames, and redundant connections are required between switches, broadcast storms might bring down the network and make it unusable. Creating VLANs and assigning the devices to specific VLANs limits the broadcast domain, reduces the chance of a broadcast storm, and logically groups the devices for easier troubleshooting. For each VLAN, a dedicated Layer 3 IP subnet is usually allocated. It is much easier to enforce security policies and limit access between VLANs at a Layer 3 device than it is to accomplish the same on a port-by-port basis on a switch. No matter the physical location in the network, devices can be part of the same VLAN (see [Figure 16-8](#)).

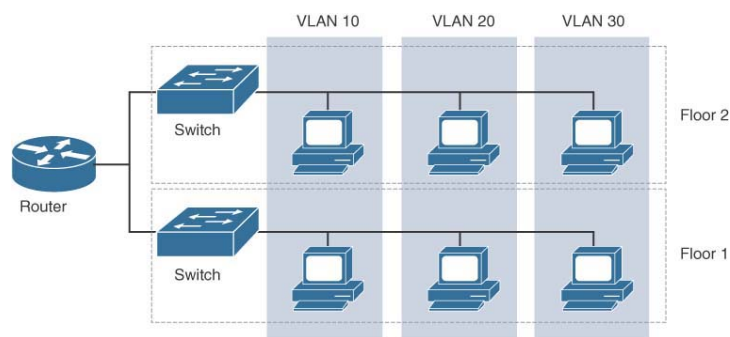


Figure 16-8 *Virtual Local-Area Networks (VLANs)*

VLANs provide network segmentation by reducing the broadcast domains and organizational flexibility by combining devices on a network, based on the needs of the organization.

Switching

Switching is the process through which a data frame is forwarded from its source toward its destination by a Layer 2 device called a *switch*. In a typical LAN, all

devices connect to the network either through an Ethernet port available on the NIC or through a wireless NIC that connects to a wireless access point that is connected to a switch port. In the end, whether wired or wireless, all client devices connect to Ethernet switches. Forwarding of data traffic between devices at Layer 2 is based on the MAC address table that is stored on each switch within the network. The MAC address table is dynamically built and contains a list of all the MAC addresses that the switch has learned and the switch ports on which it learned those addresses.

A MAC address table on a switch might look as shown in [Table 16-2](#).

Table 16-2 MAC Address Table

VLAN	MAC Address	Type	Ports
10	001b.10a0.2500	Dynamic	Gio/1
20	001b.10ae.7d00	Dynamic	Gio/2
30	0050.7966.6705	Dynamic	Gio/3

Based on this table, the switch can forward the Layer 2 data frames toward their destination. If a data frame with MAC destination address 0050.7966.6705 arrives at the switch with this MAC address table, the switch will forward that frame out its GigabitEthernet0/3 port toward the host with that specific MAC address.



The switch dynamically builds and maintains the MAC address table and indicates where each device is

connected in the network. Let's assume that a switch was just rebooted, and its MAC address table is empty. As the devices that are connected to the switch start exchanging data, the MAC address table gets populated with their addresses. From the first device that sends data on the network, the switch looks at the source MAC address in the data frame and records it in the MAC address table, together with the port on which it received the data. Since this is the first MAC address in the table, the switch doesn't have the destination MAC address of the frame, so it floods it over all the ports except the port on which it received the data. As the frame gets flooded throughout the network, eventually it will get to its destination. The destination device replies with its own data frame, in which the source and destination MAC addresses get swapped. All the other devices drop the original frame because it was not destined for them. The switch receives the frame, takes note of the source MAC address, and records it in its table, together with the port on which it was received. This way, the switch dynamically builds its MAC address switching table and can forward traffic between all the devices that are connected to it.

ROUTING CONCEPTS

Routing, or Layer 3 packet forwarding, is the process of selecting a path through a network. To understand routing, you need to understand IPv4 and IPv6 addresses. Then you can learn about the routing process itself.

IPv4 Addresses

The most common protocol at the Internet layer is Internet Protocol (IP). The Internet, the worldwide network connecting billions of devices and users, is built on top of IP. There are currently two versions of IP available: IP version 4 (IPv4) and IP version 6 (IPv6).

Both protocol versions are used in the Internet, and a slow migration toward IPv6 is in progress.



The main role of IP is to uniquely identify devices at the Internet layer in the TCP/IP reference model and Layer 3 in the OSI model. IPv4 uses 32-bit addresses, which means it can theoretically support 2^{32} (just over 4 billion) unique addresses. It is a connectionless protocol, IP data packets are forwarded individually, the delivery of packets is best effort, and there is no retransmission or data recovery functionality specified in the protocol definition. In case of lost or corrupt packets, retransmission and data recovery are left to be implemented by the higher-layer protocols. Since the physical medium and transmission are handled by the data link layer, all those specifications are abstracted from the Internet layer, and IP operates independently of the medium that is carrying the data.

IP addresses are similar to physical addresses of homes and businesses. The same way the address of a house uniquely identifies it so that packages and other mail can be delivered, the IP address uniquely identifies all the devices connected to the Internet so that data packages can be exchanged and delivered. For easier readability for humans, IPv4 addresses are split into four octets, with each decimal value separated by a dot—in a format known as dotted decimal notation. Each octet is made up of 8 bits, and each bit can have a value of 1 or 0. For example, an IP address would take the following form in the IP header: 11000000101010000011000001000001. It is very easy to make mistakes when representing IP addresses this way. The four-octet dotted decimal notation for the same address becomes 192.168.48.65, which is much easier to remember and keep track of.

Table 16-3 shows decimal and binary representations of this IPv4 address.

Table 16-3 IPv4 Address Representation with Decimal and Binary Values

Decimal	192	168	48	65
Octet/binary	110000 00	101010 00	001100 00	010000 01

An IP address consists of two parts:

- **Network ID:** The network address part starts from the leftmost bit and extends to the right. Devices on a network can communicate directly only with devices that are in the same network. If the destination IP address is on a different network than the network the source IP address is on, a router needs to forward the traffic between the two networks. The router maintains a routing table with routes to all the networks it knows about.
- **Host ID:** The host address part starts from the rightmost bit and extends to the left. The host ID uniquely identifies a specific device connected to the network. Although the host ID can be the same between different devices on different networks, the combination of network ID and host ID must be unique throughout the network.

In order to accommodate networks of different sizes, IP addresses are organized into different classes. There are in total five classes of IPv4 addresses. Classes A, B, and C are available for public use; Class D is used for multicast addresses; and Class E is reserved for research and is not available for public use. The Internet Assigned Numbers Authority (IANA) manages the assignment and allocation of IP addresses to Internet service providers (ISPs), which in turn assign IP address ranges to their customers. The five classes of IPv4 addresses are described in Table 16-4.

Table 16-4 Classes of IPv4 Addresses

Address Class	Bit Pattern of First Byte	First-Byte Decimal Range	Host Assignment Range in Dotted Decimal Format
A	0xxxxxx x	1 to 127	1.0.0.1 to 127.255.255.254
B	10xxxxx x	128 to 191	128.0.0.1 to 191.255.255.254
C	110xxxxx	192 to 223	192.0.0.1 to 223.255.255.254
D	1110xxxx	224 to 239	224.0.0.1 to 239.255.255.254
E	11110xxx	240 to 255	240.0.0.1 to 255.255.255.255

Class A uses the first byte for the network ID and the other 3 bytes (or 24 bits) for the host ID. As you can imagine, networks with 2^{24} (more than 16 million) hosts in the same network are nonexistent; technologies such as classless interdomain routing (CIDR) and variable-length subnet masking (VLSM) have been developed to address the wastage of IPv4 addresses with the original definition of classes of IPv4 addresses. The first bit of the first byte in a Class A IP address is always 0. This means that the lowest number that can be represented in the first byte is 00000000, or decimal 0, and the highest number is 01111111, or decimal 127. The 0 and 127 Class A network addresses are reserved and cannot be used as routable network addresses. Any IPv4 address that has a value between 1 and 126 in the first byte is a Class A address.

Class B uses the first 2 bytes for the network ID and the last 2 bytes for the host ID. The first 2 bits of the first

byte in a Class B IPv4 address are always 10. The lowest number that can be represented in the first byte is 10000000, or decimal 128, and the highest number that can be represented is 10111111, or decimal 191. Any IPv4 address that has a value between 128 and 191 in the first byte is a Class B address.

Class C uses the first 3 bytes for the network ID and the last byte for the host ID. Each Class C network can contain up to 254 hosts. A Class C network always begins with 110 in the first byte, meaning it can represent networks from 11000000, or decimal 192, to 11011111, or decimal 223. If an IP address contains a number in the range 192 to 223 in the first byte, it is a Class C address.

The role of the address mask is to identify which portion of the IP address is the network ID part and which portion is the host ID. Similarly to IPv4 addresses, address masks are also 32 bits long and use dotted decimal notation. IPv4 addresses and address masks go together hand in hand. Address masks specify the network ID portion of an IP address by turning the corresponding bit into a 1 and the host ID portion of the IP address by turning the corresponding address mask bit into a 0. The default address masks for the first three classes of IPv4 addresses are as follows:

- **Class A:** The default address mask is 255.0.0.0 or /8, indicating that the first 8 bits of the address contain the network ID, and the other 24 bits contain the host ID.
- **Class B:** The default address mask is 255.255.0.0 or /16, indicating that the first 16 bits of the address contain the network ID, and the last 16 bits contain the host ID.
- **Class C:** The default address mask is 255.255.255.0 or /24, indicating that the first 24 bits of the address contain the network ID, and the last 8 bits contain the host ID.

In each IP network, there are two reserved addresses. The IP address where all the host ID bits are 0 is reserved and is used to identify the network itself. For example, the Class C network address 192.16.1.0 cannot

be assigned to any host on that network because it is used to identify the network. This is how the network is represented in the routing tables of routers throughout the network. The other reserved IP address is the one where all the host ID bits are 1; this address, called the *broadcast address*, is used to send information to all the hosts on the network and cannot be assigned to any device. In the previous example with the Class C network, the broadcast address for that network would be 192.16.1.255. Class C networks can accommodate and uniquely address $256 - 2 = 254$ devices on each network.

The Internet has experienced explosive growth, and it won't be long before IANA runs out of available IPv4 address ranges to assign. A large number of the IPv4 address ranges that were initially allocated were also used in private networks, so IANA decided to create a group of private IPv4 address ranges for those networks. Private IPv4 networks, defined by the Internet Engineering Task Force (IETF) in RFC 1918: *Address Allocation for Private Internets*, are meant to preserve the IPv4 address space by having dedicated networks for private networks. Three blocks of IP addresses (1 Class A, 16 Class B, and 256 Class C networks) are reserved for private, internal use (see [Table 16-5](#)).

Table 16-5 Private IPv4 Addresses

Address Class	Private Network ID	Network Address Range
A	10.0.0.0	10.0.0.0 to 10.255.255.255
B	172.16.0.0	172.16.0.0 to 172.31.255.255
C	192.168.0.0	192.168.0.0 to 192.168.255.255

IP addresses in these ranges are not routable in the Internet. When private networks using these IPv4 addresses need to connect to the Internet, their addresses need to be translated to public, Internet-routable IPv4 addresses. This translation process is called Network Address Translation (NAT), and it is discussed in more detail in [Chapter 18, “IP Services.”](#)

The need to further divide IPv4 address classes, especially the Class A and Class B networks, and to create additional subnetworks became clear as networks started to grow in size and complexity and IPv4 addresses started to run out. As previously mentioned, a Class A network with the default address mask can address more than 16 million devices on the same network. Having more than 16 million devices on one Layer 3 network is theoretically possible but practically impossible. Class B networks have 16 bits dedicated to host IDs, which means each Class B network can address $2^{16} = 65,536 - 2 = 65,534$ devices (with two addresses reserved for the network address and the broadcast address). These networks also are in need of further subnetting. Subnetting is the process of borrowing bits from the host ID portion of a network and transforming them into network ID bits.

Classless interdomain routing (CIDR) goes beyond the class limits and the default address masks and represents a way of specifying the subnet mask for an IP address. Initially when the classes of IPv4 addresses were defined, the default masks for each class were already implied and were not even mentioned when assigning networks for customer consumption. As the available IPv4 addresses became more and more scarce, especially for Class A and Class B networks, CIDR was introduced as a way to capture the transition from a class-based Internet to a classless one. In a classless Internet, it is no longer feasible to assume that the address mask for a Class A network is 255.0.0.0 or /8 or for a Class B network that

is 255.255.0.0 or /16. The purpose of the / notation is to specify how many bits in the subnet mask are dedicated to the network ID. Network addresses can be subdivided further to create *subnets*. This is accomplished by using variable-length subnet masking (VLSM) to indicate how many bits in an IPv4 address are dedicated to the network ID and how many bits are dedicated to the host ID. As an example, the 10.0.0.0 private network can be further subdivided, and new subnets can be created out of it. Instead of having one network with more than 16 million hosts, which is not helpful in real life, 65,536 subnets with 254 hosts—each similar to 10.0.0.0/24, 10.0.1.0/24, 10.0.2.0/24, and so on—can be created out of the 10.0.0.0/8 network. This is done by borrowing 16 bits from the 24 bits that are dedicated to host IDs and transforming them into network ID bits. By applying a different subnet mask—using /24 instead of /8—we can use VLSM and CIDR notation to adjust the host portion of the network address.

Let's take a practical example and create new subnets for a fictitious enterprise called Acme, Inc. The network administrator is planning on using the 192.168.0.0/24 network to address all the devices that are connected to the Acme, Inc. enterprise network. As you would expect, there are several departments in the Acme, Inc. world, and there are different numbers of employees in each department. Let's assume that the total number of departments is 8 and include the typical corporate organization: engineering, support, human resources, finance and accounting, and so on. Acme, Inc. is still a small company, and there are not more than 20 devices connected to the network in each of the 8 departments. The network administrator would like to segment each department into its own network so that security can be more easily enforced by using Layer 3 access control lists (ACLs) and also to limit the broadcast domain for each department. The 192.168.0.0/24 CIDR notation indicates that for this subnet, there are 24 bits reserved

for the network ID, and the rest of the bits, 8, are reserved for the hosts. In order to create 8 subnets out of the original 1, 3 bits ($2^3 = 8$) would have to be borrowed from the host portion and dedicated to the new network IDs. The subnet mask in this case changes as follows:

- **Original subnet mask:** 11111111.11111111.11111111.00000000 (/24)
- **New subnet mask:** 11111111.11111111.11111111.11100000 (/27)

There are still 5 bits left for hosts on each subnet, which results in $2^5 = 32 - 2$ (1 reserved for the network address and 1 reserved for the broadcast address) = 30 usable IP addresses for each subnet. This is more than needed to accommodate the 20 devices for each department. The new subnets look like those shown in [Table 16-6](#); each one has a /27 mask:

Table 16-6 Creating Additional Subnets from 192.168.0.0/24

Subnet	Network Address	Broadcast Address	Available Host Address Range
Subnet 1	192.168.0.0	192.168.0.31	192.168.0.1 to 192.168.0.30
Subnet 2	192.168.0.32	192.168.0.63	192.168.0.33 to 192.168.0.62
Subnet 3	192.168.0.64	192.168.0.95	192.168.0.65 to 192.168.0.94
Subnet 4	192.168.0.96	192.168.0.127	192.168.0.97 to 192.168.0.126
Subnet 5	192.168.0.128	192.168.0.159	192.168.0.129 to 192.168.0.158
Subnet 6	192.168.0.160	192.168.0.191	192.168.0.161 to 192.168.0.190

Subnet 7	192.168.0 .192	192.168.0 .223	192.168.0.193 to 192.168.0.222
Subnet 8	192.168.0 .224	192.168.0 .255	192.168.0.225 to 192.168.0.254

New subnets can be easily created to accommodate any number of devices. In the previous example, each subnet can address up to 30 devices, but the subnets can be further subnetted, if needed. By borrowing additional bits from the host section, subnet masks of /28, /29, and even /30 can be specified to create new subnets for $2^4 = 16 - 2 = 14$, $2^3 = 8 - 2 = 6$, or $2^2 = 4 - 2 = 2$ hosts in each subnet. /30 subnets are especially useful on point-to-point serial connections between two routers—for example, where there are only 2 devices connected and there is no point in using larger subnets as that would only lead to wasted addresses. [Table 16-7](#) provides a breakdown of the subnet masks from /24 to /32, their binary representations, and the number of subnets and hosts for each subnet.

Table 16-7 Subnet Mask Representations and the Number of Subnets and Hosts for Each

Decimal Subnet Mask	Binary Subnet Mask	CIDR	Subnets	Hosts per Subnet
255.255.255.255	11111111.11111111.11111111.11111111	/32	1	0
255.255.255.254	11111111.11111111.11111111.11111110	/31	2	0
255.255.255.252	11111111.11111111.11111111.11111100	/30	4	2

255.255.255.248	11111111.11111111.11111111.11110000	/29	32	6
255.255.255.240	11111111.11111111.11111111.11110000	/28	16	14
255.255.255.224	11111111.11111111.11111111.11100000	/27	8	30
255.255.255.192	11111111.11111111.11111111.11000000	/26	4	62
255.255.255.128	11111111.11111111.11111111.10000000	/25	2	126
255.255.255.0	11111111.11111111.11111111.00000000	/24	1	254

IPv6 Addresses

IPv4 has valiantly served the Internet from the early days, when only a handful of devices were interconnected at college campuses in the United States for research purposes, to today, when billions of devices are interconnected and exchanging massive amounts of data. The longevity of IPv4 and the addressing scheme it defines is a testament to how robust and scalable IP actually is. Even so, IPv4 faces limitations, including the insufficiency of 32-bit addresses, the need for the IP header to be streamlined and simplified, and the need for security embedded in the protocol rather than implemented as an afterthought.

IPv6 is an evolutionary upgrade to IP that is meant to address most of the shortcomings of IPv4. IPv6 is defined in RFC 2460, *Internet Protocol, Version 6 (IPv6) Specification*, issued by the IETF. Several other RFCs that have been created following the initial definition describe the architecture and services supported by IPv6. IPv6 addresses are 128 bits long instead of 32 bits. The 128 bits provide 3.4×10^{38} addresses; this is the equivalent of a full IPv4 addressing space for each living person on Earth. Currently both IPv4 and IPv6 addresses are supported on the Internet, and a gradual transition to only IPv6 is happening, but we won't be there until sometime far in the future. Several migration technologies have been developed to ease the transition from IPv4 to IPv6. The dual-stack method involves having both IPv4 and IPv6 addresses configured on the same interface and seems to be the most popular option. Tunneling mechanisms are also available for IPv4 tunneling over IPv6 networks and IPv6 tunneling over IPv4 networks.



IPv6 brings several improvements compared to IPv4, including the following:

- Larger address space means improved global reachability and better aggregation of IP prefixes, which leads to smaller routing tables, which leads to faster routing.
- End-to-end communication is now possible, as there is no need for technologies such as Network Address Translation (NAT) or Port Address Translation (PAT) that have been developed to address the lack of public IPv4 addresses.
- Address autoconfiguration allows hosts and devices connecting to the network to easily and automatically obtain IPv6 addresses. This is especially important for mobile devices as they roam between different networks; autoconfiguration makes the transition seamless.
- The coexistence of multiple addresses on the same interface increases reliability and load balancing.
- A simplified header means better routing efficiency and increased forwarding performance.

- No broadcast addresses and no broadcast traffic mean no broadcast storms, improving the reliability of the whole network.
- Security is increased, thanks to built-in IPsec capabilities.

IPv6 addresses can be displayed as 32 hexadecimal characters split into 8 sections, as follows:

- Colons separate entries $x:x:x:x:x:x$, where x is a 16-bit hexadecimal field.
- Hexadecimal characters are case insensitive.
- Leading zeros in a field are optional.
- Successive fields of zero can be represented as $::$ once per address.

An example of an IPv6 address is

2031:0000:120F:0000:0000:09Co:9A73:242C.

Following the guidelines just specified, this address can also be written as follows:

2031:0000:120F::<9Co:9A73:242C (because successive fields of zero can be represented as $::$)

2031:0:120F::<9Co:9A73:242C (because leading zeros in a field are optional)

Figure 16-9 shows a simple illustration of how IPv6 addresses and subnetting work.

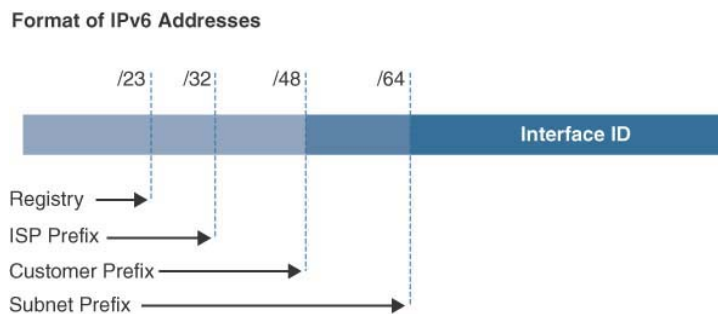


Figure 16-9 IPv6 Subnetting

The Internet Assigned Numbers Authority (IANA) is the organization that manages the assignment of IPv6 addresses at a global level. IANA assigns /23 blocks of IPv6 addresses to the registries that manage IPv6 addresses at a regional level, such as the Asia-Pacific

Network Information Center (APNIC) or the American Registry for Internet Numbers (ARIN). The regional registries assign /32 blocks of IPv6 addresses to the Internet service providers (ISPs) that operate in their geographic areas. The ISPs, in turn, assign /48 IPv6 subnets to their customers, which can subnet further and create their own subnets, usually down to a /64, giving them up to 65,536 IPv6 subnets that they can assign to their sites.

The interface ID portion of an IPv6 address is usually based on the MAC address of the device, following the EUI-64 specification. This specification creates the 64 bits needed for the interface ID portion of the address by splitting the 48-bit MAC address right down the middle and inserting the 16-bit 0xFFFE between the OUI and the vendor-assigned bits.

The hierarchical distribution of IPv6 addresses allows for better aggregation of routes, which leads to smaller Internet routing tables and faster routing.



There are several different types of IPv6 addresses:

- Global
- Link-local
- Multicast
- Loopback
- Unspecified

Global IPv6 addresses are the equivalent of public unicast IPv4 addresses. Global IPv6 addresses are assigned in a controlled fashion so that they can be easily aggregated at all levels, registry, ISPs, and customers. Currently IANA is assigning IPv6 addresses that start with the binary value 001 (2000::/3). This represents one-eighth of the total IPv6 address space. A global

unicast IPv6 address typically consists of a 48-bit global routing prefix in the most significant portion of the address, followed by a 16-bit subnet ID and 64 bits dedicated to the interface identifier. Addresses with prefixes in the range 2000::/3 to E000::/3 are required to have 64-bit interface identifiers that follow the EUI-64 format mentioned previously.

Link-local addresses in IPv6 are a new concept compared to IPv4. These addresses refer only to a particular physical link and are not forwarded by routers. They are used only for local communication between devices on the same physical network segment. Link-local addresses are part of the link-local prefix FE80::/10 and are extensively used in IPv6 routing protocols for neighbor and router discovery, as well as in automatic address configuration. They are usually dynamically created on all IPv6 interfaces by adding the 64-bit interface ID to the FE80::/10 prefix.

IPv6 multicast addresses are part of the FFO0::/8 prefix. There is no broadcast traffic in IPv6, so that functionality is taken by multicast and anycast traffic in IPv6. There are several reasons broadcast traffic has been left out of IPv6, including the fact that each broadcast packet on the network generates an interrupt on all the devices that are part of the same broadcast domain, even if the traffic is not directly meant for them; in addition, broadcast storms can bring down entire networks. With IPv6, it was decided to implement the one-to-many type of data traffic by using multicast. Multicast increases the efficiency of delivering data from one device to many, and with IPv6, it gets a larger address space, too. A data packet sent to a multicast address is delivered to all interfaces identified by the multicast address. Multicast addresses also have scope. The scope of a multicast address can be that of a node, a link, a site, an organization, or a global scope and is represented by a scope parameter of 1, 2, 5, 8, or E, respectively. As an

example, the multicast address with the prefix FF02::/16 is a multicast address with a link scope. All devices implementing IPv6 are required to join the all-nodes multicast group with the FF02:0:0:0:0:0:0:1 multicast address. IPv6 routers must also join the all-routers multicast group FF02:0:0:0:0:0:0:2. The scope on both of these multicast addresses is link-local.

Loopback IPv6 addresses are similar to the loopback addresses in IPv4: They are used for testing local traffic on a device. In IPv6 there is only one address dedicated for this purpose: 0:0:0:0:0:0:0:1 (or ::1 when using the zero compression feature in IPv6).

The unspecified address is used for situations when a device doesn't have any IPv6 address assigned and needs to populate the Source field when sending a datagram on the network, such as when obtaining an IPv6 address from a DHCPv6 server. This address is the all-zeros address: 0:0:0:0:0:0:0:0 (or just :: when using the zero compression feature in IPv6).

Anycast addresses are allocated from the unicast global space. Their purpose is to accommodate the one-to-nearest data traffic requirement. In this case, multiple devices share the same unicast address, and data traffic that is destined for the anycast address is routed through the network to the closest device that has that anycast address configured on one of its interfaces. In this case, of course, all the anycast nodes have to provide a similar, uniform service. Anycast is suitable for load balancing and content delivery services. An example of a service that can be implemented using anycast addresses is Domain Name System (DNS). DNS is covered in [Chapter 18](#); for the purposes of this example, you just need to know that DNS is a critical component of the Internet and is mainly used to resolve domain names to IP addresses. Configuring several DNS servers on a network with the same anycast address ensures a consistent DNS

configuration on all the devices in the network. Instead of having to configure several different DNS server IP addresses for redundancy purposes, in IPv6, a single address, the anycast address, can be configured on all DNS clients. As long as at least one of the DNS servers that have been configured as part of the anycast group is online, DNS redundancy is guaranteed.

Routing

Routing is the process of selecting a path through a network. This functionality is performed by routers, which are devices made to interconnect networks, find the best paths for data packets from their source toward their destination, and route data based on a routing table. As switches have a critical role at Layer 2 in connecting devices in the same physical area and creating local-area networks, routers have a critical role at Layer 3 in interconnecting networks and creating wide-area networks. The main functions of a router are the following:

- Path determination
- Data packet routing or forwarding

Path determination is the process through which routers look up information in their routing tables to determine where to forward the data packets received on their interfaces. In a typical scenario, a router has at least two interfaces—and usually more. Each interface connects to a Layer 3 network, which means the interface on the router is configured with an IP address in that network. In most cases, the router acts as the default gateway for that Layer 3 network, meaning that any traffic that is not destined for the local network will be forwarded to the router, and the router will forward it further toward its destination. Each router maintains its own routing table, which contains a list of all the destinations or networks that are known to the router and how to reach those destinations. When receiving a packet on one of its

interfaces, the router checks the destination IP address in the data packet header and looks up the best match for that destination in its routing table. If the destination IP address matches one of the networks in the routing table, it means that either the network is directly connected to the router or it can be reached via another router that is directly connected, and it becomes the next-hop router toward the final destination of the data packet. If there is no match for that destination IP address in the routing table, the router sends the data packet to the default route, and if there is no default route, the router drops the data packet.

After determining the correct path for the packet, the router forwards the packet through a network interface toward the destination. [Figure 16-10](#) shows a simple network diagram.

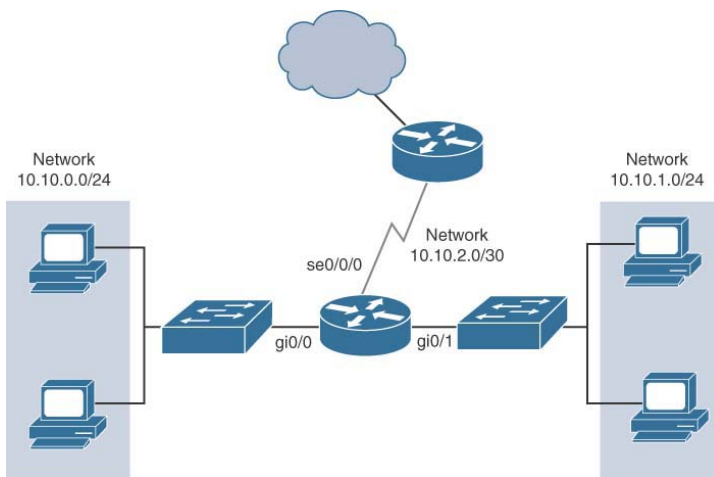


Figure 16-10 Simple Network Diagram

The routing table for the network in [Figure 16-10](#) might look like [Table 16-8](#).

Table 16-8 Routing Table

Network	Interface or Next Hop

10.10.0.0/24	directly connected: Gi0/0
10.10.1.0/24	directly connected: Gi0/1
10.10.2.0/30	directly connected: Se0/0/0
0.0.0.0/0	via 10.10.2.2 (next-hop router)

Each row in the routing table lists a destination network and the corresponding interface or next-hop address. For directly connected networks, the router has an interface that is part of that network. For example, let's assume that a router receives a data packet on its GigabitEthernet0/1 interface with destination IP address 10.10.0.15. The router looks up the destination address in its routing table and decides to route the packet out its GigabitEthernet0/0 interface toward the destination.

Following the same logic, let's assume next that the router receives a packet on its GigabitEthernet0/0 interface with destination IP address 172.16.0.25. The router performs a routing table lookup and finds that it doesn't have an explicit route for that network, but it does have a default route (0.0.0.0/0) via the next-hop router with IP address 10.10.2.2. Looking up recursively how to reach the 10.10.2.0 network, the router forwards the packet out its Serial0/0/0 interface toward the next-hop router, which should be closer to the destination. Default routes on routers—much like the default gateway configurations on end host devices—are used as a last resort for routing any data packets for which the destination doesn't explicitly exist in the routing table. Default routes are extremely helpful in maintaining a small routing table and hence decreasing the routing table lookups and also decreasing the amount of time needed to decide on which interface to route the traffic, leading to faster routing of data packets. For the majority of routers, a full Internet routing table with hundreds of

thousands of routes is not necessary, and a default route toward the next hop is sufficient.



There are several ways in which routing tables are populated:

- Directly connected networks
- Static routes
- Dynamic routes
- Default routes

We've already discussed directly connected networks, in which the router has an interface configured for that network and is routing traffic for it and the default routes, which are used as a last resort in the event that a more specific route doesn't exist in the routing table.

Static routes are configured manually by the network administrator or the person managing a router. In small networks in which there are not expected to be any changes, static routes can work just fine. In larger networks that are more dynamic in nature, with devices and routers going offline and coming online constantly, static routes do not work very well because the administrator needs to manually change the entries as the changes happen in the network. A much more scalable solution in this case is dynamic routes.

Dynamic routes, as the name implies, are learned dynamically through specialized software applications called *dynamic routing protocols*. Several routing protocols have been developed over the years, including Open Shortest Path First (OSPF), Enhanced Interior Gateway Routing Protocol (EIGRP), Routing Information Protocol (RIP), Border Control Protocol (BGP), and Intermediate System-to-Intermediate System (IS-IS). Each of these protocols has its own

characteristics, advantages, and drawbacks. It is the role of the network administrator to decide which protocol to choose to run on the network and to determine whether a dynamic routing protocol is even necessary in the first place. Medium and large networks run at least one of these protocols on all the routers in the network. The routers running dynamic routing protocols exchange data between them and dynamically build their routing tables based on the information they receive from their neighbor routers. If multiple routing protocols are configured on the routers, there needs to be a way to decide which routes to choose and to populate the routing table with them. Each routing protocol has an administrative distance associated with it that is based on the trustworthiness of the protocol. For example, if a route for the same network is learned through RIP and OSPF, the OSPF route will take precedence as OSPF has a lower administrative distance than RIP. A lower administrative distance means a higher degree of trust in that source—so a higher chance of getting added to the routing table.

Connected routes, meaning the router has an interface configured in a specific network, have the lowest administrative distance (AD), 0. Static routes are preferred next as they usually have an AD of 1. Routing protocols have different administrative distance values; for example, BGP has an AD of 20, and RIP has an AD of 120. These values for AD are default values, and they can be changed if additional tweaking of the routing table is necessary. All dynamic routing protocols mentioned support routing for both IPv4 and IPv6.

As we've seen previously in this chapter, the Internet has become classless, and there are a large number of networks and subnets that can be created from those networks. The same way that a large network address space can be subdivided into small subnets, the reverse process is also available; it is called *supernetting*. This

means that smaller subnets can be combined into a larger network, and instead of having several routing table entries for each individual subnet, a single routing table entry can encompass them all. For example, when an ISP assigns a certain prefix range to one of its customers, the ISP does not need to know all the details of how the customer is implementing subnetting in its internal network. All the ISP needs to know is that the whole prefix is assigned to the customer, and it can be reached via the customer's border routers. Supernetting is critical in maintaining an Internet routing table of a manageable size.

Another important aspect when building a routing table is the longest prefix match concept. Longest prefix match means that when the router is looking in the routing table for a match to route the data packet toward its destination, it does so by trying to match it to the longest prefix it has in the table. For example, let's assume that the routing table has two entries: 10.10.10.0/24 via interface GigabitEthernet0/0 and 10.10.0.0/16 via interface GigabitEthernet0/1. If the router receives a data packet destined for 10.10.10.15, it looks in its routing table, and it will match the longest-prefix network (in this case, 10.10.10.0/24) and route the packet out its GigabitEthernet0/0 interface. If the data packet has destination address 10.10.20.15, the router matches the other entry, 10.10.0.0/16, since that is the longest prefix match it has for that subnet.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. Table 16-9 lists these key topics and the page number on which each is found.



Table 16-9 Key Topics

Key Topic Element	Description	Page Number
Paragraph	OSI model's layered approach	48 5
Paragraph	TCP/IP model's layered approach	48 8
Paragraph	Communication between devices in a layered model	49 0
Paragraph	Client MAC addresses	49 3
Paragraph	VLAN identifier	49 4
Paragraph	MAC address table for a switch	49 5
Paragraph	IP in the TCP/IP reference model and the OSI model	49 6
List	Improvements of IPv6 over IPv4	50 1
List	Types of IPv6 addresses	50 2

List	Ways routing tables are populated	50 6
------	-----------------------------------	---------

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Open Systems Interconnection (OSI)

Transmission Control Protocol/Internet Protocol (TCP/IP)

Media Access Control (MAC)

Logical Link Control (LLC)

User Datagram Protocol (UDP)

Transmission Control Protocol (TCP)

Internet Protocol (IP)

Hypertext Transfer Protocol (HTTP)

File Transfer Protocol (FTP)

protocol data unit (PDU)

frame check sequence (FCS)

organizationally unique identifier (OUI)

virtual local-area network (VLAN)

Internet Assigned Numbers Authority (IANA)

classless interdomain routing (CIDR)

variable-length subnet masking (VLSM)

Asia-Pacific Network Information Center (APNIC)

American Registry for Internet Numbers (ARIN)

Chapter 17

Networking Components

This chapter covers the following topics:

What Are Networks?: This section describes what a network is and introduces the types of networks.

Elements of Networks: This section provides an overview of various networking elements, including hubs, switches, routers, and VLANs.

Software-Defined Networking: This section provides an overview of the fundamentals of the control, data, and management planes.

Over the past several decades, Cisco has built the backbone of what we call the internet today and has developed and created networks big and small, including enterprise networks and service provider networks. In [Chapter 16, “Network Fundamentals,”](#) you learned some networking basics. This chapter covers networking components, such as the following:

- Networking elements such as hubs, switches, and routers
- Network device functions, including the management, data, and control planes
- Software-defined networking

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics,

read the entire chapter. [Table 17-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 17-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
What Are Networks?	1-3
Elements of Networks	4-7
Software-Defined Networking	8-9

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. Ethernet is a _____ network topology.
 1. star
 2. ring
 3. bus
 4. mesh
2. _____-area networks usually use public networks like telephone, cellular, or satellite networks.
 1. Metropolitan
 2. Local

3. Wide
 4. Campus
- 3.** Which of the following is the least secure in terms of privacy of the user?
1. Router
 2. Hub
 3. Switch
 4. Bridge
- 4.** Which of the following is not true of a Layer 3 switch?
1. It uses IP addresses for forwarding.
 2. A VLAN can be implemented by using this type of switch.
 3. A Layer 3 switch creates a single broadcast domain.
 4. A Layer 3 switch usually supports Layer 2 switching as well.
- 5.** Which of the following can be used to segment a physical switch into multiple logical networks?
1. Bridge
 2. WAN
 3. VLAN
 4. Router
- 6.** FIB tables, which contain precomputed routes, reverse lookups, and so on, are used in which type of switching?
1. Process switching
 2. Cisco Express Forwarding
 3. Fast switching
 4. None of the above
- 7.** What is NAT used for?
1. Stopping local traffic from reaching the public network
 2. Mapping private IP addresses to public IP addresses
 3. Detecting traffic patterns
 4. Assigning IP addresses to local network clients
- 8.** In a network controller, which plane is concerned with administrative access to a network device?
1. Control
 2. Management
 3. Data
 4. Hardware
- 9.** In a Cisco SD-WAN solution, what is the main brain of the entire solution that manages the control plane?

1. vManage
2. vSmart
3. vEdge
4. vBond

FOUNDATION TOPICS

WHAT ARE NETWORKS?

NIST CNSSI 4009 defines a network as follows:

information system(s) implemented with a collection of interconnected components. Such components may include routers, hubs, cabling, telecommunications controllers, key distribution centers, and technical control devices.

In other words, a network is formed when two or more systems are connected to each other and are able to communicate with each other. With different types of connections, various types of networks can be built.

Three different factors are used to classify networks:

- Topology
- Standard
- Size or location



Topology refers to how the various nodes or elements connect. [Figure 17-1](#) shows three prominent network topologies:

- **Bus:** In a bus network, all the elements are connected one after the other.
- **Star:** In a star topology, all nodes in the system are connected to a central point.
- **Ring:** A ring topology is very similar to a star except that a token is passed around, and a node can transmit only when it has a valid token.

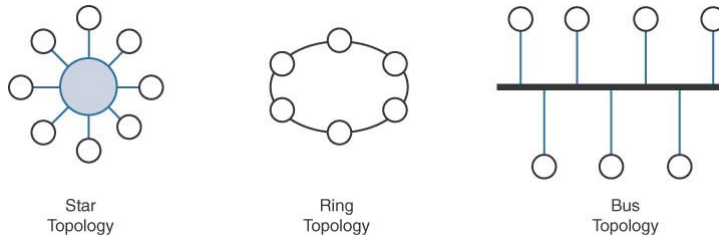


Figure 17-1 *Network Topologies*

Standard refers to the protocol that the network uses. Ethernet is one example of a standard. An Ethernet network uses a star topology with concentrators connecting all the nodes in the network. Ethernet speeds vary from 10 Mbps to 100 Mbps to 1 Gbps. Another standard is Token Ring, which, as the name suggests, is a ring topology type. Token Ring rates vary from 4 Mbps to 16 Mbps.

The *size* of a network is a bit of an arbitrary concept. Network sizes fall into basically four network types:

- **Local-area network (LAN):** A LAN is typically a single network for an office, an enterprise, or a home, and it is located completely within a single facility. [Figure 17-2](#) shows an example of a LAN that consists of various laptops, PCs, servers, and printers.

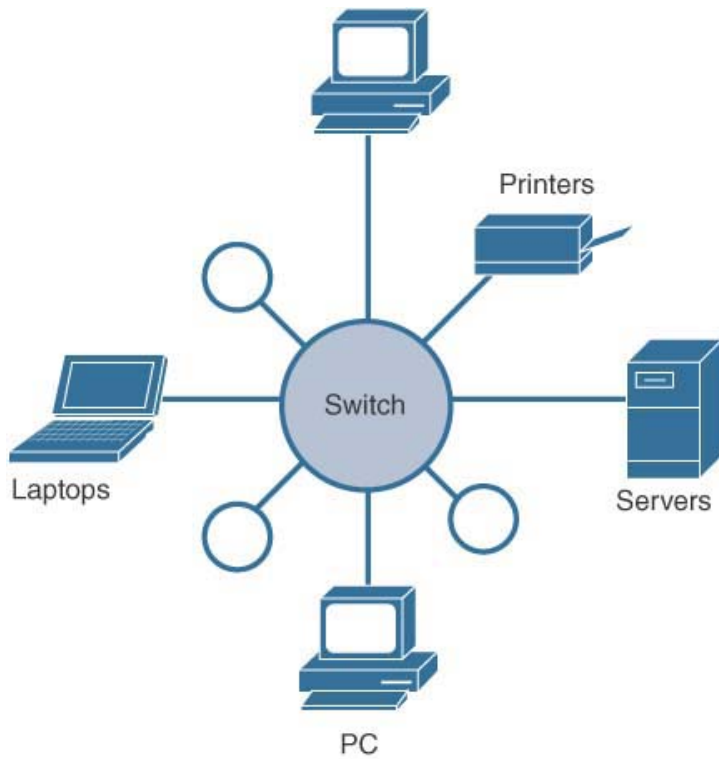


Figure 17-2 *Local-Area Network*

- Campus-area network (CAN):** A CAN consists of two or more LANs within a limited area. CANs help interconnect various buildings or departments within a campus of an enterprise or large office or school. Figure 17-3 shows an example of a CAN that connects various LAN segments.

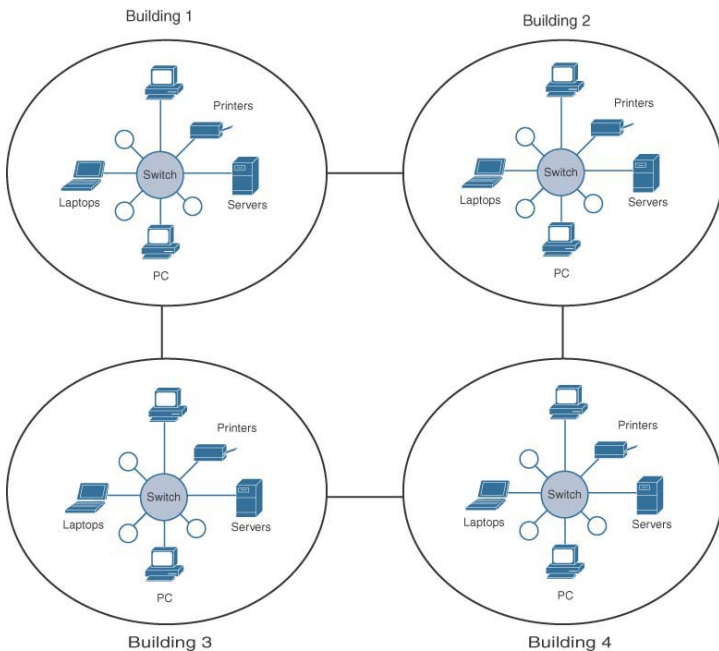


Figure 17-3 *Campus-Area Network*

- **Metropolitan-area network (MAN):** A MAN is a network that is extended to cover multiple locations within a slightly larger distance than a CAN. A classic example of a MAN is the cable network in a city. Figure 17-4 shows an example of a MAN that connects various network segments.

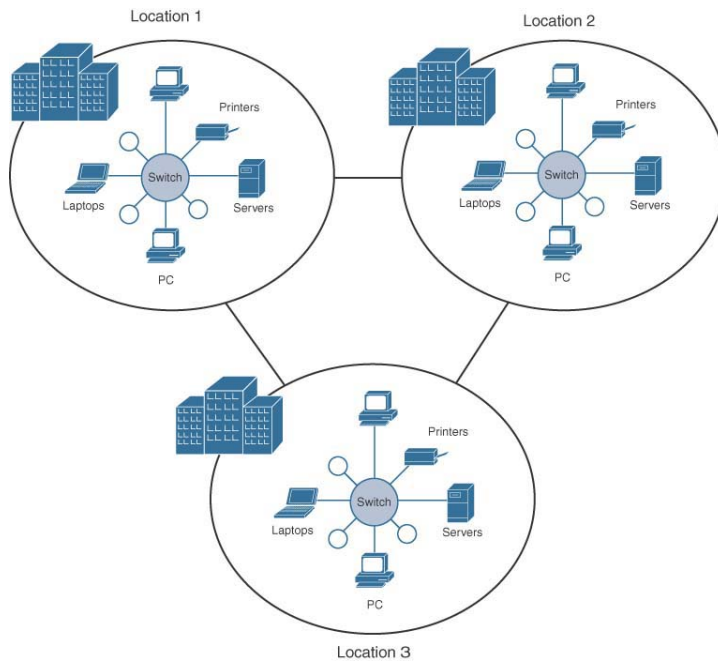


Figure 17-4 *Metropolitan-Area Network*

- **Wide-area network (WAN):** A WAN covers an even larger geographic area than a MAN. The connections are through public networks, such as the telephone system, microwaves, satellite links, or leased lines. Figure 17-5 shows an example of a WAN that connects various remote offices with the headquarters.

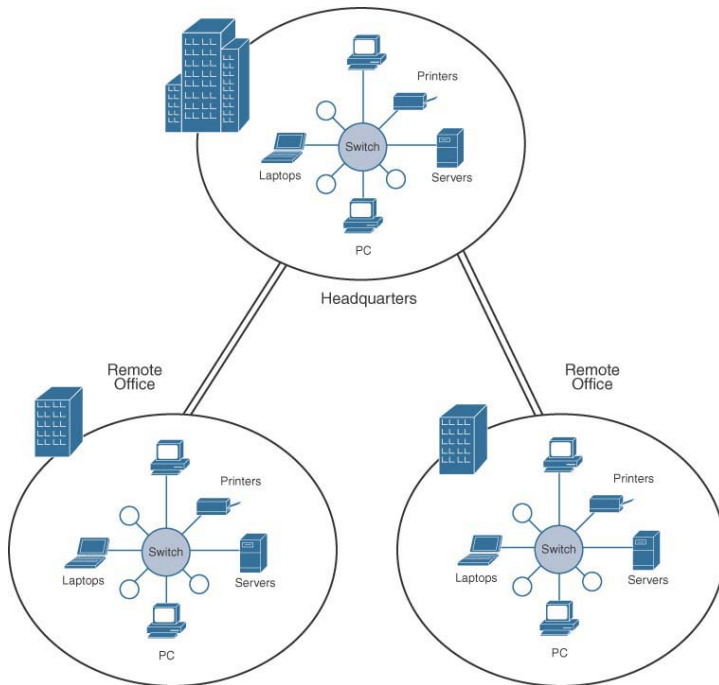


Figure 17-5 *Wide-Area Network*

ELEMENTS OF NETWORKS

Now that you have seen the various types of networks, let's look at what constitutes a network. A network has several basic components:

- Hubs
- Switches
- Bridges
- Routers

The following sections look at each of these elements in detail and also look at the difference between them. In the following sections you will see how to build a simple home network in which you want to connect two or three computers or devices.

Hubs



A hub is the least intelligent and most rudimentary network device. With a hub, traffic or data packets that come in any port are always sent out to all other ports. For example, in [Figure 17-6](#), Laptop 1 wants to communicate to Device 3. The hub replicates and sends the packet from the laptop to each of the ports. Only Device 3 responds back, and that response is again replicated to all ports. Every device connected to the hub “sees” everything. It is up to the devices themselves to decide if a packet is for them and whether they should respond. The hub remains passive and does not really know or understand the traffic that flows through it.

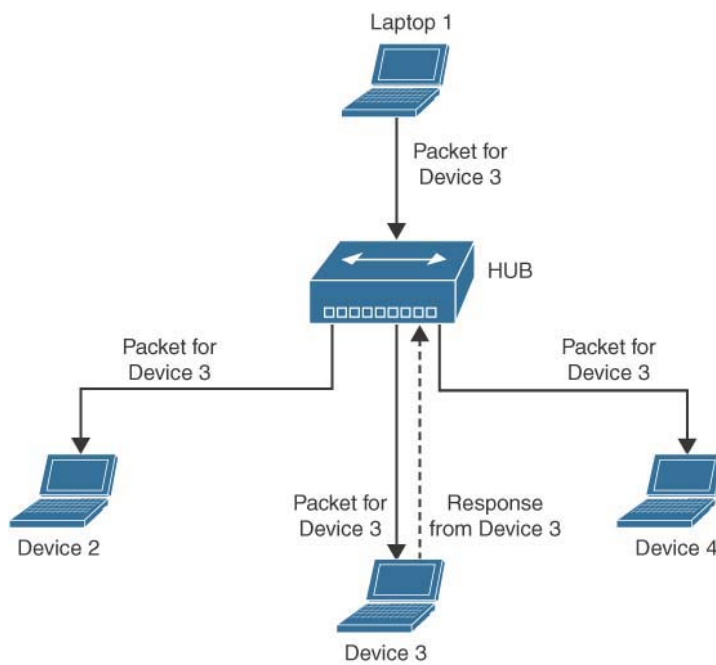


Figure 17-6 *Hub: Every Device Receives Every Packet*

One of the biggest challenges with hubs is that every frame shows up at every device attached to a hub, instead of just showing up at the intended destination. This can be a huge security problem. Another challenge has to do with performance: Because a hub can receive a lot of inbound frames, sending out frames can be difficult and can result in performance issues on the network.

Bridges

A bridge connects two physical network segments that use the same protocol. Each network bridge builds a MAC address table of all devices that are connected on its ports. When packet traffic arrives at the bridge and its target address is local to that side of the bridge, the bridge filters that frame, so it stays on the local side of the bridge.

If the bridge is unable to find the target address on the side that received the traffic, it forwards the frame across the bridge, hoping the destination will be on the other network segment. In some cases, multiple bridges are cascaded.

- [Figure 17-7](#) shows how a bridge connects two LAN segments.

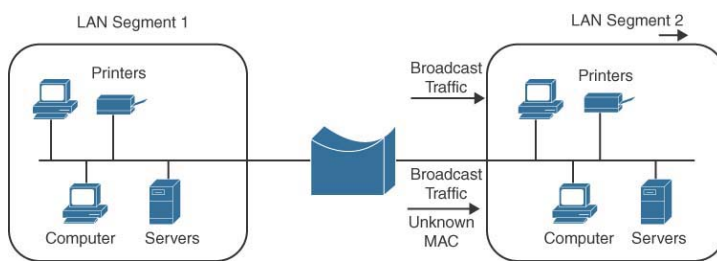


Figure 17-7 *Using a Bridge to Connect Two LAN Segments*

Switches



You can think of a switch as an “intelligent” hub. A switch does what a hub does—but more efficiently and intelligently. It analyzes the traffic as it flows. A switch builds a port-to-MAC address table as devices start communicating. This means a switch offers dramatically better performance than a hub. To analyze the traffic, specialized hardware (called application-specific integrated circuits [ASICs]) is needed. If we replace the

hub with a switch in our example, we get the setup shown in [Figure 17-8](#), where a Layer 2 switch maintains a port-to-MAC address lookup table.

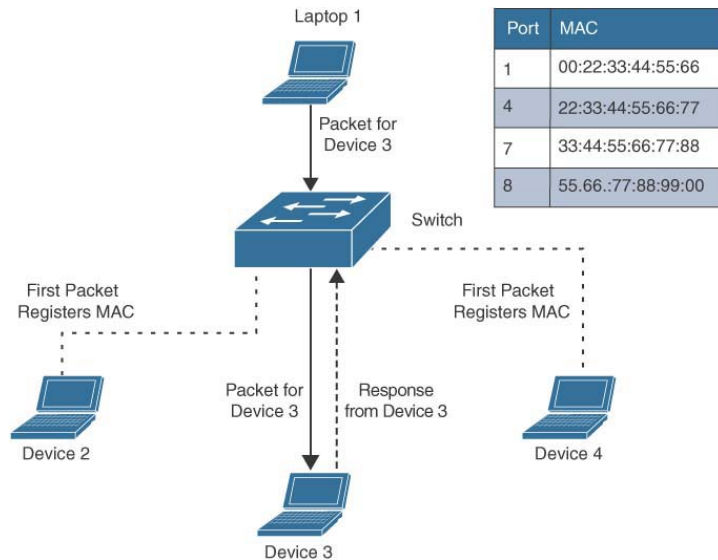


Figure 17-8 *Using a Layer 2 Switch for Port-to-MAC Address Mapping*

As you can see in [Figure 17-8](#), each of the devices is “seen,” and the MAC address is registered when the first packet is sent by the device. Over a period, the switch generates a table that contains the port number on which each MAC address was seen. It then uses this table to do a lookup for the destination MAC address and sends the incoming packet to the correct outgoing port. In the figure, a packet destined for Device 3 is sent only to the correct port.

Recall our look at the OSI model layers in [Chapter 16](#). The switch described so far in this section is a Layer 2 switch, which works at Layer 2 of the OSI model (the data link layer). The switch sends a packet to a destination port by using the MAC address lookup table, which stores the MAC address of a device associated with each port. In contrast, a Layer 3 switch works at the network layer of the OSI model (Layer 3), which means it looks at the IP address in order to do the switching.

Figure 17-9 shows a Layer 3 switch and how it maintains a port-to-IP address lookup table.

