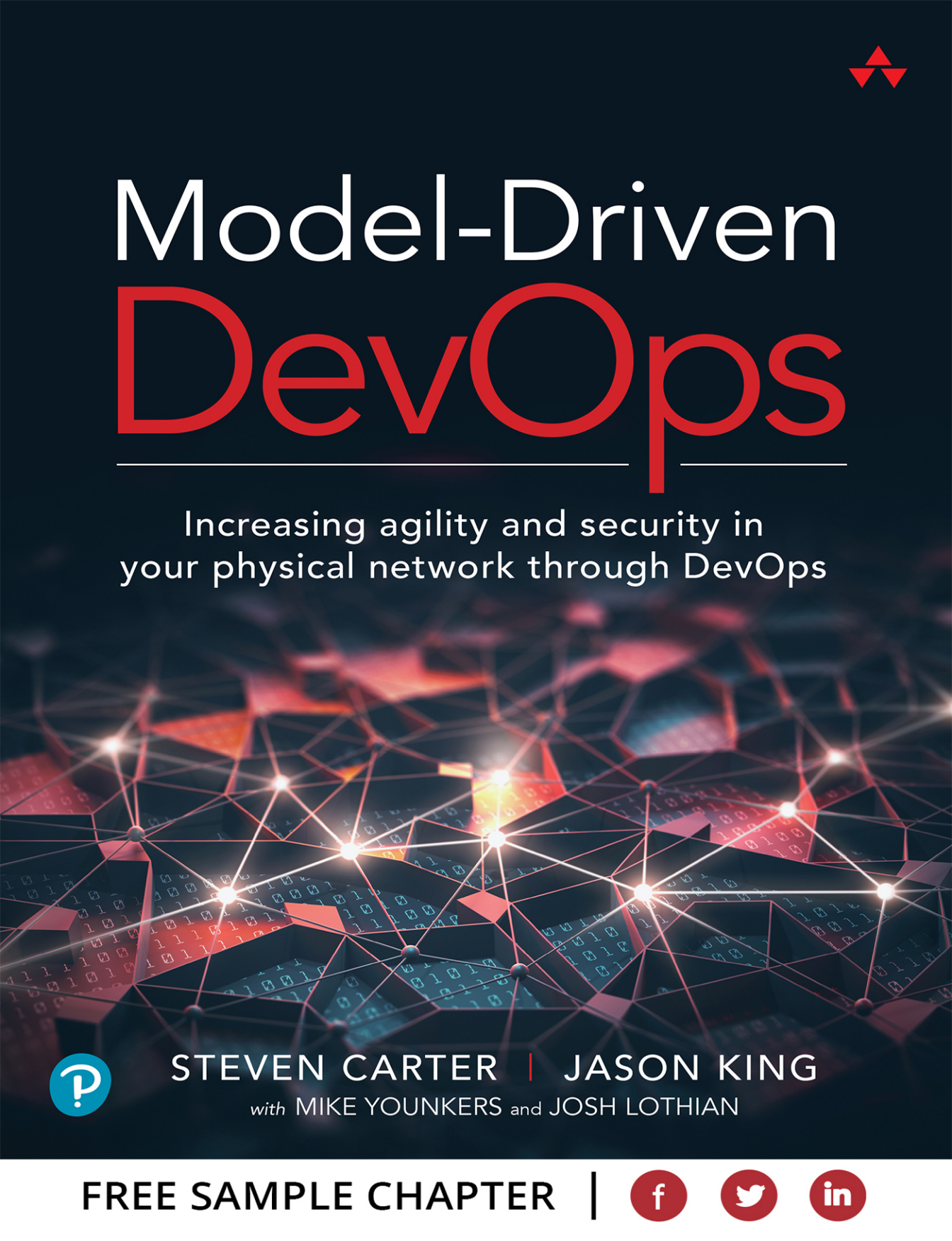
Model-Driven
DevOps
Increasing agility and security in your physical network through DevOps
STEVEN CARTER | JASON KING
with MIKE YOUNKERS and JOSH LOTHIAN
FREE SAMPLE CHAPTER

Model-Driven DevOps

This page intentionally left blank

Model-Driven DevOps

Increasing agility and security in your physical network through DevOps

Steven Carter and Jason King

with

Josh Lothian and Mike Younkers

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2022938003

ISBN-13: 978-0-13-764467-4
ISBN-10: 0-13-764467-1

Editor-in-Chief
Mark Taub

Director, ITP Product Management
Brett Bartow

Executive Editor
Nancy Davis

Development Editor
Christopher A. Cleveland

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Chuck Hutchinson

Indexer
Erika Millen

Proofreader
Barbara Mack

Technical Reviewer
Gerald Dykeman

Editorial Assistant
Cindy Teeters

Cover Designer
Chuti Prasertsith

Compositor
codeMantra

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.

I dedicate this book to my beautiful wife, Ana, and my children, Renée, Michael, Andrew, Rita Maria, Therese, Emma, and Dominic. I am grateful to have had such amazing opportunities to work at companies with talented individuals that helped me build the experience presented in this book and provide a wonderful life for my family. My wife and children support me, inspire me, and love me. I thank God for them and all that He has given me.

