Practice Tests

Flash Cards

Study Planner

Review Exercises

Official Cert Guide

Advance your IT career with hands-on learning

Cisco Certified DevNet Professional

DEVCOR 350-901

Jason Davis
Hazim Dahir
Stuart Clark
Quinn Snyder

ciscopress.com

FREE SAMPLE CHAPTER

Cisco Certified DevNet Professional

DEVCOR 350-901

Official Cert Guide

JASON DAVIS,
HAZIM DAHIR,
STUART CLARK,
QUINN SNYDER

Cisco Press

Cisco Certified DevNet Professional DEVCOR 350-901 Official Cert Guide

Jason Davis

Hazim Dahir

Stuart Clark

Quinn Snyder

Published by:
Cisco Press

ScoutAutomatedPrintCode

Library of Congress Control Number: 2022933631

ISBN-13: 978-0-13-737044-3

ISBN-10: 0-13-737044-X

Warning and Disclaimer

This book is designed to provide information about the Cisco DevNet Professional DEVCOR 350-901 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 author 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 Product Management: Brett Bartow

Executive Editor: Nancy Davis

Managing Editor: Sandra Schroeder

Development Editor: Ellie Bru

Senior Project Editor: Tonya Simpson

Copy Editor: CAH Editing

Technical Editors: Bryan Byrne, Joe Clark

Editorial Assistant: Cindy Teeters

Cover Designer: Chuti Prasertsith

Composition: Codemantra

Indexer: Cheryl Lenser

Proofreader: Donna E. 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.**

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where

- Everyone has an equitable and lifelong opportunity to succeed through learning
- Our educational products and services are inclusive and represent the rich diversity of learners
- Our educational content accurately reflects the histories and experiences of the learners we serve
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview)

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.

About the Authors

Jason Davis is a distinguished engineer for the DevNet program in the Developer Relations organization at Cisco. His role is technical strategy lead for the DevRel organization as he collaborates with various Cisco business unit leaders, partners, customers, and other industry influencers. Jason is focused on automation, orchestration, cloud-native technologies, and network management/operations technologies. He has a tenured career working with hundreds of customers, worldwide, in some of the largest network automation and management projects and is sought out for consulting and innovative leadership. His former experience as a U.S. Army Signal Corps officer has provided insights to defense, government, and public-sector projects, while his extensive work in professional services at Cisco has spanned commercial, large-enterprise, and service provider segments. Most of his customer engagements have been in automotive, manufacturing, large retail, large event venues, and health care. Jason has achieved Cisco Live Distinguished Speaker Hall of Fame status and is an automation/monitoring lead for the network operations center (NOC) at Cisco Live events in the U.S. and Europe. He resides in Apex, North Carolina, and enjoys IoT projects, home automation, and audio/video technologies in houses of worship. Jason and his wife, Amy, have four children whom they homeschool and cherish daily. Jason is found on social media @SNMPguy.

Hazim Dahir, CCIE No. 5536, is a distinguished engineer at the Cisco office of the CTO. He is working to define and influence next-generation digital transformation architectures across multiple technologies and verticals. Hazim started his Cisco tenure in 1996 as a software engineer and subsequently moved into the services organization focusing on large-scale network designs. He's currently focusing on developing architectures utilizing security, collaboration, Edge computing, and IoT technologies addressing the future of work and hybrid cloud requirements for large enterprises. Through his passion for engineering and sustainability, Hazim is currently working on advanced software solutions for electric and autonomous vehicles with global automotive manufacturers. Hazim is a Distinguished Speaker at Cisco Live and is a frequent presenter at multiple global conferences and standards bodies. He has multiple issued and pending patents and a number of innovation and R&D publications.

Stuart Clark, DevNet Expert #2022005, started his career as a hairdresser in 1990, and in 2008 he changed careers to become a network engineer. After cutting his teeth in network operations, he moved to network projects and programs. Stuart joined Cisco in 2012 as a network engineer, rebuilding one of Cisco's global networks and designing and building Cisco's IXP peering program. After many years as a network engineer, Stuart became obsessed with network automation and joined Cisco DevNet as a developer advocate for network automation. Stuart contributed to the DevNet exams and was part of one of the SME teams that created, designed, and built the Cisco Certified DevNet Expert. Stuart has presented at more than 50 external conferences and is a multitime Cisco Live Distinguished Speaker, covering topics on network automation and methodologies. Stuart lives in Lincoln, England, with his wife, Natalie, and their son, Maddox. He plays guitar and rocks an impressive two-foot beard while drinking coffee. Stuart can be found on social media
@bigevilbeard.

Quinn Snyder is a developer advocate within the Developer Relations organization inside Cisco, focusing on datacenter technologies, both on-premises and cloud-native. In this role, he aligns his passion for education and learning with his enthusiasm for helping the infrastructure automation community grow and harness the powers of APIs, structured data, and programmability tooling. Prior to his work as a DA, Quinn spent 15 years in a variety of design, engineering, and implementation roles across datacenter, utility, and service provider customers for both Cisco and the partner community. Quinn is a proud graduate of Arizona State University (go Sun Devils!) and a Cisco Network Academy alumnus. He is the technical co-chair of the SkillsUSA–Arizona Internetworking contest and is involved with programmability education at the state and regional level for the Cisco Networking Academy. Quinn resides in the suburbs of Phoenix, Arizona, with his wife, Amanda, and his two kids. In his free time, you can find him behind a grill, behind a camera lens, or on the soccer field coaching his daughter's soccer teams. Quinn can be found on social media @qsnyder, usually posting a mixture of technical content and his culinary creations.

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 a multitime 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.

Joe Clarke, CCIE No. 5384, is a Cisco Distinguished Customer Experience engineer. Joe has contributed to the development and adoption of many of Cisco's network operations and automation products and technologies. Joe evangelizes these programmability and automation skills in order to build the next generation of network engineers. Joe is a Distinguished Speaker at CiscoLive!, and is certified as a CCIE and a Cisco Certified DevNet Specialist (and a member of the elite DevNet 500). Joe provides network consulting and design support for the CiscoLive! and Internet Engineering Task Force (IETF) conference network infrastructure deployments. He also serves as co-chair of the Ops Area Working Group at the IETF. He is a coauthor of *Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI* as well as a chapter coauthor in the Springer publication *Network-Embedded Management and Applications: Understanding Programmable Networking Infrastructure*. He is an alumnus of the University of Miami and holds a bachelor of science degree in computer science. Outside of Cisco, Joe is a commercial pilot, and he and his wife, Julia, enjoy flying around the east coast of the United States.

Dedications

Jason Davis:

When you set off to write a book, how do you quantify the time needed to study, write, and refine? You need time to obtain and configure the lab equipment for examples. There are still family, career, personal health, and welfare activities that demand attention. How can you plan when your job and a global health crisis demand the utmost in flexibility and focus? To all the family and supporters of writers, thank you; there should be a special place in heaven for you. I am indebted to my wife, Amy, and my children, Alyssa, Ryan, Micaela, and Maleah. You provided me with the time and space to do this, especially through the hardships of 2021. Finally, I am thankful that my God has provided opportunities and skills to impact technology in some small part and in a positive way; because He has shown His love for me in innumerable ways, I am motivated to share the same with others.

Hazim Dahir:

To my father, my favorite librarian: I wish this book could have made it to your shelf. I think of you every day. To my mother, with heartfelt gratitude for all your love, prayers, and guidance. I am a better person, husband, father, brother, and engineer because of you two.

To my amazing wife, Angela: no words or pages can grasp how indebted I am to you for your encouragement, patience, and wisdom.

To my children, Hala, Leila, and Zayd. I love watching you grow. Thank you for being such a joy. Never give up on your dreams.

To my sisters, Hana and Dina, and brother Gary. My team.

To Bill, Sylvia, and Karen Moseley. We can do this!

Stuart Clark:

This book is dedicated to my amazing Natalie (Mouse) and our son, Maddox, and our beloved dog, Bailey, who sadly passed while I was authoring this book. Without their love, support, and countless cups of coffee, this book would have never been possible. I would like to thank my father, George, and mother, Wendy Clark, for providing me with the work ethic I have today; and Natalie's father and mother, Frank and Barbara Thomas, for giving me my very first networking book and helping me transition into a new career. A big thank you to my mentors Mandy Whaley, Denise Fishburne, Joe Clarke, and my metaldevops brother Jason Gooley, who have helped and guided me for many years at Cisco.

Quinn Snyder:

Writing a book is an unknown unknown to a first-time author. You have read the products from others, but you don't know what you don't know when you set out to write your own. The only known you have is the support of the people behind you, who make it possible to think, to work, to create, and to inspire you. They are there for your late nights, the constant tapping of the keys, listening to you ramble on about what you've just put to paper. This book would not have been possible without the support of those behind me, propping me up, cheering me along, and believing in me. To my wife, Amanda, you are my rock—my biggest fan and supporter—and the one who has always believed in me, and I wouldn't have been able to do this without you. To my children, Nicolas and Madelyn, you have given me the space to create and do, and have inspired me to show you that anything is possible. To my mother, Cynthia, you have inspired a never-quit work ethic that sticks with me to this day. I thank you all for giving me those gifts, and I dedicate this book to you.

Acknowledgments

Jason Davis:
Most who know me understand that I appreciate telling stories, especially with humor. While that humor may not be recognized, it is mine nonetheless, so thank you to those who have endured it for years. I am thankful for the opportunity to pivot from writing blogs, whitepapers, and presentations to writing this book. It has been an exercise in patience and personal growth. The support team at Cisco Press—Nancy, Ellie, and Tonya—you've been great, valuable partners in this endeavor.

I am thankful to a cadre of supportive managers at Cisco over the years who helped me grow technically and professionally; Mike, Dave, Rich, and Ghaida, you have been awesome! Hazim, Stuart, and Quinn, thank you for joining me on this journey. Besides being wonderful coworkers, our collaboration, though separated by states and countries, has turned into friendship that has made a mark.

Hazim Dahir:
This book would not have been possible without two SUPER teams. The A Team: Jason, Stuart, and Quinn. Thank you for an amazing journey and expert work. "Technically," you complete me! And, the Cisco Press team: Nancy Davis, Ellie Bru, Tonya Simpson, and their teams. Thank you for all your help, guidance, and most importantly, your patience.

Special thanks go to Kaouther Abrougui for her contributions to the Security chapter, Mithun Baphana for his Webex contributions, and to David Wang for his Git contributions. Kaouther, Mithun, and David are experts in their fields, and we're lucky to have them join us.

Many thanks go to Firas Ahmed, for his encouragement, technical reviews, and support.

A great deal of gratitude goes to Hana Dahir, an engineer at heart, who reviewed various content for me.

I am very thankful to the following Cisco colleagues and leaders: Yenu Gobena, Saad Hasan, Carlos Pignataro, Jeff Apcar, Vijay Raghavendran, Ammar Rayes, Nancy Cam-Winget, and many others who supported me throughout my career and encouraged me to break barriers.

Stuart Clark:
Being a first-time author was such a daunting task, but the great people at Cisco Press, Nancy and Ellie; my coauthors, Hazim, Jason, and the BBQ Pit Master Quinn; technical reviewers, Joe Clarke and Bryan Byrne, kept me in check and supporting every chapter. A huge thank you to my leadership at DevNet—Matt Denapoli, Eric Thiel, and Grace Francisco—for always supporting my goals and ambitions. Janel Kratky, Kareem Iskander, and Hank Preston for their vast support and encouragement since joining the DevNet team in 2017. My dearest friends, Patrick Rockholz and Du'An Lightfoot, whose faith in me is always unfaltering.

Quinn Snyder:
I'd like to thank the crew at Cisco Press for wading through my incoherent ramblings and turning them into something worth reading. To Ellie and Nancy, thank you for the patience and dealing with my inane questions. To Joe Clarke, thank you for your technical expertise in the review process.

Thank you to my DevNet leaders—Grace, Eric, Matt, and Jeff—for bringing me into this amazing team and supporting my growth and development to make this work possible.

A special thanks to the guy who taught me "how to DA," John McDonough. You set me on a path for success and showed me the ropes, and for that I am grateful. To Kareem Iskander, thanks for just being there. You always have listened to me, picked me up, and pushed me forward. To my partners in crime (both with and without beards)—Jason, Hazim, and Stuart—you guys believed that I could and gave me a shot. I have appreciated our meetings, our late-night messaging sessions, and the friendship that has developed.