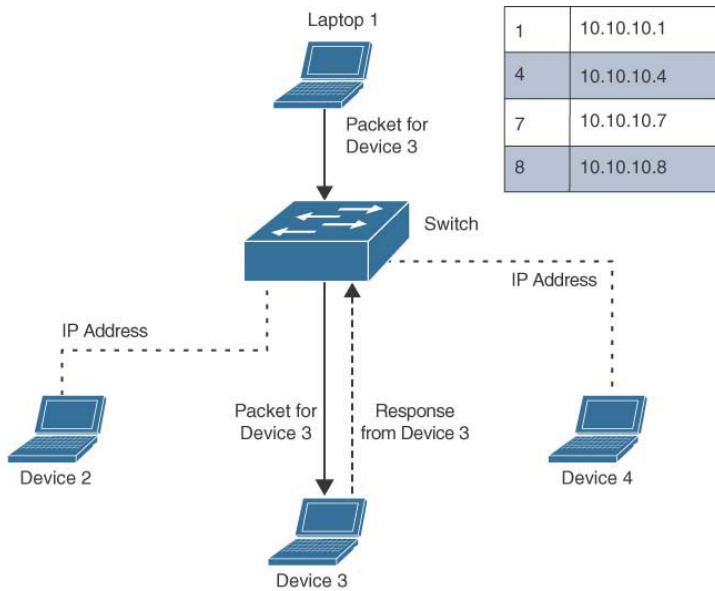


Figure 17-9 Using a Layer 3 Switch for Port-to-IP Address Mapping

Table 17-2 looks at some of the differences between Layer 2 and Layer 3 switches.

Table 17-2 Differences Between Layer 2 and Layer 3 Switches

Layer 2 Switch	Layer 3 Switch
Operates at Layer 2 (data link) of the OSI model	Operates at Layer 3 (network layer) of the OSI model
Uses MAC addresses for forwarding	Uses IP addresses for forwarding
Performs Layer 2 switching only	Performs both Layer 2 and Layer 3 switching
Can be used to reduce traffic security on the local	Can be used to implement a virtual local-area network

network	(VLAN)
Has a single broadcast domain	Has multiple broadcast domains

Virtual Local Area Networks (VLANs)



A switch can be logically segmented into multiple broadcast domains by using virtual LANs (VLANs). That is, if you have one switch, you can create multiple logical switches. A VLAN is identified by a VLAN ID. The VLAN ID is usually a value between 0 and 4095. The default VLAN on any network is VLAN 1. Every port on a switch can be assigned a different VLAN, or a group of ports can be assigned a particular VLAN ID. VLANs allow network administrators to logically split a switch, allowing multiple broadcast domains to coexist on the same hardware but maintaining the isolation, security, and performance benefits of using completely separate switches.

There are several advantages to creating VLANs:

- **Network security:** Creating VLANs within a switch also creates an automatic logical level of protection. This kind of logical separation is very useful when there is a need to create networks for various departments in an enterprise. [Figure 17-10](#) shows three VLANs—VLAN 10, VLAN 20, and VLAN 30—each assigned to a different department.

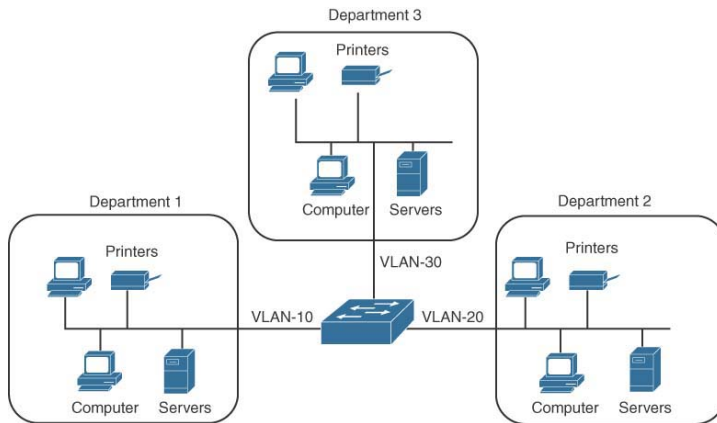


Figure 17-10 Using a VLAN for Network Security

- **Broadcast traffic distribution:** Segmenting a large LAN into smaller VLANs can reduce broadcast traffic because each broadcast packet is sent only to the relevant VLAN. For example, in Figure 17-10, which shows three VLANs—one for each of the domains—broadcast traffic will go to only the devices in the appropriate VLAN.
- **Performance increase:** Creating multiple broadcast domains reduces the broadcast traffic on the entire network tremendously, which in turn boosts the overall performance of the network.

Say that you have a single switch. By default, all of the ports on this switch are in one VLAN, such as VLAN 1. Any port can be configured to be an access port or a trunk port:

- **Access port:** An access port is essentially a port that can be assigned to only one VLAN. You can change the port membership by specifying the new VLAN ID.
- **Trunk port:** A trunk port can essentially have two or more VLANs configured. It has the ability to transport traffic for several VLANs simultaneously.

Figure 17-11 shows how trunk ports connect various VLANs across switches.

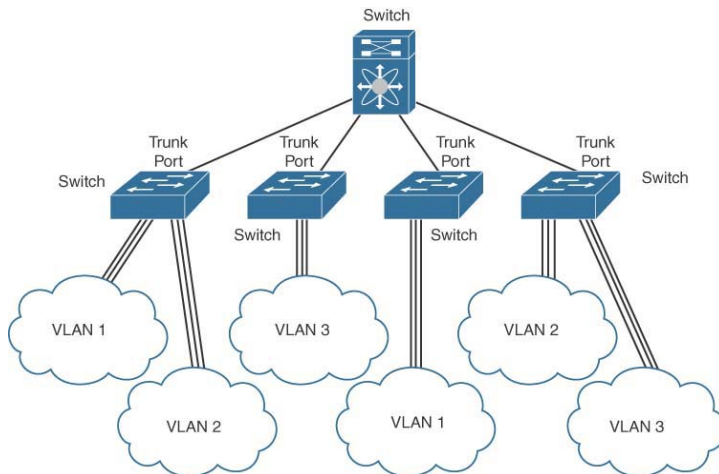


Figure 17-11 *VLAN Trunk Ports*

Routers

A router is a device that forwards packets between networks via the network layer of the OSI model (Layer 3). It forwards, or routes, packets based on the IP address of the destination device. A router also has the intelligence to determine the best path to reach a particular network or device. It can determine the next hop, or routing destination, by using routing protocols such as Routing Information Protocol (RIP), Open Shortest Path First (OSPF), and Border Gateway Protocol (BGP).

What is the difference between a Layer 3 switch and a router? Both forward packets based on IP address, right? The difference is pretty straightforward: Whereas a switch operates within a network, a router connects two or more networks. It routes packets to go across different protocols (such as Ethernet and WAN technologies such as cable, DSL, or satellite). Another difference is that switches usually have dedicated hardware (ASICs) to forward packets as they come in. In contrast, routers need more intelligence to route packets, and this intelligence is typically provided by software.

Figure 17-12 shows a router connecting different types of networks.

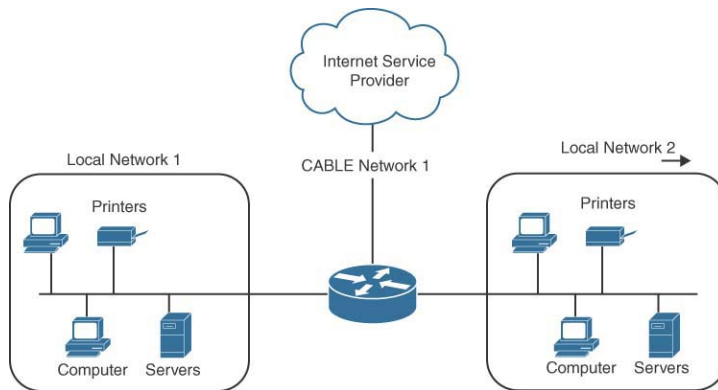


Figure 17-12 Using a Router to Connect Different Networks

Routing in Software

As discussed earlier in this chapter, routers participate in route discovery, path determination, and packet forwarding. Route discovery and path determination are part of the routing protocol, and as the router becomes part of this discovery process, it discovers and updates the routing tables accordingly. There are basically three types of forwarding:

Key Topic

- **Process switching:** The CPU is involved for every packet that is routed and requires a full routing table lookup. Process switching, shown in Figure 17-13, is the slowest type of forwarding as each of the packets that the interface driver receives (step 1 in the figure) is punted and put in the input queue for the processor for further action (step 2). In this case, the processor receives the packets (step 3), determines the next interface it needs to send them to, and rewrites the headers as needed and puts the new packets in the output queue (step 4). Finally, the kernel driver for the new network interface picks up the packets and transmits them on the interface (step 5).

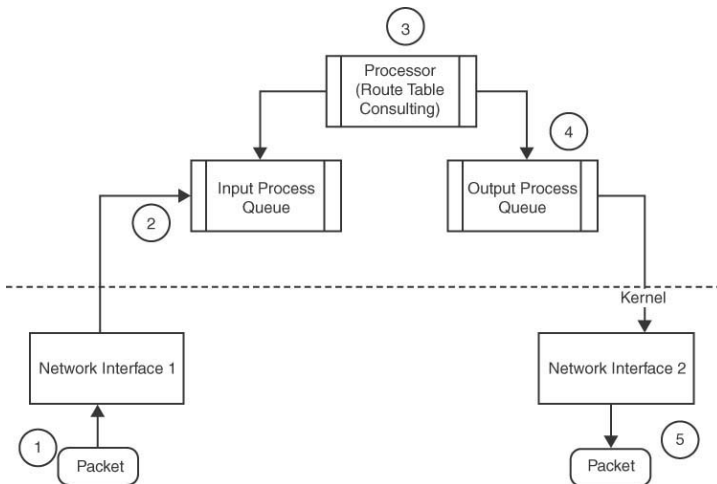


Figure 17-13 *Process Switching with the CPU Involved in Packet Routing Decisions*

- Fast switching:** Fast switching is the next-level evolution of process switching. The CPU is involved for only the first packet that is routed. It determines the outgoing interface required for the packet and updates the route cache, which typically resides in the kernel. Once the route cache is populated, the input interface driver does a route-cache lookup and fast switches the packet to the outgoing interface. [Figure 17-14](#) shows fast switching, in which the processor essentially updates the route cache after it learns the next hop. All subsequent packets are fast switched.

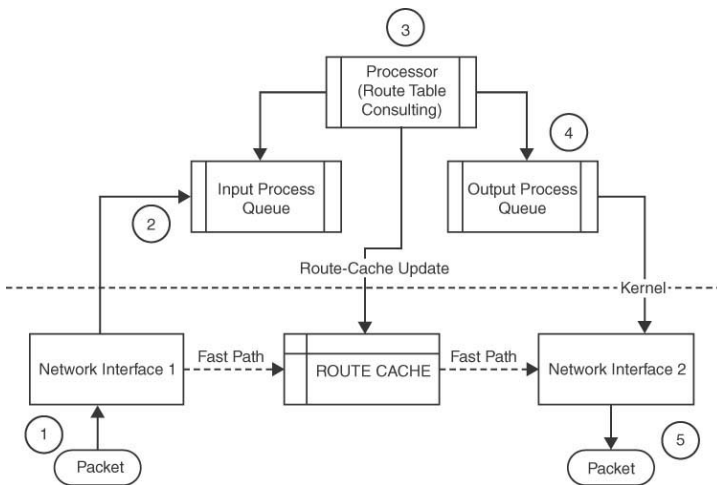


Figure 17-14 *Fast Switching and the Route Cache*

- Cisco Express Forwarding (CEF) switching:** CEF switching is an advanced Cisco-proprietary form of fast switching. CEF switching sometimes involves specialized hardware and distributed routing, and the CPU may not need to get involved in the data path at all. It does this by building a route cache with forwarding information; this is known as

the Forwarding Information Base (FIB) table. The FIB table contains precomputed reverse lookups and next-hop information for routes, including the interface and Layer 2 information to use. When a network topology or routing table change occurs, the change is also reflected in the FIB table. Figure 17-15 shows CEF switching, where both Layer 2 and Layer 3 information is used.

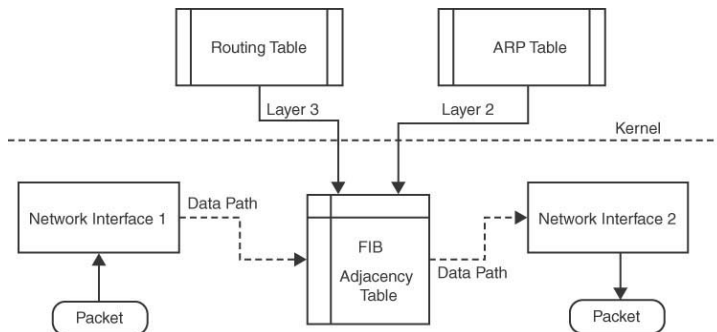


Figure 17-15 CEF Switching: Data Switching Using FIB Tables

This analogy is sometimes used to illustrate the three types of switching:

- Process switching is like doing math on paper: Write down each step and solve the problem.
- Fast switching, using the route cache, is like solving a problem by hand once and then simply recalling the answer from memory if the same problem is given again.
- CEF switching is like using formulas in an Excel spreadsheet, and when the numbers hit the cells, the answer is automatically calculated.

Functions of a Router

As discussed earlier in this chapter, a router is a device that appropriately and efficiently directs traffic from an incoming interface to the correct outgoing interface.

Apart from this primary function, a router has several other functions:

- **Network segmentation:** A router uses Network Address Translation (NAT) to map private IP addresses to public IP addresses. Using NAT at the router secures the private network and also reduces the number of IP addresses needed to get messages to the public internet. [Figure 17-16](#) shows how network segmentation is achieved by using NAT.

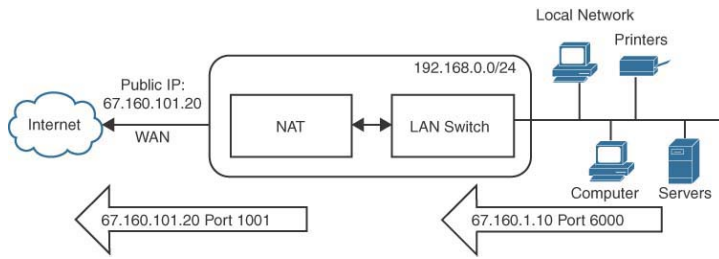


Figure 17-16 Router NAT: Translating Internal Addresses to External Addresses

- IP address management:** A router uses Dynamic Host Configuration Protocol (DHCP) to assign IP addresses to devices that connect to the network. Clients send DISCOVER broadcast message to figure out if a local DHCP server exists in the network. If a DHCP server exists, it offers configurations such as an IP address, a MAC address, a domain, and so on via an OFFER message. The client can then send a formal REQUEST to the server for allocating an IP address. The server responds with an ACK unicast message to the client, indicating that the IP address has been allocated and confirmed. Figure 17-17 shows a flow of clients requesting IP addresses via DHCP.

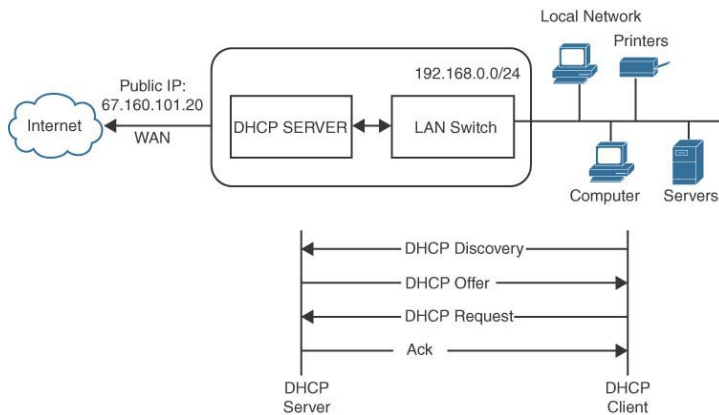


Figure 17-17 Using DHCP to Assign IP Addresses

- Firewalls:** A firewall safeguards a network from intentional or unintentional intrusion. A firewall sits at the junction point or gateway between two networks—usually a private network and a public network such as the Internet. Users on the local network of the router need to be protected from hackers and other malicious users. A firewall blocks traffic from unauthorized users. It also helps in allowing or blocking certain types of traffic on certain ports. For example, a firewall may allow access to certain applications only, such as HTTP or SSH, and block all UDP traffic. Figure 17-18 shows a firewall that allows only HTTP access to the outside world and blocks all incoming UDP traffic.

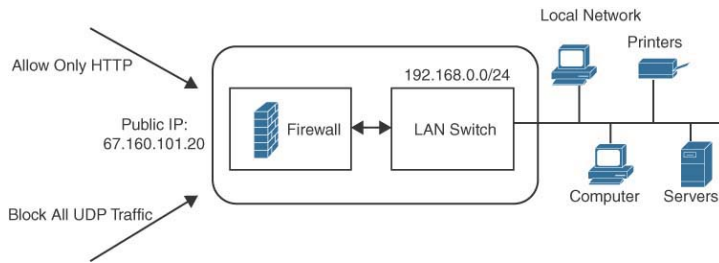


Figure 17-18 Router Firewall: Allowing/Blocking Certain Apps or Traffic

- **Domain name proxy:** Modern firewalls interact with cloud DNS servers such as Cisco Umbrella or Open DNS to resolve DNS queries. The queries and, in turn, the hosts can be stopped at the source if they are deemed harmful or malicious.

Network Diagrams: Bringing It All Together

So far in this chapter, you have learned about all the elements that constitute a network. The best way to visually represent a network topology is by using a network diagram. A network diagram maps out the structure of a network with different network element icons and connections between them. This kind of visual presentation makes it simple and easy for anyone to understand how a network is built and can help not only during initial network deployment but also in debugging.

Figure 17-19 shows a typical network topology. At the core of the network is a router (Router 1) that connects three different networks. One router interface is connected to a switch (Switch 1), a second router interface is connected to a switch (Switch 2), and a third router interface is connected to a router (Router 2):

- Switch 1 connects to Gigabit-ether 0/0 on the router.
- Switch 2 connects to Gigabit-ether 0/1 on the router.
- Router 2 connects to Fast-ethernet 0/0 on the router.

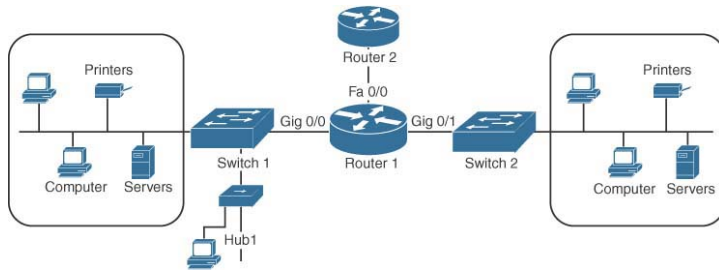


Figure 17-19 Network Diagram

Both Switch 1 and Switch 2 have networks that consist of various devices such as computers, printers, and so on. Router 2 is the Internet router with firewall functionality and a connection to the service provider.

SOFTWARE-DEFINED NETWORKING

The term software-defined networking (SDN) applies to a type of network architecture design that enables IT managers and network engineers to manage, control, and optimize network resources programmatically.

SDN essentially decouples network configuration and data flow engineering, regardless of the underlying hardware infrastructure. It allows you to consistently control the network by using standard open APIs.

Networks are often shown using a three-level architecture, as shown in [Figure 17-20](#), which consists of hardware and two planes:

- **Data plane:** As described earlier in this chapter, a router can route packets faster by using techniques such as fast switching or CEF switching. These techniques for punting packets from the incoming interface to the outgoing interface operate on what is traditionally known as the data plane. The main objective of the data plane is to determine how the incoming packet on a port must be forwarded to an outgoing port, based on specific values in the packet headers.
- **Control plane:** Routing protocols and other protocols make up the control plane. The control plane determines how a packet is routed among routers or other network elements as the packet traverses end-to-end from source host to destination host. The control plane also deals with packets that are destined for the router itself. Device and network management are also part of the control plane. Management

functions include initializing interfaces with default configurations, IP addresses, policies, user accounts, and so on.

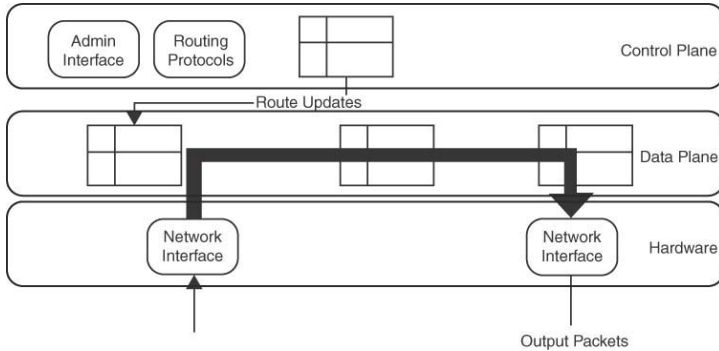


Figure 17-20 Control Plane and Data Plane

Figure 17-21 shows three network elements: a router, a switch, and a wireless access point. For each one of them, the figure shows the control plane and data plane. The management of these devices includes the functions you use to control and monitor these devices. As the controlling and monitoring function has to be done on each device, we call this architecture a “distributed control plane architecture.” The drawback is that network engineers or administrators have to manage each device independently, logging in to each device and setting it up. Imagine that you have to make a simple configuration change such as enabling a syslog server; you would have to log in to each device independently and make the changes.

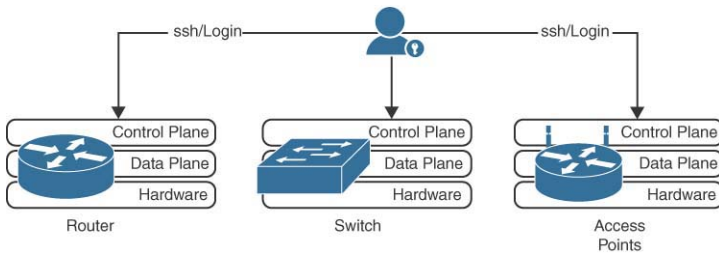


Figure 17-21 Distributed Control Plane Architecture

To prevent the inefficiency of manually managing network devices, a network controller can be used. When you use a network controller, you move the control

planes from each network device and consolidate all of them in the network controller. The network controller can then communicate with the network elements, and the network elements can send management data to the controller. Because you now have one controller, this architecture is called a centralized control plane. [Figure 17-22](#) shows how a network controller can be used to programmatically manage and control various networking elements using the northbound and southbound APIs.

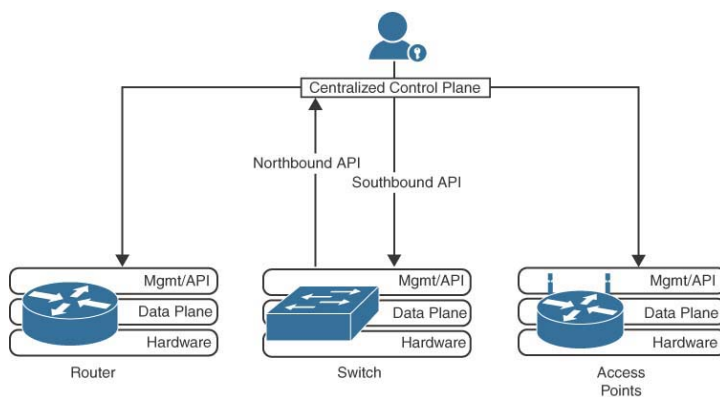


Figure 17-22 *Centralized Control Plane with a Network Controller*

Communication between the network controller and the network elements is bidirectional. Both sides use application programming interfaces (APIs), which allow the network controller and the network devices to communicate programmatically. The network controller interacts with the network devices via southbound APIs, and the network elements interact with the network controller via northbound APIs.

SDN Controllers



An SDN controller is often considered the “brain” in a modern network. The Open Networking Foundation

(ONF) defines SDN as follows:

an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of applications. This architecture decouples the network control and forwarding functions, enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

An SDN controller possesses a “global” view of the entire network. It knows about all the network elements that constitute the network, the best paths between them, and other potential routes. As indicated in the ONF SDN definition, SDN is a network architecture that separates the control and data planes for network devices and provides centralized management. There are two widely popular controllers:

- **OpenFlow:** The ONF manages this standard used for communication between the SDN controller and managed network devices.
- **OpenDayLight:** The Linux Foundation (LF) manages this standard, which uses OpenFlow to manage network devices.

As shown in [Figure 17-23](#), SDN involves three layers: application layer, control layer, and infrastructure layer.

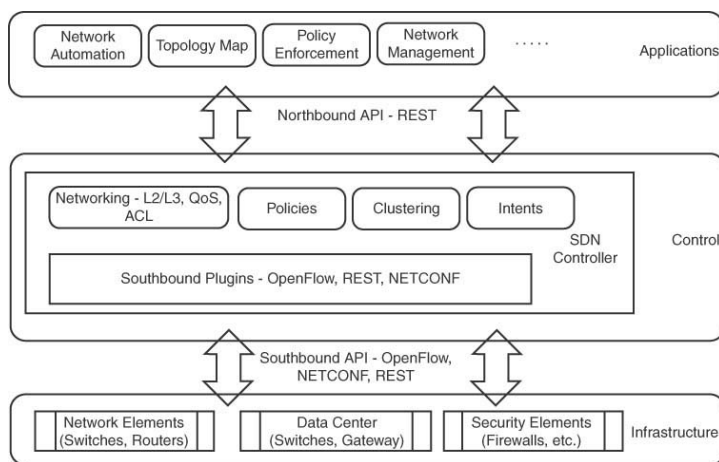


Figure 17-23 *Software-Defined Networking: Layered Architecture*

The infrastructure layer is composed of networking equipment and elements that form the actual network and elements that help to forward network traffic. It could be a set of network switches and routers in the network or data centers. The infrastructure layer is the physical layer over which network virtualization functions lay via the control layer.

The control layer is where the SDN controllers reside to control network infrastructure. The control layer has the business logic to fetch and maintain different types of network information, state details, topology details, and statistics details. The SDN controller is all about managing networks, and it has all the control logic for network use cases such as switching, routing, Layer 2 VPNs, Layer 3 VPNs, firewall security rules, DNS, DHCP, and clustering. Cisco implements various services in its SDN controllers. These services expose APIs (typically REST-based) to the upper layer (application layer), which makes life easy for network administrators who use apps on top of SDN controllers to configure, manage, and monitor the underlying network. The control layer exposes two types of interfaces:

- **Northbound interface:** This interface is used for communication with the upper layer (the application layer) and is in general realized through REST APIs of SDN controllers.
- **Southbound interface:** This interface is meant for communication with the lower layer (the infrastructure layer) of network elements and is in general realized through southbound protocols, such as OpenFlow and NETCONF.

The application layer is a developing area where a lot of innovations are happening. The applications in this layer leverage information about the network topology, network state, network statistics, and so on. Several types of applications can be developed, such as those related to network automation, network configuration and management, network monitoring, network troubleshooting, network policies, and security. Such SDN applications can provide various end-to-end

solutions for real-world enterprise and data center networks.

Cisco Software-Defined Networking (SDN)

Cisco has introduced many SDN controllers in various domains, including the data center, campus, and even service provider domains. The following is the list of Cisco's SDN products:

- **Cisco Application Centric Infrastructure (ACI):** ACI was the first Cisco SDN solution, and it has three components:
 - **Application Network Profile:** This is a collection of endpoint groups (EPGs), their connections, and the policies that define the connections.
 - **Application Policy Infrastructure Controller (APIC):** This is a centralized software controller that manages downstream switches and acts as a management plane.
 - **ACI fabric:** This is the connection between spine and leaf switches. In the ACI world, spine and leaf are the Cisco Nexus 9000 Series switches, which act as the control plane and the data plane of the ACI.
- **Cisco Digital Network Architecture (DNA):** Cisco DNA introduces the concept of intent-based networking, which interprets business needs into technical solutions.
- **Cisco Network Services Orchestrator (NSO):** This is a solution mostly for automating tasks in service provider environments. NSO is a multivendor controller installed in Linux that supports NETCONF, OpenFlow, SNMP, and APIs.
- **Cisco Software-Defined WAN (SD-WAN):** SD-WAN deals with creating and managing WAN connections in a cloud-based environment. SD-WAN solutions have several key features—including segmentation, centralized policies, zero-touch provisioning, and configuration templates—that can be very helpful to customers. Cisco SD-WAN has four main components, each with a very specific role:
 - **vManage (for management):** vManage is a GUI-based network management system that handles the management plane. vManage is a single pane of glass that provides various key stats. An operations team can use vManage for day-to-day operational activities.
 - **vSmart (controller):** vSmart is the main brain of SD-WAN, and it manages the control plane. vSmart does all the complex work of path calculation, route advertisement, and so on by allowing the data plane to do only packet forwarding.
 - **vEdge (data plane):** The vEdge router's job is to forward packets based on the policies configured with vSmart. The vEdge keeps a

constant connection with vSmart to get updates.

- **vBond (orchestrator):** vBond is the orchestrator and the gatekeeper that validates and authorizes every vEdge device trying to join the network. It also orchestrates the connectivity between vEdge and vSmart.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19, “Final Preparation,”](#) and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 17-3](#) lists these key topics and the page number on which each is found.



Table 17-3 Key Topics

Key Topic Element	Description	Page
Paragraph	Star, bus, and ring network topologies	512
Paragraph	Hubs	517
Paragraph	Layer 2 and Layer 3 switches	518
Paragraph	VLANs	520
List	Methods of routing packets in a router	522
Paragraph	SDN controllers	529



DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

local-area network (LAN)

campus-area network (CAN)

metropolitan-area network (MAN)

wide-area network (WAN)

virtual LAN (VLAN)

Cisco Express Forwarding (CEF)

Forwarding Information Base (FIB) table

Network Address Translation (NAT)

Dynamic Host Configuration Protocol (DHCP)

software-defined networking (SDN)

Chapter 18

IP Services

This chapter covers the following topics:

- **Common Networking Protocols:** This section introduces protocols that are commonly used in networks and that you should be familiar with.
- **Layer 2 Versus Layer 3 Network Diagrams:** This section covers different types of network diagrams.
- **Troubleshooting Application Connectivity Issues:** This section describes how to troubleshoot application connectivity issues.

This chapter covers IP services concepts. It starts with an introduction to common networking protocols that you are bound to use on a daily basis and then discusses them in detail:

- Dynamic Host Configuration Protocol (DHCP) is used to dynamically allocate IP configuration data to network clients so that the clients can get access to the network.
- Domain Name System (DNS) is a hierarchical naming system used mostly to resolve domain names to IP addresses.
- Network Address Translation (NAT), while not a protocol per se, deals with translations between private IP networks and public, globally routable IP networks.
- Simple Network Management Protocol (SNMP) has been developed for remote network monitoring and configuration.
- Network Time Protocol (NTP) is used to synchronize date and time between devices on a network.

The chapter also provides a comparison between Layer 2 and Layer 3 network diagrams, as well as a troubleshooting scenario for application connectivity issues.

“DO I KNOW THIS ALREADY?” QUIZ

The “Do I Know This Already?” quiz allows you to assess whether you should read this entire chapter thoroughly or jump to the “Exam Preparation Tasks” section. If you are in doubt about your answers to these questions or your own assessment of your knowledge of the topics, read the entire chapter. [Table 18-1](#) lists the major headings in this chapter and their corresponding “Do I Know This Already?” quiz questions. You can find the answers in [Appendix A, “Answers to the ‘Do I Know This Already?’ Quiz Questions.”](#)

Table 18-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section Questions	
Common Networking Protocols	1–5
Layer 2 Versus Layer 3 Network Diagrams	6
Troubleshooting Application Connectivity Issues	7

Caution

The goal of self-assessment is to gauge your mastery of the topics in this chapter. If you do not know the answer to a question or are only partially sure of the answer, you should mark that question as wrong for purposes of self-assessment. Giving yourself credit for an answer that you correctly guess skews your self-assessment results and might provide you with a false sense of security.

1. What are some of the benefits that DHCP offers?
(Choose two.)

1. Reduced network endpoint configuration tasks and costs

2. Elimination of single points of failure for network parameter configuration
3. Decentralized management of network configuration
4. Centralized management of network parameters configuration

2. What DNS component is configured on a network client?

1. Root server
2. TLD server
3. DNS resolver
4. TLC server

3. What are some of the benefits of Network Address Translation? (Choose three.)

1. Conserving public IP address space
2. Increased security due to hiding of internal addressing
3. Loss of end-to-end traceability and visibility
4. Loss of end-to-end functionality
5. Flexibility when changing ISPs

4. What version of SNMP was developed with a security focus in mind?

1. SNMPv3
2. SNMPv1
3. SNMPv2c
4. SNMPv4

5. What transport layer protocol and port are used by NTP?

1. TCP 123
2. TCP 120
3. UDP 120
4. UDP 123

6. What network devices should be included in a Layer 3 diagram?

1. Switches, routers, and firewalls
2. Routers, firewalls, and load balancers
3. Switches, firewalls, and load balancers
4. Bridges, switches, and hubs

7. What popular network utility is used to troubleshoot DNS issues?

1. **tracert**
2. **nslookup**
3. **nslookup**
4. **ping**

FOUNDATION TOPICS

COMMON NETWORKING PROTOCOLS

The following sections cover common networking protocols that network engineers and software developers alike should be familiar with. Knowledge of these protocols and technologies will give you better insight into how networks interact with applications in order to offer network clients an optimized and seamless experience.

Dynamic Host Configuration Protocol (DHCP)

Dynamic Host Configuration Protocol (DHCP), as the name suggests, is a protocol used for dynamically configuring hosts with network connectivity information. In order for any host device connected to a network to be able to send and transmit data, it needs to have network parameters such as IP address, subnet, default gateway, and DNS servers configured. This configuration can be done either manually or automatically, using protocols such as DHCP. Manual configuration of network parameters for hosts on a network is time-consuming and prone to errors—and it is therefore not very often implemented in real-world networks anymore. DHCP is extensively used for automatically distributing network configuration parameters to all network endpoints, including end-user devices and network devices. For networks that have already migrated to IP version 6 (IPv6) or that are running dual-stack IP version 4 (IPv4) and IPv6 addressing, DHCPv6 and the IPv6 autoconfiguration option can be used for dynamic and automatic network parameter assignment. IPv6 autoconfiguration can be used to quickly and dynamically assign IPv6 addresses to network clients, and DHCPv6 is used to assign not just IPv6 addresses but also DNS servers and domain names.

DHCP has two components: a protocol for delivering network device configuration information from a DHCP server to a network host and a mechanism for allocating that configuration information to hosts. DHCP works using a client/server architecture, with a designated DHCP server that allocates IP addresses and network information and that delivers that information to DHCP clients that are the dynamically configured network endpoints.

Besides basic network connectivity parameters such as IP addresses, subnet masks, default gateways, IP addresses of DNS servers, and local domain names, DHCP also supports the concept of options. With DHCP options, a DHCP server can send additional configuration information to its clients. For example, Cisco IP Phones use option 150 provided by the DHCP server to obtain IP addresses of the TFTP servers that hold configuration files for the IP Phones; Cisco wireless access points use DHCP option 43 to obtain the IP address of the Cisco wireless LAN controller that they need to connect to for management purposes.

DHCP for IPv4 is defined and described in RFC 2131: *Dynamic Host Configuration Protocol* and RFC 2132: *DHCP Options and BOOTP Vendor Extensions*. For IPv6, DHCP was initially described in RFC 3315: *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)* in 2003, but it was subsequently updated by several newer RFCs. RFC 3633: *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) Version 6* added a mechanism for prefix delegation, and RFC 3736: *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6* added stateless address autoconfiguration for IPv6.

Some of the benefits of using DHCP instead of manual configurations are



- **Centralized management of network parameters**
configuration: DHCP servers usually manage the network configuration settings for several subnets and represent the central source of truth and the configuration point for all dynamic network parameters needed for network endpoints to be able to connect to the network. This makes it much easier to manage network address assignment compared to using several disparate systems or Excel files.
- **Reduced network endpoint configuration tasks and costs:**
Dynamically allocating network connectivity information brings huge cost and time savings compared to manually performing the same tasks. This is especially true in medium to larger enterprise network environments and for Internet service providers (ISPs). Imagine ISPs needing to send out technicians to perform manual network changes to all of their customers when they start using their service every day. By using DHCP and dynamic network address configuration, the modems of clients can be configured within seconds, without any manual intervention.

DHCP is built on top of a connectionless service model using User Datagram Protocol (UDP). DHCP servers listen on UDP port 67 for requests from the clients and communicate with the DHCP clients on UDP port 68. There are several ways network configuration information is allocated by the DHCP server:

- **Automatic allocation:** With automatic allocation, the DHCP server assigns a permanent IP address to the client.
- **Dynamic allocation:** With dynamic allocation, the DHCP server assigns an IP address to the client for a limited period of time called the *lease time*.
- **Manual allocation:** With manual allocation, the network configuration of the client is done manually by the network administrator, and DHCP is used to relay that configuration information to the client.

As mentioned previously, DHCP defines a protocol and a process for how to assign network configuration information to devices connecting to the network. The process defines the methodology used to configure the DHCP server. Usually a DHCP server serves one or more client subnets. Once the client subnet is defined, a pool of addresses from that subnet is configured as available addresses for client allocation. Additional information

such as subnet mask, the IP address of the default gateway, and DNS servers is the same for the whole subnet, so these configuration parameters apply to the whole subnet rather than to each end host device. In some cases, the DHCP client and the server are located in different subnets. In such a case, a DHCP relay agent can be used to relay the DHCP packets between clients and servers. Any host on the network can act as a relay agent, but in most cases, the default router for the client subnet acts as a DHCP relay agent. Forwarding of DHCP messages between the clients and the servers by the relay agents is different from regular routing and forwarding. While regular forwarding is transparent for the endpoints involved in the exchange of data, with DHCP forwarding, DHCP relay agents receive inbound DHCP messages on the client interface and generate new DHCP messages on the interface connecting to the server. The DHCP relay agent effectively becomes a man-in-the-middle for the DHCP traffic between clients and servers. [Figure 18-1](#) illustrates DHCP relay agent functionality.



Figure 18-1 *DHCP Relay Agent*

DHCP operations fall into the following four phases (see [Figure 18-2](#)):



- Server discovery
- Lease offer
- Lease request
- Lease acknowledgment

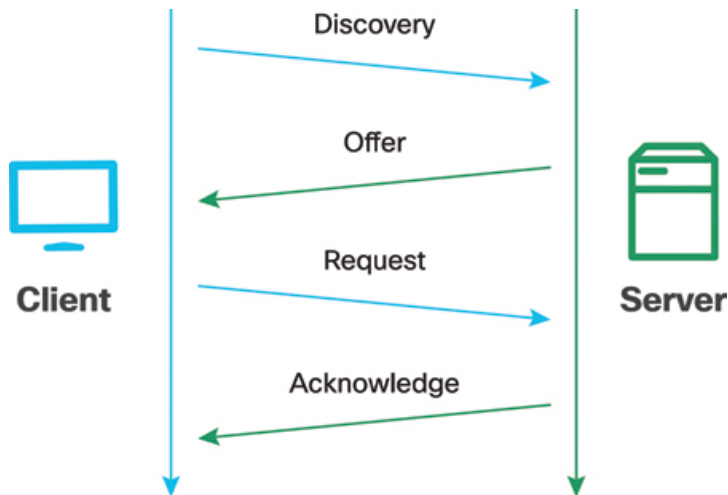


Figure 18-2 *DHCP State Machine*

Server Discovery

When a client first boots up and comes online on the network, it broadcasts a DHCPDISCOVER message with a destination address of the all-subnets broadcast address (255.255.255.255) with a source address of 0.0.0.0 since the client doesn't have any IP address at this stage. If there is a DHCP server configured for this subnet, it receives the all-subnets broadcast message and responds with a DHCPOFFER message containing the network configuration parameters for the client. If there isn't a DHCP server directly configured on the same subnet as the clients but there is a DHCP relay agent, the agent forwards the request to the DHCP server and will also forward the server offer to the requesting client.

Lease Offer

A DHCP server that receives a DHCPDISCOVER message from a client responds on UDP port 68 with a DHCPOFFER message addressed to that client. The DHCPOFFER message contains initial network configuration information for the client. There are several fields in the DHCPOFFER message that are of interest for the client:

- **chaddr:** This field contains the MAC address of the client to help the client know that the received DHCP OFFER message is indeed intended for it.
- **yiaddr:** This field contains the IP address assigned to the client by the server.
- **options:** This field contains the associated subnet mask and default gateway. Other options that are typically included in the DHCP OFFER message are the IP address of the DNS servers and the IP address lease and renewal time.

Once the client receives a DHCP OFFER message, it starts a timer and waits for further offers from other DHCP servers that might serve the same client subnet.

Lease Request

After the client has received the DHCP OFFER from the server, it responds with a DHCP REQUEST message that indicates its intent to accept the network configuration information contained in the DHCP OFFER message. The client moves to the Request state in the DHCP state machine. Because there might be multiple DHCP servers serving the same client subnet, the client might receive multiple DHCP OFFER messages, one from each DHCP server that received the DHCP DISCOVER message. The client chooses one of the DHCP OFFER messages; in most implementations of the DHCP client, this is the first DHCP OFFER received. The client replies to the server with a DHCP REQUEST message. The DHCP server chosen is specified in the Server Identifier option field of the DHCP REQUEST message. The DHCP REQUEST message has as a destination address the all-subnets broadcast address once more, so all DHCP servers receive this message and can determine if their offer was accepted by the client. The source IP address of the DHCP REQUEST message is still 0.0.0.0 since the client has not yet received a confirmation from the DHCP server that it can use the offered IP address.

Lease Acknowledgment

The DHCP server receives the DHCP REQUEST message from the client and acknowledges it with a DHCP ACK

message that contains the IP address of the DHCP server and the IP address of the client. The DHCPACK message is also sent as a broadcast message. Once the client receives the DHCPACK message, it becomes bound to the IP address and can use it to communicate on the network. The DHCP server stores the IP address of the client and its lease time in the DHCP database.

Releasing

A DHCP client can relinquish its network configuration lease by sending a DHCPRELEASE message to the DHCP server. The lease is identified by using the client identifier, the chaddr field, and the network address in the DHCPRELEASE message.

Domain Name System (DNS)

Domain Name System (DNS) is a directory of networks that maps names of endpoints to IP addresses. DNS performs a critical role in all networks and especially on the Internet. It is much easier to remember the website www.cisco.com than it is to remember the IP address to which it resolves—173.37.145.84 for IPv4 or 2600:1408:2000:1b3:0:0:0:b33 for IPv6. DNS is responsible for the process of resolving a hostname to an IP address. Each host endpoint and any device connecting to the network need to have an IP address configured in order to be able to communicate on the network. The IP address is like a street address, as every device on the Internet can be located based on its IP address. For example, when a user loads a web page, a translation must happen between the website name (cisco.com) and the machine-friendly IP address needed to locate that web page. This process is abstracted from the user because the user doesn't need to know what is happening in the background with the resolution of names into IP addresses.

There are several critical components in the DNS resolution process:



- The DNS recursive resolver is the server that receives DNS queries from client machines and is making additional requests in order to resolve the client query.
- Root name servers at the top of the DNS hierarchy are the servers that have lists of the top-level domain (TLD) name servers. They are the first step in resolving hostnames to IP addresses.
- TLD name servers host the last portion of a hostname. For example, the TLD server in the `cisco.com` example has a list for all the `.com` entries. There are TLD servers for all the other domains as well (`.net`, `.org`, and so on).
- The authoritative name server is the final step in the resolution process. It is the authoritative server for that specific domain. In the case of `cisco.com`, there are three authoritative servers: `ns1.cisco.com`, `ns2.cisco.com`, and `ns3.cisco.com`. Whenever a public domain is registered, it is mandatory to specify one or more authoritative name servers for that domain. These name servers are responsible for resolving that public domain to IP addresses.

Let's go through the steps of a DNS lookup from the perspective of a client that is trying to resolve a domain to an IP address (see [Figure 18-3](#)):

- Step 1.** The client query travels from the client machine to the configured DNS server on that machine. This DNS server is the DNS recursive resolver server.
- Step 2.** The DNS recursive resolver queries a DNS root name server.
- Step 3.** The root server responds to the recursive resolver with the TLD server for the requested last portion of the hostname. In the case of `cisco.com`, this would be the `.com` TLD server.
- Step 4.** The resolver queries the `.com` TLD server next.
- Step 5.** The TLD server responds with the IP address of the authoritative name server—in this

example, the DNS server responsible for the cisco.com domain.

Step 6. The resolver sends the query to the authoritative name server.

Step 7. The authoritative name server responds with the IP address of the cisco.com web server.

Step 8. The DNS resolver responds to the client with the IP address obtained from the authoritative name server.

Step 9. The client can finally send the web page request to the IP address of the web server.

Step 10. The web server returns the web page to the client, and the browser renders it for the user.

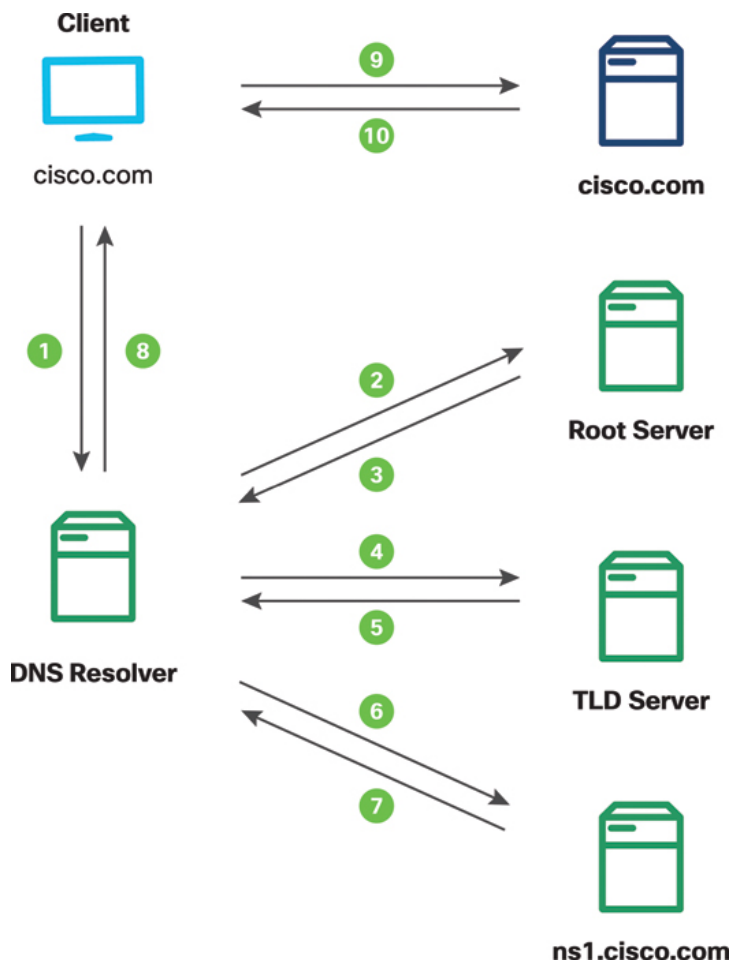


Figure 18-3 DNS Name Resolution Steps

A caching mechanism is available with DNS in order for the client queries to be resolved as quickly as possible. DNS caching means temporarily storing results obtained during previous requests on DNS servers that are close to the client. Caching DNS resolution data makes it possible to resolve client queries earlier in the DNS lookup chain, which improves resolution time and reduces bandwidth and CPU consumption.

DNS uses User Datagram Protocol (UDP) on port 53 to serve resolution queries. Several different types of records are stored in the DNS database, including IP addresses (A records for IPv4 and AAAA records for IPv6 addresses), SMTP mail exchangers (MX records), IP addresses of name servers (NS records), and alias records (CNAME). Although it was not intended to be used as a general-purpose database, DNS has been extended to store many types of additional information.



The Internet Engineering Task Force (IETF) has published several Requests for Comments (RFCs) related to DNS over the years. Some of the most important ones are RFC 1034: *Domain Names—Concepts and Facilities*, RFC 1035: *Domain Names—Implementation and Specification*, and RFC 1123: *Requirements for Internet Hosts—Application and Support*.

NETWORK ADDRESS TRANSLATION (NAT)

When Internet Protocol (IP) was created, very few people, if any, were expecting it to support a global network of billions of interconnected devices. As discussed in earlier chapters, IPv4 addresses are 32 bits long, which means they can uniquely address a bit more than 4 billion endpoints. This number was fine and out

of reach for a long time, but as the number of endpoints connecting to the Internet grew exponentially, it was clear that 4 billion addresses would not be enough to uniquely identify all the connected devices. At that point, work started for a new version of IP, IPv6, which defines 128-bit addresses and is able to uniquely identify trillions of endpoints. At the same time, it was clear that an overnight switchover from one IP version to another would be an impossible feat on the Internet, so several temporary solutions were proposed to ease the transition and extend the life of the IPv4-based Internet.

Network Address Translation (NAT) is one of the solutions to preserve the dwindling number of public IPv4 addresses. NAT reuses private IPv4 address blocks in internal networks and translates those addresses into public and unique IPv4 addresses at the borders of the internal networks. RFC 1918: *Address Allocation for Private Internets* declared a set of subnets private and unroutable on the global Internet. The subnets 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16 are extensively used in all private networks in the world, from enterprise networks to small office/home office networks. All other IPv4 addresses are public and routable on the Internet, meaning they uniquely identify endpoints on the network.



NAT is mostly used to translate between private RFC 1918 subnets and public IPv4 subnets. This translation happens at the exit points from the private networks, which in most cases are firewalls or border routers. NAT can also be used to translate between private and private networks. In the case of mergers and acquisitions, it is possible that the enterprise that was acquired uses the same private IPv4 subnets as the acquiring company. In order to be able to exchange traffic between these

networks that are addressed with the same IP addresses, NAT can be used to perform address translation. Basically, whenever an IP address needs to be translated into another address, NAT can be used.

NAT is an IETF standard described in RFC 1631: *The IP Network Address Translator (NAT)*. A large number of Cisco and third-party routers and firewalls support NAT. The devices that perform IP address translations are usually situated and route data traffic between the internal network and the outside world or the public Internet. During the NAT configuration phase, an internal subnet or a set of subnets is defined as being internal, or “inside”; these are usually the private IP subnets used internally in the enterprise. As a second step in the configuration process, single public IP address or an external, or “outside,” pool of IP addresses are defined. With this information, the border device can perform IP address translation between the internal and external worlds. Several types of NAT are available:



- **Static NAT (static NAT):** Static NAT defines a one-to-one mapping between the internal IP address and its public IP address correspondent.
- **Dynamic NAT (dynamic NAT):** With dynamic NAT, the internal subnets that are permitted to have outside access are mapped to a pool of public IP addresses. The pool of public IP addresses is generally smaller than the sum of all the internal subnets. This is usually the case in enterprise networks, where public IPv4 addresses are scarce and expensive, and a one-to-one mapping of internal to external subnets is not feasible. Reusing the pool of public IP addresses is possible as not all internal clients will access the outside world at the same time.
- **Port Address Translation (PAT or overloading):** PAT takes the dynamic NAT concept to the extreme and translates all the internal clients to one public IP address, using TCP and UDP ports to distinguish the data traffic generated by different clients. This concept is explained in more detail later in this chapter.

The type of NAT used in a particular situation depends on the number of public IP addresses defined and how

the translation process is implemented.

In order to illustrate how NAT works, let's assume that a client connected to an enterprise network uses private IPv4 addressing. As the client generates data traffic and tries to connect to the Internet, the traffic makes its way to the enterprise border device. The border device looks up the destination of the traffic and the NAT configuration. If the client IP address is part of the internal subnets that have to be translated, it creates an entry in its NAT table with the source and destination IP addresses, it changes the source IP address of the packet from the internal private IP address to the public IP address, and it forwards the packet toward its destination. As the data traffic is received at the destination, the destination node is unaware that the traffic received went through the NAT process. The IP addresses in the response traffic are swapped, and as the data traffic is being received by the border device, a lookup in the NAT table is done for the entry that was created as the traffic was exiting the network. The entry is matched, and the translation is done again—but this time in the reverse order, from the public IP address back to the private IP address of the client—and the traffic is routed back to the original source.

With PAT, the same type of table that keeps track of private to public translations and vice versa is created on the border device—but in this case TCP and UDP ports are also taken into account. For example, if the client generates web traffic and is trying to reach a web server on the Internet, the randomly generated TCP source port and the destination TCP port 443 for HTTPS are also included in the network translation table. In this way, a large number of clients—up to theoretically 65,535 for TCP and 65,535 for UDP traffic—can be translated to one public IP address. [Figure 18-4](#) illustrates PAT, which is also called *overloading* because many internal private IP addresses are translated to only one public IP address.