—Steven Carter

I would like to dedicate this book to my wife, Erika, daughter, Julia, and son, Josh. It is cliche, but honestly, it would not have happened without them. It turns out that writing a book is not an individual effort undertaken solely by the author, but rather a team effort. Everybody on the team needs to make sacrifices, and my family certainly made many during the writing of this book. I sincerely hope that people find this book useful, because Erika has informed me that it is probably my last. It was unexpected, but I found great satisfaction in writing the fictional portions of this book. Who knows? Maybe I can convince her that Bob from ACME Corp is more than just a successful DevOps engineer, if only the world knew his shocking secret…

—Jason King

Table of Contents

Preface

The Internet is built on network infrastructure. Many technologies, and by extension many economies and societies, are built on the Internet. Unfortunately, the way organizations deploy and maintain these critical networks has not changed meaningfully in 30 years. Network infrastructure operations is often a very human-intensive and manual process, making it prone to error and slow to react to business needs. The DevOps model promises to dramatically improve infrastructure operations using automation, tools, and processes designed to increase agility, scale, security, compliance, and reliability. Although DevOps has been used to great effect in applications development and management of cloud infrastructure, there has been no comprehensive, structured approach for applying DevOps to network infrastructure.

One primary way in which DevOps applied to network infrastructure differs from application DevOps is the number of elements managed and the amount of data on each of those elements. Essentially, this makes network infrastructure DevOps a data management problem. Networking vendors use data models to organize the data within each individual network element and to regularize their APIs, yet these data models are different between vendors and even between device families from the same vendor. Model-driven DevOps seeks to normalize the data models used to organize the data across the entire infrastructure as well as to normalize the code. In a sense, model-driven DevOps is intended to provide a repeatable, deterministic way to apply DevOps to network infrastructure and achieve the same benefits as DevOps applied to cloud infrastructure.

Vision

This book represents a journey that the authors have taken over the last couple of decades. We all started our careers with our hands on a keyboard, running large networks and even supercomputers. Driven partly by the demands of the organizations that we have worked for and partly by laziness, we have leveraged some form of automation through it all. As we progressed throughout our careers, some of us went into consultancy, some development, and some management. We have been privileged to work with, and for, many amazing companies with many talented people. It was our vision that this book contains the distillation of what we have learned over the years and how we apply it to solving customer challenges today. Using this experience, we seek to provide a holistic approach to applying DevOps to infrastructure operations organizations. This book lays down an extensive foundation that helps developers and operators apply and tailor the detailed, prescriptive approach laid out for infrastructure DevOps. Furthermore, it addresses the human and organizational factors that, left unaddressed, cause many organizations to fail.

We also want this book to be approachable and usable. It is our opinion that the skills required to be a network operator or network engineer have fundamentally changed. The API is the new CLI. The material in this book is meant to help network operators and engineers start retooling their skills to

operate their infrastructure in line with the way their colleagues operate cloud infrastructure. To reinforce this approach, we added a fictional storyline that, in our experience, illustrates the challenges faced in organizations that lead them to make this change.

Finally, we wanted to focus on outcomes and provide plenty of code to enable that outcome in your organization. We focus on industry standard tools and methodologies. Where possible, we use open-source or free tools. When we do have to choose a vendor solution, we do so in a way that makes it a choice for a particular implementation. That is, using different vendor implementations for various components would not significantly change the principles, framework, or even the code that we present.

Who Should Read This Book?

This book is targeted at infrastructure teams within the Information Technology sector running physical networks, although the principles apply to any infrastructure team. We take a deep dive into model-driven DevOps and define it through use cases and specific examples in our open-source, companion code repositories. In addition to IT infrastructure teams, this book is also applicable to cybersecurity teams looking to build security into their infrastructure at all stages. And finally, the human factors section is targeted at individual contributors as well as business and technical leaders who want to understand modern best practices as they relate to achieving high-quality results through teams.

How This Book Is Organized

The chapters of this book follow a logical progression. First, we examine *why* network infrastructure operations need to change, then we explore *what* needs to change, and finally we show you *how* to change it. Your journey includes a reference implementation of model-driven DevOps that will guide you through how to apply the techniques and concepts that you have learned. With this solid technical foundation in place, we end the journey with a discussion of the significant human factors to consider when making an operational change of this magnitude.

Along the way, you encounter exercises that will allow you to get hands-on experience, understand the technical details better, and test your knowledge. These exercises are based on the reference implementation and are identified throughout the book.

To provide some context and help illustrate many of the concepts in the book, each chapter starts with a fictional story involving a network engineer named Bob. Bob works for ACME Corp. ACME Corp is an intentionally generic company with a typical organizational structure including a CIO, various IT silos, and consultants. Most importantly, it operates network infrastructure in a very human-intensive, hands-on keyboards fashion. At the direction of the CIO, Bob is on a journey to DevOps. It is through his challenges, spectacular failures, and ultimate success that we see the problems of the legacy operational model and how automation, and ultimately DevOps, can enable true business transformation.

Book Structure

Each chapter in this book is intended to build on the previous chapter. Infrastructure DevOps is a journey, and the chapters are arranged in a way intended to guide you along the journey. The following is a brief summary of each chapter and how it fits into the journey.