Finally, a special thank you to the man who guided me so many years ago, Barry Williams. Without your Cisco classes, your instruction, your belief, and never accepting "just good enough," I wouldn't have had the foundation that I do today. You helped a kid from rural Arizona see the big world and what was possible if I stuck with it, and because of that, I am where I am today.

Contents at a Glance

Online Elements

Reader Services

Contents

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 preparing to take the DevNet Professional DEVCOR 350-901 exam but also for real-world skills in any production environment.

You can expect that the blueprint for the DevNet Professional DEVCOR 350-901 exam tightly aligns with the topics contained in this book. This was by design. You 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.

We are targeting any and all learners who are learning these topics for the first time as well as 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 Professional DEVCOR 350-901 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 Professional DEVCOR 350-901 exam. In fact, if the primary objective of this book were different, then the book's title would be misleading; however, the methods used in this book to help you pass the DevNet Professional exam are designed to also make you much more knowledgeable about how to do your job. Although 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 Professional 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 Professional 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 Professional DEVCOR 350-901 exam. Not only can this book help you pass the exam, but it can also help you learn the necessary topics to provide value to your organization as a network developer.

Passing the DevNet Professional DEVCOR 350-901 exam is a milestone toward becoming a better network developer, which, in turn, can help with becoming more confident with these technologies.

Strategies for Exam Preparation

The strategy you use for the DevNet Professional 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, the 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 need to be convinced 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, exist on this book's companion website.

How to Access the Companion Website

To access the companion website, which gives you access to the electronic content with this book, 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: **9780137370443**. 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.

When 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.

Note that if you want to use the web app only at this point, just navigate to www.pearsontestprep.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 to Use This Book

Although this book could 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. This book was written to include not only reference materials and study guides for the exam but also rich reference material for your day-to-day technical requirements.

The core chapters, Chapters 1 through 17, cover the following topics:

- **Chapter 1, "Software Development Essentials":** This chapter introduces software architecture, architecture requirements, and software development models.
- **Chapter 2, "Software Quality Attributes":** This chapter discusses in detail quality attributes or nonfunctional requirements and discusses in detail modularity, scalability, and high availability in application design.
- **Chapter 3, "Architectural Considerations and Performance Management":** This chapter continues to discuss another set of nonfunctional requirements and how they relate to design trade-offs. It discusses performance, observability, and database selection criteria.
- **Chapter 4, "Version Control and Release Management with Git":** This chapter discusses the basics of version control, Git's way of managing version controls and collaboration, and then covers in detail branching strategies and why they're important for the success of any project.
- **Chapter 5, "Network APIs":** This chapter covers how software developers can use application programming interfaces (APIs) to communicate with and configure networks and how APIs are used to communicate with applications and other software.
- **Chapter 6, "API Development":** This chapter focuses on application programming interface development and covers both API design and API architecture.
- **Chapter 7, "Application Deployment":** This chapter covers the code-to-production process, including organizational structures, responsibilities, and tooling required. Historical as well as current deployment models are discussed, as well as design factors to enable portable applications between hosting locations.
- **Chapter 8, "Security in Application Design":** This chapter discusses security practices for application development. It starts by defining privacy and personally identifiable information and how to protect them. Then it covers the public key infrastructure (PKI), how to secure web applications, and the OAuth Authorization framework.

- **Chapter 9, "Infrastructure":** This chapter covers aspects of network infrastructure management and automation. Some historical context is provided, but exam preparation is focused on newer programmability features that enable automation and orchestration.
- **Chapter 10, "Automation":** This chapter covers topics such as SDN, APIs, and orchestration. Additional helpful context is provided around the impact to IT service management.
- **Chapter 11, "NETCONF and RESTCONF":** This chapter covers the NETCONF, YANG, and RESTCONF technologies with examples that will be helpful in your preparation and professional use.
- **Chapter 12, "Model-Driven Telemetry":** This chapter is focused on model-driven telemetry, its purpose, and how it is implemented. In support of your learning, exam preparation, and professional use, there are also examples for using MDT.
- **Chapter 13, "Open-Source Solutions":** This chapter covers several open-source solutions that are helpful in many environments. Examples for deployment and usage provide insight and help inform your implementation decisions.
- **Chapter 14, "Software Configuration Management":** This chapter discusses software configuration management: what is it, why is it important, and how do you decide which system is best for your project? We also discuss Ansible and Terraform and their strengths and weaknesses.
- **Chapter 15, "Hosting an Application on a Network Device":** This chapter provides insights on how to run containerized workloads on a network device. Some best practices are also shared to encourage your best uses.
- **Chapter 16, "Cisco Platforms":** Finally, this chapter contains a mix of practical API and SDK usage examples across several platforms, such as Webex, Meraki, Intersight, DNA Center, and AppDynamics. If you have some of these solutions, the examples should reveal methods to integrate with them programmatically. If you don't use the platforms, this chapter should reveal the "art of the possible."
- **Chapter 17, "Final Preparation":** This chapter details a set of tools and a study plan to help you complete your preparation for the DEVCOR 350-901 exam.

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 the DevNet Professional DEVCOR 350-901 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 DEVCOR 350-901 Exam Topics and Chapter References

DEVCOR 350-901 Exam Topic	Chapter(s) in Which Topic Is Covered
1.0 Software Development and Design	Chapter 1
1.1 Describe distributed applications related to the concepts of front-end, back-end, and load balancing	Chapter 2
1.2 Evaluate an application design considering scalability and modularity	Chapter 2
1.3 Evaluate an application design considering high availability and resiliency (including on-premises, hybrid, and cloud)	Chapter 2
1.4 Evaluate an application design considering latency and rate limiting	Chapter 3
1.5 Evaluate an application design and implementation considering maintainability	Chapter 3
1.6 Evaluate an application design and implementation considering observability	Chapter 3
1.7 Diagnose problems with an application given logs related to an event	Chapter 3
1.8 Evaluate choice of database types with respect to application requirements (such as relational, document, graph, columnar, and time series)	Chapter 3
1.9 Explain architectural patterns (monolithic, services oriented, microservices, and event driven)	Chapter 2
1.10 Utilize advanced version control operations with Git 1.10.a Merge a branch 1.10.b Resolve conflicts 1.10.c git reset 1.10.d git checkout 1.10.e git revert	Chapter 4
1.11 Explain the concepts of release packaging and dependency management	Chapter 4
1.12 Construct a sequence diagram that includes API calls	Chapter 5
2.0 Using APIs	Chapter 5
2.1 Implement robust REST API error handling for timeouts and rate limits	Chapter 6
2.2 Implement control flow of consumer code for unrecoverable REST API errors	Chapter 6
2.3 Identify ways to optimize API usage through HTTP cache controls	Chapter 6
2.4 Construct an application that consumes a REST API that supports pagination	Chapter 6
2.5 Describe the steps in the OAuth2 three-legged authorization code grant flow	Chapter 8

DEVCOR 350-901 Exam Topic	Chapter(s) in Which Topic Is Covered
3.0 Cisco Platforms	Chapter 16
3.1 Construct API requests to implement ChatOps with Webex Teams API	Chapter 16
3.2 Construct API requests to create and delete objects using Firepower device management (FDM)	Chapter 16
3.3 Construct API requests using the Meraki platform to accomplish these tasks 3.3.a Use Meraki Dashboard APIs to enable an SSID 3.3.b Use Meraki location APIs to retrieve location data	Chapter 16
3.4 Construct API calls to retrieve data from Intersight	Chapter 16
3.5 Construct a Python script using the UCS APIs to provision a new UCS server given a template	Chapter 16
3.6 Construct a Python script using the Cisco DNA Center APIs to retrieve and display wireless health information	Chapter 16
3.7 Describe the capabilities of AppDynamics when instrumenting an application	Chapter 16
3.8 Describe steps to build a custom dashboard to present data collected from Cisco APIs	Chapter 16
4.0 Application Deployment and Security	Chapter 8
4.1 Diagnose a CI/CD pipeline failure (such as missing dependency, incompatible versions of components, and failed tests)	Chapter 7
4.2 Integrate an application into a prebuilt CD environment leveraging Docker and Kubernetes	Chapter 7
4.3 Describe the benefits of continuous testing and static code analysis in a CI pipeline	Chapter 7
4.4 Utilize Docker to containerize an application	Chapter 15
4.5 Describe the tenets of the “12-factor app”	Chapter 7
4.6 Describe an effective logging strategy for an application	Chapter 7
4.7 Explain data privacy concerns related to storage and transmission of data	Chapter 8
4.8 Identify the secret storage approach relevant to a given scenario	Chapter 8
4.9 Configure application specific SSL certificates	Chapter 8
4.10 Implement mitigation strategies for OWASP threats (such as XSS, CSRF, and SQL injection)	Chapter 8
4.11 Describe how end-to-end encryption principles apply to APIs	Chapter 8

DEVCOR 350-901 Exam Topic	Chapter(s) in Which Topic Is Covered
5.0 Infrastructure and Automation	Chapters 9 and 10
5.1 Explain considerations of model-driven telemetry (including data consumption and data storage)	Chapter 12
5.2 Utilize RESTCONF to configure a network device including interfaces, static routes, and VLANs (IOS XE only)	Chapter 11
5.3 Construct a workflow to configure network parameters with: 5.3.a Ansible playbook 5.3.b Puppet manifest	Chapter 13
5.4 Identify a configuration management solution to achieve technical and business requirements	Chapter 14
5.5 Describe how to host an application on a network device (including Catalyst 9000 and Cisco IOx-enabled devices)	Chapter 15

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/9780137370443. 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.

Credits

Figure 4-1 through Figure 4-29, Figure 4-32 through Figure 4-34, Figure 6-8, Figure 6-9, Figure 11-10, Figure 11-11, Figure 12-6, Figure 12-27, Figure 12-28: GitHub, Inc

Figure 5-3 through Figure 5-6, Figure 6-10, Figure 6-11, Figure 6-13: IMDb-API

Figure 5-8, Figure 11-14 through Figure 11-19, Figure 16-16 through Figure 16-19, Figure 16-31, Figure 16-41 through Figure 16-43: Postman, Inc

Figure 6-1 through Figure 6-7: SmartBear Software

Figure 7-3 through Figure 7-5: HootSuite Media Inc

Figure 7-7, Figure 7-8: Weaveworks, Inc

Figure 7-10, Figure 7-11: Jupyter

Figure 7-9, Figure 7-12 through Figure 7-16: Amazon Web Services, Inc

Figure 8-7 through Figure 8-10: DigiCert, Inc

Figure 9-3: Coleman Yuen/Pearson Education Asia Limited

Figure 9-5: vystekimages/Shutterstock

Figure 9-7: faithie/123RF

Figures 9-13 and 9-14, arm icons: De-V/Shutterstock

Figure 10-2: Twitter, Inc

Figure 10-3: Natata/Shutterstock

Figure 10-4: Rawpixel.com/Shutterstock

Figure 10-5: Andrey_Popov/Shutterstock

Figure 10-7 icons: Sergii Korolko/Shutterstock, Vitaly Korovin/Shutterstock, Gazlast/Shutterstock, dodi31/Shutterstock, Stokkete/Shutterstock, Pavel Ignatov/Shutterstock

Figure 10-12: Tribalium/Shutterstock, KJBevan/Shutterstock

Figure 12-21 through Figure 12-25, Figure E-1, Figure E-2: Grafana Labs

Logo in Figure 13-3: Puppet

Figure 13-5: Puppet

Figure 13-6, Figure 14-6 through Figure 14-8: HashiCorp

Figure 16-44, Figure 16-45: AppDynamics

Table 2-2: Permission to reproduce extracts from British Standards is granted by BSI Standards Limited (BSI). No other use of this material is permitted. British Standards can be obtained in PDF or hard copy formats from the BSI online shop: https://shop.bsigroup.com/.

CHAPTER 10

Automation

This chapter covers the following topics:

- **Challenges Being Addressed:** This section identifies the challenges that need to be addressed with the advent of software-defined networking, DevOps, and network programmability.
- **Software-Defined Networking (SDN):** This section covers the genesis of software-defined networking, its purpose, and the value gained.
- **Application Programming Interfaces (APIs):** This section provides guidance around application programming interfaces. Network engineering and operations were very different before APIs; some environments are still resistant to change, but the benefits outweigh the risks of not embracing them.
- **REST APIs:** This section provides insights to the functionality and benefit of REST APIs and how they are used.
- **Cross-Domain, Technology-Agnostic Orchestration:** This section contains material that is not covered in the DEVCOR certification test. However, as network IT continues to transform, it provides an important consideration for the transformation of environments.
- **Impact to IT Service Management and Security:** This section acknowledges the influence of IT service management and security to network programmability. With so many companies investing in ITIL and TOGAF methodologies in the early 2010s, understanding the alignments is helpful.