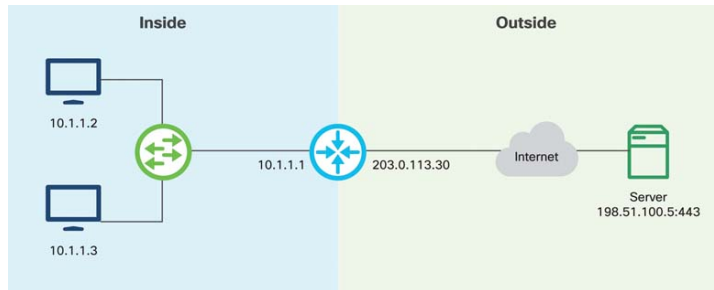


Figure 18-4 *Port Address Translation*

Some of the benefits of NAT are as follows:

- **Reduced costs for renting public IPv4 addresses:** With dynamic NAT and especially with PAT, a large number of private internal endpoints can be hidden behind a much smaller number of public IP addresses. Since public IPv4 addresses have become a rare commodity, there is a price for each public IPv4 address used. Large cost savings are possible by using a smaller number of public addresses.
- **Conserving public IPv4 address space:** The Internet would not have witnessed the exponential growth of the past few decades without NAT. Extensively reusing RFC 1918 private IP addresses for internal networks helped slow the depletion of IPv4 address space.
- **Additional security due to hiding the addressing for internal networks:** Having a whole network hidden behind one or a pool of public IPv4 addresses thwarts network reconnaissance attacks and increases the defense capabilities of the network.
- **Flexibility and cost savings when changing ISP connectivity:** In cases in which external network connectivity needs to be changed and migrations to new ISPs are required, configuration changes need to be performed only on the network border devices, but the rest of the internal network does not need to be re-addressed. This results in massive cost savings, especially in large enterprise networks.

Although the advantages of NAT, far outweigh the disadvantages in most cases, there are some disadvantages, including the following:

- **Loss of end-to-end functionality:** Some applications, especially for real-time voice and video signaling, are sensitive to changes in IP header addressing. While establishing these types of real-time voice and video sessions, headers exchanged at the application layer contain information pertaining to the private non-globally routable IP addresses of the endpoints. When using NAT in these cases, the IP

addresses in the Layer 3 headers differ from the IP addresses contained in the application layer headers. This results in an inability to establish end-to-end voice and video calls.

- **Loss of end-to-end traceability and visibility:** Troubleshooting end-to-end connectivity issues is especially challenging in networks that use NAT.
- **Degradation of network performance:** NAT operations on border devices are usually not resource intensive, and several mechanisms have been implemented to make this impact even lower. Still, a border device needs to take an additional step before forwarding the data traffic toward its destination, and that means additional delay and consumption of memory and CPU resources.

NAT is extensively used in networks that use IPv4 addressing. One of the requirements for IPv6 was to restore end-to-end connectivity between all endpoints on the network, so NAT is not popular in IPv6 networks.

Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is an application layer protocol used for monitoring and configuring devices. It was originally developed in the 1980s, and the IETF has published several RFCs covering SNMP since then. As networks were becoming larger and more complicated in those days, a need to be able to remotely monitor and manage devices arose. Following are several versions of SNMP that have been released through the years:

- SNMP version 1 (SNMPv1)
- SNMP version 2 (SNMPv2)
- SNMP version 2c (SNMPv2c)
- SNMP version 3 (SNMPv3)

While versions 1 and 2 of SNMP are rarely used anymore, versions 2c and 3 are extensively used in production environments. SNMPv2 adds 64-bit counter support and includes additional protocol operations. With SNMPv3, the focus is on security, so additional features like authentication, encryption, and message integrity were added to the protocol specification.

The following are some of the advantages of SNMP:



- It provides a single framework for monitoring many different kinds of devices.
- It is based on open standards documented in IETF RFCs.
- It is easily extensible.

There are also disadvantages with SNMP, including the following:

- The lack of writable MIBs (Management Information Bases) leads to poor configuration capabilities. SNMP is rarely used for configuration purposes.
- The lack of atomic transactions makes rollbacks to previous states difficult.
- SNMP is slow for monitoring purposes when large amounts of operational data need to be retrieved.
- It is CPU and memory resource intensive when large amounts of performance metrics are retrieved.

Even though SNMP was originally designed to support both monitoring and configuration capabilities, historically only the monitoring capabilities were used. The SNMP specification defines the following three components:

- Managed devices
- SNMP agent
- SNMP manager

Managed devices are the devices that are being monitored and managed through SNMP. They implement an SNMP interface through which the SNMP manager monitors and controls the device. The SNMP agent is the software component that runs on the managed device and translates between the local management information on the device and the SNMP version of that information. The SNMP manager, also called the Network Management Station (NMS), is the application that monitors and controls the managed

devices through the SNMP agent. The SNMP manager offers a monitoring and management interface to network and system administrators. The SNMP components and their interactions are illustrated in [Figure 18-5](#).

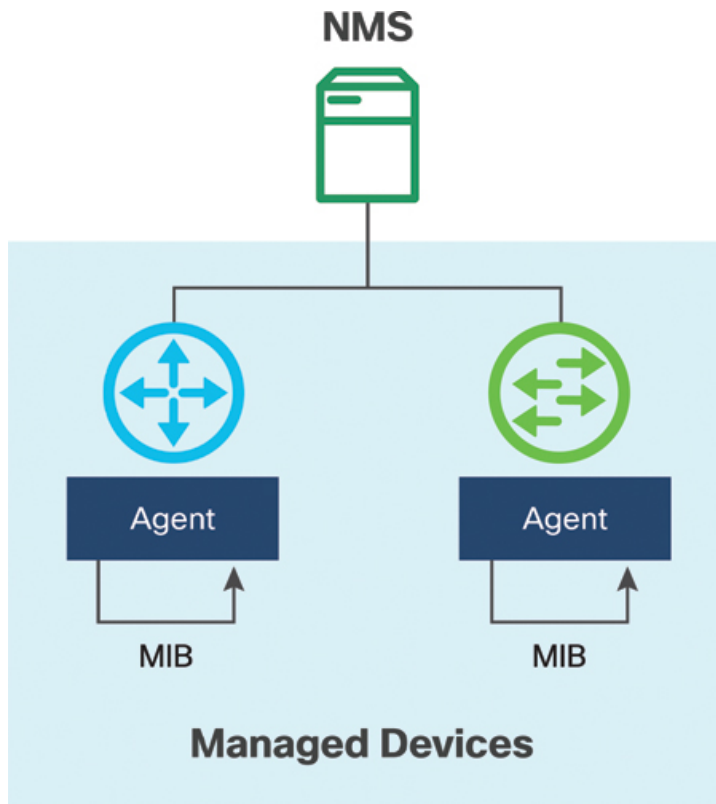


Figure 18-5 *SNMP Components*

The SNMP agent listens on UDP port 161 for requests from the SNMP manager. SNMP also supports notifications, which are SNMP messages that are generated on the managed device when significant events take place. Through notifications, the SNMP agent notifies the SNMP manager about these critical events. The NMS listens for these notifications on UDP port 162. SNMP data structures that facilitate the exchange of information between the SNMP agent and the NMS are organized as a list of data objects called a Management Information Base (MIB). A MIB can be thought of as a map of all components of a device that

are being managed by SNMP. In order to be able to monitor devices, the NMS must compile the MIB file for each device type in the network. Having the appropriate MIB, the SNMP agent and manager can exchange a wide range of information.



A MIB is organized as a tree-like structure with unique variables represented as leaves. Each variable in the tree is uniquely identified by an Object Identifier (OID). Operational data such as CPU temperature, fan speed, and outbound packets on an interface are all represented through OID values. In order for the NMS to obtain any device's operational metric, it needs to send an SNMP GET packet that includes OID values for each metric of interest. The SNMP agent receives the packet and looks up the OIDs in the MIB, and if the device implements the requested information, it returns the information to the NMS.

Several types of SNMP messages are used to communicate between the SNMP manager and the agent:



- **GetRequest:** This is the type of message used by the NMS to request the value of a variable or list of variables. The agent returns the request information in a Response message.
- **SetRequest:** The SNMP manager uses this message to request a change to a variable or a list of variables. The agent returns a Response message containing the new values for the variables.
- **GetNextRequest:** The SNMP manager uses this message to discover available variables and their values. This is a common operation used to “walk” the entire MIB of an agent. The agent returns a Response message that contains the requested information.
- **GetBulkRequest:** This optimization of the GetNextRequest message, which was introduced with SNMPv2, contains multiple iterations of the GetNextRequest call.

- **Response:** The SNMP agent generates this message, which contains the information the SNMP manager requested.
- **Trap:** The SNMP agent generates this notification message to signal to the SNMP manager when critical events take place on the managed device.
- **InformRequest:** The SNMP agent sends this acknowledged notification to the SNMP manager. Keep in mind that SNMP runs over UDP, which is a connectionless protocol, and packets might get lost during transmission. A rudimentary acknowledgment mechanism is implemented through InformRequest messages.

SNMP community names are used to establish trust between managers and agents. When community names are configured, the SNMP requests from the manager are considered valid if the community name matches the one configured on the managed device. If the names match, all agent-based MIB variables are made accessible to the manager. If they do not match, SNMP drops the request.

Network Time Protocol (NTP)

Accurately keeping track of time is critical in today's IT infrastructure. As IT infrastructure becomes more and more ingrained in business processes and success, every second of downtime when the infrastructure is not available translates into loss of revenue for the business. In extreme cases, this can lead to loss of customers and bankruptcy. Service-level agreements (SLAs) are contracts between providers and consumers of infrastructure. As an example, stringent SLA contracts require 99.999% uptime, which in the case of Internet service providers translates into no more than 5 minutes of downtime per year. It is of critical importance to make sure that there is a consistent, uniform, and correct view of time on all the devices in a network. Time is fundamental when measuring SLAs and enforcing contracts. Inaccurate time can lead to service disruptions. As a simple example, with web traffic, HTTPS TLS connections will not even be established if the time on the web server or the client is not accurate.

The system clock on each device is the heart of the time service. The system clock runs from the second the operating system starts; it keeps track of the date and time. The system clock can be set to update from different time sources and can be used to distribute time to other devices. Network Time Protocol (NTP) enables a device to update its clock from a trusted network time source and serve time to other devices, enabling groups of devices to be time synchronized. Most devices contain battery-powered clocks that keep track of date and time across restarts and power outages.

The main role of NTP is to synchronize the time on a network of devices. It was developed by IETF, and the latest version of the protocol, version 4, is defined in RFC 5905: *Network Time Protocol Version 4: Protocol and Algorithms Specification*. NTP uses UDP at the transport layer, and port 123 is reserved for it. NTP works based on a client/server architecture, with NTP servers providing the time to clients. Authoritative time sources are servers that have attached radio clocks or atomic clocks, making them extremely accurate. NTP has the role of distributing time to all the devices connected to the network. Multiple NTP servers can coexist at the same time on the same subnet, and clients can use all of them for time synchronization. NTP clients poll the time servers at intervals managed dynamically by conditions on the network such as latency and jitter. One NTP transaction per minute is sufficient to synchronize time between two machines.



The concept of strata is used in NTP to describe how many hops or devices away a client is from an authoritative time source. NTP servers that are most authoritative and are directly connected to very accurate time sources are in stratum 1. A stratum 2 time server

receives time from a stratum 1 server, and so on. When a client receives time from different NTP servers, a lower-stratum server is chosen as the most trusted source unless there is a big time difference between the lower-stratum server and all the other servers.

There are two ways communication between NTP clients and servers takes place: IT can be statically configured or can occur through broadcast messages. You can manually configure NTP clients to establish a connection and to associate and solicit time updates from NTP servers by simply statically configuring the hostname or the IP addresses of the servers. In local-area networks within the same subnet, NTP can be configured to use broadcast messages instead. Configuration in this situation is simpler, as each device can either be configured to send or receive broadcast messages. With broadcast NTP messages, there is a slight loss of accuracy since the flow of information is only one way.



Since time accuracy is critical in today's infrastructure, it is recommended to implement all security features that come with NTP. Two NTP security features are most commonly used:

- An encrypted authentication mechanism between clients and servers should always be enabled.
- NTP associations should be limited to only trusted servers through access control lists.

In most situations, it is recommended to have at least three higher-stratum NTP servers configured for each network. A large number of public NTP servers can be used for these purposes.

LAYER 2 VERSUS LAYER 3 NETWORK DIAGRAMS

Network diagrams are a critical component in the documentation process for any network. Just as software developers document their code for easier maintenance and sharing, network engineers document networks by building network diagrams. It is very important to have accurate network diagrams, especially when troubleshooting network issues. The network has become a business differentiator for all companies, and each second of downtime costs money. Having up-to-date network diagrams makes it much easier and faster to troubleshoot network issues and decrease the MTTR (mean time to resolution) when problems arise.

Several different types of network diagrams can be created, depending on what needs to be emphasized and documented. Layer 1 network diagrams can be used to show physical connections, including how network devices are connected and what type of cables (twisted pair, fiber optics, and so on) are being used. These diagrams can also show patch panel port connectivity and availability and everything else that is in the confines of physical connectivity.

Layer 2 network diagrams contain information related to all Layer 2 devices, protocols, and connectivity in a network. Such a diagram shows all the switches in the network and how they are interconnected, including how the ports on one switch connect to the ports on another switch, which VLANs are configured on each switch, which ports are configured as trunks and what VLANs are allowed on them, which ports are configured as port channels, and any other Layer 2 information. Depending on the size of the network, a Layer 2 diagram can be split into multiple documents.



A Layer 3 network diagram captures all the network information available at Layer 3. Such a diagram should show all devices that operate at Layer 3—routers, firewalls, load balancers, and so on—and how they are interconnected. Information about IP addressing and subnets, routing protocols, first-hop redundancy protocols (such as HSRP, VRRP, and GLBP), Layer 3 port channels, and Internet connectivity should also be included at a minimum. More information can also be included in these diagrams but care should be taken to avoid including too much information or dynamic information that is bound to change often.



Several software tools are available for creating network diagrams. You can create network diagrams manually by using tools such as Microsoft Visio and draw.io, or you can have them created automatically with controller-based solutions such as Cisco DNA Center. With Cisco DNA Center, the controller automatically discovers all the devices in the network and how they are interconnected, and it uses this information to build network diagrams, topologies, and databases.

It is a good idea to store network diagrams in version control systems such as Git. Networks are dynamic and evolve over time, and it is important to capture this evolution in the diagrams. Git offers a proven solution to storing and keeping track of changes in documents, so it is ideal for storing network diagram versions. With infrastructure as code and solutions such as Cisco VIRT, network diagrams and topologies are stored as YAML files and can be used as part of CI/CD infrastructure configuration automation pipelines.

TROUBLESHOOTING APPLICATION CONNECTIVITY ISSUES

Applications are becoming more and more complicated, with many moving components that need to communicate with each other over the network. With the advent of the microservices architecture for applications and the ubiquity of APIs, the network is critical to the functionality of applications. It is important to know how to troubleshoot connectivity issues and how they might affect the performance of applications.

Network connectivity may not function as expected for a variety of reasons. Whenever an application or a website or any target destination that is being accessed over a network is not behaving as expected, the network takes the blame. We will see next that while the network might be the culprit in some instances, there are many other reasons applications stop responding as expected.

Network troubleshooting usually follows the OSI layers, discussed in [Chapter 16, “Network Fundamentals.”](#) It can occur from top to bottom, beginning at the application layer and going down the layers all the way to the physical layer, or it can occur from bottom to top. This section looks at a typical bottom-to-top troubleshooting session that starts from the physical layer and goes up the stack toward the application layer. The troubleshooting steps discussed here can also be followed in the reverse order, if desired.



First and foremost, it is important to know how an endpoint device connects to the network at the physical layer: with a wired or wireless connection. If the connection to the network is wired through an Ethernet cable, the network interface card should come online, and electrical signals should be exchanged with the switch port to which the NIC is connected. Depending on the operating system of the client connecting to the

network, the status of the connection will show as solid green or will display as “enabled” or “connected” in the network settings. If the NIC status shows as connected, the physical layer is working as expected, and the troubleshooting process can proceed to the next layer. If the NIC doesn’t show as connected or enabled, there could be several causes, including the following:

- Misconfigured or disabled switch port
- Defective network cable
- Defective wall network port
- Incorrect cabling in the patch panel
- Defective network interface card

Troubleshooting at the physical layer revolves around making sure there is an uninterrupted physical connection between the client and the switch port to which it connects.

If the connection to the network is wireless, it is important to ensure that the wireless network interface card is turned on and that it can send and receive wireless signals to and from the nearest wireless access point. Being within the service range of a wireless access point is also important, and usually the closer the client is to the access point, the better network performance it should experience.

Troubleshooting at the data link layer, or Layer 2, means making sure the network client and the switch are able to learn MAC addresses from each other. On most operating systems, the client can check the MAC address table with the **arp** command, and on most Cisco switches, the client can check the MAC address table with the **show mac address-table** CLI command. If the ARP table on both the client and the switch get dynamically populated with MAC address information, it means the data link layer is functioning as expected. Some of the issues that cause the data link layer not to function as expected are as follows:

Key Topic

- Misconfigured switch port
- Layer 2 access control lists
- Misconfigured Spanning Tree Protocol
- Missing or misconfigured VLANs

At Layer 3 (the Internet layer), IP connectivity and reachability have to work. The network client should be configured with the correct IP address, subnet mask, and default gateway. This is usually done using DHCP servers, and in rare cases it may be manually configured. Built-in operating system tools such as **ifconfig** and **ping** can be used at this layer to help with troubleshooting. **ifconfig** (or **ipconfig** on Microsoft Windows) is a network utility that retrieves and can modify the configuration and status of all network interfaces present on the client endpoint. **ping** is another popular network tool that is extensively used to verify endpoint IP reachability. If the destination of the client data traffic is in a different subnet than the client is connected to, that means that the traffic has to be routed through the default gateway of the client subnet. Layer 3 connectivity between data traffic source and destination can be checked using the **ping** command. If IP connectivity is verified end to end between source and destination, troubleshooting can proceed to the next step. If not, you need to look for the problems that can cause connectivity issues at Layer 3, including these:

Key Topic

- Misconfigured IP information (IP address, subnet mask, default gateway) on the client device
- Layer 3 access control lists that blocks data traffic
- Routing protocol issues causing black-holing or incorrect routing of traffic

Troubleshooting at the transport layer means making sure that the network clients can access the TCP or UDP ports on which the destination applications are running. For example, in the case of web traffic, it is important to verify that the client can connect to TCP ports 80 (HTTP) and/or 443 (HTTPS) on the web server. In some cases, web servers are configured to listen on esoteric ports such as 8080, so it is important to know the correct port on which the destination application is running. Networking tools such as **curl** and custom **telnet** commands specifying the application port can be used to ensure that transport layer connectivity can be established end to end between the source and destination. If a transport layer connection cannot be established, you need to look for issues such as these:



- Firewall access control lists blocking data traffic based on TCP and UDP ports
- Misconfigured applications and listening ports
- Misconfigured load balancers
- Presence of proxy servers that are intercepting the traffic and denying connectivity
- Misconfigured PAT

Other common problems that affect application connectivity are DNS related. As discussed earlier in this chapter, DNS plays a critical role in resolving domain names to IP addresses. If DNS is malfunctioning for some reason, end-to-end connectivity is impacted. Network tools such as **nslookup** can be used to troubleshoot DNS functionality. The following problems commonly cause DNS issues:

- Misconfigured DNS resolver on the network client
- Wrong hostname specified
- Invalid DNS server configuration
- Missing or incorrect DNS entry

Even if end-to-end connectivity at the transport layer is verified, there can still be issues on the network that cause connections to applications to fail. These issues are usually rather difficult to discover and troubleshoot and are related to data traffic load and network delay. The difficulty with these issues comes from the fact that they are difficult to reproduce and can be temporary in nature, caused by short spikes in network traffic.

Networking tools such as **iperf** can be used to dynamically generate traffic and perform load stress on the network to ensure that large amounts of data can be transported between the source and destination.

Implementing quality of service (QoS) throughout the network can help with these problems. With QoS, traffic is categorized in different buckets, and each bucket gets separate network treatment. For example, you can classify traffic such as voice and video as real-time traffic by changing QoS header fields in the Layer 2 data frames and Layer 3 packets so that when switches and routers process this type of traffic, they give it a higher priority and guaranteed bandwidth, if necessary.



At the application layer, network tools such as **tcpdump** can be used to capture actual data traffic received on any of the network interfaces of either the source device or the destination device. Comparing between sent and received data can help in troubleshooting connectivity issues and determining the root cause of a problem. Slow or no responses at the application layer could indicate an overloaded backend database, misconfigured load balancer, or faulty code introduced through new application features.

EXAM PREPARATION TASKS

As mentioned in the section “How to Use This Book” in the Introduction, you have a couple of choices for exam preparation: the exercises here, [Chapter 19](#), “[Final Preparation](#),” and the exam simulation questions on the companion website.

REVIEW ALL KEY TOPICS

Review the most important topics in this chapter, noted with the Key Topic icon in the outer margin of the page. [Table 18-2](#) lists these key topics and the page number on which each is found.



Table 18-2 Key Topics

Key Topic Element	Description	Page Number
List	Benefits of DHCP	5 3 5
List	Four phases of DHCP operations	5 3 6
List	DNS resolution process components	5 3 8
Paragraph	User Datagram Protocol (UDP)	5 4 0
Paragraph	Network Address Translation (NAT)	5 4 0
Paragraph	IETF NAT, described in RFC 1631	5

raph		4 0
Parag raph	Port Address Translation (PAT)	5 4 1
List	Advantages of SNMP	5 4 3
Parag raph	SNMP agent listening for requests from the SNMP manager	5 4 4
List	SNMP message types	5 4 5
Parag raph	Main role of NTP	5 4 6
Parag raph	Two forms of communication between NTP clients and servers	5 4 6
Parag raph	Layer 2 network diagrams	5 4 7
Parag raph	Layer 3 network diagrams	5 4 7
Parag raph	Network troubleshooting	5 4 8
Parag raph	Troubleshooting at the data link layer	5 4 8
Parag raph	Troubleshooting at the Internet layer	5 4

		9
Parag raph	Troubleshooting at the transport layer	5 4 9
Parag raph	Network issues with application connection failures	5 5 0

DEFINE KEY TERMS

Define the following key terms from this chapter and check your answers in the glossary:

Dynamic Host Configuration Protocol (DHCP)

Domain Name System (DNS)

top-level domain (TLD) name server

Network Address Translation (NAT)

Port Address Translation (PAT)

Simple Network Management Protocol (SNMP)

Network Management Station (NMS)

Management Information Base (MIB)

Object Identifier (OID)

Network Time Protocol (NTP)

Chapter 19

Final Preparation

The first 18 chapters of this book cover the technologies, protocols, design concepts, and considerations required to be prepared to pass the 200-901 DevNet Associate DEVASC exam. While those chapters supply the detailed information, most people need more preparation than simply reading the first 18 chapters of this book. This chapter provides a set of tools and a study plan to help you complete your preparation for the exam.

This short chapter has three main sections. The first section helps you get ready to take the exam, and the second section lists the exam preparation tools useful at this point in the study process. The third section provides a suggested study plan you can follow, now that you have completed all the earlier chapters in this book.

GETTING READY

Here are some important tips to keep in mind to ensure that you are ready for this rewarding exam:

- **Build and use a study tracker:** Consider using the exam objectives shown in this chapter to build a study tracker for yourself. Such a tracker can help ensure that you have not missed anything and that you are confident for your exam. As a matter of fact, this book offers a sample Study Planner as a website supplement.
- **Think about your time budget for questions on the exam:** When you do the math, you will see that, on average, you have one minute per question. While this does not sound like a lot of time, keep in mind that many of the questions will be very straightforward, and you will take 15 to 30 seconds on those. This leaves you extra time for other questions on the exam.

- **Watch the clock:** Check in on the time remaining periodically as you are taking the exam. You might even find that you can slow down pretty dramatically if you have built up a nice block of extra time.
- **Get some earplugs:** The testing center might provide earplugs but get some just in case and bring them along. There might be other test takers in the center with you, and you do not want to be distracted by their screams. I personally have no issue blocking out the sounds around me, so I never worry about this, but I know it is an issue for some.
- **Plan your travel time:** Give yourself extra time to find the center and get checked in. Be sure to arrive early. As you test more at a particular center, you can certainly start cutting it closer time-wise.
- **Get rest:** Most students report that getting plenty of rest the night before the exam boosts their success. All-night cram sessions are not typically successful.
- **Bring in valuables but get ready to lock them up:** The testing center will take your phone, your smartwatch, your wallet, and other such items and will provide a secure place for them.
- **Take notes:** You will be given note-taking implements and should not be afraid to use them. I always jot down any questions I struggle with on the exam. I then memorize them at the end of the test by reading my notes over and over again. I always make sure I have a pen and paper in the car, and I write down the issues in my car just after the exam. When I get home—with a pass or fail—I research those items!

TOOLS FOR FINAL PREPARATION

This section lists some information about the available tools and how to access the tools.

Pearson Cert Practice Test Engine and Questions on the Website

Register this book to get access to the Pearson IT Certification test engine (software that displays and grades a set of exam-realistic multiple-choice questions). Using the Pearson Cert Practice Test Engine, you can either study by going through the questions in Study mode or take a simulated (timed) DevNet Associate DEVASC exam.

The Pearson Test Prep practice test software comes with two full practice exams. These practice tests are available to you either online or as an offline Windows application. To access the practice exams that were developed with

this book, please see the instructions in the card inserted in the sleeve in the back of the book. This card includes a unique access code that enables you to activate your exams in the Pearson Test Prep software.

Accessing the Pearson Test Prep Software Online

The online version of this software can be used on any device with a browser and connectivity to the Internet, including desktop machines, tablets, and smartphones. To start using your practice exams online, simply follow these steps:

- Step 1.** Go to <http://www.PearsonTestPrep.com>.
- Step 2.** Select **Pearson IT Certification** as your product group.
- Step 3.** Enter your email and password for your account. If you don't have an account on PearsonITCertification.com or CiscoPress.com, you need to establish one by going to PearsonITCertification.com/join.
- Step 4.** In the **My Products** tab, click the **Activate New Product** button.
- Step 5.** Enter the access code printed on the insert card in the back of your book to activate your product.
- Step 6.** The product will now be listed in your My Products page. Click the Exams button to launch the exam settings screen and start your exam.

Accessing the Pearson Test Prep Software Offline

If you wish to study offline, you can download and install the Windows version of the Pearson Test Prep software. You can find a download link for this software on the book's companion website, or you can just enter this link in your browser:

<http://www.pearsonitcertification.com/content/downloads/pcpt/engine.zip>

To access the book's companion website and the software, simply follow these steps:

- Step 1.** Register your book by going to [PearsonITCertification.com/register](http://www.pearsonitcertification.com/register) and entering the ISBN **9780136642961**.
- Step 2.** Respond to the challenge questions.
- Step 3.** Go to your account page and select the Registered Products tab.
- Step 4.** Click on the Access Bonus Content link under the product listing.
- Step 5.** Click the Install Pearson Test Prep Desktop Version link in the Practice Exams section of the page to download the software.
- Step 6.** When the software finishes downloading, unzip all the files onto your computer.
- Step 7.** Double-click the application file to start the installation and follow the onscreen instructions to complete the registration.
- Step 8.** When the installation is complete, launch the application and click the Activate Exam button on the My Products tab.
- Step 9.** Click the Activate a Product button in the Activate Product Wizard.
- Step 10.** Enter the unique access code from the card in the sleeve in the back of your book and click the Activate button.
- Step 11.** Click Next and then click the Finish button to download the exam data to your application.
- Step 12.** You can now start using the practice exams by selecting the product and clicking the Open Exam button to open the exam settings screen.

Note that the offline and online versions sync together, so saved exams and grade results recorded on one version will be available to you in the other version as well.

Customizing Your Exams

Once you are in the exam settings screen, you can choose to take exams in one of three modes:

- Study mode
- Practice Exam mode
- Flash Card mode

Study mode allows you to fully customize your exam and review answers as you are taking the exam. This is typically the mode you use first to assess your knowledge and identify information gaps. Practice Exam mode locks certain customization options in order to present a realistic exam experience. Use this mode when you are preparing to test your exam readiness. Flash Card mode strips out the answers and presents you with only the question stem. This mode is great for late-stage preparation, when you really want to challenge yourself to provide answers without the benefit of seeing multiple-choice options. This mode will not provide the detailed score reports that the other two modes provide, so you should not use it if you are trying to identify knowledge gaps.

In addition to these three modes, you will be able to select the source of your questions. You can choose to take exams that cover all of the chapters, or you can narrow your selection to just a single chapter or the chapters that make up specific parts in the book. All chapters are selected by default. If you want to narrow your focus to individual chapters, simply deselect all the chapters and then select only those on which you wish to focus in the Objectives area.

You can also select the exam banks on which to focus. Each exam bank comes complete with a full exam of questions that cover topics in every chapter. Two exams are available to you online as part of the print book, as well as two additional exams of unique questions. You can have the test engine serve up exams from all four banks or just from one individual bank by selecting the desired banks in the exam bank area.

There are several other customizations you can make to your exam from the exam settings screen, such as the time allowed for taking the exam, the number of questions served up, whether to randomize questions and answers, whether to show the number of correct answers for multiple-answer questions, and whether to serve up only specific types of questions. You can also create custom test banks by selecting only questions that you have marked or questions on which you have added notes.

Updating Your Exams

If you are using the online version of the Pearson Test Prep software, you should always have access to the latest version of the software as well as the exam data. If you are using the Windows desktop version, every time you launch the software, it will check to see if there are any updates to your exam data and automatically download any changes made since the last time you used the software. This requires that you are connected to the Internet at the time you launch the software.

Sometimes, due to a number of factors, the exam data might not fully download when you activate your exam. If you find that figures or exhibits are missing, you might need to manually update your exams.

To update a particular exam you have already activated and downloaded, simply select the **Tools** tab and select

the **Update Products** button. Again, this is only an issue with the desktop Windows application.

If you wish to check for updates to the Windows desktop version of the Pearson Test Prep exam engine software, simply select the **Tools** tab and click the **Update Application** button. This will ensure you are running the latest version of the software engine.

Premium Edition

In addition to the free practice exam provided on the website, you can purchase additional exams with expanded functionality directly from Pearson IT Certification. The Premium Edition of this title contains an additional two full practice exams and an eBook (in both PDF and ePub format). In addition, the Premium Edition title also has remediation for each question to the specific part of the eBook that relates to that question.

Because you have purchased the print version of this title, you can purchase the Premium Edition at a deep discount. There is a coupon code in the book sleeve that contains a one-time-use code and instructions for where you can purchase the Premium Edition.

To view the Premium Edition product page, go to www.informit.com/title/9780136642985.

Chapter-Ending Review Tools

Chapters 2 through 18 include several features in the “Exam Preparation Tasks” section at the end of the chapter. You might have already worked through these in each chapter. It can also be useful to use these tools again as you make your final preparations for the exam.

SUGGESTED PLAN FOR FINAL REVIEW/STUDY

This section lists a suggested study plan from the point at which you finish reading through Chapter 18 until you take the 200-901 DevNet Associate DEVASC exam. You can ignore this plan, use it as is, or just take suggestions from it.

The plan involves two steps:

Step 1. Review key topics and “Do I Know This Already?” (DIKTA?) questions: You can use the table that lists the key topics in each chapter or just flip the pages, looking for key topics. Also, reviewing the DIKTA? questions from the beginning of the chapter can be helpful for review.

Step 2. Use the Pearson Cert Practice Test engine to practice: The Pearson Cert Practice Test engine allows you to study using a bank of unique exam-realistic questions available only with this book.

SUMMARY

The tools and suggestions listed in this chapter have been designed with one goal in mind: to help you develop the skills required to pass the 200-901 DevNet Associate DEVASC exam. This book has been developed from the beginning to not just tell you the facts but to also help you learn how to apply the facts. No matter what your experience level leading up to when you take the exam, it is our hope that the broad range of preparation tools, and even the structure of the book, will help you pass the exam with ease. We hope you do well on the exam.

Appendix A

Answers to the “Do I Know This Already?” Quiz Questions

CHAPTER 1

- 1.** A, B, D. The five levels of Cisco accreditation are Entry, Associate, Professional, Expert, and Architect.
- 2.** A, B, E. Highlighting skills to employers and peers, increasing confidence, improving credibility, providing value to employers, providing a baseline of understanding, career advancement, and increased salary are some of the most common reasons candidates want to get certified.
- 3.** A, E. Only two types of exams are necessary to obtain the DevNet Professional certification: the Technology Core exam and a single concentration exam.
- 4.** A. A single exam is all that is required for the new CCNA certification.
- 5.** A, B, C, E. Cisco Automation Platform doesn't exist. The DevNet Automation Exchange is a place on DevNet to download fully tested and working use case–driven code-based examples.

CHAPTER 2

- 1.** C. Waterfall is a linear and sequential process for software development.

2. B. Agile is an implementation of the Lean management philosophy for software development.
3. A, B, D. Model-View-Controller is a software design pattern used to create interactive graphical web-centric applications. It can be found in Django (a web app framework) as well as many web applications that require support for many types of clients.
4. A, C. The Observer pattern follows the publisher/subscriber model, where a client subscribes to the publisher and synchronizes its configuration state. This is perfect for one-to-many device configuration and management of event handling in infrastructure components.
5. C. BASH stands for Bourne Again Shell.
6. B. `env | more` pipes the contents of your environment variables to the **more** command and allows for page breaks.
7. B. Software version control is also commonly known as source code management.
8. D. Linus Torvalds, the father of Linux, created Git.
9. B. The three main structures tracked by Git are local workspace, index, and local repository.
10. D. To add a specific file to the Git index, you use the command **git add** followed by the name of the file you want to add.
11. A. One of the main benefits of conducting formal code reviews is to help you the developer, create higher-quality software.

CHAPTER 3

1. D. The correct command is **python3 -m (for module) venv myvenv** (which can be whatever you choose to name your virtual environment).

2. B, C. PyPI is a repository that holds thousands of Python modules that you can import. To install it, you can use `python3 -m (module) pip install` and the name of the package you want. You can also directly install it with the `pip` command.
- 3.** B. PEP 8 is the style guide for Python syntax, and it specifies four spaces for each block of code. Tabs will work, and your editor may actually convert them automatically for you, but it is a good practice to follow the standard.
4. B. Comments are specified by the `#` or three single quotes `'''`. The benefit of using the single quotes is that you can write multiline text.
- 5.** A, B. Lists and dictionaries are both mutable, or changeable, data types. Integers and tuples must be replaced and can't be edited, which makes them immutable.
6. B, D. You can create an empty dictionary object by assigning the function `dict()` to a Python object (in this example, `a`). You can insert dictionary values as well by using braces, `{}`, and key:value pairs that you assign to an object.
- 7.** C. The `input()` function by default creates a string data type.
8. D. The only one that is valid as is would be `print("hello world")`. `print(hello, world)` would be valid only if there were two variables named `hello` and `world`; otherwise, it would produce an error. Since you don't know in this case whether these variables exist, `print("hello world")` is the correct answer.
9. D. All of the statements are correct regarding how an `If` statement operates in Python.
10. A, D. The `range()` function in Python allows you to set a range that is up to but not including the number specified. If you want to increment to 10, for example, you need to provide the number 11.

A range can also count up or down based on the sign of the number provided.

CHAPTER 4

- 1.** C. A function in Python uses the **def** keyword followed by a function name and parentheses with an optional argument inside.
- 2.** D. Python has specific rules for variable names. A variable cannot be named using reserved keywords, such **True**, and must start with a letter or an underscore but not a number.
- 3.** B. A docstring is text that describes the purpose and use of a Python function, and it is located on the very next line after the function definition. The docstring can be viewed with the **help()** function in Python.
- 4.** A, B. The key components of object-oriented programming in Python are functions that can be performed on a data structure and attributes that are stored in an object.
- 5.** A, B, C. The benefits of OOP are reusable code, easy-to-follow code, and low coupling/high cohesion between application components.
- 6.** A, B. A Python class can be defined using the keyword **class**, a name for the class, and optionally a parent class to inherit attributes and functions from.
- 7.** C. A method is a function that you can perform on an object. Methods are often defined in classes. They can also make use of externally defined functions as well.
- 8.** B. Inheritance allows you to define a base class and then define another class using the previous class as a parent. The subsequent class includes all the capabilities of the parent class and can add to or override any attribute or method that needs to be different.

- 9. D. The **os** module allows you to interact with the file system and directory structure of the operating system.
- 10. A. Cisco built **pyATS** and released it to open source for testing Cisco infrastructure software.

CHAPTER 5

- 1. C. The end-of-line is the last character of a line of text before the text wraps to the next line. This is identified as `\n` or as EoF (end of file).
- 2. A. In order to open a file for writing, you need to use the **open()** function and assign it to a file handler variable—in this case, `data`. In addition, you need to pass the name of the file and tell Python that you want to allow write access to it (using “w”).
- 3. C. Using the **with open** combo to the `text.csv` file, you map it to the filehandle object. Map **csv_writer**, which is just another Python object, to the **csv.writer(filehandle)** function. Next, you write your data to the CSV file by calling the **.writerow** method.
- 4. B. The `xmltodict` module reads XML data.
- 5. A, C. To load a native JSON file into a Python string object, you use **loads()**, which stands for load string, and to convert a Python string into native JSON, you use **dump()**.
- 6. B. YAML stands for YAML Ain’t Markup Language.
- 7. A. Error handling in Python can be conducted by using a **try-except-else-finally** block.
- 8. C. **finally** is executed every time the code runs through the **try** block. It is often used to clean up variables or alert the user to a success or failure event.

9. C. Test-driven development focuses on writing code that allows a previously written test (that naturally fails since no code was written) to succeed.
10. A, D. An integration test is for API verification, and a function test verifies that your application meets the agreed-upon requirements of how it should operate.
11. B. `unittest.TestCase` is a special class that is used to access the `unittest` module's capabilities.

CHAPTER 6

1. A. Southbound APIs send information down to devices within the network.
2. A, B. Because asynchronous APIs do not have to wait for replies, they reduce the time required to process data.
3. A, D, E. SOURCE and PURGE do not exist. GET, POST, PUT, PATCH, and DELETE are the HTTP functions.
4. A. Both API keys and custom tokens are commonly used within API authentication.
5. C. SOAP stands for Simple Object Access Protocol.
6. A, B, D, E. The four main components of a SOAP message are the envelope, header, body, and fault. The fault is an optional component.
7. A. RPCs are blocked during the waiting periods. Once a procedure is executed and the response is sent from the server and received on the client, the execution of the procedure continues. This is similar to a synchronous API.

CHAPTER 7

1. A, B. In order to make a successful call—whether it is a GET or a POST—a client must have the

URL, the method, an optional header, and an optional body.

- 2.** C. TRIGGER is not a defined HTTP method.
- 3.** B, D. Webhooks are like callback functions for the web; they handle notifications from the server. Also, these notifications are triggered by events.
- 4.** B. The 3xxx HTTP codes are for redirection. When a resource is moved, a server sends a 3xx code.
- 5.** B. Sequence diagrams model the interactions between various objects in a single use case.
- 6.** C. Code on demand provides flexibility to a client by allowing it to download code. For example, a client can download JavaScript and execute it.
- 7.** D. Rate-limiting techniques help in limiting the security surface, allowing various business models (from freemium to paid), and improving the efficiency of the entire system.
- 8.** B. The processes of encoding and decoding JSON are usually called serialization and deserialization, respectively. Python dictionaries (dicts) are used as standard data types for a lot of Python request functions.

CHAPTER 8

- 1.** A, B, C. A good SDK is easy to use, well documented, integrated well with other SDKs, has a minimal impact on hardware resources, and provides value-added functionality.
- 2.** A, B. Some of the advantages of using an SDK are quicker integration, faster and more efficient development, brand control, increased security, and the availability of metrics.
- 3.** A, B. The Cisco Meraki cloud platform provides the following APIs to developers: Captive Portal

API, Scanning API, MV Sense Camera API, and Dashboard API.

4. A. The Cisco Meraki Dashboard API authentication header is called X-Cisco-Meraki-API-Key.
5. A. The base URL for the Cisco Meraki Dashboard API is <https://api.meraki.com/api/v0>.
6. C. Cisco DNA Center API authentication is based on basic auth.
7. A. The **timestamp** parameter in Cisco DNA Center APIs is in UNIX epoch time, in milliseconds.
8. B. Cisco DNA Center allows customers to have their non-Cisco devices managed by DNA Center through a multivendor SDK. Cisco DNA Center communicates with the third-party devices through device packages. The device packages are developed using the multivendor SDK and implement southbound interfaces based on the CLI, SNMP, or NETCONF.
9. C. Cisco vManage is a centralized network management system that provides a GUI and REST API interface to the SD-WAN fabric.
10. D. The information sent over the authentication POST call is URL form encoded and contains the username and password for the vManage instance.

CHAPTER 9

1. B. The Cisco Nexus 9000 family of switches can run in two separate modes of operation, depending on the software that is loaded. The first mode is called standalone (or NX-OS) mode, which means the switches act like regular Layer 2/Layer 3 data center devices, which are usually managed individually. The second mode is called ACI mode, in which the Cisco Nexus devices are

part of an ACI fabric and are managed in a centralized fashion.

- 2.** B. Bridge domains represent the Layer 2 forwarding domains within the fabric and define the unique MAC address space and flooding domain for broadcast, unknown unicast, and multicast frames. Each bridge domain is associated with only one VRF instance, but a VRF instance can be associated with multiple bridge domains.
- 3.** A. APIC REST API username- and password-based authentication uses a special URI that includes `aaaLogin`, `aaaLogout`, and `aaaRefresh` as the DN targets of a POST operation.
- 4.** B. The service profile is a logical construct in UCS Manager that contains the complete configuration of a physical server. All the elements of a server configuration—including RAID levels, BIOS settings, firmware revisions and settings, adapter settings, network and storage settings, and data center connectivity—are included in the service profile.
- 5.** A. The Cisco UCS Python module for UCS Manager is called `ucsmsdk`. It can be installed using pip by issuing the following command at the command prompt: **`pip install ucmsdk`**.
- 6.** C. Cisco UCS Manager provides a managed object browser called Visore. Visore can be accessed by navigating to `https://<UCS-Manager-IP>/visore.html`.
- 7.** D. A workflow is a series of tasks arranged to automate a complex operation. The simplest workflow contains a single task, but workflows can contain any number of tasks.
- 8.** B. Each REST API request must be associated with an HTTP header called `X-Clouppia-Request-`

Key, with its value set to the REST API access key.

- 9.** C. The Intersight API is a programmatic interface to the Management Information Model similar to Cisco ACI and Cisco UCS Manager. Just like Cisco ACI and Cisco UCS Manager, the Cisco Intersight Management Information Model is comprised of managed objects.
- 10.** A. An Intersight API key is composed of a keyId and a keySecret. The API client uses the API key to cryptographically sign each HTTP request sent to the Intersight web service.

CHAPTER 10

- 1.** A, B, C. Cisco's collaboration portfolio allows video calling, integration of bots, Remote Expert use cases.
- 2.** A, B, D. Teams allows users, third-party apps, and bots to interact with its APIs.
- 3.** C. A JWT token is generated using the guest issuer ID and secret.
- 4.** A. Bots use webhooks to handle events.
- 5.** B. Finesse has a rich API set, and the desktop application is completely built using APIs.
- 6.** B. The Finesse Notification Service sends XMPP over BOSH messages to agents that are subscribed to certain XMPP nodes.
- 7.** A. SetMeeting lets you modify the attributes of a meeting after the meeting has been created.
- 8.** D. xAPI allows developers to programmatically invoke commands and query the status of devices that run collaboration endpoint software or Webex RoomOS software.
- 9.** A, C, D. xAPI does not work with FTP.
- 10.** B. The Administration XML (AXL) API provides a mechanism for inserting, retrieving, updating,

and removing data from Cisco Unified Communications Manager.

CHAPTER 11

- 1.** B. The Investigate API provides enrichment of security events with intelligence to SIEM or other security visibility tools.
- 2.** C. The Umbrella Enforcement API involves an HTTP POST request, which internally comprises the Investigate API to check whether the domain is safe.
- 3.** A. The response header contains the token **X-auth-access-token**, which needs to be used in all subsequent API calls.
- 4.** B. A named object is a reusable configuration that associates a name with a value.
- 5.** C. APTs allow bad actors to gain access to and control endpoint resources over an extended period to steal valuable data without being detected.
- 6.** B. ISE enables devices and users to be identified and provisioned and enables policies to be applied.
- 7.** B. Threat Grid is a unified malware analysis and threat intelligence platform.
- 8.** B. IOCs are used to indicate that the system has been affected by some form of malware.

CHAPTER 12

- 1.** A, B, D. There are three standards-based programmable interfaces for operating on the YANG data models: NETCONF, RESTCONF, and gRPC.
- 2.** B. By default, the NETCONF server on the device runs on TCP port 830 and uses the SSH process for transport.

3. D. Messages sent with NETCONF use remote procedure calls (RPCs), a standard framework for clients to send a request to a server to perform an action and return the results.
4. C. YANG defines a set of built-in types and has a mechanism through which additional types can be defined. There are more than 20 base types, including binary, enumeration, and empty. Percent is not a built-in data type.
5. C. The YANG header contains the namespace for the module. The namespace is used to uniquely identify each module on a system that implements NETCONF.
6. B. One popular NETCONF client is the Python 3 ncclient library.
7. A, D. RESTCONF data is encoded with either XML or JSON. Compared with NETCONF, RESTCONF has added support for JSON encoding.
8. A. The PATCH method provides a resource patching mechanism. It is equivalent to the NETCONF <edit-config> operation with operation=merge.
9. C. Per the RESTCONF standard, devices implementing the RESTCONF protocol should expose a resource called /.well-known/host-meta to enable discovery of root programmatically.
10. B, C. There are two types of telemetry subscriptions. With a dynamic subscription, the subscriber sends a request, usually via the **ietf-yangpush.yang** data model. A configured subscription is configured via the CLI, NETCONF, or RESTCONF and is persistent between reboots.

CHAPTER 13

1. D. A SaaS provider offers software for use and maintains all aspects of it. You may be allowed to customize parts of the software configuration, but typically you are not allowed to change anything regarding how the software functions.
2. B. IoT is heavily reliant on sensors detecting and responding to events in real time. The edge computing model allows for centralized control of IoT sensors and real-time response because computing capabilities are closer to the sensor.
3. A, C. Containers differ from virtual machines in that they are lighter (fewer resources used in storage), and they can start as fast as an application (500 ms), whereas a virtual machine requires the guest operating system to boot so takes more time to start.
4. A. Serverless deployment is great for applications that process data periodically but not designed for continuous use.
5. D. The second way of DevOps is the feedback loop for continuous improvement.
6. C. Continuous integration refers to the merging of development work with a shared code base to facilitate automatic testing and software building.
7. D. Docker natively uses the union file system for container construction.
8. C. To launch an nginx container on port 80, you run the command **docker container run -p 80:80 -d nginx**.

CHAPTER 14

1. B. A vulnerability is a weakness or gap in protection efforts. It can be exploited by threats to gain unauthorized access to an asset.
2. B, D. A man-in-the-middle attack allows devices to receive traffic as it flows in the network; a

brute-force attack involves using trial and error to crack passwords.

- 3.** A. Penetration (pen) testing is commonly used to find weak spots.
- 4.** B. Nmap is the Network Mapper tool, which is used for network discovery and security auditing.
- 5.** B. MFA uses at least two identity components to authenticate a user's identity.
- 6.** A, C. A one-way hash is used for fingerprinting data.
- 7.** A. Data needs to be secured in multiple locations: while it is in motion (network), at rest (storage), and while in use (memory).
- 8.** B. An IDS is passive, as it receives a copy of a packet, whereas an IPS is active, working on live traffic.

CHAPTER 15

- 1.** B, D. Historically, network devices were managed through command-line interfaces (CLIs) using protocols such as Telnet and Secure Shell (SSH).
- 2.** C. We've seen in Chapter 8 an example of a network controller with Cisco DNA Center. Cisco DNA Center can be used to completely configure, manage, and monitor networks.
- 3.** B, C. There are usually two types of approaches to infrastructure as code: declarative and imperative. With the declarative approach, the desired state of the system is defined, and then the system executes all the steps that need to happen in order to attain the desired state. The imperative approach defines a set of commands that have to be executed in a certain order for the system to achieve the desired state.
- 4.** A, C. Implementing infrastructure as code processes leads to shorter deployment times for

new infrastructure and faster and easier troubleshooting steps.

5. D. A key component of any CI pipeline is the build server, which reacts to developers committing their code into the central repository and starts the initial tests on the new code features.
6. B. The four steps of a CI/CD pipeline are source, build, test, and deploy.
7. B. Ansible playbooks can be run from a terminal with the **ansible-playbook** command.
8. A. The Puppet manifests are standard text files that contain Puppet Domain Specific Language (DSL) code and have the .pp extension.
9. C. Recipes are authored in Ruby, and most of them contain simple configuration patterns that get enforced through the Chef client.
10. B. The NSO Device Manager manages network devices using YANG data models and NETCONF. For devices that natively implement NETCONF and YANG models, the device manager is automatic, and devices that do not support NETCONF are integrated in the platform by using Network Element Drivers (NEDs).
11. C. Network topologies are stored as YAML files and can be easily modified and shared.
12. A, D. The pyATS solution is composed of two main components: the pyATS test framework and the pyATS library, which used to be called Genie but was renamed in an effort to simplify the nomenclature of the product.

CHAPTER 16

1. B. The transport layer, as the name suggests, is responsible for end-to-end transport of data from the source to the destination of the data traffic. Connection-oriented protocols at the transport

layer establish an end-to-end connection between the sender and the receiver, keep track of all the segments that are being transmitted, and have a retransmission mechanism in place.

- 2.** C. The Internet layer in the TCP/IP reference model corresponds in functions and characteristics to the network layer in the OSI model.
- 3.** B, D. There are a large number of application layer protocols, including the Hypertext Transfer Protocol (HTTP), which is used for transferring web pages between web browsers and web servers, and File Transfer Protocol (FTP), which is used for transferring files between a client and a server.
- 4.** D. The TCP/IP reference model transport layer PDU is called a *segment*.
- 5.** D. The Preamble field consists of 8 bytes of alternating 1s and 0s that are used to synchronize the signals of the sender and receiver.
- 6.** B. A MAC address has 48 bits organized as 12 hexadecimal numbers.
- 7.** D. If a switch doesn't have the destination MAC address of the frame, it floods the frame over all the ports except the port on which it was received. As the frame gets flooded throughout the network, eventually it will get to its destination.
- 8.** A. The bit pattern for the first byte in a Class C IPv4 address is 110xxxxx.
- 9.** C. The broadcast address for the network 192.168.0.96/27 is 192.168.0.127.
- 10.** A, B, D. An IPv6 address is 128 bits long, with colons separating entries (x:x:x:x:x:x:x, where x is a 16-bit hexadecimal field). Successive fields of zero can be represented as :: but only once per

address, and the hexadecimal characters are not case sensitive.

CHAPTER 17

- 1.** A. Ethernet is a star topology that used concentrators to connect all the nodes in the network.
- 2.** C. A WAN covers a large geographic area and uses public networks.
- 3.** B. With a hub, every frame shows up at every device attached to a hub, which can be really harmful in terms of privacy.
- 4.** C. Multiple VLANs can be implemented with Layer 3 switches, so multiple broadcast can be supported.
- 5.** C. Virtual local-area networks (VLANs) divide a single physical switch into multiple logical networks.
- 6.** B. CEF switching uses FIB tables in order to make switching very fast.
- 7.** B. NAT is used for mapping private IP addresses to public IP addresses.
- 8.** B. The management plane is concerned with administrative access to a network device.
- 9.** B. vSmart is the main brain of an SD-WAN solution and manages the control plane.

CHAPTER 18

- 1.** A, D. Some of the benefits of using DHCP instead of manual configurations are centralized management of network parameters configuration and reduced network endpoint configuration tasks and costs.
- 2.** C. The DNS recursive resolver is a server that receives DNS queries from client machines and

makes additional requests in order to resolve a client query.

- 3.** A, B, E. The benefits of Network Address Translation include conserving public IPv4 address space, additional security due to hiding the addressing for internal networks, and flexibility and cost savings when changing ISPs.
- 4.** A. With SNMPv3, the focus was on security, and additional features such as authentication, encryption, and message integrity were added to the protocol specification.
- 5.** D. The main role of NTP is to synchronize the time on a network of devices. The IETF developed NTP, and the latest version of the protocol, version 4, is defined in RFC 5905. It uses User Datagram Protocol (UDP) at the transport layer, and port 123 is reserved for NTP.
- 6.** B. Layer 3 network diagrams capture all the network information available at Layer 3. All devices that operate at Layer 3—routers, firewalls, load balancers, and so on—and how they are interconnected should be included in these diagrams.
- 7.** C. If DNS is malfunctioning for some reason, end-to-end connectivity is impacted. Network tools such as **nslookup** can be used to troubleshoot DNS functionality.