- **Chapter 1, "A Lightbulb Goes Off":** In this chapter we illustrate why the legacy operational model for network infrastructure needs to change, briefly give an overview of how DevOps might address many of the problems with the legacy model, and explore reasons that DevOps is not widely adopted for on-premises IT infrastructure.
- **Chapter 2, "A Better Way":** In this chapter we define the goal of business transformation, begin to discuss the high-level framework for model-driven DevOps, and introduce concepts such as source of truth and data models.
- **Chapter 3, "Consumable Infrastructure":** If network infrastructure is to become an enabler of business transformation, we need to get away from the box-by-box CLI management model. This chapter makes the case that the API is the new CLI and explores ways that we can leverage and scale APIs.
- **Chapter 4, "Infrastructure as Code":** Although APIs enable you to work with network infrastructure programmatically, you don't have to be a programmer to take advantage of them. This chapter explores how you can refer to the network infrastructure "as code" using concepts such as data models, source of truth, configuration management tools, and templating tools. Together, these tools enable infrastructure as code and let you operate your network infrastructure just like you would in "the cloud."
- **Chapter 5, "Continuous Integration/Continuous Deployment":** Infrastructure as code is incredibly powerful but, like many powerful things, carries a great deal of risk if applied indiscriminately. In this chapter, we explore the concepts of version control systems, data validation tools, simulation platforms, and CI/CD. Together, these tools enable the safe use of infrastructure as code at scale and automated compliance and security.
- **Chapter 6, "Implementation":** Books on DevOps often focus on the *why* and the *what*, but they often omit the *how*. In this chapter, we take the concepts and techniques covered in the previous chapters, bring them all together, and apply them to a reference implementation. The reference implementation is published as a repository on GitHub so that you can get hands-on experience with model-driven DevOps as well as modify or extend the code to meet your own needs.

- **Chapter 7, "Human Factors":** Much of the text of this book is focused on the technical aspects of implementing model-driven DevOps. However, the technical challenges are only part of the journey. The importance of the human factors involving the breakdown of organizational silos, culture change, and the skills gap cannot be overstated. This chapter outlines why it is not enough to focus only on the technical capability, but also on the human side of implementing DevOps.

Acknowledgments

This book was truly a team effort. In addition to our own experience, much of the information in this book is informed by many of the companies and federal, state, and local organizations that we have worked with over the years. It would be hard to list them all, but we'd like to thank Captain Kyle "Chet" Turco, U.S. Navy, for his relevancy to the content of this book. We would also like to thank Lee Van Ginkel, Cisco Systems, and Gerald Dykeman, Red Hat, for the collaboration and proofreading they provided. Furthermore, we'd like to thank Craig Hill and Stephen Orr for their mentorship and guidance. Finally, there is a team of systems architects and developers, without which much of the code that underpins this book would not exist; in particular, we would like to acknowledge Steven Mosher, Tim Thomas, and Mitch Mitchner.

About the Authors

Steven Carter has more than 25 years of industry experience working in large universities, government research and development laboratories, and private sector companies. He has been a speaker at several industry conferences and written blogs and articles in technical journals. He has spent time as a system administrator running some of the world's largest supercomputers and a network engineer building out the world's first SDN network for the Department of Energy. In addition, Steven has a wide range of experience in networking, including operations, embedded software development, and sales. He has spent the past 5 years working for Red Hat Ansible and Cisco Systems consulting and coding for many of the world's largest organizations as they modernize and secure their operations by incorporating DevOps. He currently works as a principal DevOps engineer for Cisco Systems creating CI/CD pipelines for deploying cloud applications and network infrastructure in secure and classified environments. He holds a BS in computer engineering, an MS in computer science, an MBA, and a CCIE in routing and switching.

Jason King is a solutions architect at Cisco, supporting the public sector community. In his 11 years at Cisco, he has focused on cloud, automation, programmability, and HPC. Prior to joining Cisco, he spent 10 years building and tuning some of the world's largest HPC clusters at Lawrence Livermore National Laboratory. He holds an MS in computer science and a CCIE in routing and switching.

About the Contributing Authors

Josh Lothian has worked in system administration and DevOps for more than 20 years. In his career he has supported everything from academic departments to the fastest supercomputer in the world. He has focused on how automation can lessen the burden on staff while increasing efficiency and reliability. In his 6 years at Cisco as a senior cloud engineer and technical leader, he has helped build small, highly collaborative teams that produce big results using DevOps principles. He holds a BS and an MS in computer science.

Mike Younkers has 30 years of industry experience working for the U.S. government and private sector companies. He has been an operator, engineer, and architect of global networks, cybersecurity capabilities, and other IT-related systems. Over the years he has held various leadership positions where he has been impacted by industry changes and led multiple teams through various transformations. He is currently a senior director of Systems Engineering on Cisco's U.S. federal team, where he has the privilege and honor to work alongside some of the brightest and most dedicated people in our industry. He holds a BS in electrical engineering (BSEE), a BS in computer science, and an MS in telecommunications and computers. It is the combination of this education and these years of experience that inspired him to want to be an active participant in this effort.