This chapter maps to the second part of the *Developing Applications Using Cisco Core Platforms and APIs v1.0 (350-901)* Exam Blueprint Section 5.0, "Infrastructure and Automation."

As we've learned about the infrastructure involved in network IT and see the continued expansion, we also recognize that static, manual processes can no longer sustain us. When we were managing dozens or hundreds of devices using manual methods of logging in to terminal servers, through a device's console interface, or through inband connectivity via SSH, it may have been sufficient. However, now we are dealing with thousands, tens of thousands, and in a few projects I've been on, *hundreds* of thousands of devices. It is simply untenable to continue manual efforts driven by personal interaction. At some point, these valuable engineering, operations, and management resources must be refocused on more impactful activities that differentiate the business. So, automation must be embraced. This chapter covers some key concepts related to automation: what challenges need to be addressed, how SDN and APIs enable us, and the impact to IT service management and security.

“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?’ Quizzes.”

Table 10-1 “Do I Know This Already?” Section-to-Question Mapping

Foundation Topics Section	Questions
Challenges Being Addressed	1–5
Software-Defined Networking (SDN)	6
Application Programming Interfaces (APIs)	11
REST APIs	7–10

1. When you are considering differences in device types and function, which technology provides the most efficiencies?
   - **a.** Template-driven management
   - **b.** Model-driven management
   - **c.** Atomic-driven management
   - **d.** Distributed EMSs
2. The SRE discipline combines aspects of _______ engineering with _______ and _______.
   - **a.** Hardware, software, firmware
   - **b.** Software, infrastructure, operations
   - **c.** Network, software, DevOps
   - **d.** Traffic, DevOps, SecOps
3. What do the Agile software development practices focus on?
   - **a.** Following defined processes of requirements gathering, development, testing, QA, and release.
   - **b.** Giving development teams free rein to engineer without accountability.
   - **c.** Pivoting from development sprint to sprint based on testing results.
   - **d.** Requirements gathering, adaptive planning, quick delivery, and continuous improvement.
4. Of the software development methodologies provided, which uses a more visual approach to the what-when-how of development?
   - **a.** Kanban
   - **b.** Agile
   - **c.** Waterfall
   - **d.** Illustrative

5. Concurrency focuses on _______ lots of tasks at once. Parallelism focuses on _______ lots of tasks at once.
   a. Doing; working with
   b. Exchanging; switching
   c. Threading; sequencing
   d. Working with; doing
6. The _______ specification, originally the _______ specification, defines a model for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services.
   a. OpenAPI; Swagger
   b. REST; CLI
   c. SDN; Clean Slate
   d. OpenWeb; CORBA
7. What would be the correct method to generate a basic authentication string on a macOS/Linux CLI?
   a. **echo -n 'username:password' | openssl md5**
   b. **echo -n 'username:password' | openssl**
   c. **echo -n 'username:password' | openssl base64**
   d. **echo -n 'username&password' | openssl base64**
8. What does XML stand for?
   a. Extendable machine language
   b. Extensible markup language
   c. Extreme machine learning
   d. Extraneous modeling language
9. In JSON, what are records or objects denoted with?
   a. Angle braces < >
   b. Square brackets [ ]
   c. Simple quotes “ ”
   d. Curly braces { }
10. Which REST API HTTP methods are *both* idempotent?
    a. PATCH, POST
    b. HEAD, GET
    c. POST, OPTIONS
    d. PATCH, HEAD
11. Which are APIs? (Choose two.)
    a. REST
    b. RMON
    c. JDBC
    d. SSH

Foundation Topics

Challenges Being Addressed

As described in the chapter introduction, automation is a necessity for growing sophisticated IT environments today. Allow me to share a personal example: if you've been to a CiscoLive conference in the US, it is common to deploy a couple thousand wireless access points in the large conference venues in Las Vegas, San Diego, and Orlando. I'm talking a million square feet plus event spaces.

Given that the network operations center (NOC) team is allowed onsite only four to five days before the event starts, that's not enough time to manually provision everything with a couple dozen event staff volunteers. The thousands of wireless APs are just one aspect of the event infrastructure (see Figure 10-1). There are still the 600+ small form-factor switches that must be spread across the venue to connect breakout rooms, keynote areas, World of Solutions, testing facilities and labs, the DevNet pavilion, and other spaces (see Figure 10-2).

Figure 10-1 *Moving a Few Wireless APs*

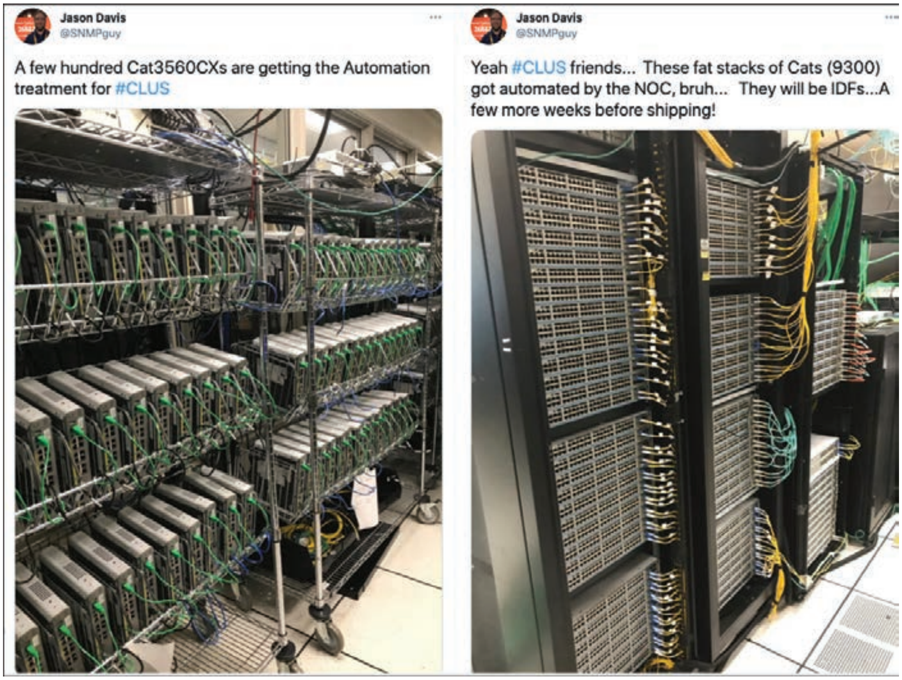

Figure 10-2 *Lots of Equipment to Stage*

Automation is a "do or die" activity for our businesses: without it, we overwork individuals and that impacts the broader organization. Automation must also extend beyond provisioning into the wide-scale collection of performance, health, and fault information.

Discerning companies are investigating how artificial intelligence and machine learning (AI/ML) can benefit them in obtaining new operational insights and reducing human effort even more.

We might even acknowledge "change is hard and slow." If you started networking after prior experience with a more programmatic environment or dealt with other industries where mass quantities of devices were managed effectively, you might wonder why network IT lags. This is a fair question, but also to be fair, enormous strides have been made in the last 10 years with an industry that found its start in ARPANET at the end of the 1960s. Cisco incorporated in 1984, and the industry has been growing in scale and functionality ever since.

Being involved in the latter part of the first wave of network evolution has been a constant career of learning and advancing skills development. The change and expansion of scope and function with networking have been very interesting and fulfilling for me.

Differences of Equipment and Functionality

Some of the challenges with networking deal with the diversity of equipment and functionality. In the last part of the 1960s and early 1970s, the aforementioned ARPANET included few network protocols and functions. A router's purpose was to move traffic across different, isolated network segments of specialized endpoints. The industry grew with shared media technologies (hubs), then to switches. Businesses started acquiring their own servers; they weren't limited to government agencies and the development labs of colleges and universities. Slowly, home PCs contributed to a burgeoning technology space.

Connectivity technology morphed from more local-based technologies like token ring and FDDI to faster and faster Ethernet-based solutions, hundred megabit and gigabit local interfaces, also influencing the speed of WAN technologies to keep up.

Switches gave advent to more intelligent routing and forwarding switches. IP-based telephony was developed. Who remembers that Cisco's original IP telephony solution, Call Manager, was originally delivered as a compact disc (CD), as much software was?

Storage was originally directly connected but then became networked, usually with different standards and protocols. The industry then accepted the efficiencies of a common, IP-based network. The rise of business computing being interconnected started influencing home networking. Networks became more interconnected and persistent. Dial-up technologies and ISDN peaked and started a downward trend in light of always-on cable-based technologies to the home. Different routing protocols needed to be created. Multiple-link aggregation requirements needed to be standardized to help with resiliency.

Wireless technologies came on the scene. Servers, which had previously been mere endpoints to the network, now became more integrated. IPv6. Mobile technologies. A lot of hardware innovations but also a lot of protocols and software developments came in parallel. So why the history lesson? Take them as cases in point of why networking IT was slow in automation. The field was changing rapidly and growing in functionality. The scope and pace of change in network IT were unlike those in any other IT disciplines.

Unfortunately, much of the early development relied on consoles and the expectation of a human administrator always creating the service initially and doing the sustaining changes. The Information Technology Information Library (ITIL) and The Open Group Architecture Framework (TOGAF) service management frameworks helped the industry define structure and operational rigor. Some of the concepts seen in Table 10-2 reflect a common vocabulary being established.

Table 10-2 Operational Lifecycle

Operational Perspective	Function
Day-0	Initial installation
Day-1	Configuration for production purpose
Day-2	Compliance and optimization
Day-X	Migration/decommissioning

The full lifecycle of a network device or service must be considered. All too often the "spin-up" of a service is the sole focus. Many IT managers have stories about finding orphaned recurring charges from decommissioned systems. Migrating and decommissioning a service are just as important as the initial provisioning. We must follow up on reclaiming precious consumable resources like disk space, IP addresses, and even power.

In the early days of compute virtualization, Cisco had an environment called CITEIS—Cisco IT Elastic Infrastructure Services, which were referred to as "cities." CITEIS was built to promote learning, speed development, and customer demos, and to prove the impact of automation. A policy was enacted that any engineer could spin up two virtual machines of any kind as long as they conformed to predefined sizing guidelines. If you needed something different, you could get it, but it would be handled on an exception basis. Now imagine the number of people excited to learn a new technology all piling on the system. VMs were spun up;

CPU, RAM, disk space, and IP addresses consumed; used once or twice, then never accessed again. A lot of resources were allocated. In the journey of developing the network programmability discipline, network engineers also needed to apply operational best practices. New functions were added to email (and later send chat messages to) the requester to ensure the resources were still needed. If a response was not received in a timely fashion, the resources were archived and decommissioned. If no acknowledgment came after many attempts over a longer period, the archive may be deleted. These kinds of basic functions formed the basis of standard IT operations to ensure proper use and lifecycle management of consumable resources.

With so many different opportunities among routing, switching, storage, compute, collaboration, wireless, and such, it's also understandable that there was an amount of specialization in these areas. This focused specialization contributed to a lack of convergence because each technology was growing in its own right; the consolidation of staff and budgets was not pressuring IT to solve the issue by building collaborative solutions. But that would change. As addressed later in the topics covering SDN, the industry was primed for transformation.

In today's world of modern networks, a difference of equipment and functionality is to be expected. Certainly, there are benefits recognized with standardizing device models to provide efficiencies in management and device/module sparing strategies. However, as network functions are separated, as seen later with SDN, or virtualized, as seen with Network Function Virtualization (NFV), a greater operational complexity is experienced. To that end, the industry has responded with model-driven concepts, which we cover in Chapter 11, "NETCONF and RESTCONF." The ability to move from device-by-device, atomic management considerations to more service and function-oriented models that comprehend the relationships and dependencies among many devices is the basis for model-driven management.