Appendix B

DevNet Associate DEVASC 200-901 Official Cert Guide Exam Updates

Over time, reader feedback allows Pearson to gauge which topics give our readers the most problems when taking the exams. To assist readers with those topics, the authors create new materials clarifying and expanding on those troublesome exam topics. As mentioned in the Introduction, the additional content about the exam is contained in a PDF on this book's companion website, at <http://www.ciscopress.com/title/9780136642961>.

This appendix is intended to provide you with updated information if Cisco makes minor modifications to the exam upon which this book is based. When Cisco releases an entirely new exam, the changes are usually too extensive to provide in a simple updated appendix. In those cases, you might need to consult the new edition of the book for the updated content. This appendix attempts to fill the void that occurs with any print book. In particular, this appendix does the following:

- Mentions technical items that might not have been mentioned elsewhere in the book
- Covers new topics if Cisco adds new content to the exam over time
- Provides a way to get up-to-the-minute current information about content for the exam

ALWAYS GET THE LATEST AT THE BOOK'S PRODUCT PAGE

You are reading the version of this appendix that was available when your book was printed. However, given

that the main purpose of this appendix is to be a living, changing document, it is important that you look for the latest version online at the book's companion website. To do so, follow these steps:

Step 1. Browse to

www.ciscopress.com/title/9780136642961.

Step 2. Click the Updates tab.

Step 3. If there is a new Appendix B document on the page, download the latest Appendix B document.

Note

The downloaded document has a version number. Comparing the version of the print Appendix B (Version 1.0) with the latest online version of this appendix, you should do the following:

- **Same version:** Ignore the PDF that you downloaded from the companion website.
- **Website has a later version:** Ignore this Appendix B in your book and read only the latest version that you downloaded from the companion website.

TECHNICAL CONTENT

The current Version 1.0 of this appendix does not contain additional technical coverage.

Glossary

A

advanced persistent threat (APT) A prolonged and targeted cyberattack in which an intruder gains access to a network and remains undetected for a period of time.

Agile A form of software development that prioritizes managing change and adding continuous incremental value.

American Registry for Internet Numbers (ARIN)

The regional organization that manages IP address assignment and allocation for the North American region.

API Application programming interface is a method of communication between various software components or systems or applications.

API key A predetermined string that is passed from a client to a server.

API token A unique auto-generated and encrypted token that is used to access protected pages or resources instead of requiring user login credentials to be passed repeatedly.

Application Centric Infrastructure (ACI) The SDN-based solution from Cisco for data center deployment, management, and monitoring.

Application Policy Infrastructure Controller (APIC) The central controller of the ACI fabric.

artifact repository The location where software builds are stored for deployment. This term can also be used to refer to a registry if your built software is container based.

Asia-Pacific Network Information Center

(APNIC) The regional organization that manages IP address assignment and allocation for the Asia-Pacific region.

asynchronous API An API that does not wait until a response is received but that can continue to function as an application and process other aspects of data.

B

Bidirectional-streams Over Synchronous HTTP

(BOSH) A transport protocol that emulates the semantics of a long-lived, bidirectional TCP connection between two entities.

Bluetooth Low Energy (BLE) A wireless personal-area network technology.

Border Gateway Protocol (BGP) The routing protocol for the Internet.

C

campus-area network (CAN) A network that consists of two or more LANs within a limited area.

Cisco Digital Network Architecture (DNA) An open, extensible, and software-driven architecture for enterprise networks.

Cisco Express Forwarding (CEF) A Cisco-proprietary packet-switching technique used in Cisco routers used to enhance network performance.

Cisco Software Defined-WAN (SD-WAN) A secure, scalable, open, and programmable network architecture for WANs.

classless interdomain routing (CIDR) A way of specifying the subnet mask for an IP address that goes beyond the default class address mask.

computer telephony integration (CTI) Technology that enables computers to interact with telephones.

Configuration Database (CDB) A database that stores all the data in the platform in a RAM database, including the NSO configuration, the configuration of all services and all managed devices, and all NSO operational data.

container A form of operating system kernel virtualization that enables you to deploy application packages that are small and that contain only the binaries and libraries that need to run on a container host.

continuous integration/continuous delivery (CI/CD) pipeline A series of automated steps that code goes through from the IDE of an individual developer, through building, testing, and finally deployment to staging and production environments.

continuous integration/continuous deployment (CI/CD) An environment with an automatic software development and deployment pipeline.

CRUD CREATE, READ, UPDATE, and DELETE resources are methods to implement an application for creating new resources, getting information back, and updating and deleting those resources.

CRUD functions Functions used to interact with or manipulate the data stored in a database. CRUD stands for CREATE, READ, UPDATE, and DELETE.

cryptography The science of transmitting information securely.

D

data as a service (DaaS) A data management solution that uses cloud services to deliver data storage, integration, and processing.

DevOps A combination of development and operations together in a single shared fate operating model for application development, deployment, and support.

distinguished name (DN) A name that describes a managed object and specifies its location in the MIT.

Dlint A static analysis tool for Python.

Docker The largest and most deployed container runtime. It is both an open-source project and a commercial product that you can purchase for support and advanced management capabilities.

Docker daemon The component of the Docker container architecture that controls cgroups and namespaces within the host kernel. It also enables management and container operations.

Docker Hub A free source for hundreds of thousands of prebuilt containers that is maintained by Docker. It includes official releases from software companies as well as community and individual contributions.

Domain Name System (DNS) A directory of networks that maps names of endpoints to IP addresses. A protocol that translates domain names into IP addresses, which computers can understand.

Dynamic Host Configuration Protocol (DHCP) A protocol used to dynamically configure hosts with network connectivity information. DHCP also assigns IP addresses to devices that connect to a network.

E

edge computing An application deployment model that is typically used when you need local computing decision-making capabilities closer to where sensors are deployed. Edge computing reduces latency and can significantly reduce the amount of data that needs to be processed and transmitted.

endpoint group (EPG) A collection of endpoints that have common policy requirements.

Enhanced Interior Gateway Routing Protocol (EIGRP) A routing protocol used in computer networks to automate routing decisions.

Extensible Messaging and Presence Protocol

(XMPP) A message-oriented and real-time communication protocol based on XML.

F

File Transfer Protocol (FTP) A protocol used for transferring files between a client and a server.

Finesse A next-generation agent and supervisor desktop designed for the contact center.

Forwarding Information Base (FIB) table A routing table that is used to switch packets quickly from one interface to another.

frame check sequence (FCS) A footer in a Layer 2 frame that ensures data is transmitted without errors between two devices on a network.

fully qualified domain name (FQDN) The complete and unambiguous domain name that specifies an exact location for an object in a Domain Name System (DNS) hierarchy.

functional test A type of test that validates software against the original design specifications to ensure that the software operates as expected.

G

Git A free distributed version control system created by Linus Torvalds.

GitHub A social network built on Git that allows developers to share and contribute to software development projects.

gRPC An open-source remote procedure call (RPC) framework that was developed by Google.

H

hybrid cloud A cloud application deployment model that stretches components of an application between on-

premises or private cloud resources and public cloud components.

Hypertext Transfer Protocol (HTTP) A protocol used for transferring web pages between web browsers and web servers.

I

integration test A type of test that tests a combination of individual units of software as a group. APIs are often tested with integration tests.

Internet Assigned Numbers Authority (IANA)

The organization that manages the assignment and allocation of IP addresses.

Internet Protocol (IP) An Internet layer protocol in the TCP/IP reference model.

intrusion detection system (IDS) A tool that passively analyzes packets for known signatures.

intrusion prevention system (IPS) A tool that analyzes packets, notifies, and takes action to help stop attacks.

J

JSON JavaScript Object Notation is a lightweight data storage format inspired by JavaScript.

JSON Web Token (JWT) A standardized, validated, and/or encrypted **format** that is used to securely transfer information between two entities.

L

Lean A management style that focuses on continuous improvement and reducing wasted effort.

local-area network (LAN) A single network for an office, an enterprise, or a home.

local workspace The files you are actively working on in a Git directory that have not been tagged for staging.

Logical Link Control (LLC) A layer that is responsible for encapsulating network layer data and frame synchronization.

M

managed object (MO) An object that is part of the MIT.

managed security service provider (MSSP) A service that provides outsourced monitoring and management of security devices and systems.

Management Information Base (MIB) An SNMP data structure that facilitates the exchange of information between the SNMP agent and the NMS.

Management Information Model (MIM) The general model that comprises all the physical and logical components of a Cisco ACI fabric.

management information tree (MIT) A hierarchical representation of the MIM.

Media Access Control (MAC) A layer that provides specifications on how devices gain access to the transmission medium.

metropolitan-area network (MAN) A network that spans multiple locations over a slightly larger distance than a CAN.

Model-View-Controller (MVC) A separation of concerns pattern for creating modern modular web applications.

MQ Telemetry Transport (MQTT) A standard lightweight publish/subscribe network protocol used to transport messages between devices.

multifactor authentication (MFA) A security mechanism that requires an individual to provide two or more credentials in order to authenticate his or her identity.

N

Network Address Translation (NAT) A mechanism of mapping private IP addresses to public IP addresses.

Network Configuration Protocol (NETCONF) A protocol that specifies the means of opening a secure management session with a network device, includes a set of operations to manipulate the configuration data and retrieve the operational state, and describes a mechanism to send out notifications.

Network Management Station (NMS) An application that monitors and controls SNMP managed devices.

Network Services Orchestrator (NSO) A Cisco network services orchestration platform that enables end-to-end service deployment across multivendor physical and virtual infrastructure.

Network Time Protocol (NTP) A protocol used to synchronize time on a network of devices.

next-generation firewall (NGFW) A network security device that provides capabilities beyond those of a traditional stateful firewall.

Nmap A network scanning and host detection tool that is very useful during several steps of penetration testing.

O

Object Identifier (OID) A variable that makes up the MIB data.

Observer The observer pattern uses a subscription model for alerting a list of dependents (called observers) of the state change of an object (called a subject). The subject can be configuration, event status, or any other piece of operational data that needs to be synchronized with other applications or devices.

Open Shortest Path First (OSPF) A routing protocol for Internet Protocol (IP) networks.

Open Systems Interconnection (OSI) An ISO reference model for networks.

Open Web Application Security Project (OWASP)

An online community that produces freely available articles, methodologies, documentation, tools, and technologies in the field of web application security.

organizationally unique identifier (OUI) A field in the MAC address that identifies the manufacturer of the network interface card.

Overlay Management Protocol (OMP) A control protocol used to exchange routing, policy, and management information within Cisco SD-WAN fabrics.

OWASP Top 10 An awareness document for web application security.

P

phishing A type of attack that aims to procure sensitive information via emails or web pages.

Plug and Play (PnP) API A secure and scalable solution for day-zero provisioning.

Port Address Translation (PAT) The process of translating one IP address and a transport layer port (TCP or UDP) into a new IP address/transport layer port pair.

private cloud A cloud application model that gives organizations more control as well as more responsibility in that they operate their own cloud environment for their dedicated use.

protocol data unit (PDU) The data format that is being exchanged at each layer in the context of OSI and TCP/IP reference models.

public cloud A cloud application deployment model in which you pay for compute, storage, and network resources based on how much you use.

Python Automated Test System (pyATS) A Cisco test and automation solution.

Python Enhancement Proposals (PEP) Documents that provide guidance and describe best practices on how Python code should be organized, packaged, released, deprecated, and so on.

R

registry An area where containers are stored in the container infrastructure.

remote-procedure call (RPC) A standard framework for clients to send a request to a server to perform an action and return the results.

repository The complete list of files and source code for an application, also referred to as head in Git.

Representational State Transfer (RST) APIs APIs that use HTTP methods to gather and manipulate data.

REST Representational State Transfer, which describes the general rules for how data and services are represented through the API so that other programs will be able to correctly request and receive the data and services that an API makes available.

RESTCONF An HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG.

Rivest–Shamir–Adleman (RSA) One of the first public-key cryptosystems.

S

Secure Sockets Layer (SSL) An encryption-based Internet security protocol.

Simple Network Management Protocol (SNMP) An application-layer protocol used for monitoring and configuring devices.

Simple Object Access Protocol (SOAP) A protocol that is used to access web services. SOAP can use Simple Mail Transfer Protocol (SMTP) or HTTP as a transport.

software as a service (SaaS) A consumption model that does not require you to manage the installation, updates, or maintenance of the application infrastructure.

software-defined networking (SDN) An approach to networking that enables an administrator to manage, control, and optimize network resources programmatically.

software development kit (SDK) A set of software development tools that developers can use to create software or applications for a certain platform, operating system, computer system, or device.

Software Development Lifecycle (SDLC) A process used in the software industry to design, develop, and test high-quality software.

Software Image Management (SWIM) API A set of processes that have been developed to automate software management on Cisco devices.

software version control A form of source code management that development teams use to manage changes and additions to application source code.

staging The step before committing changed code to the Git repository, also known as indexing.

synchronous API An API that causes an application to wait for a response from the API in order to continue processing data or function normally.

T

test-driven development (TDD) An Agile development method that involves writing a failing testing case first and then writing code to make it pass. TDD is intended to focus the developer on writing only what is needed to reduce complexity.

threat Anything you are trying to protect against.

top-level domain (TLD) name server A DNS server that hosts the last portion of the hostname (such as .com, .org, or .net).

Transmission Control Protocol (TCP) A connection-oriented transport layer protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP) A network reference model based on a layered approach.

Transport Layer Security (TLS) A widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet.

U

Unified Communications Cisco equipment, software, and services that combine multiple enterprise communications channels, such as voice, video, messaging, voicemail, and content sharing.

Unified Computing System (UCS) A Cisco compute product.

unit test A type of test that is meant to ensure that individual pieces of code are operating as they are supposed to. A unit is the smallest testable part of any software.

User Datagram Protocol (UDP) A connectionless transport layer protocol.

V

variable-length subnet masking (VLSM) A way to indicate how many bits in an IP address are dedicated to the network ID and how many bits are dedicated to the host ID.

Virtual Internet Routing Laboratory (VIRL) A powerful Cisco network simulation platform that is meant to be a flexible, all-in-one virtual networking lab.

virtual local-area network (VLAN) A software layer on top of a physical network that is used to create separate broadcast domains.

virtual routing and forwarding (VRF) instance

An isolated routing table, also called a context or a private network.

voice over IP (VoIP) Technology that routes voice and video conversations over the Internet or any other IP-based network.

vulnerability A weakness or gap in protection efforts.

W

Waterfall A linear and sequential process for software development.

webhook A user-defined HTTP callback.

wide-area network (WAN) A network that spans a large geographic area and uses public networks.

Y

YAML YAML Ain't Markup Language is a data serialization and a data storage format that is generally used to store configuration.

Yet Another Next Generation (YANG) A data modeling language that was defined by the IETF in RFC 6020 that is used to model configuration and state data manipulated by NETCONF.

Index

SYMBOLS

****kwargs, [90–91](#)**
***args, [90–91](#)**
== operator, [70](#)
!= operator, [70](#)
> operator, [70](#)
>= operator, [70](#)
< operator, [70](#)
<= operator, [70](#)
""" (triple-quote) method, [66](#)
() characters, Python, [68](#)
| character, piping commands, [34](#)
+ operator, [72](#), [73](#)
character, Python comments, [65](#)

NUMBERS

[300–401](#) ENCOR exam, [8–9](#)

A

AAA policies, Cisco ACI MIT, [220](#)
acceleration (web), reverse proxies, [445](#)
access policies, Cisco ACI MIT, [219](#)
access scopes, Webex Teams API integration, [266](#)
accessing
 API, [147–148](#)
 Dashboard API (Meraki), [179–180](#)
AccountMoid property (Cisco Intersight), [248](#)
ACI (Cisco), [216–217](#), [530](#)

EPG, [221–222](#)
fabric architectures, [217–218](#)
MIM, [219](#)
MIT, [219–220](#)
MO, [219](#)
model-driven frameworks, [218](#)
policy models, [218](#)
tenants, [219](#)

- bridge domains, [221](#)*
- components of, [220–221](#)*
- external bridged networks, [221](#)*
- external routed networks, [221](#)*
- subnets, [221](#)*
- VRF instances, [221](#)*

UCS Director, [240](#)

ACI fabric, [530](#)

acitoolkit, [227–229](#)

action batches, Dashboard API (Meraki), [181](#)

adding

- branches with Git, [48–49](#)
- files to repositories, [43–45](#)
- members to rooms, [269](#)

address masks, [498](#)

addresses

- broadcast addresses, [493, 498](#)
- IPv4 addresses, [496](#)
 - address masks, [498](#)*
 - broadcast addresses, [498](#)*
 - CIDR, [499–500](#)*
 - classes of, [497–498](#)*
 - decimal/binary values, [496](#)*
 - host ID, [497](#)*
 - network ID, [497](#)*
 - private IPv4 addresses, [498–499](#)*
 - subnets, [499–500](#)*
- IPv6 addresses, [501–503](#)

- anycast addresses, [503–504](#)*
- global addresses, [503](#)*
- IANA, [502](#)*
- link-local addresses, [503](#)*
- loopback addresses, [503](#)*
- multicast addresses, [503](#)*
- subnets, [502](#)*
- unspecified addresses, [503](#)*
- MAC addresses, [493–494](#)
- multicast addresses, [493](#)
- OUI, [493](#)
- unicast addresses, [493](#)
- vendor-assigned addresses, [493](#)

Agile model (SDLC), [29–30](#)

AMP (Cisco)

- Cisco AMP for Endpoints, [320–321](#)
 - API, creating credentials, [322–323](#)*
 - API, list of, [325–326](#)*
 - API, listing computers, [323](#)*
 - API, listing vulnerabilities, [323–325](#)*
 - automation, [321–322](#)*
 - uses of, [321–322](#)*
- Private Cloud, [320–321](#)

analyzing requirements, SDLC, [25](#)

Ancestors property (Cisco Intersight), [248](#)

Ansible, [458–462](#)

anycast addresses, [503–504](#)

API. *See also* [HTTP](#)

- accessing, [147–148](#)
- API keys, [134–135](#)
- APIC REST API, [223](#)
 - acitoolkit, [227–229](#)*
 - authentication, [223–225](#)*
 - get requests, [225–227](#)*
- asynchronous API, [132](#)
- authentication

- basic authentication*, [134](#)
 - RESTful API*, [133–134](#)
- AXL API, [294–295](#)
 - AXL SOAP API*, [296–297](#)
 - toolkit*, [295–296](#)
 - Zeep Client Library*, [296–297](#)
- basic operations, [131](#)
- Cisco AMP for Endpoints
 - creating credentials*, [322–323](#)
 - list of API*, [325–326](#)
 - listing computers*, [323](#)
 - listing vulnerabilities*, [323–325](#)
- Cisco collaboration portfolio, overview of, [261](#)
- Cisco DNA Center
 - Integration API*, [190](#)
 - Intent API*, [190, 191–192](#)
- Cisco Firepower, [315, 316–317](#)
 - creating networks*, [318–320](#)
- Cisco ISE, [327](#)
 - ERS API*, [327–331](#)
 - Session API*, [327](#)
- Cisco SD-WAN, [203](#)
- Cisco Umbrella
 - Console Reporting API*, [306](#)
 - Enforcement API*, [306, 308–310](#)
 - Investigate API*, [306, 311–313](#)
 - Management API*, [305, 307–308](#)
 - Network Device Management API*, [306](#)
 - Reporting API*, [305](#)
- Console Reporting API, [306](#)
- CreateMeeting API, [284–285](#)
- Data API, [334](#)
- defined, [146](#)
- DelMeeting API, [288–289](#)
- Dialog API, Unified CC (Finesse) API, [279–280](#)
- Enforcement API, [306, 308–310](#)

- ERS API, [327–331](#)
- Finesse Team API, Unified CC (Finesse) API, [279](#)
- Finesse User API, Unified CC (Finesse) API, [277–279](#)
- information API, [147](#)
- Investigate API, [306](#), [311–313](#)
- IOC API, [334](#)
- JSON, [113–115](#), [156](#)
 - Cisco AMP for Endpoints, listing vulnerabilities,* [324–325](#)
 - data format,* [157](#)
 - json modules,* [102](#)
 - JWT,* [272–273](#), [283](#)
 - keys,* [156](#)
 - RESTCONF,* [368](#)
- LstsummaryMeeting API, [286](#)
- Management API, [305](#), [307–308](#)
- Meetings XML API, [289](#)
- Memberships API, Webex Teams API integration, [269](#)
- Meraki
 - Captive Portal API,* [178](#)
 - Dashboard API,* [179–187](#)
 - MV Sense Camera API,* [179](#)
 - Scanning API,* [178–179](#)
- Messages API, Webex Teams API integration, [270](#)
- Network Device Management API, [306](#)
- northbound API, [130](#)
- Organizations API, Webex Teams API integration, [266](#)
- partner API, [147](#)
- PnP API, [192](#)
- private API, [147](#)
- public API, [148](#)
- Reporting API, [305](#)
- REST API
 - Cisco Intersight,* [247](#), [249](#)
 - curl,* [168–169](#)
 - debugging tools,* [172](#)

- development tools, [172](#)*
- HTTPIe, [169–170](#)*
- monetization, [163–164](#)*
- pagination, [162–163](#)*
- Postman, [164–168](#)*
- Python Requests, [171–172](#)*
- rate limiting, [163–164](#)*
- UCS Director, [242–245](#)*
- versioning, [162](#)*
- RESTCONF API, [368](#)
- RESTful API, [133](#)
 - authentication, [133–134](#)*
 - CRUD functions, [132–133](#)*
 - HTTP functions, [132–133](#)*
- reverse API. *See [webhooks](#)*
- Rooms API
 - creating, [267–268](#)*
 - listing rooms, [268](#)*
 - Webex Teams API integration, [267–268](#)*
- RPC, [140](#)
 - high-level communication, [140](#)*
 - XML-RPC reply messages, [141](#)*
 - XML-RPC request messages, [140–141](#)*
- Sample API, [334](#)
- service API, [146–147](#)
- Session API, [327](#)
- SetMeeting API, [287–288](#)
- SOAP, [136](#)
 - fault codes, [138–139](#)*
 - fault options, [138–139](#)*
 - high-level communication, [137–138](#)*
 - message format, [136–137](#)*
 - sample faults, [139–140](#)*
 - sample message, [138](#)*
- southbound API, [130](#)
- SWIM, [191](#)

- synchronous API, [131](#)
- Teams API
 - creating*, [266–267](#)
 - Webex Teams API integration*, [266–267](#)
- Threat Grid API, [332](#)
 - API keys*, [333](#)
 - Data API*, [334](#)
 - format of*, [332](#)
 - IOC API*, [334](#)
 - Sample API*, [334](#)
 - submissions searches*, [334–335](#)
 - “Who am I” queries*, [333–334](#)
- TSP API, Webex Meetings API, [281](#)
- UCS Manager, [231](#)
- Unified CC (Finesse), [275–276](#)
 - authentication*, [276–280](#)
 - Dialog API*, [279–280](#)
 - Finesse Team API*, [279](#)
 - Finesse User API*, [277–279](#)
 - gadgets*, [281](#)
- URL API, Webex Meetings API, [281](#)
- Webex Meetings API
 - architecture of*, [282](#), [284](#)
 - authentication*, [283](#)
 - CreateMeeting API*, [284–285](#)
 - DelMeeting API*, [288–289](#)
 - integrations*, [283–284](#)
 - LstsummaryMeeting API*, [286](#)
 - Meetings XML API*, [289](#)
 - overview of*, [281–282](#)
 - SetMeeting API*, [287–288](#)
 - supported services*, [282–283](#)
 - TSP API*, [281](#)
 - URL API*, [281](#)
 - XML API*, [281](#)
- Webex Teams, [261–262](#)

- access scopes, [265–266](#)*
- authentication, [262–273](#)*
- bots, [271–272](#)*
- guest issuers, [272–273](#)*
- integrations, [263–270](#)*
- Memberships API, [269](#)*
- Messages API, [270](#)*
- Organizations API, [266](#)*
- personal access tokens, [262–263](#)*
- Rooms API, [267–268](#)*
- Teams API, [266–267](#)*
- xAPI
 - authentication, [290–293](#)*
 - categories, [290](#)*
 - overview of, [290](#)*
- XML API
 - data filters, [232](#)*
 - UCS Manager, [231](#), [232](#), [234–237](#)*
 - Webex Meetings API, [281](#)*
- YAML, [117–119](#), [157–158](#), [461](#)
- APIC, [530](#)**
 - Cisco ACI, [216–217](#)
 - Cisco ACI MIT, [219](#)
- APIC REST API, [223](#)**
 - acitoolkit, [227–229](#)
 - authentication, [223–225](#)
 - get requests, [225–227](#)
- APNIC, [502](#)**
- application layer**
 - OSI network model, [488](#)
 - TCP/IP network model, [490](#)
- Application Network Profile, [530](#)**
- application profiles, EPG, [222](#), [223](#)**
- application-level (proxy) firewalls, [438–439](#)**
- applications**
 - connectivity issues, troubleshooting, [548–550](#)

- deploying, [382](#)
 - bare-metal deployments*, [382–383](#)
 - cloud computing*, [379–381](#), [384](#)
 - cloud-native deployments*, [384](#)
 - containerized applications*, [384–386](#)
 - DevOps*, [388–390](#), [391](#)
 - DevOps, calendars*, [388–389](#)
 - DevOps, continuous experimentation and learning*, [393–394](#)
 - DevOps, defined*, [390–391](#)
 - DevOps, feedback loops*, [392–393](#)
 - DevOps, implementing*, [394–397](#)
 - DevOps, pipelines*, [394–397](#)
 - DevOps, systems and flow*, [391–392](#)
 - DevOps, XebiaLabs periodic table of DevOps tools*, [394–395](#)
 - Docker*, [398](#)
 - Docker, architecture*, [400–401](#)
 - Docker, Cgroups*, [399](#)
 - Docker, command line*, [401–403](#)
 - Docker, containers*, [402–410](#)
 - Docker, daemon*, [400](#), [401](#), [404](#), [412](#)
 - Docker, Dockerfiles*, [410–411](#)
 - Docker, Hello World*, [404–405](#)
 - Docker, Hub*, [414–418](#)
 - Docker, images*, [411–414](#)
 - Docker, Kitematic*, [415–416](#)
 - Docker, namespaces*, [398–399](#)
 - Docker, registry*, [401](#), [405](#), [414–415](#)
 - Docker, Union File System*, [399–400](#)
 - Docker, using*, [401–403](#)
 - edge computing*, [381–382](#)
 - fog computing*, [381–382](#)
 - serverless applications*, [386–388](#)
 - virtualized applications*, [383–384](#)
- security

best practices, 431
common threats, 423–424
CVE records, 425–426, 427–429
Cybersecurity Framework, 422
data integrity (one-way hash), 432
data security, 433–434
device security, 431
digital signatures, 432–433
DNS, 440–443
encryption, 431–433
end-user security, 431
firewalls, 431, 437–439
IDS, 439–440
IPS, 440–441
load balancing, 443–446
MFA, 431
mitigating common threats, 423–424
Nmap, 426–429
Nslookup, 442–443
one-way hash (data integrity), 432
OWASP, 424–426
passwords (strong), 431
penetration (pen) testing, 424–425
reverse proxies, 444–446
SDLC, 434–436
software development, 434–436
three-tier application security, 430
updating software, 431

APT, 320–321
area_of_circle function, 123–124, 125–126
***args, 90–91**
arguments, Python, 89–91
ARIN, 502
artifact repositories, 397
assistant bots, 271
asynchronous API, 132

Atom text editor, [64–65](#)

augmenting, YANG data models, [355–356](#)

authentication

API

basic authentication, [134](#)

RESTful API, [133–134](#)

APIC REST API, [223–225](#)

Cisco Firepower, [315–316](#)

Cisco Intersight, [249](#)

Cisco Umbrella, [306](#)

custom tokens, [135–136](#)

ERS API, [328–329](#)

MFA, [431](#)

RESTful API, [133–134](#)

reverse proxies, [445](#)

UCS Manager, [234](#)

Unified CC (Finesse) API, [276–280](#)

Dialog API, [279–280](#)

Finesse Team API, [279](#)

Finesse User API, [277–279](#)

Webex Meetings API, [283](#)

Webex Teams API

access scopes, [265–266](#)

bots, [271–272](#)

guest issuers, [272–273](#)

integrations, [263–270](#)

Memberships API, [269](#)

Messages API, [270](#)

Organizations API, [266](#)

overview of, [262](#)

personal access tokens, [262–263](#)

Rooms API, [267–268](#)

Teams API, [266–267](#)

xAPI, [290–291](#)

creating sessions, [291](#)

current device status, [291–292](#)

event notification webhooks, 293
session authentication, 291–293
setting device attributes, 292

authorization

Cisco DNA Center, [193–194](#)
Cisco SD-WAN, [204–205](#)

automation

Cisco AMP for Endpoints, [321–322](#)
DevNet Automation Exchange, [18–20](#)
infrastructure automation, [458](#)
Ansible, 458–462
Chef, 465–466
CI/CD pipelines, 455–457
Cisco NSO, 467–473
Cisco VIRL, 457, 474–476
device-level management, 453
infrastructure as a code, 454–455
network controllers, 451–452, 453
Puppet, 462–464
pyATS, 476–479
integration automation frameworks, Cisco ACI MIT,
[220](#)
lifecycle of, [19–20](#)
napalm, [103](#)
nornir, [103–104](#)
Open Automation, UCS Director, [241](#)
UCS Director, [239–240](#)
Cisco ACI, 240
Open Automation, 241
PowerShell, 242
REST API, 242–245
retrieving user profiles, 244–245
script modules, 242
SDK, 241
tasks, 240, 242
templates, 240

workflows, 240, 241, 245

automobile as a system analogy (certification), 5

AXL API, 294–295

- AXL SOAP API, 296–297
- toolkit, 295–296
- Zeep Client Library, 296–297

AXL SDK, 297–298

B

bare-metal application deployments, 382–383

base 2 (binary) systems, Python, 69

base 8 (octal) systems, Python, 69

base 16 (hex) systems, Python, 69

base URL, Dashboard API (Meraki), 180, 181–182

Base64 encoding

- Cisco Umbrella, 306
- ERS API, 328
- Unified CC (Finesse) API, 277

BASH, 32–33

- cat command, 34, 37
- cd command, 35
- cp command, 36
- directories
 - changing, 36*
 - creating, 36*
 - listing, 36*
 - navigating, 35–36*
 - printing paths, 35*
- environment variables, 37–38
- file management, 36–37
- ls command, 36
- man command, 33
- mkdir command, 36
- more command, 34
- mv command, 37

- piping commands, [34](#)
- pwd command, [35](#)
- rm command, [37](#)
- touch command, [37](#)

BGP, [201](#)

binary (base 2) systems, Python, [69](#)

binary/decimal values, IPv4 addresses, [496](#)

BLE, [178–179](#)

Bluetooth, BLE, [178–179](#)

boolean comparisons, [70](#)

BOSH, [275](#)

bots

- assistant bots, [271](#)
- Botkit, [271](#)
- building, [271–272](#)
- controller bots, [271](#)
- Flint, [271](#)
- frameworks, [271](#)
- notification bots, [271](#)
- Webex Teams API, [271–272](#)

branches (software development), [47–48](#)

- adding, [48–49](#)
- conflicts, [52–53](#)
- merging, [50–53](#)

break statements, [82–83](#)

bridge domains, Cisco ACI tenants, [221](#)

bridged networks (external), Cisco ACI tenants, [221](#)

bridges, [517–518](#)

broadcast addresses, [493, 498](#)

brute-force attacks, [304, 424](#)

budgeting time, exam preparation, [552](#)

buffer overflow attacks, [423](#)

building stage (SDLC), [26](#)

bus networks, [512](#)

C

cache constraints (REST), [161](#)

caching

DNS resolution data, [539](#)

reverse proxies, [445](#)

calendar modules, [99](#)

calendars, DevOps, [388–389](#)

Call Manager. *See* [Unified CM](#)

camera positions, setting (xAPI), [292](#)

CAN, [513–514](#)

Captive Portal API, [178](#)

career certification (Cisco)

CCIE, [11](#)

CCNP

300–401 ENCOR exam, [8–9](#)

components of, [8–9](#)

concentration exams, [9–11](#)

core exams, [9–11](#)

levels of, [6–7](#)

overview of, [6–11, 14](#)

specialist certifications, [11](#)

tracks after restructuring, [8](#)

tracks prior to restructuring, [7](#)

cat command, [34, 37](#)

categorizing domains, Cisco Umbrella, [312–313](#)

CCIE certifications, [11](#)

CCNP

300–401 ENCOR exam, [8–9](#)

components of, [8–9](#)

concentration exams, [9–11](#)

core exams, [9–11](#)

cd command, [35](#)

CDB, Cisco NSO, [468–469](#)

CEF, [201, 523–524](#)

certification

300–401 ENCOR exam, [8–9](#)

automobile as a system analogy, [5](#)

CCIE, [11](#)

Cisco career certification

300-401 ENCOR exam, [8-9](#)

CCIE, [11](#)

CCNP, [8-11](#)

levels of, [6-7](#)

overview of, [6-11, 14](#)

specialist certifications, [11](#)

tracks after restructuring, [8](#)

tracks prior to restructuring, [7](#)

DevNet certifications, [11](#)

DEVASC, [12-13](#)

DevNet Professional, [13-14](#)

overview of, [14-15](#)

importance of, [3-6](#)

knowledge domains

DEVASC, [12](#)

DevNet Professional, [13](#)

reasons for, [3-6](#)

specialist certifications, [11](#)

DevNet Professional, [13-14](#)

Cgroups, [399](#)

chaddr field (DHCPOFFER messages), [537](#)

changing

directories, [35](#)

interface names in dictionaries, [118-119](#)

chapter-ending review tools, [556](#)

chatbots. *See* [bots](#)

Chef, [465-466](#)

Chef Supermarket, [466](#)

CI/CD pipelines, [455-457](#)

CIDR, subnets, [499-500](#)

Cisco ACI, [216-217, 530](#)

EPG, [221-222](#)

fabric architectures, [217-218](#)

- MIM, [219](#)
- MIT, [219–220](#)
- MO, [219](#)
- model-driven frameworks, [218](#)
- policy models, [218](#)
- tenants, [219](#)
 - bridge domains*, [221](#)
 - components of*, [220–221](#)
 - external bridged networks*, [221](#)
 - external routed networks*, [221](#)
 - subnets*, [221](#)
 - VRF instances*, [221](#)

- UCS Director, [240](#)

Cisco AMP

- Cisco AMP for Endpoints, [320–321](#)
 - API, creating credentials*, [322–323](#)
 - API, list of*, [325–326](#)
 - API, listing computers*, [323](#)
 - API, listing vulnerabilities*, [323–325](#)
 - automation*, [321–322](#)
 - uses of*, [321–322](#)

- Private Cloud, [320–321](#)

Cisco APIC, 530

- Cisco ACI, [216–217](#)
- Cisco ACI MIT, [219](#)

Cisco career certification

- CCIE, [11](#)
- CCNP
 - 300–401 ENCOR exam*, [8–9](#)
 - components of*, [8–9](#)
 - concentration exams*, [9–11](#)
 - core exams*, [9–11](#)
- levels of, [6–7](#)
- overview of, [6–11](#), [14](#)
- specialist certifications, [11](#)
- tracks

after restructuring, [8](#)
prior to restructuring, [7](#)

Cisco collaboration portfolio

API

overview of, [261](#)

Webex Teams, [261–273](#)

Cisco Webex, [260](#)

API, [261–273](#)

SDK, [273–274](#)

collaboration endpoints, [260–261](#)

overview of, [257](#)

Unified CC (Finesse), [259–260](#)

API, [275–280](#)

authentication, [276–280](#)

high-level flow, [274–275](#)

user states, [275](#)

Unified CM, [259](#)

AXL API, [294–297](#)

AXL SDK, [297–298](#)

overview of, [294](#)

Cisco DevNet certifications, [11](#)

DEVASC, [12–13](#)

DevNet Professional, [13–14](#)

overview of, [14–15](#)

Cisco DNA, [530](#)

Cisco DNA Center, [189–190](#)

authorization, [193–194](#)

client data, [198–199](#)

Integration API, [190](#)

Intent API, [190](#), [191–192](#)

network ID, [195–197](#)

non-Cisco device management, [191](#)

platform interface, [193](#)

Python SDK, [199–201](#)

rate limiting, [192](#)

versions of, [193](#)

webhooks, [191](#)

Cisco Firepower, [314](#)

API, [315](#), [316–317](#)

authentication, [315–316](#)

components of, [314–315](#)

features of, [314](#)

Management Center objects, [317–318](#)

networks, creating, [318–320](#)

server version information, [316](#)

session tokens, generating, [315–316](#), [318–320](#)

system information, [316–317](#)

Cisco Headset 500 Series, [261](#)

Cisco infrastructure, Python modules, [101–104](#)

Cisco Intersight, [246–247](#)

AccountMoid property, [248](#)

Ancestors property, [248](#)

authentication, [249](#)

CreateTime property, [248](#)

documentation, [249–250](#)

HATEOAS, [249](#)

managed object properties, [247–248](#)

MIM, [247](#)

ModTime property, [248](#)

Moid identifier, [248](#)

ObjectType property, [248](#)

Owners property, [248](#)

Parent property, [248](#)

Python, [249–251](#)

query language, [249](#)

REST API, [247](#), [249](#)

tagging, [248](#)

Tags property, [248](#)

URI, [248](#)

Cisco IOS XE, NETCONF sessions, [360–362](#)

Cisco ISE, [326](#)

API, [327](#)

- ERS API, 327–331*
 - Session API, 327*
- components of, 326–327
- deployments, 326–327
- profiles, 326
- Cisco NSO, 467, 470–471, 530**
 - architecture of, 467, 468
 - CDB, 468–469
 - CLI, 471
 - components of, 467
 - model-to-model mapping, 468
 - ncs-netsim, 469–470
 - NETCONF, 468
 - RESTCONF, 471–473
 - YANG, 468, 469
- Cisco SDN, 530–531**
- Cisco SD-WAN, 201–202, 203, 530–531**
 - API, 203
 - authorization, 204–205
 - device templates, 207–208
 - listing devices, 205–207
 - OpenAPI interface, 203
 - Python scripting, 209–212
 - vBond, 202
 - vEdge, 202
 - vManage, 202, 203–212
 - vSmart, 202
- Cisco security portfolio, 302–303**
- Cisco Threat Grid, 331**
 - API, 332
 - API keys, 333*
 - Data API, 334*
 - format of, 332*
 - IOC API, 334*
 - Sample API, 334*
 - submissions searches, 334–335*

- “Who am I” queries, 333–334*
 - architecture of, [331–332](#)
 - feeds, [335–337](#)
- Cisco UCS, fabric interconnects, [230](#). See also [UCS Manager](#)**
- Cisco UCS Python SDK, [237–239](#)**
- Cisco Umbrella, [304](#)**
 - API, [305–306](#)
 - authentication, [306](#)
 - Console Reporting API, [306](#)
 - domain lookups, [305](#)
 - domains
 - adding domains, [308–309](#)*
 - categorizing domains, [312–313](#)*
 - deleting domains, [310](#)*
 - listing domains, [309–310](#)*
 - Enforcement API, [306, 308–310](#)
 - flow example, [305](#)
 - Investigate API, [306, 311–313](#)
 - Management API, [305, 307–308](#)
 - Network Device Management API, [306](#)
 - Reporting API, [305](#)
- Cisco VIRT, [457, 474–476](#)**
- Cisco Webex, [260](#)**
- Cisco Webex Board, [260](#)**
- classes, Python, [92](#)**
 - creating, [92–93](#)
 - inheritance, [94–96](#)
 - router classes, [93–95](#)
- CLI, [342](#)**
 - Cisco NSO, [471](#)
 - NETCONF and, [345–346](#)
 - UCS Manager, [231](#)
- client data, Cisco DNA Center, [198–199](#)**
- client/server constraints (REST), [160](#)**
- clock (exam preparation), watching the, [552](#)**

cloning/initiating repositories, [42–43](#)

close() function, [109–110](#)

cloud computing, [376](#)

application deployments, [379–381](#)

characteristics of, [377](#)

cloud-native application deployments, [384](#)

community clouds, [381](#)

edge computing, [381–382](#)

fog computing, [381–382](#)

hybrid clouds, [380](#)

IaaS, [378](#)

Meraki, [178](#)

NIST definition, [376–377](#)

PaaS, [378](#)

Private Cloud (Cisco AMP), [320–321](#)

private clouds, [379](#)

public clouds, [379–380](#)

SaaS, [378](#)

service models, [378](#)

code

editors, [55](#), [64–65](#)

malicious code, [430](#)

reviews, [55](#)

code blocks

Atom text editor, [65–66](#)

Python and, [65–66](#)

code on demand constraints (REST), [162](#)

collaboration

IP phone, [260](#)

spaces, [261](#)

video devices, [260](#)

collaboration portfolio

API

overview of, [261](#)

Webex Teams, [261–273](#)

Cisco Webex, [260](#)

- API, [261–273](#)*
 - SDK, [273–274](#)*
- collaboration endpoints, [260–261](#)
- overview of, [257](#)
- Unified CC (Finesse), [259–260](#)
 - API, [275–280](#)*
 - authentication, [276–280](#)*
 - high-level flow, [274–275](#)*
 - user states, [275](#)*
- Unified CM, [259](#)
 - AXL API, [294–297](#)*
 - AXL SDK, [297–298](#)*
 - overview of, [294](#)*
- command line**
 - Docker, [401–403](#)
 - imported dictionaries, printing to command line, [116](#)
 - printing from, unparse() function, [117](#)
- commands, piping, [34](#)**
- comments, Python, [65–66](#)**
- committing files, [45](#), [53–54](#)**
- common tenants, Cisco ACI MIT, [219](#)**
- communications, Unified Communications, [257–258](#)**
 - Webex Calling, [258–259](#)
 - Webex Teams, [258](#)
 - API, [261–273](#)*
 - SDK, [273–274](#)*
- community clouds, [381](#)**
- community names, SNMP, [545](#)**
- Community page (DevNet), [16](#)**
- composite filters, XML API, [232](#)**
- compression, reverse proxies, [445](#)**
- compromise, IOC API, [334](#)**
- compute object lists, UCS Manager, [234–237](#)**
- concatenating**
 - lists, [73](#)

- strings, [72](#)
- concentration exams**
 - CCNP, [300–401](#) ENCOR exam, [8–9](#)
 - DevNet Professional, [13–14](#)
- configuration data stores, NETCONF, [346–347](#)**
- conflicts (branches), merging, [52–53](#)**
- CONNECT requests, [150](#)**
- connected routes, [506](#)**
- connectivity**
 - applications, troubleshooting connectivity issues, [548–550](#)
 - SingleConnect technology, UCS Manager, [230](#)
 - UCS Manager, [230](#)
- Console Reporting API, [306](#)**
- container nodes (YANG), [350–351](#)**
- containerized applications, [384–386](#)**
- containers, Docker, [402–410](#)**
- content filtering, reverse proxies, [445](#)**
- content negotiation, versioning, [162](#)**
- contexts. *See* [VRF instances](#)**
- contracts, EPG, [222–223](#)**
- control plane (networks), [527–528](#)**
- controller bots, [271](#)**
- controllers**
 - OpenDayLight controllers, [529](#)
 - OpenFlow controllers, [529](#)
 - SDN controllers, [529–530](#)
 - vSmart, [531](#)
- controllers (network), [451–452](#), [453](#)**
- cookbooks (Chef), [465–466](#)**
- copying, files/folders, [36](#)**
- core exams, CCNP, [9–11](#)**
- cp command, [36](#)**
- CreateMeeting API, Webex Meetings API, [284–285](#)**
- CreateTime property (Cisco Intersight), [248](#)**

Cross-Site Scripting (XSS), [304](#), [424](#)

CRUD

functions, RESTful API, [132–133](#)

HTTP, CRUD operations, [150–151](#)

CSV files, [110–113](#)

csv modules, [102](#)

CTI connections, [275](#), [277](#)

curated feeds, Threat Grid, [336–337](#)

curl, [168–169](#)

APIC REST API

authentication, [223–225](#)

get requests, [225–227](#)

Cisco DNA Center

authorization, [193–194](#)

network ID, [195–197](#)

Cisco SD-WAN

authorization, [205](#)

listing devices, [207](#)

Dashboard API (Meraki)

network ID, [183–184](#)

organization ID, [181–183](#)

Rooms API, listing rooms, [268](#)

UCS Director

retrieving user profiles, [244–245](#)

workflow inputs, [245](#)

current device status, getting in xAPI, [291–292](#)

custom headers, versioning, [162](#)

custom tasks, UCS Director, [242](#)

custom tokens, authentication, [135–136](#)

**customizing practice tests, Pearson Test Prep
software, [554–555](#)**

CVE records, [425–426](#), [427–429](#)

Cybersecurity Framework, [422](#)

D

daemon (Docker), [400](#), [401](#), [404](#), [412](#)

Dashboard API (Meraki)

- action batches, [181](#)
- base URL, [180](#), [181–182](#)
- hierarchy of, [180](#)
- network ID, [180](#)
 - curl*, [183–184](#)
 - Postman*, [184–187](#)
- organization ID, [180](#), [181](#)
 - curl*, [181–183](#)
 - Postman*, [183](#)
- pagination, [181](#)
- rate limiting, [180](#)
- requests, [180](#)
- use cases, [179](#)
 - accessing*, [179–180](#)

Data API, [334](#)

data encryption, [431](#)

- objectives of, [431](#)
- public key encryption, [431–432](#)
- TLS handshakes, [432](#)

data filters, XML API, [232](#)

data frame switching, [492](#)

- Ethernet switching, [492](#)
- frames, [492](#)

data integrity (one-way hash), [432](#)

data link layer (OSI network model), [486](#)

data packet routing, [489](#), [504–506](#)

data, parsing in Python, [110](#)

- CSV files, [110–113](#)
- JSON, [113–115](#)
- XML, [115–117](#)
- YAML, [117–119](#)

data plane (networks), [527](#)

data routing, TCP/IP network model, [489](#)

data security, [433–434](#)

data types

commonly used data types, [67](#)

dictionaries, [75–76](#)

floating point numbers, [68](#), [69](#)

integers, [68](#), [69](#)

lists, [72](#), [73–74](#)

concatenating, [73](#)

list indexes, [73](#)

methods, [74](#)

sets, [76](#)

strings, [70](#), [71–72](#)

concatenating, [72](#)

DevNet string index, [70–71](#)

string methods, [72](#)

tuples, [74–75](#)

DataEncodingUnknown fault code, [139](#)

datetime modules, [101](#)

deactivate command, [62](#)

debugging tools, REST API, [172](#)

decimal/binary values, IPv4 addresses, [496](#)

de-encapsulation, [491](#)

defining stage (SDLC), [26](#)

DELETE requests, [150](#), [151](#), [161](#)

deleting

directories, [37](#)

domains, Cisco Umbrella, [310](#)

files/folders, [37](#)

meetings, [288–289](#)

DelMeeting API, Webex Meetings API, [288–289](#)

deploying applications, [382](#)

bare-metal deployments, [382–383](#)

cloud computing, [379–381](#)

cloud-native deployments, [384](#)

community clouds, [381](#)

edge computing, [381–382](#)

fog computing, [381–382](#)

hybrid clouds, [380](#)

- private clouds, 379*
 - public clouds, 379–380*
- containerized applications, 384–386
- DevOps, 388–390, 391
 - calendars, 388–389*
 - continuous experimentation and learning, 393–394*
 - defined, 390–391*
 - feedback loops, 392–393*
 - implementing, 394–397*
 - pipelines, 394–397*
 - systems and flow, 391–392*
 - XebiaLabs periodic table of DevOps tools, 394–395*
- Docker, 398
 - architecture of, 400–401*
 - Cgroups, 399*
 - command line, 401–403*
 - containers, 402–410*
 - daemon, 400, 401, 404, 412*
 - Dockerfiles, 410–411*
 - Hello World, 404–405*
 - Hub, 414–418*
 - images, 411–414*
 - Kitematic, 415–416*
 - namespaces, 398–399*
 - registry, 401, 405, 414–415*
 - Union File System, 399–400*
 - using, 401–403*
- serverless applications, 386–388
- virtualized applications, 383–384
- deployment stage (SDLC), 26**
- design patterns (software), 30**
 - MVC, 30–31
 - Observer, 31–32
- designing stage (SDLC), 26**

DEVASC (DevNetv Associate Certification), [12](#)–[13](#)

development

security, [434](#)–[436](#)
software. *See* [SDLC](#)

TDD, [121](#)–[122](#)

development tools, REST API, [172](#)

device templates, Cisco SD-WAN, [207](#)–[208](#)

devices

device-level management (automation), [453](#)
security, [431](#)

devkits. *See* [SDK](#)

DevNet

certifications, [11](#)
DEVASC, [12](#)–[13](#)
DevNet Professional, [13](#)–[14](#)
overview of, [14](#)–[15](#)

Community page, [16](#)

Discover page, [15](#)

Events page, [17](#)–[18](#)

Main page, [14](#)–[15](#), [17](#)

string index, [70](#)–[71](#)

Technologies page, [15](#)–[16](#)

DevNet Automation Exchange, [18](#)–[20](#)

DevNet Express, [18](#)

DevOps, [388](#)–[390](#), [391](#)

calendars, [388](#)–[389](#)

continuous experimentation and learning, [393](#)–[394](#)

defined, [390](#)–[391](#)

feedback loops, [392](#)–[393](#)

implementing, [394](#)–[397](#)

pipelines, [394](#)–[397](#)

systems and flow, [391](#)–[392](#)

XebiaLabs periodic table of DevOps tools, [394](#)–[395](#)

DHCP, [490](#), [534](#)–[535](#)

benefits of, [535](#)

- DHCPACK messages, [537](#)
- DHCPDISCOVER messages, [536–537](#)
- DHCPOFFER messages, [537](#)
- DHCPRELEASE messages, [537](#)
- DHCPREQUEST messages, [537](#)
- IPv4, [535](#)
- lease acknowledgement phase, [537](#)
- lease offer phase, [537](#)
- lease request phase, [537](#)
- phases of operation, [536](#)
- relay agents, [535–536](#)
- releasing phase, [537](#)
- server allocation, [535](#)
- server discovery phase, [536–537](#)
- state machine, [536](#)

diagrams (network), [526](#)

Dialog API, Unified CC (Finesse) API, [279–280](#)

dictionaries, [75–76](#)

- imported dictionaries, printing to command line, [116](#)
- interface names, changing in, [118–119](#)

diff command, [53](#)

digital signatures, [432–433](#)

dir() function, [97](#)

directories. *See also* [files/folders](#)

- changing, [35](#)
- creating, [36](#)
- deleting, [37](#)
- listing, [36](#)
- moving files/folders, [37](#)
- navigating, [35–36](#)
- printing paths, [35](#)
- structure of, [35](#)

Discover page (DevNet), [15](#)

DITKA questions, [556](#)

Dlint, [436](#)

DME, UCS Manager, [231](#)

DN, UCS Manager, [232](#)

DNA (Cisco), [530](#)

DNA Center (Cisco), [189–190](#)

- [authorization, 193–194](#)
- [client data, 198–199](#)
- [Integration API, 190](#)
- [Intent API, 190, 191–192](#)
- [network ID, 195–197](#)
- [non-Cisco device management, 191](#)
- [platform interface, 193](#)
- [Python SDK, 199–201](#)
- [rate limiting, 192](#)
- [versions of, 193](#)
- [webhooks, 191](#)

DNS, [440–443, 540](#)

- [caching resolution data, 539](#)
- [lookups, 538–539](#)
- [name resolution, 539](#)
- [resolution process, 538](#)
- [resolution queries, 540](#)

Docker, [398](#)

- [architecture of, 400–401](#)
- [Cgroups, 399](#)
- [command line, 401–403](#)
- [containers, 402–410](#)
- [daemon, 400, 401, 404, 412](#)
- [Dockerfiles, 410–411](#)
- [Hello World, 404–405](#)
- [Hub, 414–418](#)
- [images, 411–414](#)
- [Kitematic, 415–416](#)
- [namespaces, 398–399](#)
- [registry, 401, 405, 414–415](#)
- [Union File System, 399–400](#)
- [using, 401–403](#)

documentation

- Cisco Intersight, [249–250](#)
- UCS Manager, [233](#)
- domain lookups, malicious, [305](#)**
- domain name proxies, [526](#)**
- domains, Cisco Umbrella**
 - adding domains, [308–309](#)
 - categorizing domains, [312–313](#)
 - deleting domains, [310](#)
 - listing domains, [309–310](#)
- DoS attacks, [303](#), [424](#)**
- dump() function, [113](#), [114](#)**
- dumps() function, [113–114](#)**
- dynamic NAT, [541](#)**
- dynamic routes, [506](#)**