Chapter 3

Consumable Infrastructure

In the preceding chapter, we explained the power of model-driven DevOps and the use of data models. To start the journey toward model-driven DevOps, your infrastructure must be consumable in a programmatic way. Therefore, in this chapter we discuss the concept of consumable infrastructure. *Consumable infrastructure*, simply put, is infrastructure that is interacted with through APIs via data models. Consumable infrastructure rapidly responds to the needs of an organization through APIs and platform-based simplification. As we explore later, we want to look at the network in terms of consuming services and capabilities and not just automating specific tasks. Consumable infrastructure greatly reduces the complexity of automation and makes it more accessible to operators as opposed to requiring deep programming expertise.

Automating Things to Do Stuff

When we last left Bob from ACME Corp, he was considering a new job offer from a competitor while the CIO, Haley, was just beginning to understand the true extent of Bob's value to the company. It was clear that ACME Corp's current methods for operating IT infrastructure were not aligned with the business's goals of greater agility and lower risk. In fact, Haley was beginning to realize that she had a very large risk of disruption to the business if Bob left for a competitor.

Haley was becoming convinced that automating her IT infrastructure and implementing DevOps could meet her twin goals of increasing agility and lowering risk. She needed to take somebody like Bob and transform his knowledge into code while making the shift to DevOps. She knew this effort was going to be a significant amount of work, but she also knew that it was the right way forward. She would start right away.

When Bob walked into the office on Monday, he had a new meeting on his calendar with the ominous title "Network Automation Discussion." Bob had built his career at ACME Corp through his knowledge of network design and protocols, combined with the ability to turn those designs into reality via the CLI. This was the value he brought to the company.

A couple years ago, he got excited and tried to do some simple automation using a script that would log in to a device and issue a few commands. His excitement did not last long. He quickly discovered that trying to parse the output of a list of commands was far from simple. The CLI was easy for humans to parse, but it turned out to be far trickier for a machine to understand the result of a configuration change or the output of a show command. He had to write code that would look for and parse certain language in the output. Writing this code was difficult, and even if he could do it, he knew that the CLI was different for each device in ACME Corp's network. Writing unique code for hundreds or maybe thousands of individual devices was not something Bob had time for. The effort might pay off in the long run, but Bob had a network to operate. Like many network engineers before him, instead of investing all that time in custom language parsing code, he fell back to what had worked for the last few decades: managing every device, individually, through the command line.

Recalling his previous failed attempts, Bob was now concerned about the notion of automating network infrastructure. At the start of the meeting, Jane, Bob's manager, laid out the grim truth to the network team. She said, "Our CIO is convinced that IT infrastructure automation will improve our efficiency, agility, and lower the risk of outages." Like virtually every meeting involving network engineers, when the word automation was mentioned, a collective groan erupted. Jane heard comments such as "Over my dead body!" and "Not on my watch!" Bob, with a somber expression, said, "We tried that before, and it just doesn't work." Although, deep down, he knew he was overworked and unhappy, he had been involved in every critical change to the environment, and because developers were deploying code multiple times a day, critical changes were becoming a regular occurrence. There had to be some merit to the CIO's automation strategy. Jane, in a firm tone, responded, "Well, what we are doing today is not working. Something needs to change. Any suggestions?" She made direct eye contact with Bob. He quickly reflected on his past automation failure and thought about what could be done to fix it. He knew that APIs are supposed to help with some of the previous issues, and even though he was not excited about learning something new, Bob offered a suggestion. "Okay, we all have had issues scripting network changes through the CLI. It is, well, painful." He heard murmured agreement from the team. He offered a half smile and continued, "So why don't we look at using APIs to automate our network infrastructure? Our vendors are always talking about how great APIs are." "That is an excellent suggestion, Bob! I would like you to take the lead on this effort," Jane replied. Bob's smile faded. "That's what I get for opening my big mouth," he thought.

A week later, Bob picked up a ticket submitted by the person who manages the ACME Corp NTP servers. Due to changes occurring in the data center infrastructure, they needed to migrate the NTP servers to new IP addressing, and doing so required that the NTP server configurations on every network device be modified. He had been studying how to interact with network devices through their APIs for the last few days, and he thought that this might be a good opportunity to put his new skills to use.

After some trial and error using a tool called Postman, Bob was able to make a change through the API on one of the more modern devices in the lab. He verified that the change occurred successfully by examining the API return codes. "Success!" he shouted as he raised his fists in triumph. Then he checked the resulting configuration to find that his change only added new servers to the list instead of replacing the list. This process was going to be more complicated than he thought.

It turns out that things like simple lists are not so simple to automate without some more complicated logic to check for existing configuration. After more trial and error, he was finally able to script the process of retrieving the existing configuration, reconciling the old list with the new one, and pushing only the needed changes to the device, all via the API.