Proximity of Management Tools and Support Staff

Another situation that needed to be addressed was the proximity of management tools and support staff. Early networks were not as interconnected, persistent, or ingrained to as many aspects of our lives as they are now. It was common to deploy multiple copies of management tools across an environment because the connectivity or basic interface link speed among sites often precluded using a central management tool. Those were the days of "Hey, can you drive from Detroit down to Dayton to install another copy of XYZ?"

Support staff existed at many large sites, sometimes with little collaboration among them or consistency of service delivery across states or countries.

Because early networks often metered and charged on traffic volume across a wide area, they were almost disincentivized to consolidate monitoring and management. "Why would I want to run more monitoring traffic and increase my cost? I only want 'business-critical traffic' across those WAN links now." However, fortunately, even this way of thinking changed.

Today networks are more meshed, persistent, highly available, and faster connected. There is little need to deploy multiple management tools, unless it is purposeful for scale or functional segmentation. The support teams today may include a "follow the sun" model where three or four different support centers are spread across the globe to allow personnel to serve others in their proximate time zone. As businesses experience higher degrees of automation and orchestration, there is reduced need for on-shift personnel. Consolidation of support teams is possible. This pivot to a more on-call or exception-based support model is desired. The implementation of self-healing networks that require fewer and fewer support personnel is even more desirable. Google's concept of site reliability engineering (SRE) is an example of addressing the industry's shortcomings with infrastructure and operations support. The SRE discipline combines aspects of software engineering with infrastructure and operations. SRE aims to enable highly scalable and reliable systems. Another way of thinking about SRE is what happens when you tell a software engineer to do an operations role.

Speed of Service Provisioning

With early networks being "small" and "specialized," there was a certain acceptance to how long it took to provision new services. The network engineer of the late 1990s and early 2000s might have experienced lead times of many *months* to get new circuits from their WAN service provider. However, this was an area of transformation in network IT also. Networks became more critical to businesses. Soon, having a web presence, in addition to any brick-and-mortar location, was a necessity. This would drive a need for faster service provisioning and delivery. Previous manual efforts that included a "truck roll," or someone driving to another location, put too much latency into the process.

Businesses that could provide a service in weeks were driving a competitive differentiator to those that took months. Then this model progressed to those that could provide services in days versus weeks, and now you see the expectation of minutes, or "while I watch from my browser."

Business models have greatly changed. The aforementioned brick-and-mortar model was the norm. As the Internet flourished, having a web presence became a differentiator, then a requirement. To that end, so many years later, it is very difficult to find impactful domain names to register. Or it may cost a lot to negotiate a transfer from another owner!

Today, the physical presence is not required and is sometimes undesirable. More agile business models mean companies can be operated out of the owner's home. Fulfillment can be handled by others, and the store or marketplace is handled through a larger e-commerce entity like Amazon, Alibaba, or eBay.

It is impossible to provide services in such a rapid fashion without automation. The customer sitting at a browser expects to see an order confirmation or expected service access right then. Indeed, some customers give up and look for alternative offers if their request is not met as they wait.

This expectation of *now* forces businesses to consolidate their offers into more consistent or templatized offers. The more consistent a service can be delivered, the better suited it is for automation. It's the exceptions that tend to break the efficiencies of automation and cause longer service delivery cycles.

This rapid pace of service delivery influenced IT service management and development with DevOps and models like Agile and Lean. Figure 10-3 depicts the Agile methodology.

Key Topic

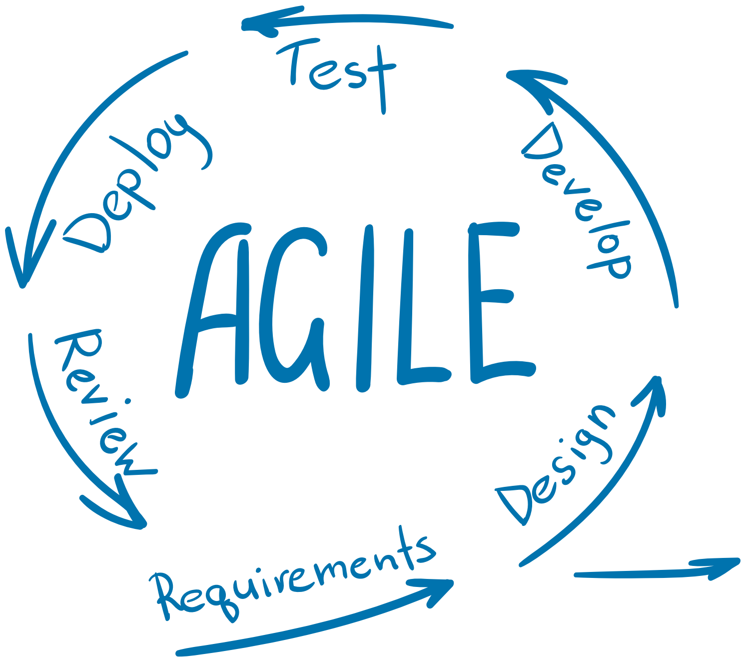

Figure 10-3 *Agile Methodology*

Agile, as a software development practice, focuses on extracting requirements and developing solutions with collaborative teams and their users. Planning with an adaptive approach to quick delivery and continuous improvement sets Agile apart from other, less flexible models. Agile is just one software development methodology, but it has a large following and is suggested for consideration in your network programmability journey. Several more of the broad spectrum of methodologies and project management frameworks are described in Table 10-3.

Table 10-3 Software Development Methodologies and Frameworks

Method Name	Description
Agile	Flexible and incremental design process focused on collaboration
Kanban	Visual framework promoting what, when, and how to develop in small, incremental changes; complements Agile
Lean	Process to create efficiencies and remove waste to produce more with less
Scrum	Process with fixed-length iterations (sprints); follows roles, responsibilities, and meetings for well-defined structure; derivative of Agile
Waterfall	Sequential design process; fully planned; execution through phases

Whatever model you choose, take time to understand the pros and cons and evaluate against your organization's capabilities, culture, motivations, and business drivers. Ultimately, the right software development methodology for you is the one that is embraced by the most people in the organization.

Accuracy of Service Provisioning

Walt Disney is known for sharing this admirable quote, "Whatever you do, do it well." That has been the aspiration of any product or service provider. The same thinking can be drawn to network service provisioning: nobody truly intends to partially deploy a service or to deploy something that will fail. One reason accuracy of service provisioning struggled before network programmability hit its stride was due to the lack of programmatic interfaces.

As we mentioned before, much of the genesis of network IT, and dare we say even IT more broadly, was founded on manual command-line interface interactions. Provisioning a device meant someone was logging into it and typing or pasting a set of configuration directives. The task wasn't quite as bad as that in Figure 10-4, but it sure felt that way!

Figure 10-4 *Not Quite This Painful*

A slightly more advanced method might be typing or pasting those directives and putting them into a file to be transferred to the device and incorporated into its running configuration state. However, these manual efforts still required human interaction and an ability to translate intent to a set of configuration statements.

Some automations were, and sometimes still are, simply the collection and push of those same CLI commands (see Figure 10-5), but in an unattended fashion by a script or management application.

Figure 10-5 *Automating the CLI*

The fact that the foundation has been based on CLI automation seems to imply that the industry was conceding the "best" way to interact with a device was through the CLI. A lot of provisioning automation occurs through CLI with many management applications and open-source solutions.

Yet the CLI, while suited for human consumption, is not optimal for programmatic use. If the command syntax or output varies between releases or among products, the CLI-based solutions need to account for the differences. Consider the command output for **show interface** in Example 10-1.

Example 10-1 *Show Interface Output*

```
Switch# show interface te1/0/2
TenGigabitEthernet1/0/2 is up, line protocol is up (connected)
  Hardware is Ten Gigabit Ethernet, address is 0023.ebdd.4006 (bia 0023.ebdd.4006)
  MTU 1500 bytes, BW 10000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive not set
  Full-duplex, 10Gb/s, link type is auto, media type is 10GBase-SR
  input flow-control is off, output flow-control is unsupported
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:04, output 00:00:00, output hang never
 Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
 Output queue: 0/40 (size/max)
```

```
  5 minute input rate 5000 bits/sec, 9 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     200689496 packets input, 14996333682 bytes, 0 no buffer
     Received 195962135 broadcasts (127323238 multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog, 127323238 multicast, 0 pause input
     0 input packets with dribble condition detected
     7642905 packets output, 1360729535 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier, 0 PAUSE output
     0 output buffer failures, 0 output buffers swapped out
```

What are the options for extracting information like number of multicast packets output?

The use of Python scripts is in vogue, so let's consider that with Example 10-2, which requires a minimum of Python 3.6.

Example 10-2 *Python Script to Extract Multicast Packets*

```
import paramiko
import time
import getpass
import re

username = input('Enter Username: ')
userpassword = getpass.getpass('Enter Password: ')
devip = input('Enter Device IP: ')
devint = input('Enter Device Interface: ')

try:
    devconn = paramiko.SSHClient()
    devconn.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    devconn.connect(devip, username=username, password=userpassword,timeout=60)
    chan = devconn.invoke_shell()
    chan.send("terminal length 0\n")
    time.sleep(1)
    chan.send(f'show interface {devint}')
    time.sleep(2)
    cmd_output = chan.recv(9999).decode(encoding='utf-8')
    devconn.close()
    result = re.search('(\d+) multicast,', cmd_output)
    if result:
        print(f'Multicast packet count on {devip} interface {devint} is {result.
group(1)}')
    else:
       print(f'No match found for {devip} interface {devint} - incorrect
interface?')
```

10

```
except paramiko.AuthenticationException:
    print("User or password incorrect - try again")
except Exception as e:
    err = str(e)
    print(f'ERROR: {err}')
```

There's a common theme in methodologies that automate against CLI output which requires some level of string manipulation. Being able to use regular expressions, commonly called regex, or the **re** module in Python, is a good skill to have for CLI and string manipulation operations. While effective, using regex can be difficult skill to master. Let's call it an acquired taste. The optimal approach is to leverage even higher degrees of abstraction through model-driven and structure interfaces, which relieve you of the string manipulation activities. You can find these in solutions like pyATS (https://developer.cisco.com/pyats/) and other Infrastructure-as-Code (IaC) solutions, such as Ansible and Terraform.

Product engineers intend to maintain consistency across releases, but the rapid rate of change and the intent to bring new innovation to the industry often result in changes to the command-line interface, either in provisioning syntax and arguments or in command-line output. These differences often break scripts and applications that depend on CLI; this affects accuracy in service provisioning. Fortunately, the industry recognizes the inefficiencies and results of varying CLI syntax and output. Apart from SNMP, which generally lacked a strong provisioning capability, one of the first innovations to enable programmatic interactions with network devices was the IETF's **NETCONF (network configuration)** protocol.