E

- earplugs, exam preparation, [552](#)**
- edge computing, [381–382](#)**
- editing code, [55](#), [64–65](#)**
- EIGRP, [201](#)**
- elif statements, [79–80](#)**
- else statements, [80](#), [82](#)**
- emulators (software), UCS Platform Emulator, [234](#)**
- encapsulation, [491](#)**
- encryption (data), [431](#)**
 - objectives of, [431](#)
 - public key encryption, [431–432](#)
 - TLS handshakes, [432](#)
- endpoints**
 - Cisco AMP for Endpoints, [320–321](#)
 - API, creating credentials, [322–323](#)*
 - API, list of, [325–326](#)*
 - API, listing computers, [323](#)*
 - API, listing vulnerabilities, [323–325](#)*
 - automation, [321–322](#)*

- uses of, 321–322*
 - collaboration endpoints, [260–261](#)
 - groups, creating with ERS API, [329–331](#)
- end-user security, 431**
- Enforcement API, 306, 308–310**
- environment variables (BASH), 37–38**
- EPG**
 - application profiles, [222, 223](#)
 - Cisco ACI, [221–222](#)
 - contracts, [222–223](#)
- error handling, Python, 119–121**
- ERS API, 327–331**
- Ethernet**
 - frames, [494](#)
 - switching, [492](#)
- event notification webhooks, 293**
- event subscription, UCS Manager, 233**
- Events page (DevNet), 17–18**
- exams**
 - 300–401 ENCOR exam, [8–9](#)
 - concentration exams
 - CCNP, 9–11*
 - DevNet Professional, 13–14*
 - core exams, CCNP, [9–11](#)
 - practice tests, Pearson Test Prep software, [556](#)
 - accessing tests, 553–554*
 - customizing tests, 554–555*
 - Premium Edition, 555–556*
 - updating tests, 555–556*
 - preparing for
 - budgeting time, 552*
 - chapter-ending review tools, 556*
 - DITKA questions, 556*
 - earplugs, 552*
 - getting rest, 553*
 - locking up valuables, 553*

Pearson Test Prep software, 553–554
practice tests, 553–554
study trackers, 552
suggested final review/study plans, 556
taking notes, 553
travel time, 552
watching the clock, 552

external bridged networks, Cisco ACI tenants, 221

external routed networks, Cisco ACI tenants, 221

F

fabric interconnects, 230

fabric policies, Cisco ACI MIT, 220

fast switching, 523, 524

FCS, 491, 492

feedback loops, DevOps, 392–393

feeds, Threat Grid, 335–337

files/folders. *See also* directories

BASH file management, 36–37

copying, 36

creating, 37

CSV files, 110–113

deleting, 37

FTP, 490

Git

committing files, 45, 53–54

file lifecycles, 40–41

git add command, 43–45

git clone command, 42, 46

git commit command, 45

git init command, 42–43, 46

git mv command, 44

git pull command, 47

git push command, 46–47

git remote command, 46

- git rm command, 44*
- git status command, 41, 43*
- Modified file status, 40*
- pushing/pulling files, 45–47*
- Staged file status, 40*
- touch command, 43*
- UnModified file status, 40*
- Untracked file status, 40*
- moving, 37
- Python, input/output, 108–110
- repositories, adding/removing files, 43
- requirements.txt files, 63
- timestamps, 37
- filtering content, reverse proxies, 445**
- final review/study plans, 556**
- Finesse (Unified CC), 259–260**
 - API, 275–276
 - authentication, 276–280*
 - Dialog API, 279–280*
 - Finesse Team API, 279*
 - Finesse User API, 277–279*
 - gadgets, 281*
 - high-level flow, 274–275
 - user states, 275
- Finesse Team API, Unified CC (Finesse) API, 279**
- Finesse User API, Unified CC (Finesse) API, 277–279**
- Firepower (Cisco), 314**
 - API, 315, 316–317
 - authentication, 315–316
 - components of, 314–315
 - features of, 314
 - Management Center objects, 317–318
 - networks, creating, 318–320
 - server version information, 316
 - session tokens, generating, 315–316, 318–320

system information, [316–317](#)

firewalls, [431](#), [437](#)

application-level (proxy) firewalls, [438–439](#)

next-generation firewalls, [439](#)

NGFW, [314](#)

packet filtering (stateless) firewalls, [437–438](#)

routers, [525](#)

stateful inspection firewalls, [438–439](#)

Flint, [271](#)

floating point numbers, [68](#), [69](#)

flow control

break statements, [82–83](#)

elif statements, [79–80](#)

else statements, [80](#), [82](#)

if statements, [78–80](#)

for loops, [79](#), [80–82](#)

while loops, [79](#), [82–83](#)

Fly category (DevNet Automation Exchange), [19](#)

fog computing, [381–382](#)

folders. *See* [files/folders](#)

for loops, [79](#), [80–82](#)

FQDN, [276](#)

Dialog API, [280](#)

Finesse Server, [277](#), [278](#), [281](#)

Finesse User API, [277](#)

Team API, [279](#)

frames. *See also* [PDU](#), [492](#), [494](#)

frameworks, Cybersecurity Framework, [422](#)

freeze command, [63](#)

FTP, [490](#)

functional tests, [122](#)

functions

Python functions, [88–89](#)

area_of_circle function, [123–124](#), [125–126](#)

arguments, [89–91](#)

close() function, [109–110](#)

- dir()* function, [97](#)
- dump()* function, [113](#), [114](#)
- dumps()* function, [113–114](#)
- getdesc()* function, [93–95](#)
- help()* function, [89](#), [97–98](#)
- items()* function, [91](#)
- load()* function, [113](#)
- loads()* function, [113](#), [114](#)
- max()* function, [89](#)
- open()* function, [109–110](#)
- parameters*, [89–91](#)
- print()* function, [88](#)
- test_value* function, [125](#)
- unparse()* function, [117](#)

RESTful API

- CRUD* functions, [132–133](#)
- HTTP* functions, [132–133](#)

G

gadgets, Unified CC (Finese) API, [281](#)

GET requests

- APIC REST API, [225–227](#)

- HTTP, [149](#), [150](#), [151–152](#), [161](#), [165](#), [168](#), [170](#)

GetBulkRequest messages (SNMP), [545](#)

getdesc() function, [93–95](#)

GetNextRequest messages (SNMP), [545](#)

GetRequest messages (SNMP), [545](#)

Git, [39](#), [42](#)

- branches, [47–48](#)

- adding*, [48–49](#)

- conflicts*, [52–53](#)

- merging*, [50–53](#)

- cloning/initiating repositories, [42–43](#)

- committing files, [45](#), [53–54](#)

- file lifecycles, [40–41](#)

- git add command, [43–45](#), [52](#)

- git branch command, [48–49](#)
- git checkout command, [49](#)
- git clone command, [42](#), [46](#)
- git commit command, [45](#), [52](#)
- git diff command, [53–54](#)
- git init command, [42–43](#), [46](#)
- git log command, [47–48](#)
- git merge command, [50](#), [51](#), [52](#)
- git mv command, [44](#)
- git pull command, [47](#)
- git push command, [46–47](#)
- git remote command, [46](#)
- git rm command, [44](#)
- git status command, [41](#), [43](#)
- GitHub and, [39](#)
- Modified file status, [40](#)
- pushing/pulling files, [45–47](#)
- repositories
 - adding/removing files*, [43–45](#)
 - cloning/initiating*, [42–43](#)
- Staged file status, [40](#)
- touch command, [43](#)
- tree structures, [40](#)
- UnModified file status, [40](#)
- Untracked file status, [40](#)
- workflows, [41](#)

GitHub

- Git and, [39](#)
- repositories, cloning, [42](#)

global addresses, [503](#)

gRPC, [343](#)

guest issuers, Webex Teams API, [272–273](#)

GUI, UCS Manager, [231](#)

H

hackers, [429](#)

- hardware API, [147–148](#)**
- hash (one-way), data integrity, [432](#)**
- HATEOAS, Cisco Intersight, [249](#)**
- HATEOS, [161](#)**
- HEAD requests, [150](#)**
- headers**
 - custom headers, versioning, [162](#)
 - HTTP, [152–153](#)
 - request headers, [153–154](#)*
 - response headers, [153–154](#)*
- Headset 500 Series (Cisco), [261](#)**
- Hello World, Docker, [404–405](#)**
- help() function, [89](#), [97–98](#)**
- hex (base 16) systems, Python, [69](#)**
- host ID, [497](#)**
- HTTP, [148](#), [490](#)**
 - BOSH, [275](#)
 - CRUD operations, [150–151](#)
 - headers, [152–153](#)
 - request headers, [153–154](#)*
 - response headers, [153–154](#)*
 - requests
 - components of, [149](#)*
 - CONNECT requests, [150](#)*
 - DELETE requests, [150](#), [151](#), [161](#)*
 - GET requests, [149](#), [150](#), [151–152](#), [161](#), [165](#), [168](#), [170](#)*
 - HEAD requests, [150](#)*
 - methods, [150–151](#)*
 - OPTIONS requests, [150](#)*
 - PATCH requests, [150](#), [151](#)*
 - POST requests, [150](#), [151](#), [152](#), [161](#), [166](#), [169](#)*
 - PUT requests, [150](#), [151](#), [161](#)*
 - request/response cycle, [148](#)*
 - TRACE requests, [150](#)*
 - response codes, [154–155](#)

REST methods, [150–151](#)
RESTCONF methods, [367–368](#)
status codes, [154–155](#)
transport layer (TCP/IP network model), [489–490](#)
webhooks, [158](#)
 development tools, [158](#)
 testing, [158](#)
 validation, [158–159](#)

HTTP functions, RESTful API, [132–133](#)

HTTPIe, [169–170](#)

Hub (Docker), [414–418](#)

hubs, [517](#)

hybrid clouds, [380](#)

I

IaaS, [378](#)

IANA, IPv6 addresses, [502](#)

ID

host ID, IPv4 addresses, [497](#)

network ID

Dashboard API (Meraki), [180](#), [183–187](#)

IPv4 addresses, [497](#)

organization ID, Dashboard API (Meraki), [180](#), [181–184](#)

IDS, [439–440](#)

IETF data models, [354](#)

if statements, [78–80](#)

images

 Docker images, [411–414](#)

 resizing via serverless applications, [386–388](#)

imported dictionaries, printing to command line, [116](#)

importing

 modules, [100–101](#)

 Python modules, [97–99](#)

information API, [147](#)

InformRequest messages (SNMP), 545

infra tenants, Cisco ACI MIT, 219

infrastructure automation, 458

Ansible, 458–462

Chef, 465–466

CI/CD pipelines, 455–457

Cisco NSO, 467, 470–471

architecture of, 467, 468

CDB, 468–469

CLI, 471

components of, 467

model-to-model mapping, 468

ncs-netsim, 469–470

NETCONF, 468

RESTCONF, 471–473

YANG, 468, 469

Cisco VIRL, 457, 474–476

device-level management, 453

infrastructure as a code, 454–455

network controllers, 451–452, 453

Puppet, 462–464

pyATS, 476–479

inheritance, Python classes, 94–96

initiating, repositories, 42–43

input() function, 77

input/output, Python files/folders, 108–110

installing Python, 61, 62–63

integers, 68, 69

Integration API (Cisco DNA Center), 190

**integration automation frameworks, Cisco ACI
MIT, 220**

integrations

tests, 122

Webex Meetings API, 283–284

Webex Teams API

access scopes, 265–266

Memberships API, 269

Messages API, 270

Organizations API, 266

Rooms API, 267–268

Teams API, 266–267

integrity of data (one-way hash), 432

Intent API (Cisco DNA Center), 190, 191–192

**interface names, changing in dictionaries, 118–
119**

Internet layer (TCP/IP network model), 489

Intersight (Cisco), 246–247

AccountMoid property, 248

Ancestors property, 248

authentication, 249

CreateTime property, 248

documentation, 249–250

HATEOAS, 249

managed object properties, 247–248

MIM, 247

ModTime property, 248

Moid identifier, 248

ObjectType property, 248

Owners property, 248

Parent property, 248

Python, 249–251

query language, 249

REST API, 247, 249

tagging, 248

Tags property, 248

URI, 248

Investigate API, 306, 311–313

IOC API, 334

IOS XE, NETCONF sessions, 360–362

IP address management, routers, 525

IP phone, collaboration, 260

IP services, 537

DHCP, 534–535

benefits of, 535

DHCPACK messages, 537

DHCPDISCOVER messages, 536–537

DHCPOFFER messages, 537

DHCPRELEASE messages, 537

DHCPREQUEST messages, 537

IPv4, 535

lease acknowledgement phase, 537

lease offer phase, 537

lease request phase, 537

phases of operation, 536

relay agents, 535–536

releasing phase, 537

server allocation, 535

server discovery phase, 536–537

state machine, 536

DNS, 540

caching resolution data, 539

lookups, 538–539

name resolution, 539

resolution process, 538

resolution queries, 540

NAT, 540–541

benefits of, 542

disadvantages of, 542–543

dynamic NAT, 541

IPv4, 543

PAT, 541–542

static NAT, 541

network diagrams, 547

NTP, 545–547

PAT, 541

SNMP, 544

advantages of, 543

community names, 545

- components of, 544*
- disadvantages of, 543*
- GetBulkRequest messages, 545*
- GetNextRequest messages, 545*
- GetRequest messages, 545*
- InformRequest messages, 545*
- managed devices, 544*
- MIB, 544–545*
- NMS, 544–545*
- OID values, 545*
- Response messages, 545*
- SetRequest messages, 545*
- SNMP agent, 544–545*
- SNMP manager, 544–545*
- Trap messages, 545*
- versions of, 543*

IPS, 440–441

IPv4

- addresses, 496
 - address masks, 498*
 - broadcast addresses, 498*
 - CIDR, 499–500*
 - classes of, 497–498*
 - decimal/binary values, 496*
 - host ID, 497*
 - network ID, 497*
 - private IPv4 addresses, 498–499*
 - subnets, 499–500*

DHCP, 535

NAT, 543

IPv6 addresses, 501–503

- anycast addresses, 503–504
- global addresses, 503
- IANA, 502
- link-local addresses, 503
- loopback addresses, 503

- multicast addresses, [503](#)
- subnets, [502](#)
- unspecified addresses, [503](#)

ISE (Cisco), [326](#)

- API, [327](#)
 - ERS API*, [327–331](#)
 - Session API*, [327](#)
- components of, [326–327](#)
- deployments, [326–327](#)
- profiles, [326](#)

ISO/IEC 12207, [26](#)

issuers (guest), Webex Teams API, [272–273](#)

items() function, [91](#)

J

JSON, [113–115](#), [156](#)

- Cisco AMP for Endpoints, listing vulnerabilities, [324–325](#)
- data format, [157](#)
- json modules, [102](#)
- JWT, [272–273](#), [283](#)
- keys, [156](#)
- RESTCONF, [368](#)

JWT, [272–273](#), [283](#)

K

Kitematic, [415–416](#)

knowledge domains (certification)

- DEVASC, [12](#)
- DevNet Professional, [13](#)

****kwargs, [90–91](#)**

L

LAMP, containerized applications, [385–386](#)

LAN, [492](#), [513](#)

- Layer 2 network diagrams, 547**
- Layer 2 switches, 518–519**
- Layer 3 network diagrams, 547**
- Layer 3 packet forwarding. *See* [routing](#)**
- Layer 3 switches, 519**
- layered system constraints (REST), 162**
- leaf nodes (YANG), 349–350**
- leaf-list nodes (YANG), 350**
- Lean model (SDLC), 28–29**
- lease acknowledgement phase (DHCP), 537**
- lease offer phase (DHCP), 537**
- lease request phase (DHCP), 537**
- lifecycle of software development. *See* [SDLC](#)**
- limiting rates**
 - Cisco DNA Center, [192](#)
 - Dashboard API (Meraki), [180](#)
 - REST API, [163–164](#)
- link-local addresses, 503**
- Linux, BASH, 32–33**
 - cat command, [34](#), [37](#)
 - cd command, [35](#)
 - cp command, [36](#)
 - directories, [35–36](#)
 - environment variables, [37–38](#)
 - file management, [36–37](#)
 - ls command, [36](#)
 - man command, [33](#)
 - mkdir command, [36](#)
 - more command, [34](#)
 - mv command, [37](#)
 - piping commands, [34](#)
 - pwd command, [35](#)
 - rm command, [37](#)
 - touch command, [37](#)
- list nodes (YANG), 351**
- listing**

- computers, Cisco AMP for Endpoints API, [323](#)
- details of curated feed types, Threat Grid, [336–337](#)
- devices, Cisco SD-WAN, [205–207](#)
- directories, [36](#)
- meetings, [286](#)
- rooms, Rooms API, [268](#)
- vulnerabilities, Cisco AMP for Endpoints API, [323–325](#)

lists, [72](#), [73–74](#)

- concatenating, [73](#)
- domains, listing in Cisco Umbrella, [309–310](#)
- list indexes, [73](#)
- methods, [74](#)

LLC layer (OSI network model), [486](#)

load balancing, [443–446](#)

load() function, [113](#)

loads() function, [113](#), [114](#)

locking up valuables (exam preparation), [553](#)

logins, Finesse User API, [278](#)

longest prefix match routing concept, [507](#)

lookups

- DNS lookups, [538–539](#)
- malicious domain lookups, [305](#)

loopback addresses, [503](#)

loopback interfaces, NETCONF, [365–367](#)

loops

- feedback loops, DevOps, [392–393](#)
- for loops, [79](#), [80–82](#)
- while loops, [79](#), [82–83](#)

ls command, [36](#)

LstsummaryMeeting API, Webex Meetings API, [286](#)

M

MAC addresses, [493–494](#), [495–496](#)

Mac computers, Python installations, [61](#)

MAC layer (OSI network model), [486](#)

Main page (DevNet), [14–15](#)

malicious code, [430](#)

malicious domain lookups, [305](#)

malware, [304](#), [320–321](#), [424](#)

MAN, [514–515](#)

man command, [33](#)

managed devices (SNMP), [544](#)

Management API, [305](#), [307–308](#)

management tenants, Cisco ACI MIT, [219](#)

managing, BASH files, [36–37](#)

manual pages, man command, [33](#)

math modules, [97–98](#)

max() function, [89](#)

meetings

creating, [284–285](#)

deleting, [288–289](#)

listing, [286](#)

setting/modifying attributes, [287–288](#)

Meetings XML API, Webex Meetings API, [289](#)

Memberships API, Webex Teams API

integration, [269](#)

Meraki, [178](#)

Captive Portal API, [178](#)

cloud computing, [178](#)

Dashboard API

accessing, [179–180](#)

action batches, [181](#)

base URL, [180](#), [181–182](#)

hierarchy of, [180](#)

network ID, [180](#), [183–187](#)

organization ID, [180](#), [181–184](#)

pagination, [181](#)

rate limiting, [180](#)

requests, [180](#)

use cases, [179](#)

Meraki Python SDK, [187–189](#)

MV Sense Camera API, [179](#)

Scanning API, [178–179](#)

merging, branches, [50–53](#)

Messages API, Webex Teams API integration, [270](#)

methods

list methods, [74](#)

Python methods, [93–94](#)

sqrt() method, [98, 99](#)

tan() method, [99](#)

string methods, [72](#)

MFA, [431](#)

MIB, SNMP, [544–545](#)

MIM

Cisco ACI, [219](#)

Cisco Intersight, [247](#)

MIT

Cisco ACI, [219–220](#)

UCS Manager, [231–232](#)

mitigating common security threats, [423–424](#)

MITM attacks, [303, 423](#)

mkdir command, [36](#)

MO

Cisco ACI, [219](#)

UCS Manager, [231](#)

model-driven frameworks, [218, 231](#)

model-driven programmability, [343–344](#)

NETCONF, [344](#)

Cisco IOS XE, NETCONF sessions, [360–362](#)

Cisco NSO, [468](#)

Cisco NX-OS, [363–365](#)

CLI commands, [345–346](#)

configuration data stores, [346–347](#)

loopback interfaces, [365–367](#)

notification messages, [356–357](#)

- operations, 346*
 - purpose of, 345*
 - RPC, 345, 356, 362–363*
 - server changes, 365–367*
 - YANG and, 344–346*
- RESTCONF, 367, 369–371**
 - API, 368*
 - Cisco NSO, 471–473*
 - HTTP methods, 367–368*
 - JSON, 368*
 - protocol stacks, 367*
 - URI, 368–369*
 - XML, 368*
- YANG, 347, 348**
 - built-in data types, 348*
 - container nodes, 350–351*
 - data models, augmenting, 355–356*
 - data models, components of, 352*
 - data models, example of, 352–354*
 - data models, IETF, 354*
 - data models, native, 354–355*
 - data models, nodes, 349*
 - data models, structure of, 357–360*
 - leaf nodes, 349–350*
 - leaf-list nodes, 350*
 - list nodes, 351*
 - NETCONF, Cisco IOS XE sessions, 360–362*
 - NETCONF, Cisco NX-OS, 363–365*
 - NETCONF, loopback interfaces, 365–367*
 - NETCONF, server changes, 365–367*
 - NETCONF notification messages, 356–357*
 - NETCONF RPC, 356, 362–363*
 - statements, 347, 348–349*
 - XML, 348*
- model-driven telemetry, 371–372**
- model-to-model mapping, 468**

- Modified status (Git files), [40](#)**
- modifier filters, XML API, [232](#)**
- modifying meeting attributes, [287–288](#)**
- ModTime property (Cisco Intersight), [248](#)**
- modules (Python), [96–97](#)**
 - calendar modules, [99](#)
 - Cisco infrastructure, [101–104](#)
 - csv modules, [102](#)
 - datetime modules, [101](#)
 - importing, [97–99](#), [100–101](#)
 - installing, [62–63](#)
 - json modules, [102](#)
 - math modules, [97–98](#)
 - napalm, [103](#)
 - ncclient modules, [103](#)
 - netmiko modules, [103](#)
 - nornir, [103–104](#)
 - os modules, [101](#)
 - pprint modules, [101](#)
 - pyang modules, [102](#)
 - pyats, [104](#)
 - pysnmp modules, [103](#)
 - Python standard library, [99](#)
 - PyYAML modules, [102](#)
 - requests modules, [102–103](#)
 - sys modules, [100](#), [101](#)
 - time modules, [102](#)
 - unittest, [104](#)
 - xmltodict modules, [102](#)
- Moid identifier (Cisco Intersight), [248](#)**
- monetization, REST API, [163–164](#)**
- more command, [34](#)**
- moving, files/folders, [37](#)**
- MQTT, [179](#)**
- MSSP, [307](#)**
- multicast addresses, [493](#), [503](#)**

multiplication, Python, [69](#)
MustUnderstand fault code, [139](#)
mv command, [37](#)
MV Sense Camera API, [179](#)
MVC software design pattern, [30–31](#)

N

name resolution, DNS, [539](#)
namespaces, Docker, [398–399](#)
napalm, [103](#)
NAT, [540–541](#)

- benefits of, [542](#)
- disadvantages of, [542–543](#)
- dynamic NAT, [541](#)
- IPv4, [543](#)
- PAT, [541–542](#)
- static NAT, [541](#)

native data models, YANG, [354–355](#)
ncclient modules, [103](#)
ncs-netsim, [469–470](#)
negotiation, content negotiation, versioning, [162](#)
NETCONF, [344](#)

- Cisco IOS XE, NETCONF sessions, [360–362](#)
- Cisco NSO, [468](#)
- Cisco NX-OS and, [363–365](#)
- CLI commands, [345–346](#)
- configuration data stores, [346–347](#)
- loopback interfaces, [365–367](#)
- notification messages, [356–357](#)
- operations, [346](#)
- purpose of, [345](#)
- RPC, [345, 356, 362–363](#)
- server changes, [365–367](#)
- YANG and, [344–346](#)

netmiko modules, [103](#)

**network access layer (TCP/IP network model),
488–489**

Network Device Management API, 306

network ID, 497

Cisco DNA Center, 195–197

Dashboard API (Meraki), 180

curl, 183–184

Postman, 184–187

network layer (OSI network model), 486–487

networking protocols, DHCP, 534–535

benefits of, 535

IPv4, 535

phases of operation, 536

relay agents, 535–536

server allocation, 535

state machine, 536

networks

bridges, 517–518

bus networks, 512

CAN, 513–514

control plane, 527–528

controllers, 451–452, 453

creating, Cisco Firepower, 318–320

data plane, 527

defined, 512

diagrams, 526, 547

elements of (overview), 516

external bridged networks, Cisco ACI tenants, 221

external routed networks, Cisco ACI tenants, 221

hubs, 517

LAN, 492, 513

Layer 2 switches, 518–519

Layer 3 switches, 519

MAN, 514–515

NFV, 474

OSI network model, 484–485, 488

- application layer, 488*
- data link layer, 486*
- layers, 485–488*
- LLC layer, 486*
- MAC layer, 486*
- network layer, 486–487*
- physical layer, 486*
- presentation layer, 487–488*
- session layer, 487*
- transport layer, 487*

private networks. *See* VRI instances

ring topology, *512*

routers, *521*

- domain name proxies, 526*
- IP address management, 525*
- network connections, 522*
- segmenting networks, 524*

routing, *496, 504, 506*

- CEF switching, 523–524*
- connected routes, 506*
- data packet routing, 504–506*
- dynamic routes, 506*
- fast switching, 523, 524*
- firewalls, 525*
- IPv4 addresses, 496–500*
- IPv6 addresses, 501–504*
- longest prefix match concept, 507*
- path determination, 504*
- process switching, 522, 524*
- static routes, 506*
- supernetting, 507*
- tables, 505–506*

SDN, *201–202, 526–527*

- Cisco SDN, 530–531*
- control plane, 527–528*
- controllers, 529–530*

- data plane, 527*
 - segmentation, [524](#)
 - standards, [513](#)
 - star topology, [512](#)
 - subnets
 - IPv4 addresses, 499–500*
 - IPv6 addresses, 502*
 - VLSM, 497, 499*
 - switching, [492](#)
 - Ethernet switching, 492*
 - frames, 492*
 - MAC addresses, 493–494, 495–496*
 - TCP/IP network model, [488](#)
 - application layer, 490*
 - data packet routing, 489*
 - data routing, 489*
 - de-encapsulation, 491*
 - encapsulation, 491*
 - Internet layer, 489*
 - network access layer, 488–489*
 - PDU, 490–491. See also frames*
 - TCP, 489–490*
 - transport layer, 489–490*
 - UDP, 490*
 - topologies, [512–513](#)
 - VLAN, [494–495, 520–521](#)
 - WAN, [515–516](#)
- next-generation firewalls, 439**
- Nexus switches, Cisco ACI, 216–217**
- NFV, 474**
- NGFW, 314**
- NIST**
 - cloud computing, [376–377](#)
 - Cybersecurity Framework, [422](#)
- Nmap**
 - CVE record detection, [427–429](#)

- vulnerability scanning, [426](#)
- NMS, [544–545](#)**
- nornir, [103–104](#)**
- northbound API, [130](#)**
- northbound interfaces, SDN controllers, [530](#)**
- notes (exam preparation), taking, [553](#)**
- notification bots, [271](#)**
- notification messages, NETCONF, [356–357](#)**
- Nslookup, [442–443](#)**
- NSO (Cisco), [467, 470–471, 530](#)**
 - architecture of, [467, 468](#)
 - CDB, [468–469](#)
 - CLI, [471](#)
 - components of, [467](#)
 - model-to-model mapping, [468](#)
 - ncs-netsim, [469–470](#)
 - NETCONF, [468](#)
 - RESTCONF, [471–473](#)
 - YANG, [468, 469](#)
- NTP, [545–547](#)**
- numeric operations, Python, [68–69](#)**

O

- ObjectType property (Cisco Intersight), [248](#)**
- Observer software design pattern, [31–32](#)**
- octal (base 8) systems, Python, [69](#)**
- OID values, [545](#)**
- OMP, [203](#)**
- one-way hash (data integrity), [432](#)**
- OOP, Python, [91–92](#)**
- Open Automation, UCS Director, [241](#)**
- open data models. *See* [IETF data models](#)**
- open() function, [109–110](#)**
- OpenAPI interface (Cisco SD-WAN), [203](#)**
- OpenConfig, YANG data models, [354–355](#)**
- OpenDayLight controllers, [529](#)**

OpenFlow controllers, [529](#)
options field (DHCP OFFER messages), [537](#)
OPTIONS requests, [150](#)
**organization ID, Dashboard API (Meraki), [180](#),
[181](#)**
 curl, [181–183](#)
 Postman, [183](#)
**Organizations API, Webex Teams API
integration, [266](#)**
os modules, [101](#)
OSI network model, [484–485](#), [488](#)
 application layer, [488](#)
 data link layer, [486](#)
 layers, [485–488](#)
 LLC layer, [486](#)
 MAC layer, [486](#)
 network layer, [486–487](#)
 physical layer, [486](#)
 presentation layer, [487–488](#)
 session layer, [487](#)
 transport layer, [487](#)
OSPF, [201](#)
OUI, [493](#)
output/input, Python files/folders, [108–110](#)
overloading. *See* [PAT](#)
OWASP, [424–426](#)
Owners property (Cisco Intersight), [248](#)

P

PaaS, [378](#)
packet filtering (stateless) firewalls, [437–438](#)
packet forwarding. *See* [routing](#)
packet routing, [489](#), [504–506](#)
pagination
 Dashboard API (Meraki), [181](#)
 REST API, [162–163](#)

- parameters, Python, [89–91](#)**
- Parent property (Cisco Intersight), [248](#)**
- parsing data, Python, [110](#)**
 - CSV files, [110–113](#)
 - JSON, [113–115](#)
 - XML, [115–117](#)
 - YAML, [117–119](#)
- partner API, [147](#)**
- passwords (strong), [431](#)**
- PAT, [541–542](#)**
- PATCH requests, [150](#), [151](#)**
- path determination, [504](#)**
- PDU, [490–491](#). *See also* [frames](#)**
- Pearson Test Prep software, practice tests, [556](#)**
 - accessing, [553–554](#)
 - customizing, [554–555](#)
 - Premium Edition, [555–556](#)
 - updating, [555–556](#)
- PEMDAS, [68](#)**
- penetration (pen) testing, [424–425](#)**
- People Presence, [294](#)**
- PEP, [177](#)**
- personal access tokens, Webex Teams API**
 - authentication, [262–263](#)
- phishing, [304](#), [424](#)**
- physical layer (OSI network model), [486](#)**
- pip command, [62–63](#)**
- pipelines (DevOps), [394–397](#)**
- piping commands, [34](#)**
- planning stage (SDLC), [25](#)**
- playbooks (Ansible), [458–459](#), [460–462](#)**
- PnP, API, [192](#)**
- polling, UCS Manager, [233](#)**
- ports**
 - system ports, [489–490](#)
 - well-known ports, [489–490](#)

POST requests, [150](#), [151](#), [152](#), [161](#), [166](#), [169](#)

Postman, [164–168](#)

Cisco DNA Center

authorization, [194–195](#)

network ID, [197](#)

Cisco SD-WAN, authorization, [205](#)

Dashboard API (Meraki)

network ID, [184–187](#)

organization ID, [183](#)

PowerShell, UCS Director, [242](#)

PowerTool suite (UCS Manager), [237](#)

pprint modules, [101](#)

practice tests, Pearson Test Prep software, [553–554](#), [556](#)

accessing tests, [553–554](#)

customizing tests, [554–555](#)

Premium Edition, [555–556](#)

updating tests, [555–556](#)

preparing for exams

budgeting time, [552](#)

chapter-ending review tools, [556](#)

DITKA questions, [556](#)

earplugs, [552](#)

getting rest, [553](#)

locking up valuables, [553](#)

Pearson Test Prep software, [556](#)

accessing, [553–554](#)

customizing, [554–555](#)

Premium Edition, [555–556](#)

updating, [555–556](#)

practice tests, [556](#)

accessing, [553–554](#)

customizing, [554–555](#)

Premium Edition, [555–556](#)

updating, [555–556](#)

study trackers, [552](#)

suggested final review/study plans, [556](#)

taking notes, [553](#)

travel time, [552](#)

watching the clock, [552](#)

presentation layer (OSI network model), [487](#)–[488](#)

print() function, [77](#)–[78](#), [88](#)

printing

from command line, `unparse()` function, [117](#)

directory paths, [35](#)

imported dictionaries to command line, [116](#)

private API, [147](#)

Private Cloud (Cisco AMP), [320](#)

private clouds, [379](#)

private IPv4 addresses, [498](#)–[499](#)

private networks. See VRI instances

private subnets, [221](#)

process switching, [522](#), [524](#)

programmability (model-driven), [343](#)–[344](#)

NETCONF, [344](#)

Cisco IOS XE, NETCONF sessions, [360](#)–[362](#)

Cisco NSO, [468](#)

Cisco NX-OS, [363](#)–[365](#)

CLI commands, [345](#)–[346](#)

configuration data stores, [346](#)–[347](#)

loopback interfaces, [365](#)–[367](#)

notification messages, [356](#)–[357](#)

operations, [346](#)

purpose of, [345](#)

RPC, [345](#), [356](#), [362](#)–[363](#)

server changes, [365](#)–[367](#)

YANG and, [344](#)–[346](#)

RESTCONF, [367](#), [369](#)–[371](#)

API, [368](#)

Cisco NSO, [471](#)–[473](#)

HTTP methods, [367](#)–[368](#)

- JSON, 368*
- protocol stacks, 367*
- URI, 368–369*
- XML, 368*
- YANG, 347, 348**
 - built-in data types, 348*
 - container nodes, 350–351*
 - data models, augmenting, 355–356*
 - data models, components of, 352*
 - data models, example of, 352–354*
 - data models, IETF, 354*
 - data models, native, 354–355*
 - data models, nodes, 349*
 - data models, structure of, 357–360*
 - leaf nodes, 349–350*
 - leaf-list nodes, 350*
 - list nodes, 351*
 - NETCONF, Cisco IOS XE sessions, 360–362*
 - NETCONF, Cisco NX-OS, 363–365*
 - NETCONF, loopback interfaces, 365–367*
 - NETCONF, server changes, 365–367*
 - NETCONF notification messages, 356–357*
 - NETCONF RPC, 356, 362–363*
 - statements, 347, 348–349*
 - XML, 348*
- property filters, XML API, 232**
- proxy (application-level) firewalls, 438–439**
- public API, 148**
- public clouds, 379–380**
- public key encryption, 431–432**
- public subnets, 221**
- Puppet, 462–464**
- Puppet Forge, 464**
- Puppet manifests, 462, 464**
- pushing/pulling, files, 45–47**
- PUT requests, 150, 151, 161**

pwd command, 35

pyang modules, 102

pyATS, 476–479

pyats, 104

pysnmp modules, 103

Python, 60–61

> operator, 70

>= operator, 70

== operator, 70

!= operator, 70

< operator, 70

<= operator, 70

() characters, 68

+ operator, 72, 73

""" (triple-quote) method, 66

acitoolkit, 227–229

APIC REST API, acitoolkit, 227–229

arguments, 89–91

Base64 encoding, 328

best practices, 436

binary (base 2) systems, 69

boolean comparisons, 70

Cisco AMP for Endpoints

listing computers, 323

listing vulnerabilities, 323

Cisco DNA Center Python SDK, 199–201

Cisco Firepower, generating session tokens, 315–316,

318–320

Cisco Intersight, 249–251

Cisco SD-WAN, 209–212

Cisco UCS Python SDK, 237–239

Cisco Umbrella

adding domains, 308–309

Base64 encoding, 306

deleting domains, 310

listing domains, 309–310

- classes, [92](#)
 - creating*, [92–93](#)
 - inheritance*, [94–96](#)
 - router classes*, [93–95](#)
- comments, [65–66](#)
- CSV files, [110–113](#)
- data types
 - commonly used data types*, [67](#)
 - dictionaries*, [75–76](#)
 - floating point numbers*, [68](#), [69](#)
 - integers*, [68](#), [69](#)
 - lists*, [72–74](#)
 - sets*, [76](#)
 - strings*, [70–72](#)
 - tuples*, [74–75](#)
- deactivate command, [62](#)
- dictionaries
 - changing interface names in dictionaries*, [118–119](#)
 - imported dictionaries, printing to command line*, [116](#)
- Flint, [436](#)
- error handling, [119–121](#)
- files/folders, input/output, [108–110](#)
- Finesse User API
 - logins*, [278](#)
 - team details*, [279](#)
 - user state changes*, [278–279](#)
- flow control
 - break statements*, [82–83](#)
 - elif statements*, [79–80](#)
 - else statements*, [80](#), [82](#)
 - if statements*, [78–80](#)
 - for loops*, [79](#), [80–82](#)
 - while loops*, [79](#), [82–83](#)
- freeze command, [63](#)

- functions, [88–89](#)
 - area_of_circle* function, [123–124](#), [125–126](#)
 - arguments, [89–91](#)
 - close()* function, [109–110](#)
 - dir()* function, [97](#)
 - dump()* function, [113](#), [114](#)
 - dumps()* function, [113–114](#)
 - getdesc()* function, [93–95](#)
 - help()* function, [89](#), [97–98](#)
 - items()* function, [91](#)
 - load()* function, [113](#)
 - loads()* function, [113](#), [114](#)
 - max()* function, [89](#)
 - open()* function, [109–110](#)
 - parameters, [89–91](#)
 - print()* function, [88](#)
 - test_value* function, [125](#)
 - unparse()* function, [117](#)
- hex (base 16) systems, [69](#)
- input()* function, [77](#)
- installing, [61](#), [62–63](#)
- Macs and, [61](#)
- Memberships API, adding members, [269](#)
- Meraki Python SDK, [187–189](#)
- Messages API, adding messages to rooms, [270](#)
- methods, [93–94](#)
 - sqrt()* method, [98](#), [99](#)
 - tan()* method, [99](#)
- modules, [96–97](#)
 - calendar* modules, [99](#)
 - Cisco infrastructure*, [101–104](#)
 - csv* modules, [102](#)
 - datetime* modules, [101](#)
 - importing, [97–99](#), [100–101](#)
 - json* modules, [102](#)
 - math* modules, [97–98](#)

- napalm*, [103](#)
- ncclient* modules, [103](#)
- netmiko* modules, [103](#)
- nornir*, [103–104](#)
- os* modules, [101](#)
- pprint* modules, [101](#)
- pyang* modules, [102](#)
- pyats*, [104](#)
- pysnmp* modules, [103](#)
- Python standard library, [99](#)
- PyYAML modules, [102](#)
- requests* modules, [102–103](#)
- sys* modules, [100](#), [101](#)
- time* modules, [102](#)
- unittest*, [104](#)
- xmldict* modules, [102](#)

multiplication, [69](#)

numeric operations, [68–69](#)

octal (base 8) systems, [69](#)

OOP, [91–92](#)

parameters, [89–91](#)

parsing data, [110](#)

- CSV* files, [110–113](#)
- JSON*, [113–115](#)
- XML*, [115–117](#)
- YAML*, [117–119](#)

PEMDAS, [68](#)

PEP, [177](#)

pip command, [62–63](#)

print() function, [77–78](#)

pyATS, [476–479](#)

Python 3, [61–62](#), [64](#)

Python standard library, [99](#)

remainders, [69](#)

Requests, [171–172](#)

requirements.txt files, [63](#)

Rooms API, creating, [267–268](#)
syntax of, [63–66](#)
TDD, [121–122](#)
Teams API, creating, [266–267](#)
testing
 functional tests, [122](#)
 integration tests, [122](#)
 TDD, [121–122](#)
 unit testing, [122–126](#)
text editors, Atom text editor, [64–65](#)
Threat Grid
 feeds, [336–337](#)
 listing details of curated feed types, [336–337](#)
Unified CC (Finesse) API
 authentication, [277](#)
 Base64 encoding, [277](#)
Unified CM, AXL API, [294–297](#)
user input, [77](#)
variables, [66–67](#)
versions of, [61–62](#)
virtual environments, [61–62](#)
whole numbers, [69](#)
xAPI
 getting endpoint status, [291–292](#)
 People Presence, [294](#)
 setting camera positions, [292](#)
Zeep Client Library, [296–297](#)
PyYAML modules, [102](#)

Q

query parameter versioning, [162](#)

R

ransomware, [320](#)

rate limiting

 Cisco DNA Center, [192](#)

- Dashboard API (Meraki), [180](#)
- REST API, [163–164](#)
- Receiver fault code, [139](#)**
- recipes (Chef), [465–466](#)**
- registry (Docker), [401](#), [405](#), [414–415](#)**
- relay agents, DHCP, [535–536](#)**
- releasing phase (DHCP), [537](#)**
- remainders, Python, [69](#)**
- removing, files from repositories, [43–45](#)**
- Reporting API, [305](#)**
- repositories**
 - adding/removing files, [43–45](#)
 - cloning/initiating, [42–43](#)
 - creating, [42–43](#)
- request headers, [153–154](#)**
- requests**
 - Dashboard API (Meraki), [180](#)
- GET requests, APIC REST API, [225–227](#)**
 - HTTP
 - components of, [149](#)*
 - CONNECT requests, [150](#)*
 - DELETE requests, [150](#), [151](#), [161](#)*
 - GET requests, [149](#), [150](#), [151–152](#), [161](#), [165](#), [168](#), [170](#)*
 - HEAD requests, [150](#)*
 - methods, [150–151](#)*
 - OPTIONS requests, [150](#)*
 - PATCH requests, [150](#), [151](#)*
 - POST requests, [150](#), [151](#), [152](#), [161](#), [166](#), [169](#)*
 - PUT requests, [150](#), [151](#), [161](#)*
 - request/response cycle, [148](#)*
 - TRACE requests, [150](#)*
 - Python Requests, [171–172](#)
- requests modules, [102–103](#)**
- requirements analysis, SDLC, [25](#)**
- requirements.txt files, [63](#)**

resizing images via serverless applications, 386–388

response codes (HTTP), 154–155

response headers, 153–154

Response messages (SNMP), 545

rest (exam preparation), getting, 553

REST API

APIC REST API, 223

acitoolkit, 227–229

authentication, 223–225

get requests, 225–227

Cisco Intersight, 247, 249

curl, 168–169

debugging tools, 172

development tools, 172

HTTPIe, 169–170

monetization, 163–164

pagination, 162–163

Postman, 164–168

Python Requests, 171–172

rate limiting, 163–164

UCS Director, 242–245

versioning, 162

REST constraints, 160

cache constraints, 161

client/server constraints, 160

code on demand constraints, 162

layered system constraints, 162

stateless constraints, 161

uniform interface constraints, 161

REST methods, 150–151

RESTCONF, 367, 369–371

API, 368

Cisco NSO, 471–473

HTTP methods, 367–368

JSON, 368

protocol stacks, [367](#)

URI, [368–369](#)

XML, [368](#)

RESTful API, [133](#)

authentication, [133–134](#)

CRUD functions, [132–133](#)

HTTP functions, [132–133](#)

reverse API. *See* [webhooks](#)

reverse proxies, [444–446](#)

reviewing

code, [55](#)

for exams

chapter-ending review tools, [556](#)

suggested final review/study plans, [556](#)

ring network topology, [512](#)

rm command, [37](#)

RN, UCS Manager, [232](#)

Room Analytics People Presence Detector, [294](#)

rooms

adding members, [269](#)

adding messages, [270](#)

Rooms API

creating, [267–268](#)

listing rooms, [268](#)

Webex Teams API integration, [267–268](#)

routed networks (external), Cisco ACI tenants, [221](#)

router classes

example of, [93–94](#)

inheritance, [94–95](#)

routing, [496, 504, 506](#)

CEF switching, [523–524](#)

CIDR, subnets, [499–500](#)

connected routes, [506](#)

data, TCP/IP network model, [489](#)

data packet routing, [504–506](#)

- dynamic routes, [506](#)
- fast switching, [523](#), [524](#)
- IPv4 addresses, [496](#)
 - address masks*, [498](#)
 - broadcast addresses*, [498](#)
 - CIDR*, [499–500](#)
 - classes of*, [497–498](#)
 - decimal/binary values*, [496](#)
 - host ID*, [497](#)
 - network ID*, [497](#)
 - private IPv4 addresses*, [498–499](#)
 - subnets*, [499–500](#)
- IPv6 addresses, [501–503](#)
 - anycast addresses*, [503–504](#)
 - global addresses*, [503](#)
 - IANA*, [502](#)
 - link-local addresses*, [503](#)
 - loopback addresses*, [503](#)
 - multicast addresses*, [503](#)
 - subnets*, [502](#)
 - unspecified addresses*, [503](#)
- longest prefix match concept, [507](#)
- path determination, [504](#)
- process switching, [522](#), [524](#)
- routers, [521](#)
 - domain name proxies*, [526](#)
 - firewalls*, [525](#)
 - IP address management*, [525](#)
 - network connections*, [522](#)
 - segmenting networks*, [524](#)
- static routes, [506](#)
- supernetting, [507](#)
- tables, [505–506](#)
- RPC, [140](#)**
 - gRPC, [343](#)
 - high-level communication, [140](#)

NETCONF and, [345](#), [356](#), [362–363](#)
XML-RPC reply messages, [141](#)
XML-RPC request messages, [140–141](#)

RSA, [432](#)

Run category (DevNet Automation Exchange), [19](#)

S

SaaS, [378](#)

Sample API, [334](#)

sandboxes, UCS Manager, [234](#)

Scanning API, [178–179](#)

script modules, UCS Director, [242](#)

scripts, XSS, [304](#), [424](#)

SDK, [176–178](#)

advantages of, [177](#)

AXL SDK, [297–298](#)

Cisco DNA Center, [189–190](#)

authorization, [193–194](#)

client data, [198–199](#)

Integration API, [190](#)

Intent API, [190](#), [191–192](#)

network ID, [195–197](#)

non-Cisco device management, [191](#)

platform interface, [193](#)

Python SDK, [199–201](#)

rate limiting, [192](#)

versions of, [193](#)

webhooks, [191](#)

Cisco UCS Python SDK, [237–239](#)

Meraki, [178](#)

Captive Portal API, [178](#)

cloud computing, [178](#)

Dashboard API, [179–187](#)

Meraki Python SDK, [187–189](#)

MV Sense Camera API, [179](#)

Scanning API, [178–179](#)

- qualities of, [177](#)
- UCS Director SDK, [241](#)
- Webex Teams, [273–274](#)

SDLC, [25](#)

- Agile model, [29–30](#)
- ISO/IEC 12207, [26](#)
- Lean model, [28–29](#)
- models, overview, [26](#)
- secure development, [434–436](#)
- stages of, [25–26](#)
- Waterfall model, [27–28](#)
 - phases of*, [27](#)
 - value problem*, [28](#)

SDN, [201–202](#), [526–527](#)

- Cisco SDN, [530–531](#)
- control plane, [527–528](#)
- controllers, [529–530](#)
- data plane, [527](#)

SD-WAN (Cisco), [201–202](#), [203](#), [530–531](#)

- API, [203](#)
- authorization, [204–205](#)
- device templates, [207–208](#)
- listing devices, [205–207](#)
- OpenAPI interface, [203](#)
- Python scripting, [209–212](#)
- vBond, [202](#)
- vEdge, [202](#)
- vManage, [202](#), [203–212](#)
- vSmart, [202](#)

security

- applications
 - best practices*, [431](#)
 - common threats*, [423–424](#)
 - CVE records*, [425–426](#), [427–429](#)
 - Cybersecurity Framework*, [422](#)
 - data integrity (one-way hash)*, [432](#)

- data security, 433–434*
- device security, 431*
- digital signatures, 432–433*
- DNS, 440–443*
- encryption, 431–433*
- end-user security, 431*
- firewalls, 431, 437–439*
- IDS, 439–440*
- IPS, 440–441*
- load balancing, 443–446*
- MFA, 431*
- mitigating common threats, 423–424*
- Nmap, 426–429*
- Nslookup, 442–443*
- one-way hash (data integrity), 432*
- OWASP, 424–426*
- passwords (strong), 431*
- penetration (pen) testing, 424–425*
- reverse proxies, 444–446*
- SDLC, 434–436*
- software development, 434–436*
- three-tier application security, 430*
- updating software, 431*

Cisco AMP

- Cisco AMP for Endpoints, 320–326*
- Private Cloud, 320–321*

Cisco Firepower, 314

- API, 315, 316–317*
- authentication, 315–316*
- components of, 314–315*
- creating networks, 318–320*
- features of, 314*
- generating session tokens, 315–316, 318–320*
- Management Center objects, 317–318*
- server version information, 316*
- system information, 316–317*

- Cisco ISE, [326](#)
 - API*, [327–331](#)
 - components of*, [326–327](#)
 - deployments*, [326–327](#)
 - profiles*, [326](#)
- Cisco security portfolio, [302–303](#)
- Cisco Umbrella, [304](#)
 - adding domains*, [308–309](#)
 - API*, [305–306](#)
 - authentication*, [306](#)
 - categorizing domains*, [312–313](#)
 - Console Reporting API*, [306](#)
 - deleting domains*, [310](#)
 - domain lookups*, [305](#)
 - Enforcement API*, [306](#), [308–310](#)
 - flow example*, [305](#)
 - Investigate API*, [306](#), [311–313](#)
 - listing domains*, [309–310](#)
 - Management API*, [305](#), [307–308](#)
 - Network Device Management API*, [306](#)
 - Reporting API*, [305](#)
- CVE records, [425–426](#), [427–429](#)
- Cybersecurity Framework, [422](#)
- data integrity (one-way hash), [432](#)
- data security, [433–434](#)
- development, [434–436](#)
- devices, [431](#)
- digital signatures, [432–433](#)
- Dllint, [436](#)
- DNS, [440–443](#)
- domain lookups, [305](#)
- encryption, [431](#)
 - objectives of*, [431](#)
 - public key encryption*, [431–432](#)
 - TLS handshakes*, [432](#)
- end-user security, [431](#)

- firewalls, [431](#), [437](#)
 - application-level (proxy) firewalls*, [438–439](#)
 - next-generation firewalls*, [439](#)
 - packet filtering (stateless) firewalls*, [437–438](#)
 - routers*, [525](#)
 - stateful inspection firewalls*, [438–439](#)
- hackers, [429](#)
- IDS, [439–440](#)
- IPS, [440–441](#)
- load balancing, [443–446](#)
- malicious code, [430](#)
- MFA, [431](#)
- MSSP, [307](#)
- NGFW, [314](#)
- Nmap
 - CVE record detection*, [427–429](#)
 - vulnerability scanning*, [426](#)
- Nslookup, [442–443](#)
- one-way hash (data integrity), [432](#)
- OWASP, [424–426](#)
- passwords (strong), [431](#)
- reverse proxies, [444–446](#)
- RSA, [432](#)
- SDLC, [434–436](#)
- software development, [434–436](#)
- SSL, [423](#), [432](#), [445–446](#)
- Threat Grid, [331](#)
 - API*, [332–337](#)
 - architecture of*, [331–332](#)
 - submissions searches*, [334–335](#)
- threats, [303](#)
 - APT*, [320–321](#)
 - brute-force attacks*, [304](#), [424](#)
 - buffer overflow attacks*, [423](#)
 - common threats*, [303–304](#), [423–424](#)
 - Cross-Site Scripting (XSS)*, [304](#), [424](#)

- defined, [303](#)*
- DoS attacks, [303](#), [424](#)*
- hackers, [429](#)*
- malicious code, [430](#)*
- malware, [304](#), [320–321](#), [424](#)*
- mitigating common threats, [423–424](#)*
- MITM attacks, [303](#), [423](#)*
- phishing, [304](#), [424](#)*
- ransomware, [320](#)*
- SQL injections, [304](#), [424](#)*
- XSS, [304](#), [424](#)*
- three-tier application security, [430](#)
- TLS, [423](#), [432](#), [434](#)
- vulnerabilities
 - CVE records, [425–426](#), [427–429](#)*
 - defined, [303](#)*
- segmenting networks, [524](#)**
- Sender fault code, [139](#)**
- sequence diagrams, [159–160](#)**
- server/client constraints (REST), [160](#)**
- serverless applications, [386–388](#)**
- servers**
 - discovery phase (DHCP), [536–537](#)
 - NETCONF server changes, [365–367](#)
 - TLD servers, [538](#)
 - version information, Cisco Firepower, [316](#)
- service API, [146–147](#)**
- service profiles, UCS Manager, [231](#)**
- Session API, [327](#)**
- session authentication, xAPI**
 - creating sessions, [291](#)
 - current device status, [291–292](#)
 - event notification webhooks, [293](#)
 - setting device attributes, [292](#)
- session layer (OSI network model), [487](#)**
- SetMeeting API, Webex Meetings API, [287–288](#)**