He then moved on to one of the older devices in the ACME Corp network and quickly discovered that it had no API. It was a CLI-only device. "Crud," he thought. "It looks like I might need to automate this device some other way." So, he looked at another common tool for network infrastructure automation called Ansible and discovered that an Ansible module supports this older device. After some trial and error with Ansible, Bob was able to make a playbook that achieved the same result on the older device, but unfortunately, it was done in a completely different tool with completely different syntax. "This is going to be ugly," Bob thought. "Even if I somehow figure out how to automate all the different devices in whatever tool or language works for that device, I still need to figure out how to verify the results of each operation and somehow scale all these different operations to thousands of network devices. This is going to be really ugly," he thought again. "There must be a better way."

APIs

An application programming interface (API) is a way for two *applications* to interact with each other. In contrast, a command-line interface is a way for a *human* to interact with an application to retrieve data or make configuration changes. In the context of IT infrastructure, an API interaction is usually a two-way communication where some data is sent from an application to a device or controller platform for the purposes of retrieving operational information or making configuration changes.

APIs are a critical component of model-driven DevOps. As the name implies, model-driven DevOps makes heavy use of data models. API software on a device takes data and, using a data model to decipher that data, configures the various components of the device the way the manufacturer intended.

When network infrastructure is treated as a set of APIs, configuration consists of moving data, generally in the form of JSON or XML, between those APIs. This capability makes network operations more like cloud and application development. This type of interaction is a significant improvement over the legacy human-optimized CLI interaction.

The most common model-driven APIs for network devices use the NETCONF protocol with YANG data models. NETCONF pushes the data models encoded in XML over a secure transport layer and provides several operational advantages over CLI, including

- Installation, manipulation, and deletion methods for configuration data
- Multiple configuration data stores (such as candidate, running, startup)
- Configuration validation and testing
- Differentiation between configuration and state data
- Configuration rollback

Why API over CLI?

For decades, network engineers have used CLIs to configure network devices. A CLI is, for the most part, an effective human-to-device interface. When it comes to automating network devices, however, a CLI is a poor computer-to-network device interface. The main reason for this is that most CLIs are meant to be human readable; therefore, most CLIs have a language-like construction that makes it easier for humans to use. Unfortunately, human languages are difficult for computers to use.

To illustrate, let's first look at a simple example of configuring the hostname on a Cisco IOS device. We use Ansible because it is one of the most popular ways of automating network devices, but the problem we are about to describe exists with most any CLI-based method of automation. Using Ansible parlance, we describe the desired end state of the hostname of a particular device. A hostname is a great use case because it is a scalar (that is, a single value). To change the hostname, the Ansible `ios_config` module does a simple textual comparison of the configuration. To set the hostname using Ansible, you would use the following YAML in a playbook:

```
- ios_config:
    lines:
      - hostname newname
```

If `hostname newname` is not present, it sends that line to the device. Even if a different hostname is present on the target device, because `hostname` is a scalar, the old hostname gets replaced by the desired hostname. However, as Bob painfully discovered, a list of NTP servers is more difficult. Suppose you've set the NTP server to 1.1.1.1 with the following YAML:

```
- ios_config:
    lines:
      - ntp server 1.1.1.1
```

Now you want to change your NTP server to 2.2.2.2, so you modify the YAML:

```
- ios_config:
    lines:
      - ntp server 2.2.2.2
```

Simple, right? But the problem is that you would end up with two NTP servers in the configuration:

```
ntp server 1.1.1.1
ntp server 2.2.2.2
```

The reason is that the Ansible `ios_config` module does not see `ntp server 2.2.2.2` present in the configuration, so it sends the line. However, because `ntp server` is a list, it adds a new NTP server instead of replacing the existing one, giving you two NTP servers (one that you do not want). To end up with just 2.2.2.2 as your NTP server, you would have to know that 1.1.1.1 was already defined as an NTP server and explicitly remove it. This is also the case with ACLs, IP prefix-lists, and any

other list in IOS. The Ansible `ios_config` module (as well as the `cli_config` module) does not have a native way to describe the desired end state of something simple like NTP servers on a network device, much less something more complex like OSPF, QoS, or Multicast.

There are clearly ways to address this situation. For example, the Ansible `ios_config` module could be improved to know how to parse IOS syntax and look for any existing NTP server configuration and remove it, just as a human would. One problem with this approach, however, is that this more capable module would be re-implementing IOS parsing rules outside of IOS. This means that vendor changes to the IOS CLI would necessitate changes to the `ios_config` Ansible module, creating a maintainability problem that would often result in lagging functionality. Furthermore, this approach would need to be taken with every vendor and/or device CLI available, making this approach unscalable.

The better way is to use an API. APIs are specifically designed for programmatic configuration of devices. To illustrate the advantages of the model-driven method using an API, let's use the `netconf-console` utility to get and set the NTP servers on a Cisco IOS-XE device. First, let's see the current list of NTP servers. Listing 3-1 illustrates how to retrieve NTP configuration using `netconf-console`.

LISTING 3-1 Using netconf-console to Retrieve NTP Configuration

```
# netconf-console -host <device IP> --port 830 -user admin -password admin -db run-
ning -get-config -xpath /native/ntp
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <ntp>
      <server xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ntp">
        <server-list>
          <ip-address>1.1.1.1</ip-address>
        </server-list>
        <server-list>
          <ip-address>2.2.2.2</ip-address>
        </server-list>
      </server>
    </ntp>
  </native>
</data>
```

Although this example is a wordier rendering of the same configuration, it is deterministic. All the NTP servers and their associated configuration are organized into one section of the tree. We can deal with it as a separate entity as opposed to being thrown in at the same level with other configuration information. Also, note that we were able to ask the device for *just* the NTP configuration information. No parsing required.