We cover NETCONF and the follow-on RESTCONF protocol in more detail later in this book. However, we can briefly describe NETCONF as an XML representation of a device's native configuration parameters. It is much more suited to programmatic use. Consider now a device configuration shown in an XML format with Figure 10-6.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply message-id="11" xmlns="urn:ietf:params:netconf:base:1.0">
  <data>
    <xml-config-data>
      <Device-Configuration xmlns="urn:cisco:xml-pi">
        <version>
          <Param>15.2</Param>
        </version>
        <service>
          <timestamps>
            <debug>
              <datetime>
                <msec/>
              </datetime>
            </debug>
          </timestamps>
        </service>
...
</rpc-reply>
```

Figure 10-6 *Partial NETCONF Device Configuration*

Although the format may be somewhat unfamiliar, you can see patterns and understand the basic structure. It is the consistent structure that allows NETCONF/RESTCONF and an XML-formatted configuration to be addressed more programmatically. By referring to tags or paths through the data, you can cleanly extract the value of a parameter without depending on the existence (or lack of existence) of text before and after the specific parameter(s) you need. This capability sets NETCONF/RESTCONF apart from CLI-based methods that rely on regex or other string-parsing methods.

A more modern skillset would include understanding XML formatting and schemas, along with XPath queries, which provide data filtering and extraction functions.

Many APIs output their data as XML- or JSON-formatted results. Having skills with XPath or JSONPath queries complements NETCONF/RESTCONF. Again, we cover these topics later in Chapter 11.

Another way the industry has responded to the shifting sands of CLI is through abstracting the integration with the device with solutions like Puppet, Chef, Ansible, and Terraform. Scripts and applications can now refer to the abstract intent or API method rather than a potentially changing command-line argument or syntax. These also are covered later in this book.

Scale

Another challenge that needs to be addressed with evolving and growing network is scale. Although early and even some smaller networks today can get by with manual efforts of a few staff members, as the network increases in size, user count, and criticality, those models break. Refer back to Figure 9-19 to see the growth of the Internet over the years.

Scalable deployments are definitely constrained when using CLI-based methodologies, especially when using paste methodologies because of flow control in terminal emulators and adapters. Slightly more efficiencies are gained when using CLI to initiate a configuration file transfer and merge process.

Let me share a personal example from a customer engagement. The customer was dealing with security access list changes that totaled thousands of lines of configuration text and was frustrated with the time it took to deploy the change. One easy fix was procedural: create a new access list and then flip over to it after it was created. The other advice was showing the customer the inefficiency of CLI flow-control based methods. Because the customer was copying/pasting the access list, they were restricted by the flow control between the device CLI and the terminal emulator.

Strike one: CLI/terminal.

Strike two: Size of access list.

Strike three: Time to import.

Pasting the customer's access list into the device's configuration took more than 10 minutes. I showed them the alternative of putting the configuration parameters into a file that could be transferred and merged with the device and the resulting seconds that this approach took instead. Needless to say, the customer started using a new process.

Using NETCONF/RESTCONF protocols to programmatically collect information and inject provisioning intent is efficient. In this case, it is necessary to evaluate the extent of deployment to gauge the next level of automation for scale. Here are some questions to ask yourself:

- How many devices, nodes, and services do I need to deploy?
- Do I have dependencies among them that require staggering the change for optimal availability? Any primary or secondary service relationships?
- How much time is permitted for the change window, if applicable?
- How quickly can I revert a change if unexpected errors occur?

Increasingly, many environments have no maintenance windows; there is no time that they are not doing mission-critical work. They implement changes during all hours of the day or night because their network architectures support high degrees of resiliency and availability. However, even in these environments, it is important to verify that the changes being deployed do not negatively affect the resiliency.

One more important question left off the preceding list for special mention is "How much risk am I willing to take?" I remember working with a customer who asked, "How many devices can we software upgrade over a weekend? What is that maximum number?" Together, we created a project and arranged the equipment to mimic their environment as closely as possible—device types, code versions, link speeds, device counts. The lab was massive—hundreds of racks of equipment with thousands of devices. In the final analysis, I reported, "You can effectively upgrade your entire network over a weekend." In this case, it was 4000 devices, which at the time was a decent-sized network. I followed by saying, "However, I wouldn't do it. Based on what I know of your risk tolerance level, I would suggest staging changes. The network you knew Friday afternoon could be very different from the one Monday morning if you run into an unexpected issue." We obviously pressed for extensive change testing, but even with the leading test methodologies of the time, we had to concede something unexpected could happen. We saved the truly large-scale changes for those that were routine and low impact. For changes that were somewhat new, such as new software releases or new features and protocols, we established a phased approach to gain confidence and limit negative exposure.

- Lab testing of single device(s) representing each model/function
- Lab testing of multiple devices, including primary/backup peers
- Lab testing of multiple devices, including primary/backup peers to maximum scale possible in lab
- Production deployment of limited device counts in low-priority environments (10 percent of total)
    - Change observation for one to two weeks (depending on criticality of change)

- Production deployment of devices in standard priority environments (25 percent of total)
  - Change observation for two to four weeks (depending on criticality of change)
- Second batch deployment in standard priority environments (25 percent of total)
  - Change observation for two to four weeks (depending on criticality of change)
- Production deployment of devices in high-priority environments (10 percent of total)
  - Change observation for two to four weeks (depending on criticality of change)
- Second batch deployment of high-priority environments (10 percent of total)
  - Change observation for two to four weeks (depending on criticality of change)
- Third batch deployment of high-priority environments (20 percent of total)

As you contemplate scale, if you're programming your own solutions using Python scripts or similar, it is worthwhile to understand multithreading and multiprocessing. A few definitions of concurrency and parallelism also are in order.

An application completing more than one task at the same time is considered concurrent. *Concurrency* is working on multiple tasks at the same time but not necessarily simultaneously. Consider a situation with four tasks executing concurrently (see Figure 10-7). If you had a virtual machine or physical system with a one-core CPU, it would decide the switching involved to run the tasks. Task 1 might go first, then task 3, then some of task 2, then all of task 4, and then a return to complete task 2. Tasks can start, execute their work, and complete in overlapping time periods. The process is effectively to start, complete some (or all) of the work, and then return to incomplete work where necessary—all the while maintaining state and awareness of completion status. One issue to observe is that concurrency may involve tasks that have no dependency among them. In the world of IT, an overall workflow to enable a new web server may not be efficient for concurrency. Consider the following activities:

1. Create the virtual network.
2. Create the virtual storage volume.
3. Create the virtual machine vCPUs and vMemory.
4. Associate the VM vNet and vStorage.
5. Install the operating system to the VM.
6. Configure the operating system settings.
7. Update the operating system.
8. Install the Apache service.
9. Configure the Apache service.

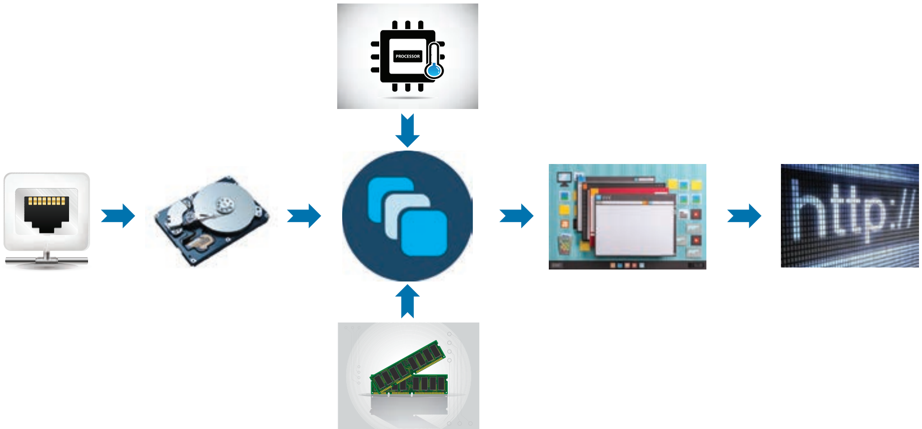

Figure 10-7 *Workflow Creating a Web Server*

Several of these steps depend on a previous step being completed. So, this workflow is not well suited to concurrency. However, deploying software images to many devices across the network would be well suited. Consider these actions on a multidevice upgrade process (see Figure 10-8):

1. Configure Router-A to download new software update (wait for it to process, flag it to return to later, move on to next router), then . . .
2. Configure Router-B to download new software update (wait for it to process, flag it to return to later, move on to next router), then . . .
3. Configure Router-C to download new software update (wait for it to process, flag it to return to later, move on to next router), then . . .
4. Check Router-A status—still going—move on to next router.
5. Configure Router-D to download new software update (wait for it to process, flag it to return to later, move on to next router).
6. Check Router-B status—complete—remove flag to check status; move to next router.
7. Configure Router-E to download new software update (wait for it to process, flag it to return to later, move on to next router).
8. Check Router-A status—complete—remove flag to check status; move to next router.
9. Check Router-C status—complete—remove flag to check status; move to next router.
10. Check Router-D status—complete—remove flag to check status; move to next router.
11. Check Router-E status—complete—remove flag to check status; move to next router.

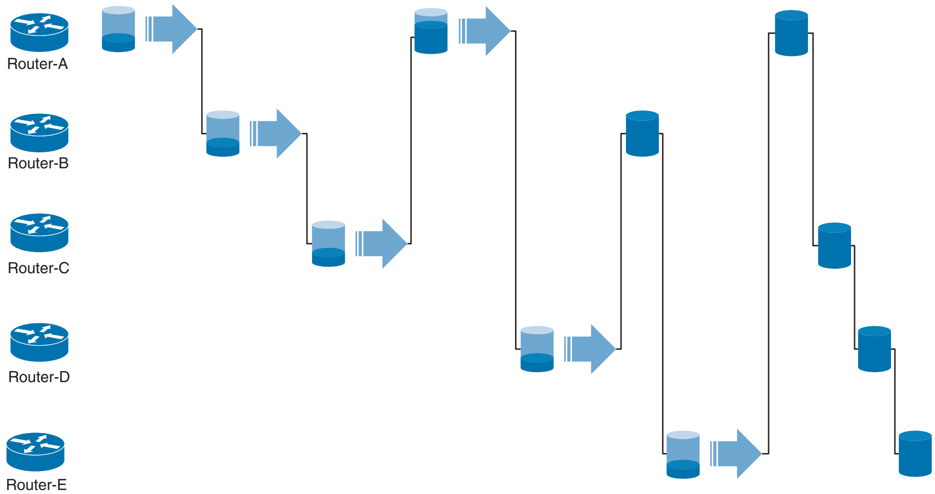

Figure 10-8 *Concurrency Example*

Parallelism is different in that an application separates tasks into smaller activities to process in parallel on multiple CPUs simultaneously. Parallelism doesn't require multiple tasks to exist. It runs parts of the tasks or multiple tasks at the same time using multicore functions of a CPU. The CPU handles the allocation of each task or subtask to a core.

Returning to the previous software example, consider it with a two-core CPU. The following actions would be involved in this multidevice upgrade (see Figure 10-9):

1. Core-1: Configure Router-A to download new software update (wait for it to process, flag it to return to later, move on to next router), while at the same time on another CPU . . .
2. Core-2: Configure Router-B to download new software update (wait for it to process, flag it to return to later, move on to next router).
3. Core-1: Configure Router-C to download new software update (wait for it to process, flag it to return to later, move on to next router).
4. Core-1: Check Router-A status—still going—move on to next router.
5. Core-2: Configure Router-D to download new software update (wait for it to process, flag it to return to later, move on to next router).
6. Core-2: Check Router-B status—complete—remove flag to check status; move to next router.
7. Core-2: Configure Router-E to download new software update (wait for it to process, flag it to return to later, move on to next router).
8. Core-1: Check Router-A status—complete—remove flag to check status; move to next router.
9. Core-1: Check Router-C status—complete—remove flag to check status; move to next router.
10. Core-1: Check Router-D status—complete—remove flag to check status; move to next router.
11. Core-2: Check Router-E status—complete—remove flag to check status; move to next router.

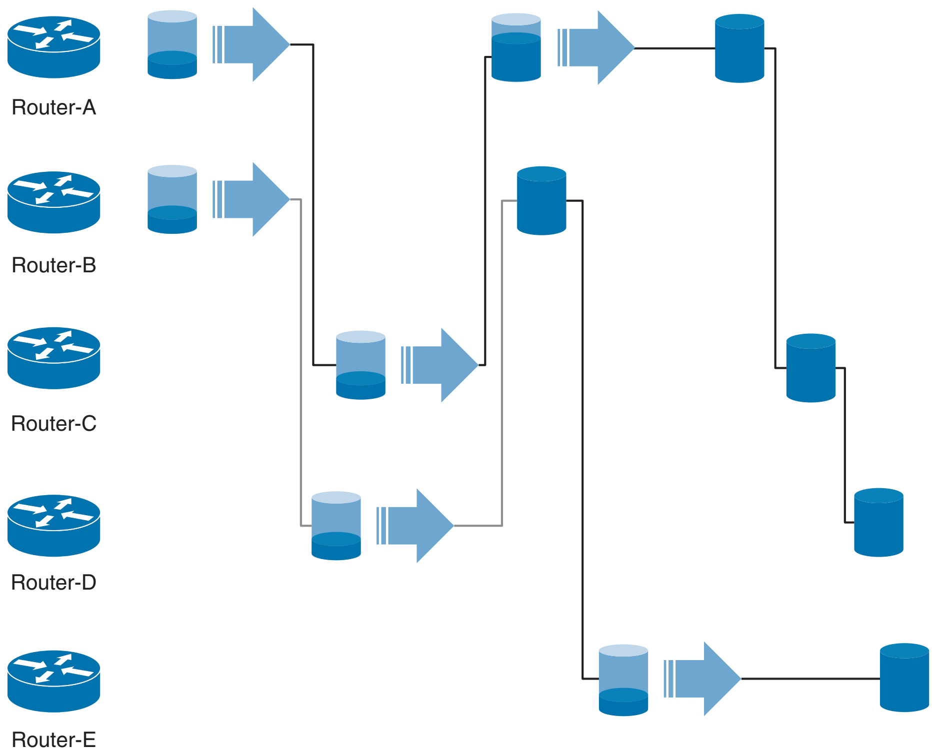

Figure 10-9 *Parallelism Example*

Because two tasks are executed simultaneously, this scenario is identified as parallelism. Parallelism requires hardware with multiple processing units, cores, or threads.

To recap, a system is concurrent if it can support two or more tasks in progress at the same time. A system is parallel if it can support two or more tasks executing simultaneously. Concurrency focuses on *working with* lots of tasks at once. Parallelism focuses on *doing* lots of tasks at once.

So, what is the practical application of these concepts? In this case, I was dealing with the Meraki Dashboard API; it allows for up to five API calls per second. Some API resources like Get Organization (**GET /organizations/{organizationId}**) have few key-values to return, so they are very fast. Other API resources like Get Device Clients (**GET /devices/{serial}/clients**) potentially return many results, so they may take more time. Using a model of parallelism to send multiple requests across multiple cores—allowing for some short-running tasks to return more quickly than others and allocating other work—provides a quicker experience over doing the entire process sequentially.

To achieve this outcome, I worked with the Python asyncio library and the semaphores feature to allocate work. I understood each activity of work had no relationship or dependency on the running of other activities; no information sharing was needed, and no interference across threads was in scope, also known as thread safe. The notion of tokens to perform work was easy to comprehend. The volume of work was created with a loop building a list of tasks; then the script would allocate as many tokens as were available in the semaphore bucket. When the script first kicked off, it had immediate access to do parallel processing of the four tokens I had allocated. As short-running tasks completed, tokens were returned to the bucket and made available for the next task. Some tasks ran longer than others, and that was fine because the overall model was not blocking other tasks from running as tokens became available.

Doing More with Less

Continuing in the theme of challenges being addressed, we must acknowledge the business pressures of gaining efficiencies to reduce operation expenses (OpEx) and potentially improve margins, if applicable. Network IT varies between a necessary cost center and a competitive differentiating profit center for many businesses. It is not uncommon for the cost center–focused businesses to manage budgets by reducing resources and attempting to get more productivity from those remaining. The profit center–focused businesses may do the same, but mostly for margin improvement.

Automation, orchestration, and network programmability provide the tools to get more done with less. If tasks are repetitive, automation reduces the burden—and burnout—on staff. Team members are able to focus on more strategic and fulfilling endeavors.

In reflection with the previous section on scale, if you have a lot of tasks that would benefit from parallel execution, if they are not dependent on each other, then it makes sense to allocate more threads/cores to the overall work. Efficient use of existing resources is desirable. It is a waste of resources if a system with many cores is often idle.

When building automated solutions, observe the tasks and time the original manual process from end to end. After you have automated the process, measure the runtime of the newly automated process and provide reporting that shows time and cost savings with the automation. Having practical examples of return on investment (ROI) helps decision makers understand the benefits of automation and encourage its implementation. You're building the automation; you can create your own telemetry and instrumentation!

Software-Defined Networking (SDN)

The catalyst for software-defined networking is largely attributed to Stanford University's Clean Slate Program in 2008. Cisco was a sponsor of this project, which reimagined what a new Internet would look like if we set aside conventional norms of traditional networks and worked from a clean slate. It was difficult to develop next-generation routing or connectivity protocols if the equipment available was purposely programmed to follow the original conventions. Programmable logic arrays (PLAs) were pretty expensive to test theories, so a more software-based approach was proposed.

What Is SDN and Network Programmability?

Key Topic

Definitions of SDN and network programmability varied among network IT vendors, but some points were generally agreed upon. As illustrated in Figure 10-10, SDN is

- An approach and architecture in networking where control and data planes are decoupled, and intelligence and state are logically centralized
- An enabling technology where the underlying network infrastructure is abstracted from the applications (network virtualization)