SetRequest messages (SNMP), 545

sets, 76

setting/modifying meeting attributes, 287–288

shared subnets, 221

shells, BASH, 32–33

cat command, 34, 37

cd command, 35

cp command, 36

directories, 35–36

environment variables, 37–38

file management, 36–37

ls command, 36

man command, 33

mkdir command, 36

more command, 34

mv command, 37

piping commands, 34

pwd command, 35

rm command, 37

touch command, 37

signatures (digital), 432–433

simple filters, XML API, 232

SingleConnect technology, UCS Manager, 230

SLA, 545–546

SNMP, 342, 343–344, 544

advantages of, 543

community names, 545

components of, 544

disadvantages of, 543

GetBulkRequest messages, 545

GetNextRequest messages, 545

GetRequest messages, 545

InformRequest messages, 545

managed devices, 544

MIB, 544–545

NMS, 544–545

- OID values, [545](#)
- Response messages, [545](#)
- SetRequest messages, [545](#)
- SNMP agent, [544–545](#)
- SNMP manager, [544–545](#)
- Trap messages, [545](#)
- versions of, [543](#)

SOAP, [136](#)

- AXL SOAP API, [296–297](#)
- fault codes, [138–139](#)
- fault options, [138–139](#)
- high-level communication, [137–138](#)
- message format, [136–137](#)
- sample faults, [139–140](#)
- sample message, [138](#)

software

- design patterns, [30](#)
 - MVC*, [30–31](#)
 - Observer*, [31–32](#)
- emulators, UCS Platform Emulator, [234](#)
- routing, [522–524](#)
- SDLC, [25](#)
 - Agile model*, [29–30](#)
 - ISO/IEC 12207*, [26](#)
 - Lean model*, [28–29](#)
 - models, overview*, [26](#)
 - secure development*, [434–436](#)
 - stages of*, [25–26](#)
 - Waterfall model*, [27–28](#)
- SDN, [526–527](#)
 - Cisco SDN*, [530–531](#)
 - control plane*, [527–528](#)
 - controllers*, [529–530](#)
 - data plane*, [527](#)
- secure development, [434–436](#)
- updating, [431](#)

version control

Git, [39–54](#)

SVC, [38–39](#)

southbound API, [130](#)

southbound interfaces, SDN controllers, [530](#)

spaces (characters)

in Atom text editor, [64–66](#)

Python and, [64–66](#)

spaces, collaboration, [261](#)

special characters

**kwargs, [90–91](#)

*args, [90–91](#)

== operator, [70](#)

!= operator, [70](#)

> operator, [70](#)

>= operator, [70](#)

< operator, [70](#)

<= operator, [70](#)

""" (triple-quote) method, [66](#)

() characters, Python, [68](#)

| character, piping commands, [34](#)

+ operator, [72, 73](#)

character, Python comments, [65](#)

specialist certifications, [11, 13–14](#)

SQL injections, [304, 424](#)

sqrt() method, [98, 99](#)

SSL, [423, 432, 445–446](#)

Staged status (Git files), [40](#)

staging, [40, 41, 43–44](#)

standards (networks), [513](#)

star network topology, [512](#)

state changes (users), Finesse User API, [278–279](#)

stateful inspection firewalls, [438–439](#)

stateless (packet filtering) firewalls, [437–438](#)

stateless constraints (REST), [161](#)

static NAT, [541](#)

static routes, 506

status codes (HTTP), 154–155

strings, 70, 71–72

concatenating, 72

DevNet string index, 70–71

string methods, 72

strong passwords, 431

study trackers, 552

submissions searches, Threat Grid API, 334–335

subnets

Cisco ACI tenants, 221

IPv4 addresses, 499–500

IPv6 addresses, 502

private subnets, 221

public subnets, 221

shared subnets, 221

VLSM, 497, 499

subscription (event), UCS Manager, 233

suggested final review/study plans, 556

supernetting, 507

Support page (DevNet), 17

SVC (Software Version Control), 38–39

SWIM, 191

switching, 492

CEF switching, 523–524

Ethernet switching, 492

fast switching, 523, 524

frames, 492

Layer 2 switches, 518–519

Layer 3 switches, 519

MAC addresses, 493–494, 495–496

Nexus switches, Cisco ACI, 216–217

process switching, 522, 524

VLAN, 494–495

symbols

**kwargs, 90–91

- *args, [90–91](#)
- == operator, [70](#)
- != operator, [70](#)
- > operator, [70](#)
- >= operator, [70](#)
- < operator, [70](#)
- <= operator, [70](#)
- """ (triple-quote) method, [66](#)
- () characters, Python, [68](#)
- | character, piping commands, [34](#)
- + operator, [72](#), [73](#)
- # character, Python comments, [65](#)

synchronous API, [131](#)

sys modules, [100](#), [101](#)

system ports, [489–490](#)

T

tabs

- in Atom text editor, [65](#)
- Python and, [65](#)

tagging, Cisco Intersight, [248](#)

Tags property (Cisco Intersight), [248](#)

tan() method, [99](#)

tasks, UCS Director, [240](#), [242](#)

TCP, [489–490](#)

TCP/IP network model, [488](#)

- application layer, [490](#)
- data packet routing, [489](#)
- data routing, [489](#)
- de-encapsulation, [491](#)
- encapsulation, [491](#)
- Internet layer, [489](#)
- network access layer, [488–489](#)
- PDU, [490–491](#)
- PDU. *See also* [frames](#)
- TCP, [489–490](#)

transport layer, [489–490](#)

UDP, [490](#)

TDD, [121–122](#)

Teams API

creating, [266–267](#)

Webex Teams API integration, [266–267](#)

Technologies page (DevNet), [15–16](#)

telemetry (model-driven), [371–372](#)

templates

device templates, Cisco SD-WAN, [207–208](#)

UCS Director, [240](#)

tenants

Cisco ACI, [219](#)

bridge domains, [221](#)

components of, [220–221](#)

external bridged networks, [221](#)

external routed networks, [221](#)

subnets, [221](#)

VRF instances, [221](#)

Cisco ACI MIT, [219](#)

Terraform, [455](#)

test_value function, [125](#)

testing

pyATS, [476–479](#)

pyats, [104](#)

pyramid, [122–123](#)

Python

functional tests, [122](#)

integration tests, [122](#)

TDD, [121–122](#)

unit testing, [122–123](#)

unittest, [104](#)

webhooks, [158](#)

testing stage (SDLC), [26](#)

text editors, Atom text editor, [64–65](#)

Threat Grid, [331](#)

API, [332](#)

API keys, [333](#)

Data API, [334](#)

format of, [332](#)

IOC API, [334](#)

Sample API, [334](#)

submissions searches, [334–335](#)

“Who am I” queries, [333–334](#)

architecture of, [331–332](#)

feeds, [335–337](#)

threats (security), [303](#)

APT, [320–321](#)

brute-force attacks, [304](#), [424](#)

buffer overflow attacks, [423](#)

common threats, [303–304](#), [423–424](#)

Cross-Site Scripting (XSS), [304](#), [424](#)

defined, [303](#)

DoS attacks, [303](#), [424](#)

hackers, [429](#)

malicious code, [430](#)

malware, [304](#), [320–321](#), [424](#)

mitigating common threats, [423–424](#)

MITM attacks, [303](#), [423](#)

phishing, [304](#), [424](#)

ransomware, [320](#)

SQL injections, [304](#), [424](#)

XSS, [304](#), [424](#)

three-tier application security, [430](#)

time

exam preparation

budgeting time, [552](#)

travel time, [552](#)

watching the clock, [552](#)

NTP, [545–547](#)

time modules, [102](#)

timestamps, files/folders, [37](#)

TLD servers, [538](#)

TLS, [423](#), [432](#), [434](#)

tokens

 custom authentication tokens, [135–136](#)

 personal access tokens, Webex Teams API
 authentication, [262–263](#)

topologies (networks), [512–513](#)

touch command, [37](#), [43](#)

TRACE requests, [150](#)

transport layer

 OSI network model, [487](#)

 TCP/IP network model, [489–490](#)

Trap messages (SNMP), [545](#)

travel time, exam preparation, [552](#)

triple-quote method (""), [66](#)

troubleshooting applications, connectivity

 issues, [548–550](#)

TSP API, Webex Meetings API, [281](#)

tuples, [74–75](#)

turning on/off, virtual environments, [62](#)

U

UCS Director, [239–240](#)

 Cisco ACI, [240](#)

 Open Automation, [241](#)

 PowerShell, [242](#)

 REST API, [242–245](#)

 script modules, [242](#)

 SDK, [241](#)

 tasks, [240](#), [242](#)

 templates, [240](#)

 user profiles, retrieving, [244–245](#)

 workflows, [240](#), [241](#), [245](#)

UCS Manager, [230](#), [232](#)

 API, [231](#)

 authentication, [234](#)

- Cisco UCS Python SDK, [237–239](#)
- CLI, [231](#)
- compute object lists, [234–237](#)
- connectivity, [230](#)
- DME, [231](#)
- DN, [232](#)
- documentation, [233](#)
- event subscription, [233](#)
- GUI, [231](#)
- MIT, [231–232](#)
- MO, [231](#)
- polling, [233](#)
- PowerTool suite, [237](#)
- sandboxes, [234](#)
- service profiles, [231](#)
- SingleConnect technology, [230](#)
- UCS Platform Emulator, [234](#)
- Visore, [233](#)
- XML API, [231](#), [232](#)
 - authentication*, [234](#)
 - compute object lists*, [234–237](#)

UCS Platform Emulator, [234](#)

UDP, [490](#)

Umbrella (Cisco), [304](#)

- API, [305–306](#)
- authentication, [306](#)
- Console Reporting API, [306](#)
- domain lookups, [305](#)
- domains
 - adding domains*, [308–309](#)
 - categorizing domains*, [312–313](#)
 - deleting domains*, [310](#)
 - listing domains*, [309–310](#)
- Enforcement API, [306](#), [308–310](#)
- flow example, [305](#)
- Investigate API, [306](#), [311–313](#)

Management API, [305](#), [307–308](#)
Network Device Management API, [306](#)
Reporting API, [305](#)

unicast addresses, [493](#)

Unified CC (Finesse), [259–260](#)

API, [275–280](#)
high-level flow, [274–275](#)
user states, [275](#)

Unified CC (Finesse) API

Dialog API, [279–280](#)
Finesse Team API, [279](#)
Finesse User API, [277–279](#)
gadgets, [281](#)

Unified CM, [259](#)

AXL API, [294–295](#)
 AXL SOAP API, [296–297](#)
 toolkit, [295–296](#)
 Zeep Client Library, [296–297](#)
AXL SDK, [297–298](#)
overview of, [294](#)

Unified Communications, [257–258](#)

Webex Calling, [258–259](#)
Webex Teams, [258](#)
 API, [261–273](#)
 SDK, [273–274](#)

uniform interface constraints (REST), [161](#)

Union File System, [399–400](#)

unit testing, [122–126](#)

unittest, [104](#)

UNIX

directories
 changing, [35](#)
 creating, [36](#)
 listing, [36](#)
 navigating, [35–36](#)
 printing paths, [35](#)

- structure of, 35*
 - file management, [36–37](#)
- UnModified status (Git files), [40](#)**
- unparse() function, [117](#)**
- unspecified IPv6 addresses, [503](#)**
- Untracked status (Git files), [40](#)**
- updating**
 - practice tests, Pearson Test Prep software, [555–556](#)
 - software, [431](#)
- URI**
 - Cisco Intersight, [248](#)
 - path versioning, [162](#)
 - RESTCONF URI, [368–369](#)
- URL**
 - Dashboard API (Meraki), [180](#), [181–182](#)
 - HTTP requests, [149](#)
- URL API, Webex Meetings API, [281](#)**
- user input, Python, [77](#)**
- user profiles, UCS Director, retrieving user profiles, [244–245](#)**
- user state changes, Finesse User API, [278–279](#)**
- user tenants, Cisco ACI MIT, [219](#)**

V

- validation, webhooks, [158–159](#)**
- valuables (exam preparation), locking up, [553](#)**
- variables**
 - BASH environment variables, [37–38](#)
 - Python, [66–67](#)
- vBond, [202](#), [531](#)**
- vEdge, [202](#), [531](#)**
- vendor-assigned addresses, [493](#)**
- version control (software)**
 - Git, [39](#), [42](#)
 - adding/removing files to repositories, [43–45](#)*
 - branches, [47–52](#)*

cloning/initiating repositories, [42-43](#)
committing files, [45](#), [53-54](#)
file lifecycles, [40-41](#)
git add command, [43-45](#), [52](#)
git branch command, [48-49](#)
git checkout command, [49](#)
git clone command, [42](#), [46](#)
git commit command, [45](#), [52](#)
git diff command, [53-54](#)
git init command, [42-43](#), [46](#)
git log command, [47-48](#)
git merge command, [50](#), [51](#), [52](#)
git mv command, [44](#)
git pull command, [47](#)
git push command, [46-47](#)
git remote command, [46](#)
git rm command, [44](#)
git status command, [41](#), [43](#)
GitHub and, [39](#)
Modified file status, [40](#)
pushing/pulling files, [45-47](#)
Staged file status, [40](#)
touch command, [43](#)
tree structures, [40](#)
UnModified file status, [40](#)
Untracked file status, [40](#)
workflows, [41](#)

SVC, [38-39](#)

versioning

content negotiation, [162](#)
custom headers, [162](#)
query parameter versioning, [162](#)
REST API, [162](#)
URI path versioning, [162](#)

VersionMismatch fault code, [139](#)

video devices, collaboration, [260](#)

VIRL (Cisco), [457](#), [474–476](#)

virtual environments

Python, [61–62](#)

turning on/off, [62](#)

virtualization

application deployments, [383–384](#)

NFV, [474](#)

Visore, [233](#)

VLAN, [494–495](#), [520–521](#)

VLSM, [497](#), [499](#)

VM

domains, Cisco ACI MIT, [220](#)

UCS Platform Emulator, [234](#)

vManage, [202](#), [203–212](#), [531](#)

VRF instances, Cisco ACI tenants, [221](#)

vSmart, [202](#), [531](#)

vulnerabilities

CVE records, [425–426](#)

detecting with Nmap, [427–429](#)

listing, Cisco AMP for Endpoints API, [323–325](#)

Nmap

CVE record detection, [427–429](#)

vulnerability scanning, [426](#)

vulnerabilities (security), defined, [303](#)

W

Walk category (DevNet Automation Exchange), [19](#)

WAN, [515–516](#). See also [Cisco SD-WAN](#)

Waterfall model (SDLC), [27](#), [28](#)

phases of, [27](#)

value problem, [28](#)

web acceleration, reverse proxies, [445](#)

Webex (Cisco), [260](#)

Webex Board (Cisco), [260](#)

Webex Calling, [258–259](#)

Webex Devices, overview of, [289–290](#)

Webex Meetings API

- architecture of, [282, 284](#)
- authentication, overview of, [283](#)
- CreateMeeting API, [284–285](#)
- DelMeeting API, [288–289](#)
- integrations, [283–284](#)
- LstsummaryMeeting API, [286](#)
- Meetings XML API, [289](#)
- overview of, [281–282](#)
- SetMeeting API, [287–288](#)
- supported services, [282–283](#)
- TSP API, [281](#)
- URL API, [281](#)
- XML API, [281](#)

Webex Share, [260](#)

Webex Teams, [258](#)

- API, [258, 261–262](#)
 - access scopes, [265–266](#)*
 - authentication, [262–273](#)*
 - bots, [271–272](#)*
 - guest issuers, [272–273](#)*
 - integrations, [263–270](#)*
 - Memberships API, [269](#)*
 - Messages API, [270](#)*
 - Organizations API, [266](#)*
 - personal access tokens, [262–263](#)*
 - Rooms API, [267–268](#)*
 - Teams API, [266–267](#)*

SDK, [273–274](#)

webhooks, [158](#)

- Cisco DNA Center, [191](#)
- development tools, [158](#)
- event notification webhooks, xAPI, [293](#)
- testing, [158](#)
- validation, [158–159](#)

well-known ports, 489–490

while loops, 79, 82–83

whitespace

in Atom text editor, 64–66

Python and, 64–66

“Who am I” queries, Threat Grid API, 333–334

whole numbers, Python, 69

workflows

Git, 41

UCS Director, 240, 241, 245

X

xAPI

authentication, 290–291

creating sessions, 291

current device status, 291–292

event notification webhooks, 293

session authentication, 291–293

setting device attributes, 292

categories, 290

overview of, 290

People Presence, 294

XML, 115–117, 155–156

AXL API, 294–295

AXL SOAP API, 296–297

toolkit, 295–296

Zeep Client Library, 296–297

data format, 156

RESTCONF, 368

YANG, 348

XML API

data filters, 232

UCS Manager, 231, 232

authentication, 234

compute object lists, 234–237

Webex Meetings API, 281

XML-RPC

reply messages, [141](#)

request messages, [140–141](#)

xmltodict modules, [102](#)

XMPP, [256](#), [275](#)

XSS, [304](#), [424](#)

Y

YAML, [117–119](#), [157–158](#), [461](#)

YANG, [347](#), [348](#)

built-in data types, [348](#)

Cisco NSO, [468](#), [469](#)

container nodes, [350–351](#)

data models

augmenting, [355–356](#)

components of, [352](#)

example of, [352–354](#)

IETF data models, [354](#)

native data models, [354–355](#)

nodes, [349](#)

structure of, [357–360](#)

leaf nodes, [349–350](#)

leaf-list nodes, [350](#)

list nodes, [351](#)

NETCONF, [344–346](#), [360–362](#)

Cisco NX-OS and, [363–365](#)

loopback interfaces, [365–367](#)

notification messages, [356–357](#)

RPC, [356](#), [362–363](#)

server changes, [365–367](#)

statements, [347](#), [348–349](#)

XML, [116](#), [348](#)

yiaddr field (DHCPOFFER messages), [537](#)

Z

Zeep Client Library, [296–297](#)



Connect, Engage, Collaborate

The Award Winning Cisco Support Community

Attend and Participate in Events

- Ask the Experts
- Live Webcasts

Knowledge Sharing

- Documents
- Blogs
- Videos

Top Contributor Programs

- Cisco Designated VIP
- Hall of Fame
- Spotlight Awards

Multi-Language Support



<https://supportforums.cisco.com>

Exclusive Offer – 40% OFF

Cisco Press Video Training

livelessons®

ciscopress.com/video

Use coupon code **CPVIDEO40** during checkout.



Video Instruction from Technology Experts



Advance Your Skills

Get started with fundamentals, become an expert, or get certified.



Train Anywhere

Train anywhere, at your own pace, on any device.



Learn

Learn from trusted author trainers published by Cisco Press.

Try Our Popular Video Training for FREE!

ciscopress.com/video

Explore hundreds of **FREE** video lessons from our growing library of Complete Video Courses, LiveLessons, networking talks, and workshops.

Cisco Press

ciscopress.com/video

ALWAYS LEARNING

PEARSON



**REGISTER YOUR PRODUCT at [CiscoPress.com/register](https://www.ciscopress.com/register)
Access Additional Benefits and SAVE 35% on Your Next Purchase**

- Download available product updates.
- Access bonus material when applicable.
- Receive exclusive offers on new editions and related products.
(Just check the box to hear from us when setting up your account.)
- Get a coupon for 35% for your next purchase, valid for 30 days.
Your code will be available in your Cisco Press cart. (You will also find it in the Manage Codes section of your account page.)

Registration benefits vary by product. Benefits will be listed on your account page under Registered Products.

CiscoPress.com – Learning Solutions for Self-Paced Study, Enterprise, and the Classroom
Cisco Press is the Cisco Systems authorized book publisher of Cisco networking technology, Cisco certification self-study, and Cisco Networking Academy Program materials.

At [CiscoPress.com](https://www.ciscopress.com) you can

- Shop our books, eBooks, software, and video training.
- Take advantage of our special offers and promotions ([ciscopress.com/promotions](https://www.ciscopress.com/promotions)).
- Sign up for special offers and content newsletters ([ciscopress.com/newsletters](https://www.ciscopress.com/newsletters)).
- Read free articles, exam profiles, and blogs by information technology experts.
- Access thousands of free chapters and video lessons.

Connect with Cisco Press – Visit [CiscoPress.com/community](https://www.ciscopress.com/community)

Learn about Cisco Press community events and programs.



Cisco Press

ALWAYS LEARNING

PEARSON

Appendix C

Study Planner

Key:

Practice Test
Reading
Review

Element	Task	Goal Date	First Date Completed	Second Date Completed (Optional)	Notes
Introduction	Read Introduction				
Your Study Plan	Read Your Study Plan				
1. Introduction to Cisco DevNet Associate Certificate				Read Foundation Topics	
2. Software Development and Design				Read Foundation Topics	
2. Software Development and Design				Review Key Topics using the book or companion website	
2. Software Development and Design				Define Key Terms using the book or companion website	
2. Software Development and Design				Repeat DIKTA questions using the book or PTP exam engine	

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

3. Introduction to Python | Read Foundation Topics

3. Introduction to Python	Review Key Topics using the book or companion website
---------------------------	---

3. Introduction to Python	Define Key Terms using the book or companion website
---------------------------	--

3. Introduction to Python	Repeat DIKTA questions using the book or PTP exam engine
---------------------------	--

3. Introduction to Python	Complete all memory tables in this chapter using the companion website
---------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

4. Python Functions, Classes, and Modules | Read Foundation Topics

4. Python Functions, Classes, and Modules	Review Key Topics using the book or companion website
---	---

4. Python Functions, Classes, and Modules	Define Key Terms using the book or companion website
---	--

4. Python Functions, Classes, and Modules	Repeat DIKTA questions using the book or PTP exam engine
---	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

5. Working with Data in Python | Read Foundation Topics

5. Working with Data in Python	Review Key Topics using the book or companion website
--------------------------------	---

5. Working with Data in Python	Define Key Terms using the book or companion website
--------------------------------	--

5. Working with Data in Python	Repeat DIKTA questions using the book or PTP exam engine
--------------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

Test	test software for this chapter
------	--------------------------------

6. Application Programming Interface (API)	Read Foundation Topics
--	------------------------

6. Application Programming Interface (API)	Review Key Topics using the book or companion website
--	---

6. Application Programming Interface (API)	Define Key Terms using the book or companion website
--	--

6. Application Programming Interface (API)	Repeat DIKTA questions using the book or PTP exam engine
--	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

7. REST APIs	Read Foundation Topics
--------------	------------------------

7. REST APIs	Review Key Topics using the book or companion website
--------------	---

7. REST APIs	Define Key Terms using the book or companion website
--------------	--

7. REST APIs	Repeat DIKTA questions using the book or PTP exam engine
--------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

8. Cisco Enterprise Networking Management Platforms and APIs	Read Foundation Topics
--	------------------------

8. Cisco Enterprise Networking Management Platforms and APIs	Review Key Topics using the book or companion website
--	---

8. Cisco Enterprise Networking Management Platforms and APIs	Define Key Terms using the book or companion website
--	--

8. Cisco Enterprise Networking Management Platforms and APIs	Repeat DIKTA questions using the book or PTP exam engine
--	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

9. Cisco Data Center and Compute Management Platforms and APIs	Read Foundation Topics
--	------------------------

9. Cisco Data Center and Compute	Review Key Topics using the
----------------------------------	-----------------------------

Management Platforms and APIs	book or companion website
-------------------------------	---------------------------

9. Cisco Data Center and Compute Management Platforms and APIs	Define Key Terms using the book or companion website
--	--

9. Cisco Data Center and Compute Management Platforms and APIs	Repeat DIKTA questions using the book or PTP exam engine
--	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

10. Cisco Collaboration Platforms and APIs	Read Foundation Topics
--	------------------------

10. Cisco Collaboration Platforms and APIs	Review Key Topics using the book or companion website
--	---

10. Cisco Collaboration Platforms and APIs	Define Key Terms using the book or companion website
--	--

10. Cisco Collaboration Platforms and APIs	Repeat DIKTA questions using the book or PTP exam engine
--	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

11. Cisco Security Platforms and APIs	Read Foundation Topics
---------------------------------------	------------------------

11. Cisco Security Platforms and APIs	Review Key Topics using the book or companion website
---------------------------------------	---

11. Cisco Security Platforms and APIs	Define Key Terms using the book or companion website
---------------------------------------	--

11. Cisco Security Platforms and APIs	Repeat DIKTA questions using the book or PTP exam engine
---------------------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

12. Model-driven Programmability	Read Foundation Topics
----------------------------------	------------------------

12. Model-driven Programmability	Review Key Topics using the book or companion website
----------------------------------	---

12. Model-driven Programmability	Define Key Terms using the book or companion website
----------------------------------	--

--	--

12. Model-driven Programmability	Repeat DIKTA questions using the book or PTP exam engine
----------------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

13. Deploying Applications | Read Foundation Topics

13. Deploying Applications	Review Key Topics using the book or companion website
----------------------------	---

13. Deploying Applications	Define Key Terms using the book or companion website
----------------------------	--

13. Deploying Applications	Repeat DIKTA questions using the book or PTP exam engine
----------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

14. Application Security | Read Foundation Topics

14. Application Security	Review Key Topics using the book or companion website
--------------------------	---

14. Application Security	Define Key Terms using the book or companion website
--------------------------	--

14. Application Security	Repeat DIKTA questions using the book or PTP exam engine
--------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

15. Infrastructure Automation | Read Foundation Topics

15. Infrastructure Automation	Review Key Topics using the book or companion website
-------------------------------	---

15. Infrastructure Automation	Define Key Terms using the book or companion website
-------------------------------	--

15. Infrastructure Automation	Repeat DIKTA questions using the book or PTP exam engine
-------------------------------	--

15. Infrastructure Automation	Complete all memory tables in this chapter using the companion website
-------------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

16. Network Fundamentals	Read Foundation Topics
--------------------------	------------------------

16. Network Fundamentals	Review Key Topics using the book or companion website
--------------------------	---

16. Network Fundamentals	Define Key Terms using the book or companion website
--------------------------	--

16. Network Fundamentals	Repeat DIKTA questions using the book or PTP exam engine
--------------------------	--

16. Network Fundamentals	Complete all memory tables in this chapter using the companion website
--------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

17. Networking Components	Read Foundation Topics
---------------------------	------------------------

17. Networking Components	Review Key Topics using the book or companion website
---------------------------	---

17. Networking Components	Define Key Terms using the book or companion website
---------------------------	--

17. Networking Components	Repeat DIKTA questions using the book or PTP exam engine
---------------------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

18. IP Services	Read Foundation Topics
-----------------	------------------------

18. IP Services	Review Key Topics using the book or companion website
-----------------	---

18. IP Services	Define Key Terms using the book or companion website
-----------------	--

18. IP Services	Repeat DIKTA questions using the book or PTP exam engine
-----------------	--

Practice Test	Take practice test in study mode using Exam #1 in practice test software for this chapter
---------------	---

Practice Test	Take practice test in study mode using Part Review exam in practice test software for this part
---------------	---

Final Review	Take practice test in study mode for all Book Questions in practice test software
--------------	---

Final Review	Review all Key Topics in all chapters or in the Key Topics App using the companion website
--------------	--

Final Review	Review all Key Terms in all chapters or using the Key Terms Flashcards on the companion website
--------------	---

Final Review	Complete all memory tables and practice exercises for all chapters using the companion website
--------------	--

Final Review	Take practice test in practice exam mode using Exam Bank #1 questions for all chapters
--------------	--

Final Review	Take practice test in practice exam mode using Exam Bank #2 questions for all chapters
--------------	--



Where are the companion content files?

Register this digital version of [Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide](#) to access important downloads.

Register this eBook to unlock the companion files. Follow these steps:

1. Go to ciscopress.com/account and log in or create a new account.
2. Enter the ISBN: **9780136642961** (NOTE: Please enter the print book ISBN provided to register the eBook you purchased.)
3. Answer the challenge question as proof of purchase.
4. Click on the “Access Bonus Content” link in the Registered Products section of your account page, to be taken to the page where your downloadable content is available.

This eBook version of the print title does not contain the practice test software that accompanies the print book.

You May Also Like—Premium Edition eBook and Practice Test. To learn about the Premium Edition eBook and Practice Test series, visit ciscopress.com/practicetest

The Professional and Personal Technology Brands of Pearson



Cisco Press

informIT

PEARSON IT Certification

QUE

SAMS

Cisco Certified

DevNet Associate

DEVASC 200-901

Official Cert Guide

ISBN: 978-0-13-664296-1

**See other side for your Pearson Test Prep
Practice Test activation code and special
offers ▶▶▶**

Complete Video Course

To enhance your preparation, Cisco Press also sells Complete Video Courses for both streaming and download. Complete Video Courses provide you with hours of expert-level instruction mapped directly to exam objectives.

Special Offer—Save 70%

This single-use coupon code will allow you to purchase the Complete Video Course at a 70% discount. Simply go to the product URL below, add the Complete Video Course to your cart, and apply the coupon code at checkout.

*Cisco Certified DevNet Associate DEVASC 200-901
Complete Video Course*
www.ciscopress.com/title/9780136904427

Coupon Code:

Cisco Certified

DevNet Associate

DEVASC 200-901 Official

Cert Guide

Premium Edition and Practice Tests

To enhance your preparation, Cisco Press also sells a digital Premium Edition of this book. The Premium Edition provides you with three eBook files (PDF, EPUB, and MOBI/Kindle) as well as an enhanced edition of the Pearson IT Certification Practice Test. The Premium Edition includes two additional practice exams with links for every question mapped to the PDF eBook.

Special Offer—Save 80%

This single-use coupon code will allow you to purchase a copy of the Premium Edition at an 80% discount. Simply go to the URL below, add the Premium Edition to your cart, and apply the coupon code at checkout.

www.ciscopress.com/title/9780136642985

Coupon Code:

DO NOT DISCARD THIS NUMBER

You will need this activation code to activate your practice test in the Pearson Test Prep practice test software. To access the online version, go to www.PearsonTestPrep.com. Select **Pearson IT Certification** as your product group. Enter your email/password for your account. If you don't have an account on PearsonITCertification.com or CiscoPress.com, you will need to establish one by going to www.PearsonITCertification.com/join. In the My Products tab, click the **Activate New Product button**. Enter the access code printed on this insert card to activate your product. The product will now be listed in your My Products page.

If you wish to use the Windows desktop offline version of the application, simply register your book at www.ciscopress.com/register, select the Registered Products tab on your account page, click the Access Bonus Content link, and download and install the software from the companion website.

This access code can be used to register your exam in both the online and offline versions.

Activation Code:

Cisco Certified

DevNet Associate

DEVASC 200-901

Official Cert Guide

Enhance Your Exam Preparation

Save 70% on Complete Video Course

The sample *Complete Video Course*, available for both streaming and download, provides you with hours of expert-level instruction mapped directly to exam objectives.

Save 80% on Premium Edition eBook and Practice Test

The *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide Premium Edition eBook and Practice Test* provides three eBook files (PDF, EPUB, and MOBI/Kindle) to read on your preferred device and an enhanced edition of the Pearson Test Prep practice test software. You also receive two additional practice exams with links for every question mapped to the PDF eBook.

See the card insert in the back of the book for your Pearson Test Prep activation code and special offers.

Cisco Certified DevNet Associate DEVASC

200-901

Official Cert Guide

Companion Website

Access interactive study tools on this book's companion website, including practice test software, video training, memory tables, review exercises, Key Term flash card application, study planner, and more!

To access the companion website, simply follow these steps:

1. Go to www.ciscopress.com/register.
2. Enter the print book ISBN: **9780136642961**.
3. Answer the security question to validate your purchase.
4. Go to your account page.
5. Click on the **Registered Products** tab.
6. Under the book listing, click on the **Access Bonus Content** link.

If you have any issues accessing the companion website, you can contact our support team by going to pearsonitp.ehelp.org.

Code Snippets

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

```
$ man man

man(1)                                man(1)

NAME
    man - format and display the on-line manual pages

SYNOPSIS
    man [-acdfPhkKtW] [--path] [-m system] [-p string] [-C config_file]
    [-M pathlist] [-P pager] [-B browser] [-H htmlpager] [-S section_list]
    [section] name ...

DESCRIPTION
    man formats and displays the on-line manual pages.  If you specify section, man only looks in that section of the manual.  name is normally the name of the manual page, which is typically the name of a command, function, or file.  However, if name contains a slash (/) then man interprets it as a file specification, so that you can do man ./foo.5 or even man /cd/foo/bar.1.gz.

    See below for a description of where man looks for the manual page files.

<output cut for brevity>
```

```

$cat weather.py | more

import json
import urllib.request
from pprint import pprint

def get_local_weather():

    weather_base_url = 'http://forecast.weather.gov/MapClick.php?FcstType=json&'

    places = {
        'Austin': ['30.3074624', '-98.0335911'],
        'Portland': ['45.542094', '-122.9346037'],
        'NYC': ['40.7053111', '-74.258188']
    }

    for place in places:
        latitude, longitude = places[place][0], places[place][1]
        weather_url = weather_base_url + "lat=" + latitude + "&lon=" + longitude
        # Show the URL we use to get the weather data. (Paste this URL into your
        browser!)
        # print("Getting the current weather for", place, "at", weather_url, ":")

        page_response = urllib.request.urlopen(weather_url).read()
<output cut for brevity>

```

```
$ echo $PATH
```

```

/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Applications/VMware
Fusion.app/Contents/Public:/opt/X11/bin

```

```
$ export PATH=$PATH:/Home/chrijack/bin
```

```
$ echo "export PATH=$PATH:/Home/chrijack/bin" >> .bashrc
git clone (url to repository) (directory to clone to)
```

```

#git clone https://github.com/CiscoDevNet/pyats-coding-101.git
Cloning into 'pyats-coding-101'...
remote: Enumerating objects: 71, done.
remote: Total 71 (delta 0), reused 0 (delta 0), pack-reused 71
Unpacking objects: 100% (71/71), done.
#cd pyats-coding-101
#pyats-coding-101 git:(master) ls
COPYRIGHT          coding-102-parsers
LICENSE            coding-103-yaml
README.md          coding-201-advanced-parsers
coding-101-python
git init (directory name)

```

```
#git init newrepo
```

```

Initialized empty Git repository in /Users/chrijack/I
GitHub/newrepo/.git/

```

```
#newrepo git:(master)
```

```
#newrepo git:(master) touch newfile
```

```
#newrepo git:(master) ls
```

```
newfile
```

```
# git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    newfile

nothing added to commit but untracked files present (use "git add"
to track)
# git add newfile

# git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   newfile
git rm (-r) (-f) (folder/file.py)
git mv (-f) (source) (destination)
# ls

oldfile.py

# git mv oldfile.py newfile.py

# ls

newfile.py
git commit [-a] [-m] <"your commit message">
# git commit -a -m "bug fix 21324 and 23421"

[master e1fec3d] bug fix 21324 and 23421

 1 file changed, 0 insertions(+), 0 deletions(-)

delete mode 100644 newfile
# git remote add origin https://github.com/chrijack/devnetccna.git
# git remote -v

origin  https://github.com/chrijack/devnetccna.git (fetch)
origin  https://github.com/chrijack/devnetccna.git (push)
git push (remotename) (branchname)
```

```
# git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 210 bytes | 210.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/chrijack/devnetccna.git
* [new branch]      master -> master

Branch 'master' set up to track remote branch 'master' from 'ori-
gin'.
git pull (remotename) (branchname)
```

```
# git pull origin master
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
From https://github.com/chrijack/devnetccna
* branch      master      -> FETCH_HEAD
   8eb16e3..40aafla master  -> origin/master
Updating 8eb16e3..40aafla
Fast-forward
 2README.md      | 3 +++
 Picture1.png    | Bin 0 -> 83650 bytes
 Picture2.jpg    | Bin 0 -> 25895 bytes
 Picture3.png    | Bin 0 -> 44064 bytes
 4 files changed, 3 insertions(+)
 create mode 100644 2README.md
 create mode 100644 Picture1.png
 create mode 100644 Picture2.jpg
 create mode 100644 Picture3.png
```

```
#git log

commit 40aaflaf65ae7226311a01209b62ddf7f4ef88c2 (HEAD -> master, origin/master)
Author: Chris Jackson <chrijack@cisco.com>
Date: Sat Oct 19 00:00:34 2019 -0500

    Add files via upload

commit 1a9db03479a69209bf722b21d8ec50f94d727e7d
Author: Chris Jackson <chrijack@cisco.com>
Date: Fri Oct 18 23:59:55 2019 -0500

    Rename README.md to 2README.md

commit 8eb16e3b9122182592815falcc029493967c3bca
Author: Chris Jackson <chrijack@me.com>
Date: Fri Oct 18 20:03:32 2019 -0500

    first commit
```

```
git branch (-d) <branchname> [commit]
git checkout [-b] (branchname or commit)
#git checkout newfeature
```

```
Switched to branch 'newfeature'
git merge (branch to merge with current)
#git add .
```

```
#git commit -a -m "new feature"
```

```
#git checkout master

Switched to branch 'master'
# git merge newfeature

Updating 77f786a..dd6bce5

Fast-forward

 text1 | 1 +

 1 file changed, 1 insertion(+)
#git merge newfeature
Auto-merging text1
CONFLICT (content): Merge conflict in text1
Automatic merge failed; fix conflicts and then commit the result.
#git add .

#git commit -m "merge conflict fixed"

[master fe8f42d] merge conflict fixed
git diff [--stat] [branchname or commit]
#git diff --cached
diff --git a/text2 b/text2

new file mode 100644

index 0000000..b9997e5

--- /dev/null

+++ b/text2

@@ -0,0 +1 @@

+new bit of code.....
```

```
#git diff newfeature text1
diff --git a/text1 b/text1
index 45c2489..ba0a07d 100644
--- a/text1
+++ b/text1
@@ -1,3 +1,4 @@
 line 1
 line 2
+line 3
..new feature code.....
```

```
# MacOS or Linux

python3 -m venv myenv

source myenv/bin/activate
```

```
# Windows
```

```
C:\py -3 -m venv myenv
```

```
C:\myenv\Scripts\activate.bat
```

```
pip install package==1.1.1. To install a specific version
```

```
pip install package>=1.0 To install a version greater than or
equal to 1.0
```

```
pip install -r requirements.txt
```

```
pip freeze > requirements.txt
```

```
>>> for kids in ["Caleb", "Sydney", "Savannah"]:
```

```
... print("Clean your room,", kids, "!")
```

```
File "<stdin>", line 2
```

```
    print("Clean your room,", kids, "!")
```

```
>>> ^
for kids in ["Caleb", "Sydney", "Savannah"]:
```

```
...     print("Clean your room,", kids, "!")
```

```
...
```

```
Clean your room, Caleb !
```

```
Clean your room, Sydney !
```

```
Clean your room, Savannah !
```

```
#get input from user in numeric format
```

```
>>> '10' + 1
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
TypeError: can only concatenate str (not "int") to str
```

```
>>> 'DevNet Rocks ' * 5
```

```
'DevNet Rocks DevNet Rocks DevNet Rocks DevNet Rocks DevNet Rocks '
```

```
>>> kids = ['Caleb', 'Sydney', 'Savannah']
```

```
>>> kids
```

```
['Caleb', 'Sydney', 'Savannah']
```

```

>>> kids
['Caleb', 'Sidney', 'Savannah']
>>> kids[1]="Sydney"
>>> kids
['Caleb', 'Sydney', 'Savannah']

>>>
>>> c= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> c[1:4]
[2, 3, 4]
>>> c[ :-4]
[1, 2, 3, 4, 5, 6]
>>> c[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> person = (2012, 'Mike', 'CCNA')
>>> person
(2012, 'Mike', 'CCNA')
>>> type(person)
<class 'tuple'>
>>> person[0]=15
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> (a, b, c) = (12, 'Fred',18)
>>> c
18
>>> cabinet = { "scores":(98,76,95), "name":"Chris",
"company":"Cisco"}
>>> type(cabinet)
<class 'dict'>

```



```

>>> cabinet["address"] = {"street": "123 Anywhere Dr",
"city": "Franklin", "state": "TN"}
>>> cabinet["address"]
{'street': '123 Anywhere Dr', 'city': 'Franklin', 'state': 'TN'}
>>> numbs = {1, 2, 4, 5, 6, 8, 10}

>>> odds = {1, 3, 5, 7, 9}
>>> numbs | odds
-----
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
>>> inpt = input('Type your name: ')

Type your name: Chris Jackson

>>> inpt

'Chris Jackson'
>>> inpt = float(input('What is the Temperature in F: '))

What is the Temperature in F: 83.5

>>> inpt

83.5
>>> print('Numbers in set', 1, ':', numbs )

Numbers in set 1 : {1, 2, 4, 5, 6, 8, 10}
>>> print('Numbers in set ', 1, ': ', numbs, sep=' ')

Numbers in set 1: {1, 2, 4, 5, 6, 8, 10}
>>> name = 'Piper'

>>> name2 = 'Chris'

>>> print(f'{name2} says Hi to {name}!')

Chris says Hi to Piper!
>>> n = 20

>>> if n == 20:

...     print('The number is 20')

...

The number is 20
-----

```

```
>>> if n == 20:
...     print('oops')
File "<stdin>", line 2
    print('oops')
    ^
IndentationError: expected an indented block
>>> n = 3
```

```
>>> if n == 17:
...     print('Number is 17')
... elif n < 10:
...     print('Number is less than 10')
... elif n > 10:
...     print('Number is greater than 10')
... 
```

Number is less than 10

```
score = int(input('What was your test score?:'))

if score >= 90:
    print('Grade is A')
elif score >= 80:
    print('Grade is B')
elif score >= 70:
    print('Grade is C')
elif score >= 60:
    print('Grade is D')
else:
    print('Grade is F')

What was your test score?:53
Grade is F
>>>
```

```
>>> count = 1
>>> while (count < 5):
...     print("Loop count is:", count)
...     count = count + 1
... else:
...     print("loop is finished")
...
Loop count is: 1
Loop count is: 2
Loop count is: 3
Loop count is: 4
loop is finished
```

```
while True:
    string = input('Enter some text to print. \nType "done" to quit> ')
    if string == 'done' :
        break
    print(string)
print('Done!')
```

```
Enter some text to print.
Type "done" to quit> Good luck on the test!
Good luck on the test!
Enter some text to print
Type "done" to quit> done
Done!
```

```
Python 3.8.1 (v3.8.1:1b293b6006, Dec 18 2019, 14:08:53)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>> def devnet():
    '''prints simple function'''
    print('Simple function')

>>> devnet()
Simple function
>>> help(devnet)

Help on function devnet in module __main__:

devnet()
    prints simple function
>>> def hello(*args):
    for arg in args:
        print("Hello", arg, "!")

>>> hello('Caleb', 'Sydney', 'Savannah')
Hello Caleb !
Hello Sydney !
Hello Savannah !
>>> def hello(**kwargs):
    for key, value in kwargs.items():
        print("Hello", value, "!")

>>> hello(kwarg1='Caleb', kwarg2='Sydney', kwarg3='Savannah')

Hello Caleb !
Hello Sydney !
Hello Savannah !
-----
```

```
>>> def greeting(name, message="Good morning!"):
    print("Hello", name + ', ' + message)
```

```
>>> greeting('Caleb')
```

```
Hello Caleb, Good morning!
```

```
>>> greeting('Sydney', "How are you?")
```

```
Hello Sydney, How are you?
```

```
class Router:
```

```
    '''Router Class'''
```

```
    def __init__(self, model, swversion, ip_add):
```

```
        '''initialize values'''
```

```
        self.model = model
```

```
        self.swversion = swversion
```

```
        self.ip_add = ip_add
```

```
rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
```

```
>>> rtr1.desc = 'virtual router'-----
```

```
>>> rtr1.desc
```

```
'virtual router'
```

```
>>> rtr2= Router('ISR4221', '16.9.5', '10.10.10.5')
```

```
>>> rtr2.model
```

```
'ISR4221'
```

```
class Router:
    '''Router Class'''
    def __init__(self, model, swversion, ip_add):
        '''initialize values'''
        self.model = model
        self.swversion = swversion
        self.ip_add = ip_add

    def getdesc(self):
        '''return a formatted description of the router'''
        desc = f'Router Model           :{self.model}\n'\
              f'Software Version         :{self.swversion}\n'\
              f'Router Management Address:{self.ip_add}'
        return desc

rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
rtr2 = Router('ISR4221', '16.9.5', '10.10.10.5')

print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
print('Rtr2\n', rtr2.getdesc(), sep='')
```

Rtr1

Router Model :iosV
Software Version :15.6.7
Router Management Address:10.10.10.1

Rtr2

Router Model :isr4221
Software Version :16.9.5
Router Management Address:10.10.10.5

```
class Router:
    '''Router Class'''
    def __init__(self, model, swversion, ip_add):
        '''initialize values'''
        self.model = model
        self.swversion = swversion
        self.ip_add = ip_add

    def getdesc(self):
        '''return a formatted description of the router'''
        desc = (f'Router Model           :{self.model}\n'
               f'Software Version           :{self.swversion}\n'
               f'Router Management Address:{self.ip_add}')
        return desc

class Switch(Router):
    def getdesc(self):
        '''return a formatted description of the switch'''
        desc = (f'Switch Model           :{self.model}\n'
               f'Software Version           :{self.swversion}\n'
               f'Switch Management Address:{self.ip_add}')
        return desc

rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
rtr2 = Router('isr4221', '16.9.5', '10.10.10.5')
sw1 = Switch('Cat9300', '16.9.5', '10.10.10.8')

print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
print('Rtr2\n', rtr2.getdesc(), '\n', sep='')
print('Sw1\n', sw1.getdesc(), '\n', sep='')
```

```
Rtr1
Router Model           :iosV
Software Version       :15.6.7
Router Management Address:10.10.10.1

Rtr2
Router Model           :isr4221
Software Version       :16.9.5
Router Management Address:10.10.10.5

Sw1
Switch Model           :Cat9300
Software Version       :16.9.5
Switch Management Address:10.10.10.8
```

```
>>> import math
>>> dir(math)
['_doc_', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees',
 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgam-
 ma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi',
 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt',
 'tan', 'tanh', 'tau', 'trunc']
```

```
>>> help(math)
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is automatically generated from the Python
    source files. It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations. When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.
-Snip for brevity-
```

```
>>> help(math.sqrt)
```

```
Help on built-in function sqrt in module math:
```

```
sqrt(x, /)
```

```
Return the square root of x.
```

```
>>> print(cal.month(2020, 2, 2, 1))
```

February 2020

```
Mo Tu We Th Fr Sa Su
```

```
          1  2
```

```
 3  4  5  6  7  8  9
```

```
10 11 12 13 14 15 16
```

```
17 18 19 20 21 22 23
```

```
24 25 26 27 28 29
```

```
>>> from math import sqrt,tan
```

```
>>> sqrt(15)
```

```
3.872983346207417
```

```
>>> import sys
```

```
>>> sys.path
```

```
['', '/Users/chrijack/Documents', '/Library/Frameworks/Python.  
framework/Versions/3.8/lib/python38.zip', '/Library/Frameworks/  
Python.framework/Versions/3.8/lib/python3.8', '/Library/Frameworks/  
Python.framework/Versions/3.8/lib/python3.8/lib-dynload', '/  
Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/  
site-packages']
```

```
from device import Router, Switch
```

```
rtr1 = Router('iosV', '15.6.7', '10.10.10.1')
```

```
rtr2 = Router('isr4221', '16.9.5', '10.10.10.5')
```

```
sw1 = Switch('Cat9300', '16.9.5', '10.10.10.8')
```

```
print('Rtr1\n', rtr1.getdesc(), '\n', sep='')
```

```
print('Rtr2\n', rtr2.getdesc(), '\n', sep='')
```

```
print('Sw1\n', sw1.getdesc(), '\n', sep='')
```



```
Rtr1
Router Model      :iosV
Software Version  :15.6.7
Router Management Address:10.10.10.1

Rtr2
Router Model      :isr4221
Software Version  :16.9.5
Router Management Address:10.10.10.5

Sw1
Switch Model      :Cat9300
Software Version  :16.9.5
Router Management Address:10.10.10.8
```

```
from netmiko import ConnectHandler
from nornir.core import InitNornir
pip install pyats (just installs the core framework, check
documentation for more options)
-----
```

```
readdata = open("textfile.txt", "r")
print(readdata.read())
```

Line one of a text file

Line two of a text file, just like line one, but the second one.

Third line of a text file.

```
with open("textfile.txt", "r") as data:
```

```
    print(data.read())
```

```
with open("textfile.txt", "a+") as data:
```

```
    data.write('\nFourth line added by Python')
```

```
with open ("textfile.txt", "r") as data:
```

```
    print(data.read())
```

Line one of a text file

Line two of a text file, just like line one, but the second one.

Third line of a text file.