Now let's change the NTP servers. First, we take the previous output, change the IP address of the second NTP server, and specify that this operation should replace the server section with `operation='replace'`. The content of Listing 3-2 would normally go into a file named ntp.xml.

LISTING 3-2 XML to Change NTP Configuration

```
<native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
  <ntp>
    <server xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ntp" operation='replace'>
      <server-list>
        <ip-address>1.1.1.1</ip-address>
      </server-list>
      <server-list>
        <ip-address>3.3.3.3</ip-address>
      </server-list>
    </server>
  </ntp>
</native>
```

Then we can push the XML payload to the device using `netconf-console`, as shown in Listing 3-3.

LISTING 3-3 Using netconf-console to Change NTP Configuration

```
# netconf-console -host <device IP> --port 830 -user admin -password admin -db
running -edit-config ntp.xml
<ok xmlns="urn:ietf:params:xml:ns:8equire:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:8equire:base:1.0"/>
```

The `ok` in the XML response indicates that the device accepted the change. Now let's retrieve the NTP configuration and verify the results, as shown in Listing 3-4.

LISTING 3-4 Using netconf-console to Verify Configuration Change

```
# netconf-console -host <device IP> --port 830 -user admin -password admin -db
running -get-config -xpath /native/ntp
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
    <ntp>
      <server xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-ntp">
        <server-list>
          <ip-address>1.1.1.1</ip-address>
        </server-list>
        <server-list>
```

```
          <ip-address>3.3.3.3</ip-address>
        </server-list>
      </server>
    </ntp>
  </native>
</data>
```

The first thing that you might notice is that this was a lot of work just to change the NTP server. In the same way that human-to-device interfaces are not optimal for computers, computer-to-device interfaces are not optimal for humans. Having a machine-friendly, deterministic, and repeatable way of making changes in the environment is the necessary foundation for effective automation. Assembling a series of programmable operations into more complex workflows is where you start to see real efficiency advantages.

Even though computer-to-device interfaces might look daunting at first and are not optimal for humans, humans still need to understand them in order to help computers use them for automation. Figure 3-1 takes a closer look at this concept. Here we take data (the list of NTP servers that should be configured) from our source of truth, encode it into a payload as defined by a data model (XML in the case of NETCONF), and send the payload to the API (NETCONF in this case).

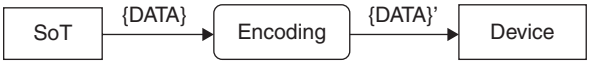

FIGURE 3-1 Encoding Data

This generic workflow can accommodate a host of use cases and APIs. For example, we can use a RESTCONF interface by simply changing the encoding to JSON (but using the same data model) and sending it to a RESTful interface. We can even stretch this notion to CLI-only devices by taking the data, encoding it as textual configuration tailored to that device, and delivering it via SSH. This yields a flexible framework, illustrated in Figure 3-2, with the flexibility to deliver the data via any encoding to any device using any API.

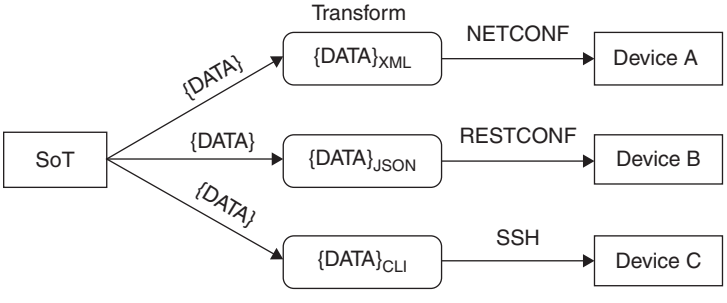

FIGURE 3-2 Encoding Data for Different APIs

This approach works for physical network infrastructure, but it also works for cloud infrastructure. For example, AWS CloudFormation is just source of truth data encoded in JSON using a data model and delivered via an API using an SDK like Boto3 for Python. Therefore, thinking in this manner allows you to use the same methodology for your entire IT infrastructure.

Platforms

A platform, in the context of model-driven DevOps, is a consolidation and simplification point for the devices in a network. Without platforms, you would need to configure each device individually, making IT much more cumbersome and time-consuming. The best examples of a platform are cloud infrastructure providers like AWS, Azure, and Google. They created platforms that provide abstracted services, such as compute, storage, or networking to an IT organization. No longer does an organization need to think about acquiring hardware, configuring that hardware into a system to support applications, and maintaining that hardware over time. This process is all abstracted as a service and provided via API. For example, a customer does not care what types of nodes are used for their compute service or how to configure them; that customer just wants the service to work. Platforms come in different forms with different capabilities, but they all aim to simplify IT and generally contain many of the attributes we describe in the remainder of this chapter.