- A concept that leverages programmatic interfaces to enable external systems to influence network provisioning, control, and operations

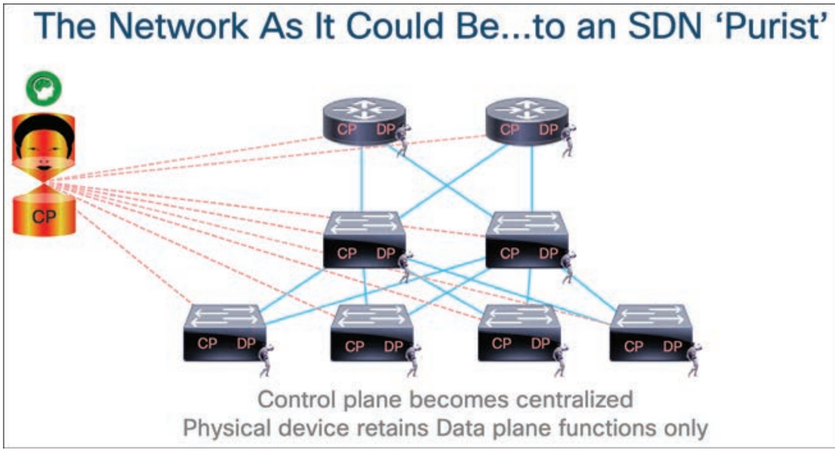

Figure 10-10 *The SDN Concept*

Although all of these definitions were exciting and transformative, the last item of leveraging programmatic interfaces appeals mostly to the network programming crowd. The last item also enables us to influence the first two through provisioning and monitoring network assets.

In my talks at CiscoLive, I would share that SDN was

- An approach to network transformation*
- Empowering alternative, nontraditional entities to influence network design and operations
- Impacting the networking industry, challenging the way we think about engineering, implementing, and managing networks
- Providing new methods to interact with equipment and services via controllers and APIs
- Normalizing the interface with equipment and services
- Enabling high-scale, rapid network and service provisioning and management
- Providing a catalyst for traditional route/switch engineers to branch out

Approach

So, why the asterisk next to an approach to network transformation? Well, it wasn't the first attempt at network transformation. If we consider separation of the control plane and data plane, we can look no further than earlier technologies, such as SS7, ATM LANE, the wireless LAN controller, and GMPLS. If we were considering network overlays/underlays and encapsulation, the earlier examples were MPLS, VPLS, VPN, GRE Tunnels, and LISP. Finally, if our consideration was management and programmatic interfaces, we had SNMP, NETCONF and EEM. Nonetheless, SDN was a transformative pursuit.

Nontraditional Entities

What about those nontraditional entities influencing the network? As new programmatic interfaces were purposely engineered into the devices and controllers, a new wave of network programmers joined the environment. Although traditional network engineers skilled up to learn programming (and that may be you, reading this book!), some programmers who had little prior networking experience decided to try their hand at programming a network. Or the programmers decided it was in their best interests to configure an underpinning network for their application themselves, rather than parsing the work out to a network provisioning team.

Regardless of the source of interaction with the network, it is imperative that the new interfaces, telemetry, and instrumentation be secured with the same, if not more, scrutiny as the legacy functions. The security policies can serve to protect the network from unintentional harm by people who don't have deep experience with the technology *and* from the intentional harm of bad actors.

Industry Impact

The impact to the network industry with operations and engineering was greatly influenced by control plane and data plane separation and the development of centralized controllers. The network management teams would no longer work as hard to treat each network asset as an atomic unit but could manage a network en masse through the controller. One touchpoint for provisioning and monitoring of all these devices! The ACI APIC controller is acknowledged as one of the first examples of an SDN controller, as seen in Figure 10-11. It was able to automatically detect, register, and configure Cisco Nexus 9000 series switches in a data center fabric.

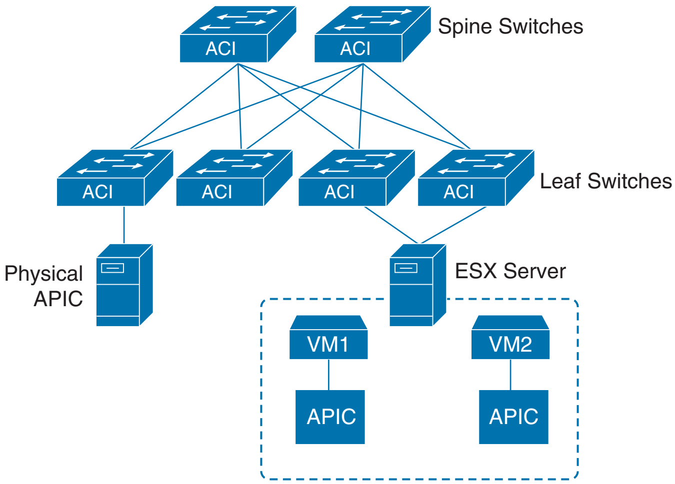

Figure 10-11 *Cisco ACI Architecture with APIC Controllers*

New Methods

With respect to new methods, protocols, and interfaces to managed assets, APIs became more prolific with the SDN approach. Early supporting devices extended a style of REST-like interface and then more fully adopted the model. First NETCONF and then RESTCONF became the desired norm. Centralized controllers, like the wireless LAN controller, ACI's

APIC controller, Meraki, and others, prove the operational efficiency of aggregating the monitoring and provisioning of fabrics of devices. This model has coaxed the question "What else can we centralize?"

Normalization

SDN's impact on network normalization is reflected in the increasingly standardized interfaces. While SNMP had some utility, SDN provided a fresh opportunity to build and use newer management technologies that had security at their core, not just a "bolt-on" consideration. Although the first API experiences felt a bit like the Wild Wild West, the Swagger project started to define a common interface description language to **REST** APIs. Swagger has since morphed into the OpenAPI initiative, and specification greatly simplifies API development and documentation tasks.

Enabling Operations

Network operations, service provisioning, and management were influenced with SDN through the new interfaces, their standardization, and programmatic fundamentals. Instead of relying on manual CLI methods, operators began to rely on their growing knowledge base of REST API methods and sample scripts in growing their operational awareness and ability to respond and influence network functions.

Besides the REST API, other influences include gRPC Network Management Interface (gNMI), OpenConfig, NETCONF, RESTCONF, YANG, time-series databases, AMQP pub-sub architectures, and many others.

Enabling Career Options

Finally, SDN provided traditional network engineers an opportunity to extend their skills with new network programming expertise. The traditional network engineer with years of domain experience could apply that knowledge in an impactful way with these programmatic interfaces. They could deploy more services at scale, with fewer errors and more quickly.

How impactful could SDN be? Let's consider the early days of IP telephony: it didn't ramp up as quickly as desired. On one side there were the traditional "tip-ring telco" team members; on the other side was the new "packet-switch" team. IP telephony technology was slow to gain momentum because few individuals crossed the aisle to learn the other side and become change and translation agents for the greater good. When people started to understand and share the nuanced discipline of the other side, then SDN started to make strides.

Network programmability is in that same transition: there are *strong* network engineers who understand their tradition route/switch technology. Likewise, there are very strong software developers who understand how to build apps and interact with systems; they just don't have the network domain expertise. As network engineers skill up with the automation and network programming discipline, they bring their experience of networks with them. So, let's do IT!

Use Cases and Problems Solved with SDN

SDN aimed to address several use cases. The research and academic communities were looking for ways to create experimental network algorithms and technologies. The hope was to turn these into new protocols, standards, and products. Because existing products closely adhered to well-defined routing protocol specifications, SDN was to help separate the current norms from new, experimental concepts.

The massively scalable data center community appreciated SDN for the ability to separate the control plane from the data plane and use APIs to provide deep insight into network traffic. Cloud providers drew upon SDN for automated provisioning and programmable network overlays. Service providers aligned to policy-based control and analytics to optimize and monetize service delivery. Enterprise networks latched onto SDN's capability to virtualize workloads, provide network segmentation, and orchestrate security profiles.

Nearly all segments realized the benefits of automation and programmability with SDN:

- Centralized configuration, management control, and monitoring of network devices (physical or virtual)
- The capability to override traditional forwarding algorithms to suit unique business or technical needs
- The capability of external applications or systems to influence network provisioning and operation
- Rapid and scalable deployment of network services with lifecycle management

Several protocols and solutions contributed to the rise of SDN. See Table 10-4 for examples.

Key Topic

Table 10-4 Contributing Protocols and Solutions to SDN

Protocol/Solution	Definition	Function
OpenFlow		Layer-2 programmable forwarding protocol and specification for switch manufacturing
I2RS	Interface to Routing System	Layer-3 programmable protocol to the routing information base (RIB); allowed manipulation and creation of new routing metrics
PCEP	Path Computation Element Protocol	L3 protocol capable of computing a network path or route based on a network graph and applying computational constraints
BGP-LS/FS	BGP Link-State / Flow Spec	The ability to gather IGP topology of the network and export to a central SDN controller or alternative method to remotely triggered black hole filtering useful for DDoS mitigation

Protocol/Solution	Definition	Function
OpenStack		Hypervisor technology for virtualization of workloads
OMI	Open Management Infrastructure	Open-source Common Information Model with intent to normalize management
Puppet		Agent-based configuration management solution embedded in devices (later updated to agentless)
Ansible		Agentless configuration management solution
NETCONF	Network Configuration standard	IETF working group specification normalizing configuration across vendors using XML schemas (later updated with YANG)
YANG	Data Modeling Language	Data modeling language for defining IT technologies and services

Overview of Network Controllers

One of the main benefits of SDN was the notion of control plane and data plane separation. You can think of the control plane as the brains of the network: it makes forwarding decisions based on interactions with adjacent devices and their forwarding protocols. The data plane is the muscle, acting on the forwarding decisions programmed into it. Routing protocols like OSPF and BGP operate in the control plane. A link aggregation protocol like LACP or the MAC address table would be representative of the data plane.

The first traditional networks combined the functionality of the control plane/data plane into the same device. Each device acted autonomously, creating and executing its own forwarding decisions. With the advent of SDN, the notion of centralizing that brain function into one control unit yet keeping the data plane function at the managed device was the new architectural goal. These centralized controllers aggregated the monitoring and management function. They oftentimes also provided that centralized forwarding path determination and programming function.

The separation of functional planes also resulted in the definition of new overlay and underlay functionality. Network overlays defined that tunnel endpoints terminated on routers and switches. The physical devices executed the protocols to handle resiliency and loops. Some examples are OTV, VXLAN, VPLS, and LISP.

Host overlays defined that tunnel endpoints terminated on virtual nodes. Examples of host overlays are VXLAN, NVGRE, and STT. Finally, integrated overlays allowed for physical or virtual endpoints in tunnel termination. The Cisco ACI fabric with Nexus 9000 series switches are examples of integrated overlays.

The Cisco Solutions

Cisco has many offerings in the SDN space, the most prominent being the Cisco ACI fabric with Nexus 9000 series switches. Software-defined access (SDA) is enabled by Cisco DNA Center on enterprise fabric-enabled devices. Software-defined wide-area networks (SD-WANs) can be seen in the acquired technologies of Viptela, resulting in the vManage solution for central cloud management, authentication, and licensing.

Network Function Virtualization (NFV) enables cloud technology to support network functions, such as the Cisco Integrated Services Virtual Router (ISRv), ASAv, and vWLC. The Cisco Managed Services Accelerator (MSX) provides automated end-to-end SD-WAN services managed from the service provider cloud.

Application Programming Interfaces (APIs)

Application programming interfaces are the foundational method for interacting with devices, applications, controllers, and other networked entities. Although the command-line interface has reigned supreme for years, we must admit, if an entity has an API, it is the desirable method for interacting with it.

APIs are common in many fashions: some are application to application, whereas others are application to hardware entity. Consider some of the following interactions as examples:

- DNAC software controller API call to Cisco Support API endpoint for opening cases: software to software
- Cisco Intersight with UCS IMC for device registration, monitoring, and provisioning from the cloud: software to hardware
- Network Services Orchestrator (NSO) to ASR1000 for provisioning and monitoring: software to hardware

To use an API, you must know the way to make requests, how to authenticate to the service, how to handle the return results (data encoding), and other conventions it may use, such as cookies. Public APIs often involve denial-of-service protections beyond authentication, such as rate limiting the number of requests per time period, the number of requests from an IP endpoint, and pagination or volume of data returned.

For the purposes of network IT, we mostly focus on web APIs, as we discuss in the next section on REST APIs, but other common APIs you may experience are the Java Database Connectivity (JDBC) and Microsoft Open Database Connectivity (ODBC) APIs. JDBC and ODBC permit connections to different types of databases, such as Oracle, MySQL, and Microsoft SQL Server, with standard interfaces that ease application development.

The Simple Object Access Protocol (SOAP) is also a well-known design model for web services. It uses XML and schemas with a strongly typed messaging framework. A web service definition (WSDL) defines the interaction between a service provider and the consumer. In Cisco Unified Communications, the Administrative XML Web Service (AXL) is a SOAP-based interface enabling insertion, retrieval, updates, and removal of data from the Unified Communication configuration database.

Because a SOAP message has the XML element of an "envelope" and further contains a "body," many people draw the parallel of SOAP being like a postal envelope, with the necessary container and the message within, to REST being like a postcard that has none of the “wrapper” and still contains information. Figure 10-12 illustrates this architecture.

Figure 10-12 *Cisco ACI Architecture with APIC Controllers*

APIs are used in many Internet interactions—logins, data collection, performance reporting, analytics. Microservices are associated with APIs as self-contained, lightweight endpoints that you can interact with to gather data or effect change. They are usually specific to a function, such as validate login, and are engineered with resiliency in mind, so continuous integration/continuous deployment (CI/CD) processes allow for routine maintenance without service impact to users.

REST APIs

RESTful APIs (or representational state transfer APIs) use Web/HTTP services for read and modification functions. This stateless protocol has several predefined operations, as seen in Table 10-5. Because it’s a stateless protocol, the server does not maintain the state of a request. The client’s request must contain all information needed to complete a request, such as session state.

Key Topic

Table 10-5 REST API Operation Types

Method	Function	Idempotency	Safety/Read-only Function
GET	Reads resource data, settings	YES	YES
HEAD	Tests the API endpoint for validity, accessibility, and recent modifications; similar to GET without response payload	YES	YES
POST	Creates a new resource	NO	NO
PUT	Updates or replaces a resource	YES	NO
PATCH	Modifies changes to a resource (not complete replacement)	NO	NO
DELETE	Deletes a resource	YES	NO
CONNECT	Starts communications with the resource; opens a tunnel	YES	YES
OPTIONS	Provides information about the capabilities of a resource, without initiating a resource retrieval function	YES	YES

API Methods

Another important aspect of a RESTful API is the API method's *idempotency*, or capability to produce the same result when invoked, regardless of the number of times. The same request repeated to an idempotent endpoint should return an identical result regardless of two executions or hundreds.

API Authentication

Authentication to a RESTful API can take any number of forms: basic authentication, API key, bearer token, OAuth, or digest authentication, to name a few. Basic authentication is common, where the username is concatenated with a colon and the user's password. The combined string is then Base64-encoded. You can easily generate the authentication string on macOS or other Linux derivatives using the built-in **openssl** utility. Windows platforms can achieve the same result by installing OpenSSL or obtaining a Base64-encoding utility.

NOTE Do *not* use an online website to generate Base64-encoded authentication strings. You do not know whether the site is logging user input, and security may be undermined.

Example 10-3 shows an example of generating a basic authentication string with **openssl** on macOS.

Example 10-3 *Generating Base64 Basic Authentication String on MacOS*