Fourth line added by Python

```
"router1", "192.168.10.1", "Nashville"
```

```
"router2", "192.168.20.1", "Tampa"
```

```
"router3", "192.168.30.1", "San Jose"
```

```
>>> import csv
```

```
>>> samplefile = open('routerlist.csv')
```

```
>>> samplereader = csv.reader(samplefile)
```

```
>>> sampledata = list(samplereader)
```

```
>>> sampledata
```

```
[['router1', '192.168.10.1', 'Nashville'], ['router2',  
'192.168.20.1', 'Tampa'], ['router3', '192.168.30.1', 'San Jose ']]
```

```
>>> sampledata[0]
```

```
['router1', '192.168.10.1', 'Nashville']
```

```
>>> sampledata[0][1]
```

```
'192.168.10.1'
```

```
import csv
```

```
with open("routerlist.csv") as data:
```

```
    csv_list = csv.reader(data)
```

```
    for row in csv_list:
```

```
        device = row[0]
```

```
        location = row[2]
```

```
        ip = row[1]
```

```
        print(f"{device} is in {location.rstrip()} and has IP  
{ip}.")
```

router1 is in Nashville and has IP 192.168.10.1.

router2 is in Tampa and has IP 192.168.20.1.

router3 is in San Jose and has IP 192.168.30.1.

```
import csv

print("Please add a new router to the list")
hostname = input("What is the hostname? ")
ip = input("What is the ip address? ")
location = input("What is the location? ")

router = [hostname, ip, location]

with open("routerlist.csv", "a") as data:
    csv_writer = csv.writer(data)
    csv_writer.writerow(router)

<Below is interactive from the terminal after running the above code>
Please add a new router to the list
What is the hostname? router4
What is the ip address? 192.168.40.1
What is the location? London
```

router1 is in Nashville and has IP 192.168.10.1.

router2 is in Tampa and has IP 192.168.20.1.

router3 is in San Jose and has IP 192.168.30.1.

router4 is in London and has IP 192.168.40.1.

```
{
  "interface": {
    "name": "GigabitEthernet1",
    "description": "Router Uplink",
    "enabled": true,
    "ipv4": {
      "address": [
        {
          "ip": "192.168.1.1",
          "netmask": "255.255.255.0"
        }
      ]
    }
  }
}
```

```
import json
```

```
with open("json_sample.json") as data:
```

```
    json_data = data.read()
```

```
    json_dict = json.loads(json_data)
```

```

>>> type(json_dict)
<class 'dict'>

>>> print(json_dict)
{'interface': {'name': 'GigabitEthernet1', 'description':
'Router Uplink', 'enabled': True, 'ipv4': {'address':
[{'ip': '192.168.0.2', 'netmask': '255.255.255.0'}]}}}
>>> json_dict["interface"]["description"] = "Backup Link"
>>> print(json_dict)
{'interface': {'name': 'GigabitEthernet1', 'description': 'Backup
Link', 'enabled': True, 'ipv4': {'address': [{'ip': '192.168.0.2',
'netmask': '255.255.255.0'}]}}}
with open("json_sample.json", "w") as fh:
    json.dump(json_dict, fh, indent = 4)

```

```

>>> with open ("json_sample.json") as data:
    json_data = data.read()
    print(json_data)

{
  "interface": {
    "name": "GigabitEthernet1",
    "description": "Backup Link",
    "enabled": true,
    "ipv4": {
      "address": [
        {
          "ip": "192.168.0.2",
          "netmask": "255.255.255.0"
        }
      ]
    }
  }
}
>>>

```

```

<?xml version="1.0" encoding="UTF-8" ?>
<interface xmlns="ietf-interfaces">
  <name>GigabitEthernet2</name>
  <description>Wide Area Network</description>
  <enabled>true</enabled>
  <ipv4>
    <address>
      <ip>192.168.1.5</ip>
      <netmask>255.255.255.0</netmask>
    </address>
  </ipv4>
</interface>

```

```

import xmltodict

with open("xml_sample.xml") as data:
    xml_example = data.read()

xml_dict = xmltodict.parse(xml_example)

>>> print(xml_dict)
OrderedDict([('interface', OrderedDict([('xmlns', 'ietf-interfaces'), ('name',
'GigabitEthernet2'), ('description', 'Wide Area Network'), ('enabled', 'true'),
('ipv4', OrderedDict([('address', OrderedDict([('ip', '192.168.0.2'), ('netmask',
'255.255.255.0')]))]))]))])

```

```

xml_dict["interface"]["ipv4"]["address"]["ip"] = "192.168.55.3"

```

```
>>> print(xmltodict.unparse(xml_dict, pretty=True))
<?xml version="1.0" encoding="utf-8"?>
<interface xmlns="ietf-interfaces">
  <name>GigabitEthernet2</name>
  <description>Wide Area Network</description>
  <enabled>true</enabled>
  <ipv4>
    <address>
      <ip>192.168.55.3</ip>
      <netmask>255.255.255.0</netmask>
    </address>
  </ipv4>
</interface>
```

```
with open("xml_sample.xml", "w") as data:
```

```
    data.write(xmltodict.unparse(xml_dict, pretty=True))
```

```
interface:
```

```
    name: GigabitEthernet2
```

```
    description: Wide Area Network
```

```
    enabled: true
```

```
    ipv4:
```

```
        address:
```

```
            - ip: 172.16.0.2
```

```
            netmask: 255.255.255.0
```

```
addresses:
```

```
    - ip: 172.16.0.2
```

```
      netmask: 255.255.255.0
```

```
    - ip: 172.16.0.3
```

```
      netmask: 255.255.255.0
```

```
    - ip: 172.16.0.4
```

```
      netmask: 255.255.255.0
```

```
import yaml

with open("yaml_sample.yaml") as data:
    yaml_sample = data.read()

yaml_dict = yaml.load(yaml_sample, Loader=yaml.FullLoader)
>>> type(yaml_dict)
<class 'dict'>
```

```
>>> yaml_dict
{'interface': {'name': 'GigabitEthernet2', 'description': 'Wide Area Network', 'enabled': True, 'ipv4': {'address': [{'ip': '192.168.0.2', 'netmask': '255.255.255.0'}]}}}
```

```
>>> yaml_dict["interface"]["name"] = "GigabitEthernet1"

>>> print(yaml.dump(yaml_dict, default_flow_style=False))
interface:
  description: Wide Area Network
  enabled: true
  ipv4:
    address:
      - ip: 192.168.0.2
        netmask: 255.255.255.0
  name: GigabitEthernet1
```

```
with open("yaml_sample.yaml", "w") as data:
    data.write(yaml.dump(yaml_dict, default_flow_style=False))
```

```
x = 0
while True:
    try:
        filename = input("Which file would you like to open? :")
        with open(filename, "r") as fh:
            file_data = fh.read()
    except FileNotFoundError:
        print(f'Sorry, {filename} doesn't exist! Please try again.')
    else:
        print(file_data)

    x = 0
    break
finally:
    x += 1
    if x == 3:
        print('Wrong filename 3 times.\nCheck name and Rerun.')
        break
```

```
Which file would you like to open? :test
Sorry, test doesn't exist! Please try again.
Which file would you like to open? :test.txt
Test file with some text.
Two lines long.
-----
```

```
Which file would you like to open? :test
Sorry, test doesn't exist! Please try again.
Which file would you like to open? :test2
Sorry, test2 doesn't exist! Please try again.
Which file would you like to open? :test3
Sorry, test3 doesn't exist! Please try again.
Wrong filename 3 times.
```

```
Check name and Rerun.
import unittest
```

```
from areacircle import area_of_circle
```

```
from math import pi
```

```
class Test_Area_of_Circle_input(unittest.TestCase):
    def test_area(self):
        # Test radius >= 0
        self.assertAlmostEqual(area_of_circle(1), pi)
        self.assertAlmostEqual(area_of_circle(0), 0)
        self.assertAlmostEqual(area_of_circle(3.5), pi * 3.5**2)
```

```
-----
Ran 1 test in 0.000s
```

```
OK
test_values(self):
    # Test that bad values are caught
    self.assertRaises(ValueError, area_of_circle, -1)
```

```
.F
-----
FAIL: test_values (__main__.Test_Area_of_Circle_input)
-----
Traceback (most recent call last):
  File "/Users/chrijack/Documents/ccnadevnet/test_areacircle.py", line 14, in
  test_values
    self.assertRaises(ValueError, area_of_circle, -1)
AssertionError: ValueError not raised by area_of_circle
-----
Ran 2 tests in 0.001s
FAILED (failures=1)
```

```
from math import pi
```

```
def area_of_circle(r):
    if r < 0:
        raise ValueError('Negative radius value error')
    return pi*(r**2)
-----
```

```
>>> area_of_circle(-1)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
  File "/Users/chrijack/Documents/ccnadevnet/areacircle.py",
    line 5, in area_of_circle
    raise ValueError('Negative radius value error')
ValueError: Negative radius value error
..
-----
Ran 2 tests in 0.000s

OK
-----
```



```
GET /something?api_key=abcdef12345
```

```
GET /something HTTP/1.1
```

```
Cookie: X-API-KEY=abcdef12345
```

```
POST /InStock HTTP/1.1
```

```
Host: www.example.org
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: 299
```

```
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
```

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://
www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>CSCO</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.example.org/timeouts"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <env:Body>
  <env:Fault>
    <env:Code>
      <env:Value>env:Sender</env:Value>
      <env:Subcode>
        <env:Value>m:MessageTimeout</env:Value>
      </env:Subcode>
    </env:Code>
    <env:Reason>
      <env:Text xml:lang="en">Sender Timeout</env:Text>
    </env:Reason>
    <env:Detail>
      <m:MaxTime>P5M</m:MaxTime>
    </env:Detail>
  </env:Fault>
</env:Body>
</env:Envelope>
```

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>21</i4></value>
    </param>
  </params>
</methodCall>
```

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Illinois</string></value>
    </param>
  </params>
</methodResponse>
```

request-header-name: request-header-value1, request-header-value2, ...
Host: myhouse.cisco.com

Connection: Keep-Alive

Accept: image/gif, image/jpeg, */*

Accept-Language: us-en, fr, cn

response-header-name: response-header-value1, response-header-value2, ...
Content-Type: text/html

Content-Length: 35

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

The response message body contains the resource data requested.

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <home>this is my house</home>
  <home>located in San Jose, CA</home>
  <rooms>
    <living_room>true</living_room>
    <kitchen>false</kitchen>
    <study_room>
      <size>20x30</size>
    </study_room>
    <study_room>
      <desk>true</desk>
    </study_room>
    <study_room>
      <lights>On</lights>
    </study_room>
  </rooms>
</root>
```

returns the devices between 100-115

returns the devices 0 to 200

```
GET /devices?offset=100&limit=10
{
  "pagination": {
    "offset": 100,
    "limit": 10,
    "total": 220,
  },
  "device": [
    //...
  ],
  "links": {
    "next": "http://myhouse.cisco.com/devices?offset=110&limit=10",
    "prev": "http://myhouse.cisco.com/devices?offset=90&limit=10"
  }
}
```

```
$ curl -sD - https://postman-echo.com/get?test=123
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 04:59:34 GMT
ETag: W/"ca-42Kz98xXW2nwFREN74xZNS6JeJk"
Server: nginx
set-cookie: sails.sid=s%3AxZUPHE3Ojk1yts3qrUFqTj_MzBQZZR5n.NrjPkNm0Wp1J7%2F%2BX907VU
TFpKHpJySLzBytRbnlzYCW; Path=/; HttpOnly
Vary: Accept-Encoding
Content-Length: 202
Connection: keep-alive

{"args":{"test":"123"},"headers":{"x-forwarded-proto":"https","host":"postman-echo.com","accept":"*/*","user-agent":"curl/7.54.0","x-forwarded-port":"443"},"url":"https://postman-echo.com/get?test=123"}
```

```
$ curl -sD - -X POST https://postman-echo.com/post -H 'cache-control: no-cache' -H 'content-type: text/plain' -d 'hello DevNet'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 05:16:58 GMT
ETag: W/"13a-01MLfkx17vDVWfb06pyVxdZlaug"
Server: nginx
set-cookie: sails.sid=s%3AwiFXmSNJpzY0ONduxUCAE8IodwNg9Z2Y.j%2BJ5%2B0mch8XEq8j01vzH8
kjNBi8ecJijlrGT8DlnBhE; Path=/; HttpOnly
Vary: Accept-Encoding
Content-Length: 314
Connection: keep-alive

{"args":{},"data":"hello DevNet","files":{},"form":{},"headers":{"x-forwarded-proto":"https","host":"postman-echo.com","content-length":"12","accept":"*/*","cache-control":"no-cache","content-type":"text/plain","user-agent":"curl/7.54.0","x-forwarded-port":"443"},"json":null,"url":"https://postman-echo.com/post"}
```

```
$ curl -sD - -X GET https://postman-echo.com/basic-auth -H 'authorization: Basic cG9zdG1hbGpwYXNzd29yZA==' -H 'cache-control: no-cache'
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 05:21:00 GMT
ETag: W/"16-sJz8uwjdDv0wvm7//BYdNw8vMbU"
Server: nginx
set-cookie: sails.sid=s%3A4i3UW5-DQCmpey8Z1Ayrqq0izt4KZR5-.B18QDnt44B690E8J06qyC-
s8oyCLpUfEsFxlEFTSWSC4; Path=/; HttpOnly
Vary: Accept-Encoding
Content-Length: 22
Connection: keep-alive
{"authenticated":true}
```

```
$ http https://postman-echo.com/get?test=123
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 179
Content-Type: application/json; charset=utf-8
Date: Tue, 27 Aug 2019 05:27:17 GMT
ETag: W/"ed-mB0Pm0M3ExozL3fgwq7U1H9aozQ"
Server: nginx
Vary: Accept-Encoding
set-cookie: sails.sid=s%3AYCeNAWJG7Kap5wvKpg8HYlZI5SHZogEf.r7Gi96fe5g7%2FSp0jaJK%2FaVRpHZp3Oj5tDxiM8TPZ%2Bpc; Path=/; HttpOnly

{
  "args": {
    "test": "123"
  },
  "headers": {
    "accept": "*/*",
    "accept-encoding": "gzip, deflate",
    "host": "postman-echo.com",
    "user-agent": "HTTPIe/1.0.2",
    "x-forwarded-port": "443",
    "x-forwarded-proto": "https"
  },
  "url": "https://postman-echo.com/get?test=123"
}
```

```
import requests
url = "https://postman-echo.com/get"
querystring = {"test": "123"}
headers = {}
response = requests.request("GET", url, headers=headers, params=querystring)
print(response.text)
```

```
import requests
url = "https://postman-echo.com/post"
payload = "hello DevNet"
headers = {'content-type': 'text/plain'}
response = requests.request("POST", url, data=payload, headers=headers)
print(response.text)
```

```
import requests
url = "https://postman-echo.com/basic-auth"
headers = {
  'authorization': "Basic cG9zdGhhbjpwYXNzd29yZA=="
}
response = requests.request("GET", url, headers=headers)
print(response.text)
```

```
curl -I -X GET \  
  --url 'https://api.meraki.com/api/v0/organizations' \  
  -H 'X-Cisco-Meraki-API-Key: 15da0c6ffff295f16267f88f98694cf29a86ed87'
```

```
HTTP/1.1 302 Found  
Server: nginx  
Date: Sat, 17 Aug 2019 19:05:25 GMT  
Content-Type: text/html; charset=utf-8  
Transfer-Encoding: chunked  
Connection: keep-alive  
Cache-Control: no-cache, no-store, max-age=0, must-revalidate  
Pragma: no-cache  
Expires: Fri, 01 Jan 1990 00:00:00 GMT  
X-Frame-Options: sameorigin  
X-Robots-Tag: none  
Location: https://n149.meraki.com/api/v0/organizations  
X-UA-Compatible: IE=Edge,chrome=1  
X-Request-Id: 87654dbd16ae23fbc7e3282a439b211c  
X-Runtime: 0.320067  
Strict-Transport-Security: max-age=15552000; includeSubDomains
```

```
curl -X GET \  
  --url 'https://n149.meraki.com/api/v0/organizations' \  
  -H 'X-Cisco-Meraki-API-Key:  
15da0c6ffff295f16267f88f98694cf29a86ed87' \  
  -H 'Accept: application/json'
```

```
[  
  
  {  
  
    "name" : "DevNet Sandbox",  
  
    "id" : "549236"  
  
  }  
  
]
```

```
curl -X GET \  
  --url 'https://n149.meraki.com/api/v0/organizations/549236/  
networks' \  
  -H 'X-Cisco-Meraki-API-Key: 15da0c6ffff295f16267f88f98694c-  
f29a86ed87' \  
  -H 'Accept: application/json'
```

```
[
  {
    "timeZone" : "America/Los_Angeles",
    "tags" : " Sandbox ",
    "organizationId" : "549236",
    "name" : "DevNet Always On Read Only",
    "type" : "combined",
    "disableMyMerakiCom" : false,
    "disableRemoteStatusPage" : true,
    "id" : "L_646829496481099586"
  },
  {
    "organizationId" : "549236",
    "tags" : null,
    "timeZone" : "America/Los_Angeles",
    "id" : "N_646829496481152899",
    "disableRemoteStatusPage" : true,
    "name" : "test - mx65",
    "disableMyMerakiCom" : false,
    "type" : "appliance"
  }, ... omitted output
],
```

```
curl -X GET \
--url 'https://n149.meraki.com/api/v0/networks/L_646829496481099586/
devices' \
-H 'X-Cisco-Meraki-API-Key: 15da0c6ffff295f16267f88f98694cf29a86ed87'\
-H 'Accept: application/json'
```

```
[
  {
    "wan2Ip" : null,
    "networkId" : "L_646829496481099586",
    "lanIp" : "10.10.10.106",
    "serial" : "QYYY-WWW-ZZZZ",
    "tags" : " recently-added ",
    "lat" : 37.7703718,
    "lng" : -122.3871248,
    "model" : "MX65",
    "mac" : "e0:55:3d:17:d4:23",
    "wan1Ip" : "10.10.10.106",
    "address" : "500 Terry Francois, San Francisco"
  },
  {
    "switchProfileId" : null,
    "address" : "",
    "lng" : -122.098531723022,
    "model" : "MS220-8P",
    "mac" : "88:15:44:df:f3:af",
    "tags" : " recently-added ",
    "serial" : "QAAA-BBBB-CCCC",
    "networkId" : "L_646829496481099586",
    "lanIp" : "192.168.128.2",
    "lat" : 37.4180951010362
  }
]
```

```

#!/usr/bin/env python
from meraki_sdk.meraki_sdk_client import MerakiSdkClient

#Cisco DevNet Sandbox Meraki API key
X_CISCO_MERAKI_API_KEY = '15da0c6ffff295f16267f88f98694cf29a86ed87'

#Establish a new client connection to the Meraki REST API
MERAKEI = MerakiSdkClient(X_CISCO_MERAKI_API_KEY)

#Get a list of all the organizations for the Cisco DevNet account
ORGS = MERAKEI.organizations.get_organizations()
for ORG in ORGS:
    print("Org ID: {} and Org Name: {}".format(ORG['id'], ORG['name']))

PARAMS = {}
PARAMS["organization_id"] = "549236" # Demo Organization "DevNet Sandbox"

#Get a list of all the networks for the Cisco DevNet organization
NETS = MERAKEI.networks.get_organization_networks(PARAMS)
for NET in NETS:
    print("Network ID: {0:20s}, Name: {1:45s},Tags: {2:<10s}".format(\
        NET['id'], NET['name'], str(NET['tags'])))

#Get a list of all the devices that are part of the Always On Network
DEVICES = MERAKEI.devices.get_network_devices("L_646829496481099586")
for DEVICE in DEVICES:
    print("Device Model: {0:9s},Serial: {1:14s},MAC: {2:17}, Firmware:{3:12s}"\
        .format(DEVICE['model'], DEVICE['serial'], DEVICE['mac'], \
            DEVICE['firmware']))

```

```

curl -X POST \
    https://sandboxdnac2.cisco.com/dna/system/api/v1/auth/token \
    -H 'Authorization: Basic ZGV2bmV0dXNlcjpdXjZyYmE='
{"Token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Y2U3MTJiMDhlZTY2MjAyZmEyZWl4ZjgiLCJhdXRoU291cmNlIjoiaW50ZXJyZWwiLCJ0ZW5hbnROYWw1IjoiVE5UMCIiInJvbGVzIjpbIjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLCJ0ZW5hbnRJCi6IjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyZmYzImV4cCI6MTU2NjU0Mzk3OCwidXNlcm5hbWUiOiJkZXZuZXRlc2VyIn0.Qv6vU6d1tqFGx9GETj6S1Da8Ts6uJNk9624onLSNSnU"}
{
  "Token":
  "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Y2U3MTJiMDhlZTY2MjAyZmEyZWl4ZjgiLCJhdXRoU291cmNlIjoiaW50ZXJyZWwiLCJ0ZW5hbnROYWw1IjoiVE5UMCIiInJvbGVzIjpbIjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLCJ0ZW5hbnRJCi6IjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyZmYzImV4cCI6MTU2NjU0Mzk3OCwidXNlcm5hbWUiOiJkZXZuZXRlc2VyIn0.ubXSmZYrI-yoCWmzCSY486y-HWhwdTlnrrWqYip5lv6Y"
}
curl -X GET \
    https://sandboxdnac2.cisco.com/dna/intent/api/v1/network-device \
    -H 'X-Auth-Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI1Y2U3MTJiMDhlZTY2MjAyZmEyZWl4ZjgiLCJhdXRoU291cmNlIjoiaW50ZXJyZWwiLCJ0ZW5hbnROYWw1IjoiVE5UMCIiInJvbGVzIjpbIjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyJdLCJ0ZW5hbnRJCi6IjVlNmNmZGZmNDMwOTkwMDA4OWYwZmYzNyZmYzImV4cCI6MTU2NjU0Mzk3OCwidXNlcm5hbWUiOiJkZXZuZXRlc2VyIn0.YXc_2o8FDzSQLYBhUxUIoxwzYXXWYeNJRkBoKBlIHI'

```



```
{
  "response" : [ {
    "siteId" : "global",
    "scoreDetail" : [ {
      "scoreCategory" : {
        "scoreCategory" : "CLIENT_TYPE",
        "value" : "ALL"
      },
      "scoreValue" : 27,
      "clientCount" : 82,
      "clientUniqueCount" : 82,
      "starttime" : 1566506189000,
      "endtime" : 1566506489000,
      "scoreList" : [ ]
    }, ... output omitted
  ]
}
```

```
#!/usr/bin/env python
from dnacentersdk import api

# Create a DNACenterAPI connection object;
# it uses DNA Center sandbox URL, username and password
DNAC = api.DNACenterAPI(username="devnetuser",
    password="Cisco123!",
    base_url="https://sandboxdnac2.cisco.com")

# Find all devices
DEVICES = DNAC.devices.get_device_list()

# Print select information about the retrieved devices
print('{0:25s}{1:1}{2:45s}{3:1}{4:15s}'.format("Device Name", "|", \
    "Device Type", "|", "Up Time"))
print('-'*95)
for DEVICE in DEVICES.response:
    print('{0:25s}{1:1}{2:45s}{3:1}{4:15s}'.format(DEVICE.hostname, \
        "|", DEVICE.type, "|", DEVICE.upTime))
print('-'*95)

# Get the health of all clients on Thursday, August 22, 2019 8:41:29 PM GMT
CLIENTS = DNAC.clients.get_overall_client_health(timestamp="1566506489000")

# Print select information about the retrieved client health statistics
print('{0:25s}{1:1}{2:45s}{3:1}{4:15s}'.format("Client Category", "|", \
    "Number of Clients", "|", "Clients Score"))
print('-'*95)
for CLIENT in CLIENTS.response:
    for score in CLIENT.scoreDetail:
        print('{0:25s}{1:1}{2:<45d}{3:1}{4:<15d}'.format(
            score.scoreCategory.value, "|", score.clientCount, "|", \
            score.scoreValue))
print('-'*95)
```

```
curl -c - -X POST -k \

    https://sandboxsdwan.cisco.com:8443/j_security_check \

    -H 'Content-Type: application/x-www-form-urlencoded' \

    -d 'j_username=devnetuser&j_password=Cisco123!'
# Netscape HTTP Cookie File
# https://curl.haxx.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.
#HttpOnly_sandboxsdwan.cisco.com. FALSE / TRUE 0 JSESSIONID.
v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.4854266f-a8ad-4068-9651-
d4e834384f51
curl -X GET -k \

    https://sandboxsdwan.cisco.com:8443/dataservice/device \

    -H 'Cookie: JSESSIONID=v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.4
854266f-a8ad-4068-9651-d4e834384f51'
```

```

{
  ... omitted output
  "data" : [
    {
      "state" : "green",
      "local-system-ip" : "4.4.4.90",
      "status" : "normal",
      "latitude" : "37.666684",
      "version" : "18.3.1.1",
      "model_sku" : "None",
      "connectedVManages" : [
        "\"4.4.4.90\""
      ],
      "statusOrder" : 4,
      "uuid" : "4854266f-a8ad-4068-9651-d4e834384f51",
      "deviceId" : "4.4.4.90",
      "reachability" : "reachable",
      "device-groups" : [
        "\"No groups\""
      ],
      "total_cpu_count" : "2",
      "certificate-validity" : "Valid",
      "board-serial" : "01",
      "platform" : "x86_64",
      "device-os" : "next",
      "timezone" : "UTC",
      "uptime-date" : 1567111260000,
      "host-name" : "vmanage",
      "device-type" : "vmanage",
      "personality" : "vmanage",
      "domain-id" : "0",
      "isDeviceGeoData" : false,
      "lastupdated" : 1567470387553,
      "site-id" : "100",
      "controlConnections" : "5",
      "device-model" : "vmanage",
      "validity" : "valid",
      "system-ip" : "4.4.4.90",
      "state_description" : "All daemons up",
      "max-controllers" : "0",
      "layoutLevel" : 1,
      "longitude" : "-122.777023"
    },
    ... omitted output
  ]
}

```

```
curl -X GET -k \
```

```

https://sandboxsdwan.cisco.com:8443/dataservice/template/device \
-H 'Cookie: JSESSIONID=v9QcTVL_ZBdIQZRsI2V95vBi7Bz47IMxRY3XAYA6.48
54266f-a8ad-4068-9651-d4e834384f51'

```

```

{
  "data" : [
    {
      "templateDescription" : "VEDGE BASIC TEMPLATE01",
      "lastUpdatedOn" : 1538865915509,
      "templateAttached" : 15,
      "deviceType" : "vedge-cloud",
      "templateId" : "72babaf2-68b6-4176-92d5-fa8de58e19d8",
      "configType" : "template",
      "devicesAttached" : 0,
      "factoryDefault" : false,
      "templateName" : "VEDGE_BASIC_TEMPLATE",
      "lastUpdatedBy" : "admin"
    },
    ... output omitted
  ]
}

```

```

#!/usr/bin/env python
import json
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

# Specify Cisco vManage IP, username and password
VMANAGE_IP = 'sandboxswan.cisco.com'
USERNAME = 'devnetuser'
PASSWORD = 'Cisc0123!'

BASE_URL_STR = 'https://{}:8443/'.format(VMANAGE_IP)

# Login API resource and login credentials
LOGIN_ACTION = '_j_security_check'
LOGIN_DATA = {'j_username': USERNAME, 'j_password': PASSWORD}

# URL for posting login data
LOGIN_URL = BASE_URL_STR + LOGIN_ACTION

# Establish a new session and connect to Cisco vManage
SESS = requests.session()
LOGIN_RESPONSE = SESS.post(url=LOGIN_URL, data=LOGIN_DATA, verify=False)

# Get list of devices that are part of the fabric and display them
DEVICE_RESOURCE = 'dataservice/device'
# URL for device API resource
DEVICE_URL = BASE_URL_STR + DEVICE_RESOURCE

DEVICE_RESPONSE = SESS.get(DEVICE_URL, verify=False)
DEVICE_ITEMS = json.loads(DEVICE_RESPONSE.content)['data']

```

```

print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}{6:16s}{7:1}{8:7s}'\
      .format("Host-Name", "|", "Device Model", "|", "Device ID", \
            "|", "System IP", "|", "Site ID"))
print('-'*105)

for ITEM in DEVICE_ITEMS:
    print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}{6:16s}{7:1}{8:7s}'\
          .format(ITEM['host-name'], "|", ITEM['device-model'], "|", \
                ITEM['uuid'], "|", ITEM['system-ip'], "|", ITEM['site-id']))
print('-'*105)

# Get list of device templates and display them
TEMPLATE_RESOURCE = 'dataservice/template/device'
# URL for device template API resource
TEMPLATE_URL = BASE_URL_STR + TEMPLATE_RESOURCE

TEMPLATE_RESPONSE = SESS.get(TEMPLATE_URL, verify=False)
TEMPLATE_ITEMS = json.loads(TEMPLATE_RESPONSE.content)['data']

print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}{6:16s}{7:1}{8:7s}'\
      .format("Template Name", "|", "Device Model", "|", "Template ID", \
            "|", "Attached devices", "|", "Template Version"))
print('-'*105)

for ITEM in TEMPLATE_ITEMS:
    print('{0:20s}{1:1}{2:12s}{3:1}{4:36s}{5:1}{6:<16d}{7:1}{8:<7d}'\
          .format(ITEM['templateName'], "|", ITEM['deviceType'], "|", \
                ITEM['templateId'], "|", ITEM['devicesAttached'], "|", \
                ITEM['templateAttached']))
print('-'*105)

```

```
curl -k -X POST \  
https://sandboxapicdc.cisco.com/api/aaaLogin.json \  
-d '{  
  "aaaUser" : {  
    "attributes" : {  
      "name" : "admin",  
      "pwd" : "ciscopsdt"  
    }  
  }  
}'
```

```
{  
  "totalCount" : "1",  
  "imdata" : [  
    {  
      "aaaLogin" : {  
        "attributes" : {  
          "remoteUser" : "false",  
          "firstLoginTime" : "1572128727",  
          "version" : "4.1(1k)",  
          "buildTime" : "Mon May 13 16:27:03 PDT 2019",  
          "siteFingerprint" : "Z29SSG/BAVPY04Vv",  
          "guiIdleTimeoutSeconds" : "1200",  
          "firstName" : "",  
          "userName" : "admin",  
          "refreshTimeoutSeconds" : "600",  
          "restTimeoutSeconds" : "90",  
          "node" : "topology/pod-1/node-1",  
          "creationTime" : "1572128727",  
          "changePassword" : "no",  
          "token" : "pRgAAAAAAAAAAAAAAAAAGNPF39fZd71fV6DJWidJoxqJmHt1Fephm-  
w6Q0I5byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVRFZrUCeggsPg8+dbjb8kWX02FJLcw9Qp  
sg98s5QfOaMDQWHSyqw0ObKOGxxglLeQbkgxM8/fgOAFZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA==",  
          "unixUserId" : "15374",  
          "lastName" : "",  
          "sessionId" : "1IZw4uthRVSmYWWH/+S9aA==",  
          "maximumLifetimeSeconds" : "86400"  
        }  
      }  
      ...omitted output  
    }  
  ]  
}
```

```
curl -k -X GET \  
https://sandboxapicdc.cisco.com/api/node/class/fabricPod.json \  
-H 'Cookie: APIC-cookie=pRgAAAAAAAAAAAAAAAAAGNPF39fZd71fV6DJWidJoxqJmHt1Fephmw6Q0  
I5byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVRFZrUCeggsPg8+dbjb8kWX02FJLcw9Qp  
sg98s5QfOaMDQWHSyqw0ObKOGxxglLeQbkgxM8/fgOAFZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA='
```

```
{  
  "totalCount" : "1",  
  "imdata" : [  
    {  
      "fabricPod" : {  
        "attributes" : {  
          "id" : "1",  
          "monPolDn" : "uni/fabric/monfab-default",  
          "dn" : "topology/pod-1",  
          "status" : "",  
          "childAction" : "",  
          "modTs" : "2019-10-26T18:01:13.491+00:00",  
          "podType" : "physical",  
          "lcOwn" : "local"  
        }  
      }  
    }  
  ]  
}
```

```
curl -k -X GET \
https://sandboxapicdc.cisco.com/api/node/class/topology/pod-1/topSystem.json \
-H 'Cookie: APIC-cookie=pRgAAAAAAAAAAAAAAAAAGNPF39fZd71fV6DJWidJoqxJmHt1Fepmw6Q0
15byoafVMZ29a6pL+4u5krJ0G2Jdrv101219cMx/o0ciIbVrfFZruCEgqsPg8+dbjBkWX02FJLcw9Qpsg
98s5QfOaMDQWHSyqWOObKOGxglLeQbkqM8/fgOAPZxbKHMw0+09ihdiu7jTb7AAJVZEzYzXA=='
```

```
{
  "imdata" : [
    {
      "topSystem" : {
        "attributes" : {
          "role" : "controller",
          "name" : "apic1",
          "fabricId" : "1",
          "inbMgmtAddr" : "192.168.11.1",
          "oobMgmtAddr" : "10.10.20.14",
          "systemUpTime" : "00:04:33:38.000",
          "siteId" : "0",
          "state" : "in-service",
          "fabricDomain" : "ACI Fabric1",
          "dn" : "topology/pod-1/node-1/sys",
          "podId" : "1"
        }
      }
    },
    {
      "topSystem" : {
        "attributes" : {
          "state" : "in-service",
          "siteId" : "0",
          "address" : "10.0.80.64",
          "fabricDomain" : "ACI Fabric1",
          "dn" : "topology/pod-1/node-101/sys",
          "id" : "101",
          "podId" : "1",
          "role" : "leaf",
          "fabricId" : "1",
          "name" : "leaf-1"
        }
      }
    }
  ]
}
```

```
},
{
  "topSystem" : {
    "attributes" : {
      "podId" : "1",
      "id" : "102",
      "dn" : "topology/pod-1/node-102/sys",
      "address" : "10.0.80.66",
      "fabricDomain" : "ACI Fabric1",
      "siteId" : "0",
      "state" : "in-service",
      "role" : "leaf",
      "name" : "leaf-2",
      "fabricId" : "1"
    }
  }
},
{
  "topSystem" : {
    "attributes" : {
      "fabricId" : "1",
      "name" : "spine-1",
      "role" : "spine",
      "podId" : "1",
      "id" : "201",
      "dn" : "topology/pod-1/node-201/sys",
      "state" : "in-service",
      "siteId" : "0",
      "fabricDomain" : "ACI Fabric1",
      "address" : "10.0.80.65"
    }
  }
}
],
"totalCount" : "4"
}
```

```

#!/usr/bin/env python
import sys
import acitoolkit.acitoolkit as aci

APIC_URL = 'https://sandboxapicdc.cisco.com'
USERNAME = 'admin'
PASSWORD = 'ciscopsdt'

# Login to APIC
SESSION = aci.Session(APIC_URL, USERNAME, PASSWORD)
RESP = SESSION.login()
if not RESP.ok:
    print('Could not login to APIC')
    sys.exit()

ENDPOINTS = aci.Endpoint.get(SESSION)
print('{0:19s}{1:14s}{2:10s}{3:8s}{4:17s}{5:10s}'.format(
    "MAC ADDRESS",
    "IP ADDRESS",
    "ENCAP",
    "TENANT",
    "APP PROFILE",
    "EPG"))
print('-'*80)

for EP in ENDPOINTS:
    epg = EP.get_parent()
    app_profile = epg.get_parent()
    tenant = app_profile.get_parent()
    print('{0:19s}{1:14s}{2:10s}{3:8s}{4:17s}{5:10s}'.format(
        EP.mac,
        EP.ip,
        EP.encap,
        tenant.name,
        app_profile.name,
        epg.name))

```

```

curl -k -X POST https://10.10.20.110/nuova \
    -H 'Content-Type: application/xml' \
    -d '<aaaLogin inName="ucspe" inPassword="ucspe"></aaaLogin>'
<aaaLogin cookie="" response="yes" outCookie="1573019916/7c901636-
c461-487e-bbd0-c74cd68c27be" outRefreshPeriod="600"
outPriv="aaa,admin,ext-lan-config,ext-lan-policy,ext-lan-
qos,ext-lan-security,ext-san-config,ext-san-policy,ext-san-
security,fault,operations,pod-config,pod-policy,pod-qos,pod-
security,read-only" outDomains="org-root" outChannel="noencssl"
outEvtChannel="noencssl" outSessionId="" outVersion="3.2(2.5)"
outName="" />
curl -k -X POST https://10.10.20.110/nuova \
    -H 'Content-Type: application/xml' \
    -d '<configFindDnsByClassId
classId="computeItem"
cookie="1573019916/7c901636-c461-487e-bbd0-c74cd68c27be" />'
-----

```

```
<configFindDnsByClassId cookie="1573019916/7c901636-c461-487e-bbd0-c74cd68c27be"
response="yes" classId="computeItem">
  <outDns>
    <dn value="sys/chassis-4/blade-8"/>
    <dn value="sys/chassis-5/blade-8"/>
    <dn value="sys/chassis-6/blade-8"/>
    <dn value="sys/chassis-6/blade-1"/>
    <dn value="sys/chassis-3/blade-1"/>
    ... omitted output
    <dn value="sys/rack-unit-9"/>
    <dn value="sys/rack-unit-8"/>
    <dn value="sys/rack-unit-7"/>
    <dn value="sys/rack-unit-6"/>
    <dn value="sys/rack-unit-5"/>
    <dn value="sys/rack-unit-4"/>
    <dn value="sys/rack-unit-3"/>
    <dn value="sys/rack-unit-2"/>
    <dn value="sys/rack-unit-1"/>
  </outDns>
</configFindDnsByClassId>
```

```
curl -k -X POST https://10.10.20.110/nuova \
-H 'Content-Type: application/xml' \
-d '<configResolveDn
  cookie="1573019916/7c901636-c461-487e-bbd0-c74cd68c27be"
  dn="sys/chassis-4/blade-8" />'
```

```
<configResolveDn dn="sys/chassis-4/blade-8" cookie="1573019916/7c901636-c461-487e-bbd0-
c74cd68c27be" response="yes">
  <outConfig>
    <computeBlade adminPower="policy" adminState="in-service" assetTag=""
    assignedToDn=""
    association="none" availability="available" availableMemory="49152"
    chassisId="4"
    checkpoint="discovered" connPath="A,B" connStatus="A,B" descr=""
    discovery="complete"
    discoveryStatus="" dn="sys/chassis-4/blade-8" fltAggr="0" fsmDescr=""
    fsmFlags=""
    fsmPrev="DiscoverSuccess" fsmProgr="100" fsmRmtInvErrCode="none"
    fsmRmtInvErrDescr=""
    fsmRmtInvRslt="" fsmStageDescr="" fsmStamp="2019-11-06T04:02:03.896"
    fsmStatus="nop"
    fsmTry="0" intId="64508" kmipFault="no" kmipFaultDescription=""
    lc="undiscovered"
    lctTs="1970-01-01T00:00:00.000" localId="" lowVoltageMemory="not-applicable"
    managingInst="A" memorySpeed="not-applicable" mfgTime="not-applicable"
    model="UCSB-
    B200-M4" name="" numOf40GAdaptorsWithOldPw="0"
    numOf40GAdaptorsWithUnknownPw="0"
    numOfAdaptors="1" numOfCores="8" numOfCoresEnabled="8" numOfCpus="2"
    numOfEthHostIfs="0" numOfFcHostIfs="0" numOfThreads="16" operPower="off"
    operPwrTransSrc="unknown" operQualifier="" operSolutionStackType="none"
    operState="unassociated" operability="operable"
    originalUuid="1b4e28ba-2fa1-11d2-
    0408-b9a761bde3fb" partNumber="" policyLevel="0" policyOwner="local"
    presence="equipped" revision="0" scaledMode="none" serial="SRV137"
    serverId="4/8"
    slotId="8" totalMemory="49152" usrLbl=""
    uuid="1b4e28ba-2fa1-11d2-0408-b9a761bde3fb"
    vendor="Cisco Systems Inc" vid=""/>
  </outConfig>
</configResolveDn>
```



```
#!/usr/bin/env python
from ucsmSdk.ucshandle import UcsHandle
HANDLE = UcsHandle("10.10.20.110", "ucspe", "ucspe")

# Login into Cisco UCS Manager
HANDLE.login()

# Retrieve all the compute blades
BLADES = HANDLE.query_classid("ComputeBlade")

print('{0:23s}{1:8s}{2:12s}{3:14s}{4:6s}'.format(
    "DN",
    "SERIAL",
    "ADMIN STATE",
    "MODEL",
    "TOTAL MEMORY"))
print('-'*70)

# Extract DN, serial number, admin state,
# model, and total memory for each blade
for BLADE in BLADES:
    print('{0:23s}{1:8s}{2:12s}{3:14s}{4:6s}'.format(
        BLADE.dn,
        BLADE.serial,
        BLADE.admin_state,
        BLADE.model,
        BLADE.total_memory))

HANDLE.logout()
```

```
curl -k -L -X GET \
-g 'https://10.10.10.66/app/api/rest?formatType=json&opName=userAPIGetMyLoginProfile&opData={}' \
-H 'X-Cloupia-Request-Key: 8187C34017C3479089C66678F32775FE'
```

```
{
  "opName" : "userAPIGetMyLoginProfile",
  "serviceName" : "InfraMgr",
  "serviceResult" : {
    "email" : null,
    "groupName" : null,
    "role" : "Admin",
    "userId" : "admin",
    "groupId" : 0,
    "firstName" : null,
    "lastName" : null
  },
  "serviceError" : null
}
```

```
curl -k -L -X GET \
-g 'http://10.10.10.66/app/api/rest?formatType=json&opName=userAPIGetWorkflowInput&opData={param0:%22VMware%20OVF%20Deployment%22}' \
-H 'X-Cloupia-Request-Key: 8187C34017C3479089C66678F32775FE'
```

```

{
  "serviceResult" : {
    "details" : [
      {
        "inputFieldValidator" : "VdcValidator",
        "label" : "vDC",
        "type" : "vDC",
        "inputFieldType" : "embedded-lov",
        "catalogType" : null,
        "isOptional" : false,
        "name" : "input_0_vDC728",
        "isMultiSelect" : false,
        "description" : "",
        "isAdminInput" : false
      },
      {
        "isAdminInput" : false,
        "description" : "",
        "label" : "OVF URL",
        "type" : "gen_text_input",
        "isMultiSelect" : false,
        "isOptional" : false,
        "inputFieldType" : "text",
        "catalogType" : null,
        "name" : "input_1_OVF_URL465",
        "inputFieldValidator" : null
      },
      ...omitted output
    ]
  },
  "serviceName" : "InfraMgr",
  "opName" : "userAPIGetWorkflowInputs",
  "serviceError" : null
}

```

```

#!/usr/bin/env python
from intersight.intersight_api_client import IntersightApiClient
from intersight.apis import equipment_device_summary_api

# Create an Intersight API Client instance
API_INSTANCE = IntersightApiClient(
    host="https://intersight.com/api/v1",\
    private_key="/Path_to/SecretKey.txt",\
    api_key_id="your_own_api_key_id")

# Create an equipment device handle
D_HANDLE = equipment_device_summary_api.EquipmentDeviceSummaryApi(API_INSTANCE)
DEVICES = D_HANDLE.equipment_device_summaries_get().results

print('{0:35s}{1:40s}{2:13s}{3:14s}'.format(
    "DN",
    "MODEL",
    "SERIAL",
    "OBJECT TYPE"))
print('-'*105)

# Loop through devices and extract data
for DEVICE in DEVICES:
    print('{0:35s}{1:40s}{2:13s}{3:14s}'.format(
        DEVICE.dn,
        DEVICE.model,
        DEVICE.serial,
        DEVICE.source_object_type))

```

```
""" Create Webex Team """

import json
import requests

URL = "https://webexapis.com/v1/teams"
PAYLOAD = {
    "name": "DevNet Associate Certification Team"
}
HEADERS = {
    "Authorization": "Bearer MDA0Y2VlMzktNDc2Ni00NzI5LWFiNmYtZmNmYzYzOTkyNjMxNmI0ND-
VmNDktNGE1_Pf84_consumer",
    "Content-Type": "application/json"
}
RESPONSE = requests.request("POST", URL, data=json.dumps(PAYLOAD), headers=HEADERS)
print(RESPONSE.text)
```

```
""" Create Webex Room """

import json
import requests
import pprint

URL = "https://webexapis.com/v1/rooms"
PAYLOAD = {
    "title": "DevAsc Team Room"
}
HEADERS = {
    "Authorization": "Bearer MDA0Y2VlMzktNDc2Ni00NzI5LWFiNmYtZmNmYzYzOTkyNjMxNmI0ND-
VmNDktNGE1_Pf84_consumer",
    "Content-Type": "application/json"
}
RESPONSE = requests.request("POST", URL, data=json.dumps(PAYLOAD), headers=HEADERS)
pprint.pprint(json.loads(RESPONSE.text))
```

```
$ python3 CreateRoom.py
{'created': '2020-02-15T23:13:35.578Z',
 'creatorId': 'Y2lzY29zcGFyazovL3VzL1BFTlBMRs8wYWZmMmPhNC1mN2IyLTQ3MwU-
tYTIzMj0xOTEyNDgwYmDEADB',
 'id': 'Y2lzY29zcGFyazovL3VzL1JPT00vY2FhMzJiYTAtNTA0OC0xMWVhLWJiZWItYmYlMWQyNGRm-
MTU0',
 'isLocked': False,
 'lastActivity': '2020-02-15T23:13:35.578Z',
 'ownerId': 'consumer',
 'title': 'DevAsc Team Room',
 'type': 'group'}
$
```

```
$ curl -X GET \
https://webexapis.com/v1/rooms \
-H 'Authorization: Bearer DeadBeefMTAtN2UzZi00YjRlWiZMGETmThjMzliNWQzGEyZTlJN-
WQxZTktNTRL_Pf84_leb65fdf-9643-417f-9974-ad72cae0e10f'
```

```

""" Add new Member to a Webex Room """

import json
import requests
import pprint

URL = "https://webexapis.com/v1/memberships"
PAYLOAD = {
    "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vY2FhMzJiYTAtNTA0OC0xMwVhLWJiZ-
WitYmY1MWQyNGRDEADB",
    "personEmail": "newUser@devasc.com",
    "personDisplayName": "Cisco DevNet",
    "isModerator": "false"
}
HEADERS = {
    "Authorization": "Bearer MDA0Y2V1MzktNDc2Ni00NzI5LWFhNmYtZmNmYzY3OTkyNjMxNmI0ND-
VmNDktNGE1PF84_consumer",
    "Content-Type": "application/json"
}
RESPONSE = requests.request("POST", URL, data=json.dumps(PAYLOAD), headers=HEADERS)
pprint.pprint(json.loads(RESPONSE.text))

```

```

""" Send Webex Message """

import json
import requests
import pprint

URL = "https://webexapis.com/v1/messages"
PAYLOAD = {
    "roomId": "Y2lzY29zcGFyazovL3VzL1JPT00vY2FhMzJiYTAtNTA0OC0xMwVhLWJiZ-
WitYmY1MWQyNGRmMTU0",
    "text": "This is a test message"
}
HEADERS = {
    "Authorization": "Bearer NDkzODZkZDUtZDExNC00ODM5LTk0YmYtZmY4NDI0ZTE5ZDA1MGI-
5YTY3OWUtZGYyPF84_consumer",
    "Content-Type": "application/json",
}
RESPONSE = requests.request("POST", URL, data=json.dumps(PAYLOAD), headers=HEADERS)
pprint.pprint(json.loads(RESPONSE.text))

```

```

""" Generate JWT """

import base64
import time
import math
import jwt

EXPIRATION = math.floor(time.time()) + 3600 # 1 hour from now
PAYLOAD = {
    "sub": "devASC",
    "name": "devASC-guest",
    "iss": "GUEST_ISSUER_ID",
    "exp": EXPIRATION
}

SECRET = base64.b64decode('GUEST_ISSUE_SECRET')

TOKEN = jwt.encode(PAYLOAD, SECRET)

print(TOKEN.decode('utf-8'))
HEADERS = {
    'Authorization': 'Bearer ' + TOKEN.decode('utf-8')
}

```

"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"

```

""" Generate Base64 Encoding """

import base64
ENCODED = base64.b64encode('devasc:strongpassword'.encode('UTF-8'))
print(ENCODED.decode('utf-8'))

```

```
""" Finesse - User Login"""
import requests

URL = "http://hq-uccx.abc.inc:8082/finesse/api/User/Agent001"
PAYLOAD = (
    "<User>" +
    "  <state>LOGIN</state>" +
    "  <extension>6001</extension>" +
    "</User>"
)

HEADERS = {
    'authorization': "Basic QWdlbnQwMDE6Y2lzMzY29wc2R0",
    'content-type': "application/xml",
}

RESPONSE = requests.request("PUT", URL, data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```

```
""" Finesse - User State Change"""
import requests

URL = "http://hq-uccx.abc.inc:8082/finesse/api/User/Agent001"

PAYLOAD = (
    "<User>" +
    "  <state>READY</state>" +
    "</User>"
)

HEADERS = {
    'authorization': "Basic QWdlbnQwMDE6Y2lzMzY29wc2R0",
    'content-type': "application/xml",
}

RESPONSE = requests.request("PUT", URL, data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```

```
import requests
url = "https://hq-uccx.abc.inc:8445/finesse/api/Team/2"
headers = {
    'authorization': "Basic QWdlbnQwMDE6Y2lzMzY29wc2R0",
    'cache-control': "no-cache",
}
response = requests.request("GET", url, headers=headers)
print(response.text)
```

```
""" Finesse - Initiate a dialog between two numbers """
import requests

URL = "http://hq-uccx.abc.inc:8082/finesse/api/User/Agent001/Dialogs"
PAYLOAD = (
    "<Dialog>" +
    "  <requestedAction>MAKE_CALL</requestedAction>" +
    "  <fromAddress>6001</fromAddress>" +
    "  <toAddress>6002</toAddress>" +
    "</Dialog>"
)

HEADERS = {
    'authorization': "Basic QWdlbnQwMDE6Y2lzMzY29wc2R0",
    'content-type': "application/xml",
    'cache-control': "no-cache",
}

RESPONSE = requests.request("POST", URL, data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
print(RESPONSE.status_code)
```

Name: myname mylastname
Email: myname@devasc.com
New API key: deadbeaf-d1e3-4000-993f-0d9479ab4944
New Passcode: 2BEEF
New Secret key: DEADBEEFNHYH54RP8THOJSM8OJKOUJVHXK7QQZSL1KF2D967JY5XZIR64B3GY5N-GRWMBVXG132XDGTOT5XJ62VQU4558I13J8IQ7TPEIPFB9MNG48WJM67F5C82P01VYTOTT4428T7RYQNO-YMSHXAX8ARLQHHFNCC1RJZYRIUF3AI000A6BA3BQN54DJ9VN3V9XPKUZM1E570X071KTSHDVTPW8CW-MU5Y9XZG76Z9FFA5UN1DJ7N39RXQQE99TRFA5HDBL2BJBPUOX6NMY6BDBZ4JP2IQ3RP3D6D7CUBP5U7W5EK-BWVTRZQXBY0L4M98Y1BR9HX5FD9D9HWYA
Expiration time: 4/27/20 6:22 AM GMT

```
curl -X POST \  
  https://api.Webex.com/WBXService/XMLService \  
  -H 'cache-control: no-cache' \  
  -H 'content-type: application/xml' \  
  -d '<?xml version="1.0" encoding="UTF-8"?>  
<serv:message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
  <header>  
    <securityContext>  
      <WebexID>devasc</WebexID>  
      <password>Kc5Ac4M1</password>  
      <siteName>apidemoeu</siteName>  
    </securityContext>  
  </header>  
  <body>  
    <bodyContent xsi:type="java:com.Webex.service.binding.meeting.  
    CreateMeeting">  
      <metaData>  
        <confName>Branding Meeting</confName>  
      </metaData>  
      <schedule>  
        <startDate/>  
      </schedule>  
    </bodyContent>  
  </body>  
</serv:message>'
```

```
curl -X POST \  
  https://api.Webex.com/WBXService/XMLService \  
  -H 'cache-control: no-cache' \  
  -H 'content-type: application/xml' \  
  -d '<serv:message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
<header>  
  <securityContext>  
    <WebexID>devasc</WebexID>  
    <password>Kc5Ac4M1</password>  
    <siteName>apidemoeu</siteName>  
  </securityContext>  
</header>  
<body>  
  <bodyContent  
    xsi:type="java:com.Webex.service.binding.meeting.LstsummaryMeeting">  
    <order>  
      <orderBy>STARTTIME</orderBy>  
    </order>  
  </bodyContent>  
</body>  
</serv:message>'
```

```

curl -X POST \
https://api.Webex.com/WBXService/XMLService \
-H 'cache-control: no-cache' \
-H 'content-type: application/xml' \
-d '<serv:message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<header>
  <securityContext>
    <WebexID>devasc</WebexID>
    <password>Kc5Ac4Ml</password>
    <siteName>apidemoeu</siteName>
  </securityContext>
</header>
<body>
  <bodyContent
    xsi:type="java:com.Webex.service.binding.meeting.SetMeeting">
    <meetingkey>625579604</meetingkey>
    <participants>
      <attendees>
        <attendee>
          <person>
            <email>student@devasc.com</email>
          </person>
        </attendee>
      </attendees>
    </participants>
    <attendeeOptions>
      <emailInvitations>true</emailInvitations>
    </attendeeOptions>
    <schedule>
      <openTime>300</openTime>
    </schedule>
  </bodyContent>
</body>
</serv:message>'

```

```

curl -X POST \
https://api.Webex.com/WBXService/XMLService \
-H 'cache-control: no-cache' \
-H 'content-type: application/xml' \
-d '<serv:message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<header>
  <securityContext>
    <WebexID>devasc</WebexID>
    <password>Kc5Ac4Ml</password>
    <siteName>apidemoeu</siteName>
  </securityContext>
</header>
<body>
  <bodyContent
    xsi:type="java:com.Webex.service.binding.meeting.DelMeeting">
    <meetingKey>625579604</meetingKey>
  </bodyContent>
</body>
</serv:message>'

```

```

""" Webex Devices - Get Session Cookie """
import requests
URL = "http://10.10.20.159/xmlapi/session/begin"
HEADERS = {
    'Authorization': "Basic ZGV2YXNjOkMxc2NvITiz"
}

RESPONSE = requests.request("POST", URL, headers=HEADERS)
print(RESPONSE.headers["Set-Cookie"])

```

```

$ python3 sess.py
SessionId=031033bebe67130d4d94747b6b9d4e4f6bd2965162ed542f22d32d75a9f238f9; Path=/;
HttpOnly

```

```
""" Webex Device - Endpoint Status """

import requests
URL = "http://10.10.20.159/status.xml"
HEADERS = {
    'Cookie': "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b"
}

RESPONSE = requests.request("GET", URL, headers=HEADERS)
print(RESPONSE.text)
```

```
""" Webex Device - Set Camera Position """

import requests

URL = "http://10.10.20.159/put.xml"

PAYLOAD = (
    '<Command>' +
    ' <Camera>' +
    ' <PositionSet command="True">' +
    ' <CameraId>1</CameraId>' +
    ' <Pan>150</Pan>' +
    ' <Tilt>150</Tilt>' +
    ' </PositionSet>' +
    ' </Camera>' +
    '</Command>'
)

HEADERS = {
    'Content-Type': "application/xml",
    'Cookie': "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b"
}

RESPONSE = requests.request("POST", URL, data=PAYLOAD, headers=HEADERS)
print(RESPONSE.text)
```

```
FeedbackSlot: <1.4> ServerUrl(r): <S: 1, 2048> Format: <XML/JSON>
Expression: <S: 1, 255> Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255> Expression: <S: 1, 255>
Expression: <S: 1, 255> Expression: <S: 1, 255> Expression: <S: 1, 255>
```



```
""" Webex Devices - Set Webhook """
```

```
import requests
```

```
URL = "http://10.10.20.159/put.xml"
```

```
PAYLOAD = {
```

```
    '<Command>' +  
    ' <HttpFeedback>' +  
    '   <Register command="True">' +  
    '     <FeedbackSlot>1</FeedbackSlot>' +  
    '     <ServerUrl>http://127.0.0.1/devasc-webhook</ServerUrl>' +  
    '     <Format>JSON</Format>' +  
    '     <Expression item="1">/Configuration</Expression>' +  
    '     <Expression item="2">/Event/CallDisconnect</Expression>' +  
    '     <Expression item="3">/Status/Call</Expression>' +  
    '   </Register>' +  
    ' </HttpFeedback>' +  
    '</Command>' +  
}
```

```
HEADERS = {
```

```
    'Content-Type': "application/xml",  
    'Cookie': "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b; SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b; SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b"
```

```
}
```

```
RESPONSE = requests.request("POST", URL, data=PAYLOAD, headers=HEADERS)
```

```
print (RESPONSE.text)
```

```
""" Webex Devices - Set People Presence detector ON """
```

```
import requests
```

```
URL = "http://10.10.20.159/put.xml"
```

```
PAYLOAD = {
```

```
    '<Configuration>' +  
    ' <RoomAnalytics>' +  
    '   <PeoplePresenceDetector>On</PeoplePresenceDetector>' +  
    ' </RoomAnalytics>' +  
    '</Configuration>' +  
}
```

```
HEADERS = {
```

```
    'Content-Type': "application/xml",  
    'Cookie': "SessionId=c6ca2fc23d3f211e0517d4c603fbe4205c77d13dd6913c7bc12eef4085b7637b"
```

```
}
```

```
RESPONSE = requests.request("POST", URL, data=PAYLOAD, headers=HEADERS)
```

```
print (RESPONSE.text)
```

```

""" Update and Retrieve Client Details """

from zeep import Client
from zeep.cache import SqliteCache
from zeep.transports import Transport
from requests import Session
from requests.auth import HTTPBasicAuth
import urllib3
from urllib3.exceptions import InsecureRequestWarning

urllib3.disable_warnings(InsecureRequestWarning)

USERNAME = 'administrator'
PASSWORD = 'ciscopsdt'
IP_ADDRESS = "10.10.20.1"
WSDL = 'schema/12.0//AXLAPI.wsdl'
BINDING_NAME = "{http://www.cisco.com/AXLAPIService/}AXLAPIBinding"
ADDRESS = "https://{ip}:8443/axl/".format(ip=IP_ADDRESS)

def update_phone_by_name(client, name, description):
    """ Update Phone by Name """
    return client.updatePhone(**{'name': name, 'description': description})

def get_phone_by_name(client, name):
    """ Get Phone by Name """
    return client.getPhone(name=name)

def main():
    """ Main """
    session = Session()
    session.verify = False
    session.auth = HTTPBasicAuth(USERNAME, PASSWORD)
    transport = Transport(cache=SqliteCache(), session=session, timeout=60)
    client = Client(wsdl=WSDL, transport=transport)
    axl = client.create_service(BINDING_NAME, ADDRESS)

    update_phone_by_name(axl, "SEP001122334455", "DevAsc: adding new Desc")
    print(get_phone_by_name(axl, "SEP001122334455"))

if __name__ == '__main__':
    main()