Physical Hardware Provisioning

The idea of a platform also extends to the physical, on-premises network. Unlike cloud, where you don't have to know or care about physical hardware, with on-premises infrastructure, it is a significant concern. To ease the deployment and provisioning of hardware, many platforms support technologies such as "plug and play" or "zero-touch provisioning." This is also commonly known as *Day 0 provisioning*, where a minimal configuration is automatically applied to a piece of physical hardware on bootup so that it can communicate with the platform to get a more complete, or Day 1, configuration. This book focuses mainly on Day 1 configuration and Day 2 operations, but it is useful to know that most platforms also ease the Day 0 provisioning of physical hardware.

Consolidated Control Point

Whether it is cloud or on-premises, the main benefit of a platform is the consolidated control point. The idea of a consolidated control point in a network came to prominence with the advent of software-defined networking (SDN). SDN decouples the control plane (the part that decides where to send packets) of a network from the data plane (the part that does the forwarding of the packet). The devices in a pure SDN network have little to no ability to operate autonomously, rendering them inoperable without a central controller. Over time, however, this pure approach largely found equilibrium in devices that can either operate autonomously or as part of a controller-based fabric. A more pragmatic approach to SDN evolved whereby devices in the network operated as part of a distributed control

plane with a centralized controller managing network configuration and policy. This approach took the best parts of pure SDN (consolidated control point) and married it with the best parts of distributed control planes (scale and resiliency). In the context of networking, this more pragmatic approach to an SDN controller is what we refer to as a platform.

Northbound vs. Southbound APIs

In the IT infrastructure space, it is often useful to think of platform APIs as either northbound or southbound. A typical controller platform exposes a "northbound" API that is intended to provide functionality for other applications. A good example is the UI for the controller itself. Often the UI for the controller uses this same northbound API to retrieve data and make configuration changes. When a request is received via the northbound API, and the controller software determines that it needs to make changes to one or more devices, it uses a "southbound" API to talk with the various devices. These southbound APIs are specific to whatever API is exposed by the devices in the network. Different device vendors often have different APIs for their devices. As shown in Figure 3-3, a controller platform can consolidate many disparate vendor or device APIs into a single, unified, northbound API.

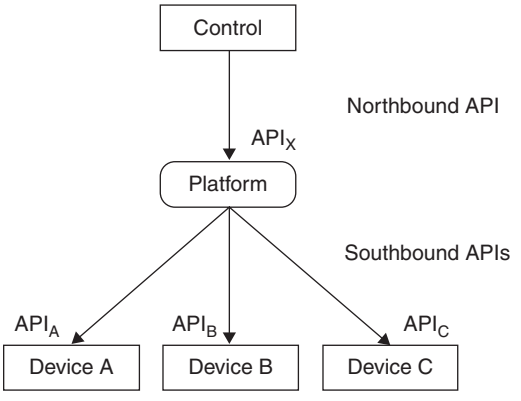

FIGURE 3-3 Northbound vs. Southbound APIs

API and Feature Normalization

One important role that a platform can play is to normalize the API across a set of dissimilar devices. From our previous example, the platform would perform any data transformations internally and transparently while presenting a single API to the user (see Figure 3-4).

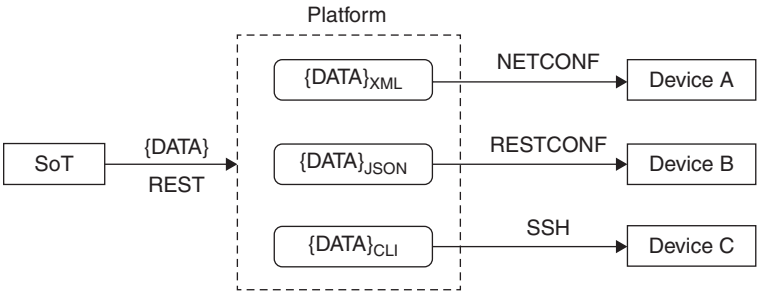

FIGURE 3-4 Platform API Normalization

This normalization greatly reduces the complexity of automating the network by allowing the automation tooling to work on a single data model against a single API. Without this regularization, the tooling would have to do the data model conversion and then call the correct API for each type of device available on the network. Recall that when Bob from ACME Corp realized that each type of device on the network had a different command line, requiring its own custom code, it seemed unmanageable. A platform with the ability to normalize to a single data model with a single API solves this issue.

Platforms further help by normalizing features across many dissimilar devices of varying capabilities. For example, many network devices do not have the capability to roll back configuration changes to a previous state. If state is added to the platform, then the platform can track changes as they occur so that it can return a device's state back to its configuration before a change occurred. Furthermore, the storing of state in the platform allows for the comparison of what the state of a device should be in case out-of-band changes are made. If the device is out of sync with the platform, then the local change can either be adopted or overridden.

Fabricwide Services

In addition to normalization, a platform can provide fabricwide services to the network. One common fabricwide service is Ethernet Virtual Private Network (EVPN). EVPN is used to extend Ethernet Layer 2 services across a large campus or between sites over a Layer 3 routed network. It is considered a fabric technology because it relies on a central control plane based on BGP to distribute MAC addresses and other information that enables connectivity between end nodes. Without the control plane, the fabric does not function even though the individual boxes can function autonomously.