```
Mac ~ % echo -n 'myusername:DevNet4U!' | openssl base64
bXl1c2VybmFtZTpEZXZOZXQ0VSE=
Mac ~ %
```

This method is not considered secure due to the encoding; at a minimum, the connection should be TLS-enabled so that the weak security model is at least wrapped in a layer of transport encryption.

Either API key or bearer token is more preferable from a security perspective. These models require you to generate a one-time key, usually from an administrative portal or user profile page. For example, you can enable the Meraki Dashboard API by first enabling the API for your organization: **Organization > Settings > Dashboard API access**. Then the associated Dashboard Administrator user can access the My Profile page to generate an API key. The key can also be revoked and a new key generated at any time, if needed.

API Pagination

API pagination serves as a method to protect API servers from overload due to large data retrieval requests. An API may limit return results, commonly rows or records, to a specific count. For example, the DNA Center REST API v2.1.2.x limits device results to 500 records at a time. To poll inventory beyond that, you would use pagination:


```
GET /dna/intent/api/v1/network-device/{index}/{count}
```

For example, if you had 1433 devices in inventory, you would use these successive polls:

```
GET /dna/intent/api/v1/network-device/1/500
GET /dna/intent/api/v1/network-device/501/500
GET /dna/intent/api/v1/network-device/1000/433
```

Other APIs may provide different cues that pagination was in effect. The API return results may include the following parameters:

```
Records: 2034

First: 0

Last: 999

Next: 1000
```

Payload Data Formats JSON XML

When dealing with REST APIs, you often need to provide a request payload containing parameters. The parameters could be anything—username to provision, interface name to poll, operating system template for a virtual machine. The API response payload body likewise has information to be consumed. In either case, it is common to work with XML- or JSON-formatted data, although others are possible and less common. These two data encoding models are conducive to programmatic use.

XML

Key Topic

The **Extensible Markup Language (XML)** is a markup language and data encoding model that has similarities to HTML. It is used to describe and share information in a programmatic but still humanly readable way.

XML documents have structure and can represent records and lists. Many people look at XML as information and data wrapped in tags. See Example 10-4 for context.

Example 10-4 *XML Document*

```
<Document>
   <Nodes>
      <Node>
         <Name>Router-A</Name>
         <Location>San Jose, CA</Location>
         <Interfaces>
            <Interface>
               <Name>GigabitEthernet0/0/0</Name>
               <IPv4Address>10.1.2.3</IPv4Address>
               <IPv4NetMask>255.255.255.0</IPv4NetMask>
               <Description>Uplink to Switch-BB</Description>
            </Interface>
            <Interface>
               <Name>GigabitEthernet0/0/1</Name>
               <IPv4Address>10.2.2.1</IPv4Address>
               <IPv4NetMask>255.255.255.128</IPv4NetMask>
               <Description />
            </Interface>
         </Interfaces>
      </Node>
   </Nodes>
</Document>
```

In this example, the structure of this XML document represents a router record. <Document>, <Nodes>, <Node>, <Name>, and <Location> are some of the tags created by the document author. They also define the structure. Router-A, San Jose, CA, and GigabitEthernet0/0/0 are values associated with the tags. Generally, when an XML document or schema is written, the XML tags should provide context for the value(s) supplied. The values associated with the tags are plaintext and do not convey data type. As a plaintext document, XML lends well to data exchange and compression, where needed.

XML has a history associated with document publishing. Its functional similarity with HTML provides value: XML defines and stores data, focusing on content; HTML defines format, focusing on how the content looks. The Extensible Stylesheet Language (XSL) provides a data transformation function, XSL Transformations (XSLT), for converting XML documents from one format into another, such as XML into HTML. When you consider that many APIs output results in XML, using XSLTs to convert that output into HTML is an enabling feature. This is the basis for simple "API to Dashboard" automation.

Referring to Example 10-4, you can see that XML documents contain starting tags, such as <Node>, and ending (or closing) tags, such as </Node>. There is also the convention of an empty element tag; note the <Description /> example. All elements must have an end tag or be described with the empty element tag for well-formed XML documents. Tags are case sensitive, and the start and end tags must match case. If you're a document author, you are able to use any naming style you wish: lowercase, uppercase, underscore, Pascal case, Camel case, and so on. It is suggested that you do not use dashes (-) or periods (.) in tags to prevent misinterpretation by some processors.

All elements must be balanced in nesting, but the spacing is not prescribed. A convention of three spaces aids the reader. It is acceptable for no spacing in a highly compressed document, but the elements must still be nested among start and end tags.

XML can have attributes, similar to HTML. In the HTML example , you can recognize attributes of src and alt with values of "devnet_logo.png" and "DevNet Logo". Similarly, in XML, data can have attributes—for example, <interface type="GigabitEthernet">0/0/0</interface>.

Attribute values, such as "GigabitEthernet", must be surrounded by double quotes. Values between tags, such as 0/0/0, do not require quotes.

XML documents usually start with an XML declaration or prolog to describe the version and encoding being used, but it is optional:

```
<?xml version="1.0" encoding="UTF-8"?>
```

XML documents are often described as trees composed of elements. The root element starts the document. Branches and child elements define the structure with elements potentially having subelements, or children. In Example 10-4, the root element is <Document>. Because XML does not predefine tags, you may see other root element tags. Some common ones are <Root>, <DocumentRoot>, <Parent>, and <rootElement>. It is up to the document author to define the tags and structure.

10

The <Nodes> element is a child. The <Node> elements are also children. The <Node> elements are also siblings to each other, as a list of records. The <Name>, <IPv4Address>, <IPv4NetMask>, and <Description> elements are children to <Node>, siblings to each other and form a record. Because there are multiple <Node> elements, a list is formed.

XML documents can be viewed in browsers, typically through an Open File function. The browser may render the XML with easy-to-understand hierarchy and expand or collapse functions using + and -or ^ and > gadgets. See Figure 10-13 for another example of ACI XML data, rendered in a browser.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<polUni>
  ▼<fvTenant name="tenant1">
    ▼<vnsAbsGraph name="WebGraph">
      ▼<vnsAbsTermNodeCon name="Input1">
         <vnsAbsTermConn name="C1"/>
       </vnsAbsTermNodeCon>
       <!--  FW1 Provides FW functionality  -->
      ▼<vnsAbsNode name="FW1" funcType="GoThrough">
         <vnsRsDefaultScopeToTerm tDn="uni/tn-tenant1/AbsGraph-WebGraph/AbsTermNodeProv-Output1/outtmnl"/>
        ▼<vnsAbsFuncConn name="external" attNotify="yes">
           <vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-{dp_version}/mFunc-Firewall/mConn-external"/>
         </vnsAbsFuncConn>
        ▼<vnsAbsFuncConn name="internal" attNotify="yes">
           <vnsRsMConnAtt tDn="uni/infra/mDev-CISCO-ASA-{dp_version}/mFunc-Firewall/mConn-internal"/>
         </vnsAbsFuncConn>
        ▶<vnsAbsDevCfg>
         ...
         </vnsAbsDevCfg>
        ▶<vnsAbsFuncCfg>
         ...
         </vnsAbsFuncCfg>
         <vnsRsNodeToMFunc tDn="uni/infra/mDev-CISCO-ASA-{dp_version}/mFunc-Firewall"/>
       </vnsAbsNode>
      ▶<vnsAbsTermNodeProv name="Output1">
       ...
       </vnsAbsTermNodeProv>
      ▶<vnsAbsConnection name="CON1">
       ...
       </vnsAbsConnection>
      ▶<vnsAbsConnection name="CON2" unicastRoute="no">
       ...
       </vnsAbsConnection>
     </vnsAbsGraph>
   </fvTenant>
</polUni>
```

Figure 10-13 *ACI XML Data Rendered in Browser*

JSON

Key Topic

JavaScript Object Notation (JSON) is a newer data encoding model than XML and is growing in popularity and use with its more compact notation, ease of understanding, and closer integration with Python programming. It is lightweight, self-describing, and programming language independent. If your development includes JavaScript, then JSON is an easy choice for data encoding with its natural alignment to JavaScript syntax.

The JSON syntax provides data being written as name-value pairs. Data is separated by commas. Records or objects are defined by curly braces { }. Arrays and lists are contained within square brackets [ ].

The name of a name-value pair should be surrounded by double quotes. The value should have double quotes if representing a string. It should not have quotes if representing a numeric, Boolean (true/false), or null value. See Example 10-5 for a sample JSON record.

Example 10-5 *REST API Payload as JSON*


```
{
   "Document": {
      "Nodes": {
         "Node": {
            "Name": "Router-A",
            "Location": "San Jose, CA",
            "InterfaceCount": 2,
            "Interfaces": {
               "Interface": [
                  {
                     "Name": "GigabitEthernet0/0/0",
                     "IPv4Address": "10.1.2.3",
                     "IPv4NetMask": "255.255.255.0",
                     "Description": "Uplink to Switch-BB",
                     "isConnected": true
                  },
                  {
                     "Name": "GigabitEthernet0/0/1",
                     "IPv4Address": "10.2.2.1",
                     "IPv4NetMask": "255.255.255.128",
                     "Description": null,
                     "isConnected": false
                  }
               ]
            }
         }
      }
   }
}
```


Using this example, you can compare the structure with the previous XML representation. There is a list of interfaces; each interface is a record or object.

With APIs, the system may not give you a choice of data formatting; either XML or JSON may be the default. However, content negotiation is supported by many APIs. If the server drives the output representation, a Content-Type header shows “application/xml” or “application/json” as the response body payload type.

If the requesting client can request what’s desired, then an Accept header specifies similar values for preference. With some APIs, appending .xml or .json to the request URI returns the data with the preferred format.

Cross-Domain, Technology-Agnostic Orchestration (CDTAO)

This section is not part of the official DEVCOR certification; however, in the spirit of growing network programmability skills, it does seem appropriate to discuss. You may skip this section if you prefer.

Most work in network IT tends to be very domain-specific. It's not unusual to see engineers and operators focus on specific technologies—enterprise networking, route/switch, wireless, storage networks, compute, wide-area networking, MPLS, security, and so on. However, many do embrace multidomain expertise.

Often the management applications follow a similar segmentation. It is easy to appreciate, then, when management apps bring a multidomain perspective to monitoring, provisioning, and management. However, consider *why* you're doing IT: there's *a lot* to support a business and the apps it depends on for the services it provides. Some typical supporting technologies include DNS, server connectivity, link aggregation, routing, switching, storage, compute, virtualized workloads, authentication, databases, firewall security, content filtering security, threat mitigation security, and application hosting. I'm sure you can think of many more!

So, is your operational perspective keeping up with the scope of your IT services? If you end up using multiple tools for different domains or scale or geographical segments or security segmentation, do you have an aggregate view of the health of your IT services, or do you switch back and forth between multiple tools? Doesn't this issue get exacerbated when you pull in other IT vendors and open-source solutions? Is this something you accept as "the way it is" or do you try to "glue" together these systems for more unified operational insight?

How do you glue these systems together?

APIs are the unifying capability that enable you to achieve that glue. Most partner-vendors, Cisco included, strive to provide the best customer experience possible with their product and service offers. However, there are many customer segments, different sizes, different areas of concentration, and constraints. I have been asked, "Why isn't there just one management tool?" Can you imagine the size in server requirements, cost, and maintenance necessary to provide such a solution? Would the broad functions, some of which don't apply to your circumstances, distract your focus or enable it? In a friendly recognition to Hasbro, the movie series, and the legacy Cisco management suite, we would have to call it "Cisco Optimus Prime"! Most would agree that's a bit unrealistic. Even building an uber-modular framework to allow the specific selection of desired functions and device support would increase complexity.

So is there an answer? Most providers enable their tools with APIs. If you pick the tools and apps you need based on function, need, cost, and preference, then you can obtain a converged operational perspective by using orchestration to collect the key health indicators from the individual tools and controllers. The orchestrator's workflow would also include activities to create dashboards and portals unifying the information into converged operational portals that direct your attention to the domain-specific management tools, as necessary.

Is this possible? It's not provided out of the box, again due to the variety of device types and functions, but it is doable. Consider the portal developed for the CiscoLive NOC in Figure

10-14. This example represents, essentially, a mashup of key health metrics from several tools: Prime Infrastructure, DNA Center, vCenter, Prime Network Registrar, Hyperflex, and so on.

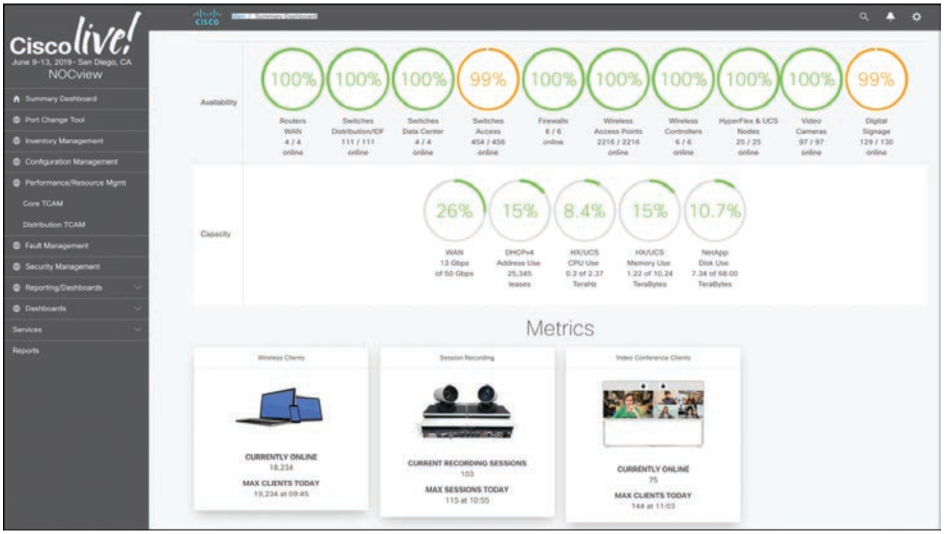

Figure 10-14 *NOC Dashboard*

So what does the technology-agnostic part of Cross-Domain, Technology-Agnostic Orchestration (CDTAO) entail? It's a wonderful concept to glue together network IT services in a cross-domain perspective. What about some out-of-the-box thinking that also brings in non-networking IT? From Figure 10-14, you can observe collaboration, digital signage, and NetApp storage. What other network-connected technology (think IoT) can be accessed and operational insight retrieved?

What industry do you work in?

- **Healthcare:** Pull in network-connected systems, such as blood-pressure cuffs, pulse ox monitors, and crash carts.
- **Financial:** Pull in ATM (cash, not legacy networking!), vault/deposit box status.
- **Retail:** Fork lifts, credit card and point-of-sale terminals.
- **Education:** Digital projector status, teacher location/availability, bus/parking lot, camera status.

If you add "business care-abouts" to the network IT perspectives, does that allow you to see contribution and impact of the supporting infrastructure to the broader company? Sure, it does!

Impact to IT Service Management and Security

This section is a continuation and amplification of the earlier "Software-Defined Networking (SDN)" section mentioning the impact of other nontraditional entities influencing the network. In a traditional networking case, you probably wrapped security around your change management and provisioning of the network devices, even if performed manually. SSH was enabled; access lists permitting only NOC or other specific personnel and network ranges were configured; logging and accounting were enabled; possibly two-factor or multifactor authentication was provisioned. In any case, security was given a strong consideration.

So now that network devices, management applications, and controllers have programmatic interfaces to extract and change functions of networks, are you continuing the same scrutiny? Are you the main source of API integration, or were other people with strong programming experience brought in to beef up automation? Do they have strong networking experience in concert with their programming skills? Are they keeping in touch with you about changes? Oh no! Did that network segment go down?

Okay, enough of the histrionic "what if" scenario. You just need to make sure the same rigor applied to traditional network engineering and operations is also being applied to newer, SDN, and programmatic environments.

What are the leading practices related to programmable networks? First, consider your risk. What devices and services are managed through controllers? They should be secured first because they have the broadest scope of impact with multiple devices in a fabric. Enable all the security features the controller provides with the least amount of privileges necessary to the fewest number of individuals (and other automated systems). If the controller has limited security options, consider front-ending it with access lists or firewall services to limit access and content. Remember to implement logging and accounting; then *review* it periodically.

The next order of business should be high-priority equipment where the loss of availability has direct service, revenue, or brand recognition impact. It's the same activity: tighten up access controls to the newer programmatic interfaces and telemetry.

Finally, go after the regular and low-priority equipment to shore up their direct device management interfaces in a similar fashion.

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 17, "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 10-6 lists a reference of these key topics and the page numbers on which each is found.

Key Topic

Table 10-6 Key Topics for Chapter 10

Key Topic Element	Description	Page Number
Figure 10-3	Agile Methodology	318
Paragraph	SDN concepts	329
Table 10-4	Contributing Protocols and Solutions to SDN	333
Table 10-5	REST API Operation Types	336
Paragraph	XML description	338
Paragraph	JSON description	340

Complete Tables and Lists from Memory

Print a copy of Appendix C, “Memory Tables” (found on the companion website), or at least the section for this chapter, and complete the tables and lists from memory. Appendix D, “Memory Tables Answer Key,” also on the companion website, includes completed tables and lists to check your work.

Define Key Terms

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

JavaScript Object Notation (JSON), Network Configuration Protocol (NETCONF), REST, Extensible Markup Language (XML), YANG

References

URL	QR Code
https://developer.cisco.com/pyats/	

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z