```

pip install ciscoaxl or pip3 install ciscoaxl

```

""" Using CiscoAXL SDK to get phone info"""

from ciscoaxl import axl

CUCM = '10.10.20.1'
CUCM_USER = "administrator"
CUCM_PASSWORD = "ciscopsdt"
CUCM_VERSION = '12.0'
ucm = axl(username=CUCM_USER,password=CUCM_PASSWORD,cucm=CUCM,cucm_version=CUCM_
VERSION)
print (ucm)
for phone in ucm.get_phones():
    print (phone.name)
for user in ucm.get_users():
    print (user.firstName)

```

"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"

```
""" Generate Base64 encoding using the base64 library """
import base64
encoded = base64.b64encode('devasc:strongpassword'.encode('UTF-8')).decode('ASCII')
print(encoded)
```

```
""" Get Customer details given a customerID """
import requests
url = "https://management.api.umbrella.com/v1/serviceproviders/serviceProviderId/
customers/customerId"
headers = {
    'accept': "application/json",
    'authorization': "Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"
}
response = requests.request("GET", url, headers=headers)
print(response.text)
```

```
""" Add domain using the Enforcement API """

import json
import requests

url = "https://s-platform.api.opendns.com/1.0/events"

querystring = {"customerKey": "XXXXXXXX-YYYY-ZZZZ-YYYY-XXXXXXXXXXXX"}

payload = [
    {
        "alertTime": "2020-01-01T09:33:21.0Z",
        "deviceId": "deadbeaf-e692-4724-ba36-c28132c761de",
        "deviceVersion": "13.7a",
        "dstDomain": "looksfake.com",
        "dstUrl": "http://looksfake.com/badurl",
        "eventTime": "2020-01-01T09:33:21.0Z",
        "protocolVersion": "1.0a",
        "providerName": "Security Platform"
    }
]

headers = {
    'Content-Type': "text/plain",
    'Accept': "*/*",
    'Cache-Control': "no-cache",
    'Host': "s-platform.api.opendns.com",
    'Accept-Encoding': "gzip, deflate",
    'Connection': "keep-alive",
    'cache-control': "no-cache"
}

response = requests.request(
    "POST",
    url,
    data=json.loads(payload),
    headers=headers,
    params=querystring)

print(response.text)
```

```
""" List domains using the Enforcement API """

import requests
url = "https://s-platform.api.opendns.com/1.0/domains"
querystring = {"customerKey": "XXXXXXXX-YYYY-ZZZZ-YYYY-XXXXXXXXXXXX"}
response = requests.request("POST", url, headers=headers, params=querystring)
print(response.text)
```

```
{
  "meta":{
    "page":1,
    "limit":200,
    "prev":false,
    "next":"https://s-platform.api.opendns.com/1.0/
domains?customerKey=XXXXXXXX-YYYY-ZZZZ-YYYY-XXXXXXXXXXXX&page=2&limit=200"
  },
  "data":[
    {
      "id":1,
      "name":"baddomain1.com"
    },
    {
      "id":2,
      "name":"looksfake.com"
    },
    {
      "id":3,
      "name":"malware.dom"
    }
  ]
}
```

https://s-platform.api.opendns.com/1.0/domains/looksfake.com?customerKey= XXXXXXXX-YYYY-ZZZZ-YYYY-XXXXXXXXXXXX

```
""" Delete domain using the Enforcement API """

import requests

url = "https://s-platform.api.opendns.com/1.0/domains/looksfake.com"

querystring = {"customerKey":"XXXXXXXX-YYYY-ZZZZ-YYYY-XXXXXXXXXXXX"}

response = requests.request("DELETE", url, headers=headers, params=querystring)
print(response.text)
```

```
""" Domains categorization using the Investigate API """

import requests

url = "https://investigate.api.umbrella.com/domains/categorization/cisco.com"
querystring = {"showLabels":""}
headers = {
  'authorization': "Bearer deadbeef-24d7-40e1-a5ce-3b064606166f",
  'cache-control': "no-cache",
}

response = requests.request("GET", url, headers=headers, params=querystring)
print(response.text)
```

```
{
  "cisco.com": {
    "status": 1,
    "security_categories": [],
    "content_categories": [
      "Software/Technology",
      "Business Services"
    ]
  }
}
```

```
""" Generate the session token for FMC """
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

url = "https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/auth/generatetoken"
headers = {
    'Content-Type': "application/xml",
    'Authorization': "Basic YXNodXRvc2g6V0JVdkE5TXk=",
}
response = requests.request("POST", url, headers=headers)
print(response.headers)
```

```
""" Get Server Version """
import requests
url = "https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/info/serverversion"
headers = {
    'X-auth-access-token': "2abd7bdc-16f8-477f-8022-7f193e71c847",
}
response = requests.request("GET", url, headers=headers, verify=False)
print(response.text)
```

```
""" generate a session token and create a new network object """

import json
import requests

## Globals used in this file
url = "https://fmcrestapisandbox.cisco.com/api/fmc_platform/v1/auth/generatetoken"
server = "https://fmcrestapisandbox.cisco.com"
username = "johnsmith"
password = "pwgDvQt3"
domain = "Global"
token = ""
headers = {
    'Content-Type': "application/json",
}

## Definition of Lab Network (10.10.10.0)

network_lab = {
    "name": "labnetwork-1",
    "value": "10.10.10.0/24",
    "overridable": False,
    "description": "Lab Network Object",
    "type": "Network"
}

def networkObject(network, uuid):
    """ Create a new Network object """

    netpath = "/api/fmc_config/v1/domain/" + uuid + "/object/networks"
    url = server + netpath
    print("-----")
    print(headers)
    try:
        response = requests.post(url, data=json.dumps(network), headers=headers,
                                verify=False)
        status_code = response.status_code
        resp = response.text
        json_response = json.loads(resp)
        print("status code is: " + str(status_code))
        if status_code == 201 or status_code == 202:
            print("Successfully network created")
        else:
```

```

        response.raise_for_status()
        return json_response["name"], json_response["id"]
    except requests.exceptions.HTTPError as err:
        print("Reason Code: " + str(err))
    finally:
        if response:
            response.close()

def generateSessionToken():
    """ Generate a new session token using the username and password """
    global uuid
    global headers
    tokenurl = "/api/fmc_platform/v1/auth/generatetoken"
    url = server + tokenurl
    response = requests.request(
        "POST",
        url,
        headers=headers,
        auth=requests.auth.HTTPBasicAuth(username, password),
        verify=False
    )
    print(response.headers)
    status_code = response.status_code
    if status_code == 201 or status_code == 202:
        print("Successfully network created")
    else:
        response.raise_for_status()

    auth_headers = response.headers
    token = auth_headers.get('X-auth-access-token', default=None)
    headers['X-auth-access-token'] = token
    domains = auth_headers.get('DOMAINS', default=None)
    domains = json.loads("{\"domains\": " + domains + "}")
    for item in domains["domains"]:
        if item["name"] == domain:
            uuid = item["uuid"]
        else:
            print("no UUID for the domain found!")

    print(domains)
    print(uuid)
    print(headers)

## Main - Entry point - Invoke generate token and create network object
if __name__ == "__main__":
    generateSessionToken()
    networkObject(network_lab, uuid)

```

https://<clientID>:<clientKEY>@<api_endpoint>
 Authorization: Basic ZGVhZGJlZWYxMjM0NDhjY2MwMQQ6WFhYWPhYWFGtWVlZWS1aWlpalTAwMDAtZTM4NGVmMmR4eHh4
 WVlZWS1aWlpalTAwMDAtZTM4NGVmMmR4eHh4

```

""" GET list of all computers via API """
import requests
url = "https://api.amp.cisco.com/v1/computers"
headers = {
    'authorization': "Basic ZGVhZGJlZWYxMjM0NDhjY2MwMQQ6WFhYWPhYWFGtWVlZWS1aWlpal-
    TAwMDAtZTM4NGVmMmR4eHh4",
    'cache-control': "no-cache",
}
response = requests.request("GET", url, headers=headers)
print(response.text)

```

```
""" GET list of all vulnerabilities via API """
import requests
url = "https://api.amp.cisco.com/v1/vulnerabilities"
querystring = {"offset": "0", "limit": "1"}
headers = {
    'authorization': "Basic ZGVhZGJlZWYxMjM0NDhjY2MwMGQ6WFhYWPhYWFgtWVlZWS1aWlpaL-
    TAwMDAtZTM4NGVmMmR4eHh4",
    'cache-control': "no-cache",
}
response = requests.request("GET", url, headers=headers, params=querystring)
print(response.text)
```

```
{
  "version": "v1.2.0",
  "metadata": {
    "links": {
      "self": "https://api.amp.cisco.com/v1/vulnerabilities?offset=0&limit=1"
    },
    "results": {
      "total": 1,
      "current_item_count": 1,
      "index": 0,
      "items_per_page": 1
    }
  },
  "data": [
    {
      "application": "Adobe Flash Player",
      "version": "11.5.502.146",
      "file": {
        "filename": "FlashPlayerApp.exe",
        "identity": {
          "sha256": "c1219f0799e60ff48a9705b63c14168684aed911610fec68548ea08f-
          605cc42b"
        }
      },
      "cves": [
        {
          "id": "CVE-2013-3333",
          "link": "https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-3333",
          "cvss": 10
        }
      ],
    }
  ],
}
```

```

    {
      "id": "CVE-2014-0502",
      "link": "https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0502",
      "cvss": 10
    }
  ],
  "latest_timestamp": 1574442349,
  "latest_date": "2019-11-22T17:05:49+00:00",
  "groups": [
    {
      "name": "Triage",
      "description": "Triage Group for FireAMP API Docs",

      "guid": "68665863-74d5-4bc1-ac7f-5477b2b6406e"
    }
  ],
  "computers_total_count": 1,
  "computers": [
    {
      "connector_guid": "17d71471-805b-4183-9121-3924b8982fac",
      "hostname": "Demo_ZAccess",
      "active": true,
      "links": {
        "computer": "https://api.amp.cisco.com/v1/computers/17d71471-805b-4183-9121-3924b8982fac",
        "trajectory": "https://api.amp.cisco.com/v1/computers/17d71471-805b-4183-9121-3924b8982fac/trajectory",
        "group": "https://api.amp.cisco.com/v1/groups/68665863-74d5-4bc1-ac7f-5477b2b6406e"
      }
    }
  ]
}

```

"Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3Jk"

```

import base64
encoded = base64.b64encode('devasc:strongpassword'.encode('UTF-8')).decode('ASCII')
print(encoded)

```

```

Data - {
  "EndPointGroup" : {
    "name" : "DevNet Associate Group",
    "description" : "DevNet Associate Group"
  }
}
-----

```



```

""" create a new endpointgroup """
import json
import requests
url = "https://ise.devnetsandbox.com/ers/config/endpointgroup"
payload = {
    "EndPointGroup": {
        "name": "DevNet Associate Group",
        "description": "DevNet Associate Group"
    }
}
headers = {
    'content-type': "application/json",
    'accept': "application/json",
    'authorization': "Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3JkZjw=",
    'cache-control': "no-cache",
}
response = requests.request(
    "POST",
    url,
    data=json.dumps(payload),
    headers=headers
)
print(response.text)

```

Location: <https://ise.devnetsandbox.com:9060/ers/config/endpointgroup/00000000-1111-2222-3333-444444444444>

```

Data - {
    "ERSEndPoint" : {
        "name" : "DevNet_Endpoint",
        "description" : "DevNet Endpoint-1",
        "mac" : "FF:EE:DD:03:04:05",
        "groupId" : " 00000000-1111-2222-3333-444444444444",
        "staticGroupAssignment" : true
    }
}

```

```

""" create a new endpoint """
import json
import requests
url = "https://ise.devnetsandbox.com/ers/config/endpoint"
payload = {
    "ERSEndPoint": {
        "name": "DevNet_Endpoint",
        "description": "DevNet Endpoint-1",
        "mac": "FF:EE:DD:03:04:05",
        "groupId": " 00000000-1111-2222-3333-444444444444",
        "staticGroupAssignment": True
    }
}
headers = {
    'content-type': "application/json",
    'accept': "application/json",
    'authorization': "Basic ZGV2YXNjOnN0cm9uZ3Bhc3N3b3JkZjw=",
    'cache-control': "no-cache",
}
response = requests.request(
    "POST",
    url,
    data=json.dumps(payload),
    headers=headers
)
print(response.text)

```

Location: <https://ise.devnetsandbox.com:9060/ers/config/endpoint/deadbeef-1111-2222-3333-444444444444>

```
""" Threat Grid - who am I """
import requests
url = "https://panacea.threatgrid.com/api/v3/session/whoami"
querystring = {"api_key": "deadbeefelcpgib9ec0909"}
headers = {
    'cache-control': "no-cache",
}
response = requests.request(
    "GET",
    url,
    headers=headers,
    params=querystring
)
print(response.text)
```

```
{
  "api_version": 3,
  "id": 1234567,
  "data": {
    "role": "user",
    "properties": {},
    "integration_id": "z1ci",
    "email": "devasc@student.com",
    "organization_id": 666777,
    "name": "devasc",
    "login": "devasc",
    "title": "DevNet Associate",
    "api_key": " deadbeefelcpgib9ec0909",
    "device": false
  }
}
```

```
""" Threat Grid - search submissions API """
import requests
url = "https://panacea.threatgrid.com/api/v2/search/submissions"
querystring = {
  "q": "8fbb3bd96b80e28ea41107728ce8c073cbadb0dd",
  "api_key": "deadbeefelcpgib9ec0909"
}
headers = {
  'cache-control': "no-cache",
}
response = requests.request(
  "GET",
  url,
  headers=headers,
  params=querystring
)
print(response.text)
```

```
{
  "api_version": 2,
  "id": 4482656,
  "data": {
    "index": 0,
    "total": 1,
    "took": 3956,
    "timed_out": false,
    "items_per_page": 100,
    "current_item_count": 1,
    "items": [
      {
        "item": {
          "properties": {
            "metadata": null
          },
          "tags": [],
          "vm_runtime": 300,
          "md5": "b5c26cab57c41208bd6bf92d495ee1f0",
          "state": "succ",
          "sha1": "8fbb3bd96b80e28ea41107728ce8c073cbadb0dd",
          "sample": "b7d3dd2a6d47ea640f719cc76c2122f8",
          "filename": "FlashPlayer.exe",
          ....cut

```

```
""" Threat Grid - List details for each curated feed type """
import requests
FEED_URL = "https://panacea.threatgrid.com/api/v3/feeds"
FEEDS_NAME = {
  "autorun-registry": "Autorun Registry Malware",
  "banking-dns": "Banking Trojans",
  "dga-dns": "Domain Generation Algorithm Destinations",
  "dll-hijacking-dns": "DLL Hijackers / Sideloaders",
  "doc-net-com-dns": "Document File Network Communication",
  "downloaded-pe-dns": "Dropper Communication",
  "dynamic-dns": "Dynamic DNS Communication",
  "irc-dns": "IRC Communication",
  "modified-hosts-dns": "Modified HOSTS File Communication",
  "public-ip-check-dns": "Public IP Checkers",
  "ransomware-dns": "Ransomware Communication",
  "rat-dns": "Remote Access Trojans",
  "scheduled-tasks": "Scheduled Task Communication",
  "sinkholed-ip-dns": "Sinkholed IPs",
  "stolen-cert-dns": "Stolen Certificates",
}
for name, desc in FEEDS_NAME.items():
  url = "{}/{}/.json".format(FEED_URL, name)
  querystring = {"api_key": "2kdn3muq7uafelcpgib9eccua7"}

  headers = {
    'cache-control': "no-cache",
    'Content-type': 'application/json',
    'Accept': 'application/json'
  }

  response = requests.request("GET", url, headers=headers, params=querystring)

  print(response.text)
```

```
typedef percent {
    type uint16 {
        range "0 .. 100";
    }
}
```

Description "Percentage":

```
}
leaf intf-name {
    type string;
    description "The name of the interface";
}
leaf-list trunk-interfaces {
    type string;
    description "List of trunk interfaces";
}
```

```
<trunk-interfaces>TenGigabitEthernet0/1</trunk-interfaces>
<trunk-interfaces>TenGigabitEthernet0/2</trunk-interfaces>
<trunk-interfaces>TenGigabitEthernet0/3</trunk-interfaces>
<trunk-interfaces>TenGigabitEthernet0/4</trunk-interfaces>
```

```
container statistics {
    description "A collection of interface statistics.";
    leaf in-octets {
        type yang:counter64;
        description "The total number of octets received on the interface.";
    }
    leaf in-errors {
        type yang:counter32;
        description "Number of inbound packets that contained errors.";
    }
    leaf out-octets {
        type yang:counter64;
        description "The total number of octets sent out on the interface.";
    }
    leaf out-errors {
        type yang:counter32;
        description "Number of outbound packets that contained errors.";
    }
}
```

<statistics>

<in-octets>5983247896</in-octets>

<in-errors>2578</in-errors>

<out-octets>678845633</out-octets>

<out-errors>0</out-errors>

</statistics>

```
<user>
  <name>john</name>
  <uid>1000</uid>
  <full-name>John Doe</full-name>
</user>
<user>
  <name>jeanne</name>
  <uid>1001</uid>
  <full-name>Jeanne Doe</full-name>
</user>
```

```
// Contents of "bogus-interfaces.yang"
module bogus-interfaces {
  namespace "http://bogus.example.com/interfaces";
  prefix "bogus";

  import "ietf-yang-types" {
    prefix yang;
  }

  organization "Bogus Inc.";
  contact "john@bogus.example.com";
  description
    "The module for entities implementing the Bogus interfaces .";

  revision 2020-01-06 {
    description "Initial revision.";
  }
  container interfaces {
    leaf intf-name {
      type string;
      description "The name of the interface";
    }

    leaf-list trunk-interfaces {
      type string;
      description "List of trunk interfaces";
    }
  }

  container statistics {
    description "A collection of interface statistics.";
    leaf in-octets {
      type yang:counter64;
      description "Total number of octets received on the
        interface.";
    }
  }
}
```

```

        leaf in-errors {
            type yang:counter32;
            description "Number of inbound packets that contained
            errors.";
        }

        leaf out-octets {
            type yang:counter64;
            description "Total number of octets sent out on the
            interface.";
        }

        leaf out-errors {
            type yang:counter32;
            description "Number of outbound packets that contained
            errors.";
        }
    }
    list user {
        key name;
        leaf name {
            type string;
        }
        leaf uid {
            type uint32;
        }
        leaf full-name {
            type string;
        }
    }
}
}

```

```

container statistics {
    description "A collection of interface statistics.";
    leaf in-octets {
        type yang:counter64;
        description "The total number of octets received on the interface.";
    }
    leaf in-errors {
        type yang:counter32;
        description "Number of inbound packets that contained errors.";
    }
    leaf out-octets {
        type yang:counter64;
        description "The total number of octets sent out on the interface.";
    }
    leaf out-errors {
        type yang:counter32;
        description "Number of outbound packets that contained errors.";
    }
}
}

```

In order to augment the module, the "augment" statement is used as follows:

```

import interface-statistics {
    prefix "intf-stats";
}
augment "/intf-stats:statistics" {
    leaf in-unicast-pkts {
        type yang:counter64;
        description "Number of unicast packets received on the interface.";
    }
    leaf out-unicast-pkts {
        type yang:counter64;
        description "Number of unicast packets sent out the interface.";
    }
}
}

```

```
rpc activate-software-image {
  input {
    leaf image {
      type binary;
    }
  }
  output {
    leaf status {
      type string;
    }
  }
}
```

```
notification config-change {
  description "The configuration change.";
  leaf operator-name {
    type string;
  }

  leaf-list change {
    type instance-identifier;
  }
}
```

```
module ietf-interfaces {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
  prefix if;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF NETMOD (Network Modeling) Working Group";

  contact
    "WG Web: <https://datatracker.ietf.org/wg/netmod/>
    WG List: <mailto:netmod@ietf.org>

    Editor: Martin Bjorklund
           <mailto:mbj@tail-f.com>";

  description
    "This module contains a collection of YANG definitions for
    managing network interfaces.+

    Copyright (c) 2018 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).

    This version of this YANG module is part of RFC 8343; see
    the RFC itself for full legal notices.";

  revision 2018-02-20 {
    description
      "Updated to support NMDA.";
    ... omitted output
  }
}
```

```

module: ietf-interfaces
+--rw interfaces
| +--rw interface* [name]
|   +--rw name                string
|   +--rw description?        string
|   +--rw type                 identityref
|   +--rw enabled?            boolean
|   +--rw link-up-down-trap-enable? enumeration {if-mib}?
|   +--ro admin-status        enumeration {if-mib}?
|   +--ro oper-status         enumeration
|   +--ro last-change?        yang:date-and-time
|   +--ro if-index            int32 {if-mib}?
|   +--ro phys-address?       yang:phys-address
|   +--ro higher-layer-if*    interface-ref
|   +--ro lower-layer-if*    interface-ref
|   +--ro speed?              yang:gauge64
|   +--ro statistics
|
|       +--ro discontinuity-time yang:date-and-time
|       +--ro in-octets?         yang:counter64
|       +--ro in-unicast-pkts?   yang:counter64
|       +--ro in-broadcast-pkts? yang:counter64
|       +--ro in-multicast-pkts? yang:counter64
|       +--ro in-discards?       yang:counter32
|       +--ro in-errors?         yang:counter32
|       +--ro in-unknown-protos? yang:counter32
|       +--ro out-octets?        yang:counter64
|       +--ro out-unicast-pkts?  yang:counter64
|       +--ro out-broadcast-pkts? yang:counter64
|       +--ro out-multicast-pkts? yang:counter64
|       +--ro out-discards?      yang:counter32
|       +--ro out-errors?        yang:counter32
x--ro interfaces-state
  x--ro interface* [name]
    x--ro name                string
    x--ro type                 identityref
    x--ro admin-status        enumeration {if-mib}?
    x--ro oper-status         enumeration
    x--ro last-change?        yang:date-and-time
    x--ro if-index            int32 {if-mib}?
    x--ro phys-address?       yang:phys-address
    x--ro higher-layer-if*    interface-state-ref
    x--ro lower-layer-if*    interface-state-ref
    x--ro speed?              yang:gauge64
    x--ro statistics
      x--ro discontinuity-time yang:date-and-time
      x--ro in-octets?         yang:counter64
      x--ro in-unicast-pkts?   yang:counter64
      x--ro in-broadcast-pkts? yang:counter64
      x--ro in-multicast-pkts? yang:counter64
      x--ro in-discards?       yang:counter32
      x--ro in-errors?         yang:counter32
      x--ro in-unknown-protos? yang:counter32
      x--ro out-octets?        yang:counter64
      x--ro out-unicast-pkts?  yang:counter64
      x--ro out-broadcast-pkts? yang:counter64
      x--ro out-multicast-pkts? yang:counter64
      x--ro out-discards?      yang:counter32
      x--ro out-errors?        yang:counter32

```



```

<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:xpath:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-Ω
    <capability>urn:ietf:params:netconf:capability:yang-library:1.0?revision=2016-06-21
&module-set-id=0de3f66ee656759ede57b7f3b6cd310d</capability>
    <capability>http://tail-f.com/ns/netconf/actions/1.0</capability>
    <capability>http://tail-f.com/ns/netconf/extensions</capability>
    <capability>http://cisco.com/ns/cisco-xe-ietf-ip-deviation?module=cisco-
xe-ietf-ip-deviation&revision=2016-08-10</capability>
    <capability>http://cisco.com/ns/cisco-xe-ietf-ipv4-unicast-routing-
deviation?module=cisco-xe-ietf-ipv4-unicast-routing-deviation&revis
ion=2015-09-11</capability>
    <capability>http://cisco.com/ns/cisco-xe-ietf-ipv6-unicast-routing-
deviation?module=cisco-xe-ietf-ipv6-unicast-routing-deviation&revis
ion=2015-09-11</capability>
    <capability>http://cisco.com/ns/cisco-xe-ietf-ospf-
deviation?module=cisco-xe-ietf-ospf-deviation&revision=2015-09-11</
capability>

    <capability>http://cisco.com/ns/cisco-xe-ietf-routing-
deviation?module=cisco-xe-ietf-routing-deviation&revision=2016-07-09</
capability>
    <capability>http://cisco.com/ns/cisco-xe-openconfig-acl-
deviation?module=cisco-xe-openconfig-acl-deviation&revision=2017-08-21</
capability>
    <capability>http://cisco.com/ns/mpls-static/
devs?module=common-mpls-static-devs&revision=2015-09-11</capability>
    <capability>http://cisco.com/ns/nvo/devs?module=nvo-devs&revision=2015-09-11</
capability>
    <capability>http://cisco.com/ns/yang/Cisco-IOS-XE-aaa?module=Cisco-IOS-XE-
aaa&revision=2017-09-05</capability>
    <capability>http://cisco.com/ns/yang/Cisco-IOS-XE-acl?module=Cisco-IOS-XE-
acl&revision=2017-08-01</capability>
    <capability>http://cisco.com/ns/yang/Cisco-IOS-XE-acl-
oper?module=Cisco-IOS-XE-acl-oper&revision=2017-02-07</
capability>
    ...
  </capabilities>
</session-id>5546</session-id></hello>]]>
<?xml version="1.0" encoding="UTF-8"?>

<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

  <capabilities>

    <capability>urn:ietf:params:netconf:base:1.0</capability>

  </capabilities>

</hello>]]>]]>
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:
base:1.0">

  <get-config>

    <source>

      <running/>

    </source>

  </get-config>

</rpc>]]>]]>

```

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
  <data>
    <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
      <version>16.6</version>
      <boot-start-marker />
      <boot-end-marker />
      <service>
        <timestamps>
          <debug>
            <datetime>
              <msec />
            </datetime>
          </debug>
          <log>
            <datetime>
              <msec />
            </datetime>
          </log>
        </timestamps>
      </service>
      <platform>
        <console xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-platform">
          <output>serial</output>
        </console>
      </platform>
      <hostname>csr1kv</hostname>
      <enable>
        <password>
          <secret>cisco</secret>
        </password>
        <secret>
          <type>5</type>
          <secret>$1$A1e2$Vp95LGeOXX3CfOK05K5Nf1</secret>
        </secret>
      </enable>
      ... omitted output
    </data>
  </rpc-reply>
```

```
#!/usr/bin/env python
""" Get the capabilities of a remote device with NETCONF """

from ncclient import manager

NXOS_HOST = "10.10.10.10"
NETCONF_PORT = "830"
USERNAME = "admin"
PASSWORD = "password"

# create a get_capabilities() method
def get_capabilities():
    """
    Method that prints NETCONF capabilities of remote device.
    """
    with manager.connect(
        host=NXOS_HOST,
        port=NETCONF_PORT,
        username=USERNAME,
        password=PASSWORD,
        hostkey_verify=False
    ) as device:

        # print all NETCONF capabilities
        print('\n***NETCONF Capabilities for device {}***\n'.format(NXOS_HOST))
        for capability in device.server_capabilities:
            print(capability)

if __name__ == '__main__':
    get_capabilities()
```

```
***NETCONF Capabilities for device 10.10.10.10***

urn:ietf:params:netconf:base:1.0
urn:ietf:params:netconf:base:1.1
urn:ietf:params:netconf:capability:writable-running:1.0
urn:ietf:params:netconf:capability:rollback-on-error:1.0
urn:ietf:params:netconf:capability:candidate:1.0
urn:ietf:params:netconf:capability:validate:1.1
urn:ietf:params:netconf:capability:confirmed-commit:1.1
http://cisco.com/ns/yang/cisco-nx-os-device?revision=2019-02-17&module=Cisco-NX-OS-device&deviations=Cisco-NX-OS-device-deviations
```

```

#!/usr/bin/env python
""" Add a loopback interface to a device with NETCONF """

from ncclient import manager

NXOS_HOST = "10.10.10.10"
NETCONF_PORT = "830"
USERNAME = "admin"
PASSWORD = "password"
LOOPBACK_ID = "01"
LOOPBACK_IP = "1.1.1.1/32"

# create add_loopback() method
def add_loopback():
    """
    Method that adds loopback interface and configures IP address
    """

    add_loop_interface = """<config>
<System xmlns="http://cisco.com/ns/yang/cisco-nx-os-device">
  <intf-items>
    <lb-items>
      <LbRtdIf-list>
        <id>lo{id}</id>
        <adminSt>up</adminSt>
        <descr>Intf configured via NETCONF</descr>
      </LbRtdIf-list>
    </lb-items>
  </intf-items>
  <ipv4-items>
    <inst-items>
      <dom-items>
        <Dom-list>
          <name>default</name>
          <if-items>
            <If-list>
              <id>lo{id}</id>
              <addr-items>
                <Addr-list>
                  <addr>{ip}</addr>
                </Addr-list>
              </addr-items>
            </If-list>
          </if-items>
        </Dom-list>
      </dom-items>
    </inst-items>
  </ipv4-items>
</System>
</config>""".format(id=LOOPBACK_ID, ip=LOOPBACK_IP)

    with manager.connect(
        host=NXOS_HOST,
        port=NETCONF_PORT,
        username=USERNAME,
        password=PASSWORD,
        hostkey_verify=False
    ) as device:

        # Add loopback interface
        print("\n Adding Loopback {} with IP address {} to device {...}\n".
              format(LOOPBACK_ID, LOOPBACK_IP, NXOS_HOST))
        netconf_response = device.edit_config(target='running',
                                              config=add_loop_interface)
        # Print the XML response
        print(netconf_response)

if __name__ == '__main__':
    add_loopback()

```

Adding Loopback 01 with IP address 1.1.1.1/32 to device
10.10.10.10...

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<rpc-reply message-id="urn:uuid:6de4444b-9193-4b74-837b-  
e3994d75a319" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
```

```
  <ok/>
```

```
</rpc-reply>
```

```
$ docker
```

```
Usage: docker [OPTIONS] COMMAND
```

```
A self-sufficient runtime for containers
```

```
<cut for brevity>
```

```
Management Commands:
```

```
builder    Manage builds
checkpoint  Manage checkpoints
config      Manage Docker configs
container   Manage containers
context     Manage contexts
image       Manage images
network     Manage networks
node        Manage Swarm nodes
plugin      Manage plugins
secret      Manage Docker secrets
service     Manage services
stack       Manage Docker stacks
swarm       Manage Swarm
system      Manage Docker
trust       Manage trust on Docker images
volume      Manage volumes
```

```
<cut for brevity>
```

```
Run 'docker COMMAND --help' for more information on a command.
```

```
$ docker container --help
```

```
Usage: docker container COMMAND
```

```
Manage containers
```

```
Commands:
```

```
attach      Attach local standard input, output, and error streams to a running
            container
commit      Create a new image from a container's changes
cp          Copy files/folders between a container and the local filesystem
create      Create a new container
diff        Inspect changes to files or directories on a container's filesystem
exec        Run a command in a running container
export      Export a container's filesystem as a tar archive
inspect     Display detailed information on one or more containers
kill        Kill one or more running containers
logs        Fetch the logs of a container
ls          List containers
pause       Pause all processes within one or more containers
port        List port mappings or a specific mapping for the container
prune       Remove all stopped containers
rename      Rename a container
restart     Restart one or more containers
rm          Remove one or more containers
run         Run a command in a new container
start       Start one or more stopped containers
stats       Display a live stream of container(s) resource usage statistics
stop        Stop one or more running containers
top         Display the running processes of a container
unpause     Unpause all processes within one or more containers
update      Update configuration of one or more containers
wait        Block until one or more containers stop, then print their exit codes
```

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:4df8ca8a7e309c256d60d7971ea14c27672fc0d10c5f303856d7bc48f8cc17ff
Status: Downloaded newer image for hello-world:latest
```

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

```
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
\$ docker container ls -a						

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bac0c2a2cca8	hello-world	"/hello"	43 minutes ago	Exited(0)	43 minutes ago	

```
$ docker container run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
7ddb47eeb70: Pull complete
c1bbdc448b72: Pull complete
8c3b70e39044: Pull complete
45d437916d57: Pull complete
Digest: sha256:6e9f67fa63b0323e9a1e587fd71c561ba48a034504fb804fd26fd8800039835d
Status: Downloaded newer image for ubuntu:latest
root@a583eac3cadb:/#
```

```
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
a583eac3cadb	ubuntu	"bash"	About a minute ago	Up	About a minute	musings_bartik

```
$ docker container attach a583eac3cadb
```

```
root@a583eac3cadb:/#
```

```
$ docker container inspect a583eac3cadb
[
  {
    "Id": "a583eac3cadbafca855bec9b57901e1325659f76b37705922db67ebf22fdd925",
    "Created": "2019-11-27T23:35:12.537810374Z",
    "Path": "bash",
    "Args": [],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 127,
      "Error": "",
      "StartedAt": "2019-11-27T23:35:13.185535918Z",
      "FinishedAt": "2019-11-27T23:53:15.898516767Z"
    },
    "Image": "sha256:775349758637aff77bf85e2ff0597e86e3e859183ef0baba8b3e8fc8d3c
ba51c",
    "ResolvConfPath": "/var/lib/docker/containers/a583eac3cadbafca-
855bec9b57901e1325659f76b37705922db67ebf22fdd925/resolv.conf"
  }
]
<cut for brevity>
```

```
$ docker container logs a583eac3cadb
```

```
root@a583eac3cadb:/# ps
```

PID	TTY	TIME	CMD
1	pts/0	00:00:00	bash
11	pts/0	00:00:00	ps

```
root@a583eac3cadb:/# uname
```

Linux

```
$ docker container run --name test-nginx -p 80:80 -d nginx
```

```
dfe3a47945d2aa1cdc170ebf0220fe8e4784c9287eb84ab0bab7048307b602b9
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dfe3a47945d2	nginx	"nginx -g 'daemon of...'"	21 seconds ago	Up	20 seconds 0.0.0.0:80->80/tcp	test-nginx

```
$ docker container run --name test-nginx -p 80:80 -d -v
```

```
~/Documents/html:/usr/share/nginx/html nginx
```

```
d0d5c5ac86a2994ea1037bd9005cc8d6bb3970bf998e5867fe392c2f35d8bc1a
```

```
$ docker container stop test-nginx
```

```
test-nginx
```



```
$ docker container rm test-nginx
```

```
test-nginx
```

```
$ docker container prune
```

```
WARNING! This will remove all stopped containers.
```

```
Are you sure you want to continue? [y/N] y
```

```
Deleted Containers:
```

```
a583eac3cadbafca855bec9b57901e1325659f76b37705922db67ebf22fdd925  
FROM ubuntu:16.04
```

```
MAINTAINER Cisco Champion (user@domain.com)  
RUN apt-get update && apt-get upgrade -y
```

```
RUN apt-get install nginx -y  
CMD ["nginx", "-g", "daemon off;"]
```

```
FROM ubuntu:latest
```

```
MAINTAINER Cisco Champion (user@domain.com)
```

```
RUN apt-get update && apt-get upgrade -y
```

```
RUN apt-get install nginx -y
```

```
EXPOSE 80 443
```

```
VOLUME /usr/share/nginx/html
```

```
CMD ["nginx", "-g", "daemon off;"]
```

```
$ docker build -t myimage:latest .  
Sending build context to Docker daemon 2.048kB  
Step 1/8 : FROM ubuntu:latest  
--> 775349758637  
Step 2/8 : MAINTAINER Cisco Champion (user@domain.com)  
--> Using cache  
--> f83e0f07db18  
Step 3/8 : RUN apt-get update  
--> Using cache  
--> 646cc0e9f256  
Step 4/8 : RUN apt-get upgrade -y  
--> Using cache  
--> c2701f555b0f  
Step 5/8 : RUN apt-get install nginx -y  
--> Using cache  
--> 4abf50fd4a02  
Step 6/8 : EXPOSE 80 443  
--> Using cache  
--> a9a1533064b2  
Step 7/8 : VOLUME /usr/share/nginx/html  
--> Using cache  
--> 2ecd13c5af2b  
Step 8/8 : CMD ["nginx", "-g", "daemon off;"]  
--> Running in e03bd2387319  
Removing intermediate container e03bd2387319  
--> 04ae8c714993  
Successfully built 04ae8c714993  
Successfully tagged myimage:latest
```

```
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
myimage latest 04ae8c714993 48 minutes ago 154MB
```

```
$ docker image inspect myimage
[
  {
    "Id": "sha256:04ae8c7149937b2ce8b8963d6c34810d8dc53607c9a97064e1f3c85cdc3abb46",
    "RepoTags": [
      "myimage:latest"
    ],
    "RepoDigests": [],
    "Parent": "sha256:2ecd13c5af2b41349b58f2397cbbbc70f5c1c604262113bae443f1f6fc3758cc",
    "Comment": "",
    "Created": "2019-11-28T05:53:37.794817007Z",
    "Container": "e03bd238731932ca2cab6c8b7fal346105b839ce2a8604c0c7d-34352b907a4af",
    "ContainerConfig": {
      "Hostname": "e03bd2387319",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      "AttachStdout": false,
      "AttachStderr": false,
      "ExposedPorts": {
        "443/tcp": {},
        "80/tcp": {}
      },
      "Tty": false,
      "OpenStdin": false,
      "StdinOnce": false,
      "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
      ]
    }
  }
]
<Cut for brevity>
```

```
$ docker image history myimage
IMAGE CREATED BY SIZE COMMENT
04ae8c714993 56 minutes ago /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon... 0B
2ecd13c5af2b 57 minutes ago /bin/sh -c #(nop) VOLUME [/usr/share/nginx/... 0B
a9a1533064b2 57 minutes ago /bin/sh -c #(nop) EXPOSE 443 80 0B
4abf50fd4a02 57 minutes ago /bin/sh -c apt-get install nginx -y 60.2MB
c2701f555b0f 58 minutes ago /bin/sh -c apt-get upgrade -y 1.55MB
646cc0e9f256 58 minutes ago /bin/sh -c apt-get update 27.6MB
f83e0f07db18 58 minutes ago /bin/sh -c #(nop) MAINTAINER Cisco Champion... 0B
775349758637 3 weeks ago /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing> 3 weeks ago /bin/sh -c mkdir -p /run/systemd && echo 'do... 7B
<missing> 3 weeks ago /bin/sh -c set -xe && echo '#!/bin/sh' > /... 745B
<missing> 3 weeks ago /bin/sh -c [ -z "$(apt-get indextargets)" ] 987kB
<missing> 3 weeks ago /bin/sh -c #(nop) ADD file:a48a5dclb9dfc632... 63.2MB
```

```
$ docker container run -p 80:80 -p 443:443 -d myimage
bf0889f6b27b034427211f105e86cc1bfeae8c3b5ab279ccaf08c114e6794d94

$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bf0889f6b27b myimage "nginx -g 7 seconds ago Up 6 seconds 0.0.0.0:80->80/tcp, elastic_maxwell
'daemon of...' 0.0.0.0:443->443/tcp
```

```
$ docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
myimage latest 04ae8c714993 About an hour ago 154MB
$ docker image tag 04ae8c714993 chrijack/newrepo:firsttry
```

```
$ docker image push chrijack/newrepo:firsttry
The push refers to repository [docker.io/chrijack/newrepo]
3e92d80e0ac4: Pushed
a87bf84680fc: Pushed
02d0765ebf97: Pushed
e0b3afb09dc3: Pushed
6c01b5a53aac: Pushed
2c6ac8e5063e: Pushed
cc967c529ced: Pushed
firsttry: digest: sha256:01cf95854003cd1312fb09286d41bc6bfd9fe3fb82f63e66e5060c7fd5a
6230a size: 1786
```

```
$ nmap --help
Nmap 7.80 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host

<cut for brevity>
```

```
$ nmap -vv www.google.com
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-08 22:10 PST
Warning: Hostname www.google.com resolves to 2 IPs. Using 216.58.194.196.
Initiating Ping Scan at 22:10
Scanning www.google.com (216.58.194.196) [2 ports]
Completed Ping Scan at 22:10, 0.02s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 22:10
Completed Parallel DNS resolution of 1 host. at 22:10, 0.06s elapsed
Initiating Connect Scan at 22:10
Scanning www.google.com (216.58.194.196) [1000 ports]
Discovered open port 443/tcp on 216.58.194.196
Discovered open port 80/tcp on 216.58.194.196
Completed Connect Scan at 22:10, 8.49s elapsed (1000 total ports)
Nmap scan report for www.google.com (216.58.194.196)
Host is up, received syn-ack (0.025s latency).
Other addresses for www.google.com (not scanned): 2607:f8b0:4005:804::2004
rDNS record for 216.58.194.196: sfo03s01-in-f196.1e100.net
Scanned at 2019-12-08 22:10:05 EST for 9s
Not shown: 998 filtered ports
Reason: 998 no-responses
PORT      STATE SERVICE REASON
80/tcp    open  http   syn-ack
443/tcp   open  https  syn-ack
```

```
$ nmap -Pn --script vuln 10.168.243.179
Starting Nmap 7.80 ( https://nmap.org ) at 2019-12-08 22:18 PST
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_ Hosts are all up (not vulnerable).
Illegal character(s) in hostname -- replacing with '**'
Nmap scan report for RX-V677*B9772F.hsd1.ca.domain.net (10.168.243.179)
Host is up (0.029s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE
80/tcp    open  http
|_ clamav-exec: ERROR: Script execution failed (use -d to debug)
| http-csrf:
| Spidering limited to: maxdepth=3; maxpagecount=20; withinhost=RX-V677*B9772F.hsd1.
ca.domain.net
|   Found the following possible CSRF vulnerabilities:
|
|   Path: http://RX-V677*B9772F.hsd1.ca.domain.net:80/
|   Form id: recoveryform
|_   Form action: /Config/avr_recovery.cgi
|_ http-dombased-xss: Couldn't find any DOM based XSS.
| http-fileupload-exploiter:
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
|
|   Couldn't find a file-type field.
```

```

|
| Couldn't find a file-type field.
|
| Couldn't find a file-type field.
|
| Couldn't find a file-type field.
|
| Failed to upload and execute a payload.
|
| Failed to upload and execute a payload.
|
| Failed to upload and execute a payload.
|
| Failed to upload and execute a payload.
|
| Failed to upload and execute a payload.
|
|_ http-stored-xss: Couldn't find any stored XSS vulnerabilities.
1029/tcp open ms-lsa
|_ clamav-exec: ERROR: Script execution failed (use -d to debug)
1900/tcp open upnp
|_ clamav-exec: ERROR: Script execution failed (use -d to debug)
8080/tcp open http-proxy
|_ clamav-exec: ERROR: Script execution failed (use -d to debug)
|_ http-aspnet-debug: ERROR: Script execution failed (use -d to debug)
| http-enum:
|_ /test.html: Test page
| http-slowloris-check:
| VULNERABLE:
| Slowloris DOS attack
| State: LIKELY VULNERABLE
| IDs: CVE:CVE-2007-6750
| Slowloris tries to keep many connections to the target web server open and
| hold
| them open as long as possible. It accomplishes this by opening connections
| to
| the target web server and sending a partial request. By doing so, it starves
| the http server's resources causing Denial Of Service.
|
| Disclosure date: 2009-09-17
| References:
| https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
|_ http://ha.ckers.org/slowloris/
50000/tcp open ibm-db2
|_ clamav-exec: ERROR: Script execution failed (use -d to debug)

Nmap done: 1 IP address (1 host up) scanned in 130.29 seconds

```

nslookup [-option] [name | -] [server]

```

nslookup stanford.edu :
nslookup with a simple domain name will return the IP address of the stanford.edu

$ nslookup standford.edu
Server: 2601:647:5500:1ea:9610:3eff:fe18:22a5
Address: 2601:647:5500:1ea:9610:3eff:fe18:22a5#53

** server can't find standford.edu: NXDOMAIN

$ nslookup stanford.edu
Server: 2601:647:5500:1ea:9610:3eff:fe18:22a5
Address: 2601:647:5500:1ea:9610:3eff:fe18:22a5#53

Non-authoritative answer:
Name: stanford.edu
Address: 171.67.215.200

```

```
nslookup -type=any stanford.edu :
```

```
The -type=any parameter allows us to get all the DNS records for that domain,  
including the mail servers, etc.
```

```
$ nslookup -type=any stanford.edu  
;; Truncated, retrying in TCP mode.  
Server: 10.168.243.90  
Address: 10.168.243.90#53
```

```
Non-authoritative answer:
```

```
stanford.edu mail exchanger = 10 mxa-00000d03.gslb.pphosted.com.  
stanford.edu mail exchanger = 10 mxb-00000d03.gslb.pphosted.com.  
stanford.edu nameserver = ns5.dnsmadeeasy.com.  
stanford.edu nameserver = ns6.dnsmadeeasy.com.  
stanford.edu nameserver = ns7.dnsmadeeasy.com.  
stanford.edu nameserver = argus.stanford.edu.  
stanford.edu nameserver = atalante.stanford.edu.  
stanford.edu nameserver = avallone.stanford.edu.  
stanford.edu  
origin = argus.stanford.edu  
mail addr = hostmaster.stanford.edu  
serial = 2019138199  
refresh = 1200  
retry = 600  
expire = 1296000  
minimum = 1800
```

```
$ ansible [pattern] -m [module] -a "[module options]"
$ cat hosts
```

```
[iosxe]
```

```
10.10.30.171
```

```
[iosxe:vars]
```

```
ansible_network_os=ios
```

```
ansible_connection=network_cli
```

```
$ cat site.yml
---
- name: Test Ansible ios_command on Cisco IOS XE
  hosts: iosxe

  tasks:
    - name: show version and ip interface brief
      ios_command:
        commands:
          - show version
          - show ip interface brief
```

```
$ ansible-playbook -i hosts site.yml -u admin -k
```

```
$ ansible-playbook -i hosts site.yml -u admin -k
SSH password:

PLAY [Test Ansible ios_command on Cisco IOS XE] *****
*****

TASK [show version and ip interface brief] *****
*****
ok: [10.10.30.171]

PLAY RECAP *****
*****
10.10.30.171      : ok=1    changed=0    unreachable=0    failed=0
                  skipped=0    rescued=0    ignored=0
```

```
# Configuring the interface using Puppet
```

```
cisco_interface { "Ethernet1/3" :
    switchport_mode    =>  enabled,
}
-----
```



```

cisco_interface 'Ethernet1/3' do

    action :create
    ipv4_address '10.1.1.1'

    ipv4_netmask_length 24

    ipv4_proxy_arp true

    ipv4_redirects true

    shutdown false

    switchport_mode 'disabled'

end

```

```
$ ncs-netsim --help
```

```

Usage ncs-netsim [--dir <NetsimDir>]
    create-network <NcsPackage> <NumDevices> <Prefix> |
    create-device <NcsPackage> <DeviceName> |
    add-to-network <NcsPackage> <NumDevices> <Prefix> |
    add-device <NcsPackage> <DeviceName> |
    delete-network |
    [-a | --async] start [devname] |
    [-a | --async ] stop [devname] |
    [-a | --async ] reset [devname] |
    [-a | --async ] restart [devname] |
    list |
    is-alive [devname] |
    status [devname] |
    whichdir |
    ncs-xml-init [devname] |
    ncs-xml-init-remote <RemoteNodeName> [devname] |
    [--force-generic] |
    packages |
    netconf-console devname [XPathFilter] |
    [-w | --window] [cli | cli-c | cli-i] devname |
    get-port devname [ipc | netconf | cli | snmp]

See manpage for ncs-netsim for more info. NetsimDir is optional and defaults to
./netsim, any netsim directory above in the path, or $NETSIM_DIR if set.

```

```

ncs-netsim create-network <NCS package> <#N devices> <PrefixY>
$ ncs-netsim create-network $NCS_DIR/packages/neds/cisco-
ios-cli-3.8 3 ios

```

```
DEVICE ios0 CREATED
```

```
DEVICE ios1 CREATED
```

```
DEVICE ios2 CREATED
```

```
$ ncs-netsim list
```

```
ncs-netsim list for $NCS_DIR/nso-run/netsim
```

```

name=ios0 netconf=12022 snmp=11022 ipc=5010 cli=10022 dir=$NCS_DIR/
nso-run/netsim/ios/ios0

```

```

name=ios1 netconf=12023 snmp=11023 ipc=5011 cli=10023 dir=$NCS_DIR
/nso-run/netsim/ios/ios1

```

```

name=ios2 netconf=12024 snmp=11024 ipc=5012 cli=10024 dir=$NCS_DIR
/nso-run/netsim/ios/ios2
-----

```

```
$ ncs-netsim cli-i ios0

admin connected from 127.0.0.1 using console on *
ios0> en
ios0# show running-config
no service pad
no ip domain-lookup
no ip http server
no ip http secure-server
ip routing
ip source-route
ip vrf my-forward
  bgp next-hop Loopback 1
!
```

```
$ ncs_cli -C -u admin

admin connected from 127.0.0.1 using console on *
admin@ncs# config term
Entering configuration mode terminal
admin@ncs(config)# devices device ios0
admin@ncs(config-device-ios0)# address 127.0.0.1
admin@ncs(config-device-ios0)# port 10022
admin@ncs(config-device-ios0)# authgroup default
admin@ncs(config-device-ios0)# device-type cli ned-id cisco-ios-cli-3.8
admin@ncs(config-device-ios0)# state admin-state unlocked
admin@ncs(config-device-ios0)# exit
```

```
$ curl --request GET 'http://localhost:8080/restconf/data/taif-ncs:devices/device' \
```

```
--header 'Content-Type: application/yang-data+json' \
```

```
--header 'Authorization: Basic YWRtaW46YWRtaW4='
```

```
{
  "taif-ncs:device": [
    {
      "name": "ios0",
      "address": "127.0.0.1",
      "port": 10022,
      "authgroup": "default",
      "device-type": {
        "cli": {
          "ned-id": "cisco-ios-cli-3.8:cisco-ios-cli-3.8"
        }
      },
      "commit-queue": {
        "queue-length": 0
      },
      "active-settings": {
        "connect-timeout": 20,
        "read-timeout": 20,
        "write-timeout": 20,
        "ssh-keep-alive": {
          "interval": 20,
          "count": 3
        },
        "ned-keep-alive": {
          "count": 3
        }
      },
      ... omitted output
    }
  ]
}
```

```
lab:
  description: ''
  notes: ''
  timestamp: 1581966586.8872395
  title: DEVASC official guide
  version: 0.0.3
nodes:
- id: n0
  label: iosv-0
  node_definition: iosv
  x: -500
  y: 50
  configuration: ''
  image_definition: iosv-158-3
  tags: []
  interfaces:
  - id: i0
    label: Loopback0
    type: loopback
  - id: i1
    slot: 0
    label: GigabitEthernet0/0
    type: physical
- id: n1
  label: csr1000v-0
  node_definition: csr1000v
...omitted output
```

```
---
devices:
  csr1000v-1:
    type: 'router'
    os: 'iosxe'
    platform: asrlk
    alias: 'uut'
    credentials:
      default:
        username: vagrant
        password: vagrant
    connections:
      cli:
        protocol: ssh
        port: 2222
        ip: "127.0.0.1"
```

```
#!/usr/bin/env python
from genie.testbed import load

# Load the testbed
tb = load('testbed.yaml')

# Find the device with alias uut
dev = tb.devices['uut']

# Connect to the device
dev.connect()

# Parse the output of the show version command
output = dev.parse('show version')

# Extract the version number and print it to the screen
print('IOS-XE version number: ' + output['version']['version'])
```