A platform can provide both a fabricwide view of the network and the services necessary to run that fabric (see Figure 3-5). These capabilities result in a substantial simplification of the network and enable services that would not be possible without this central function.

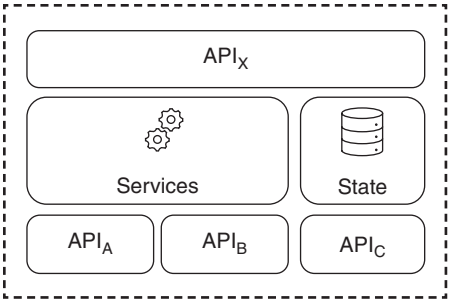

FIGURE 3-5 Fabricwide Services

Scalability

Platforms also enable a greater scale in automating networks. Without a platform, the control node used for automation needs to communicate directly with each device instead of being able to optimize communications by taking advantage of state in the platform (see Figure 3-6). As an example, let's look at the way that tools like Ansible make changes to a device. Ansible's goal is to get the devices to a desired end state. To do that, it needs to check the current state of the device, compare that state to the desired end state, and then send the changes to the device. This operation doubles the communication between the control node and the end devices. To further complicate the issue, many operations work only on discrete parts of the configuration, meaning multiple operations need to occur to make one change.

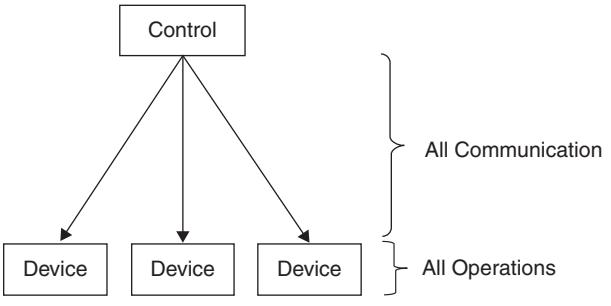

FIGURE 3-6 Scaling Control Communications

When we introduce a platform, the communication between the platform and the device can be reduced to a minimized set of consolidated changes (see Figure 3-7).

Figure 3-8 illustrates how this architecture can also scale geographically. For geographically dispersed networks, these intermediary platforms can provide regional aggregation and other control plane services.

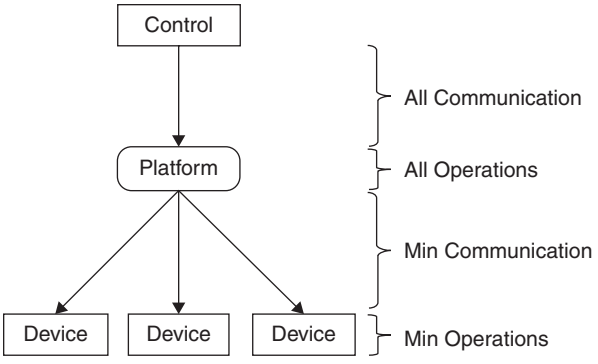

FIGURE 3-7 Better Scale Through Platforms

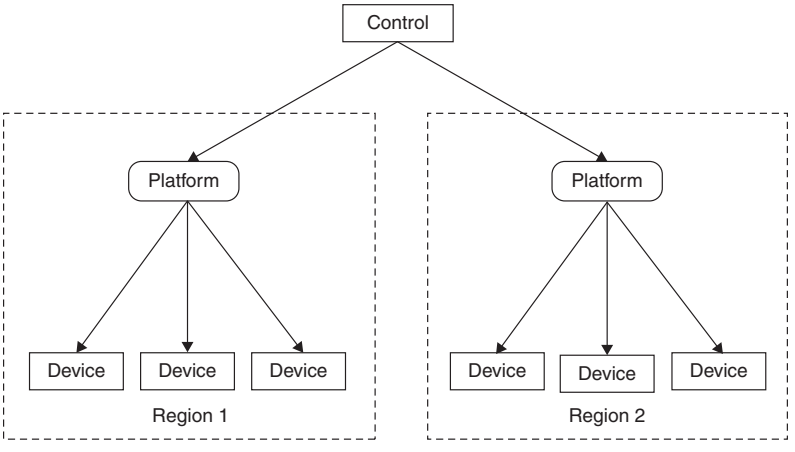

FIGURE 3-8 Scaling Platforms Geographically

Summary

This chapter covered the two main factors for successful infrastructure automation: APIs and platforms. We defined consumable infrastructure and demonstrated how the APIs coupled with data models discussed in the preceding chapter enable it. Furthermore, we illustrated how APIs enable deterministic and efficient machine-to-machine interactions and are critical for successful automation. Platforms enable you to better scale operations made via API, provision physical hardware, and enable you to build more complex automated services on top of your infrastructure. We also examined what makes cloud platforms so powerful and how those same platform concepts can be applied to on-prem infrastructure. In the following chapters, we cover how to represent your infrastructure as code and then how to take some of the high-level things you have learned so far and assemble them into a framework for infrastructure automation at scale.

Index

A

B

C

D

E

F

G

H

I

J

P

Q-R

S

T

U

V

W

X

Y-Z