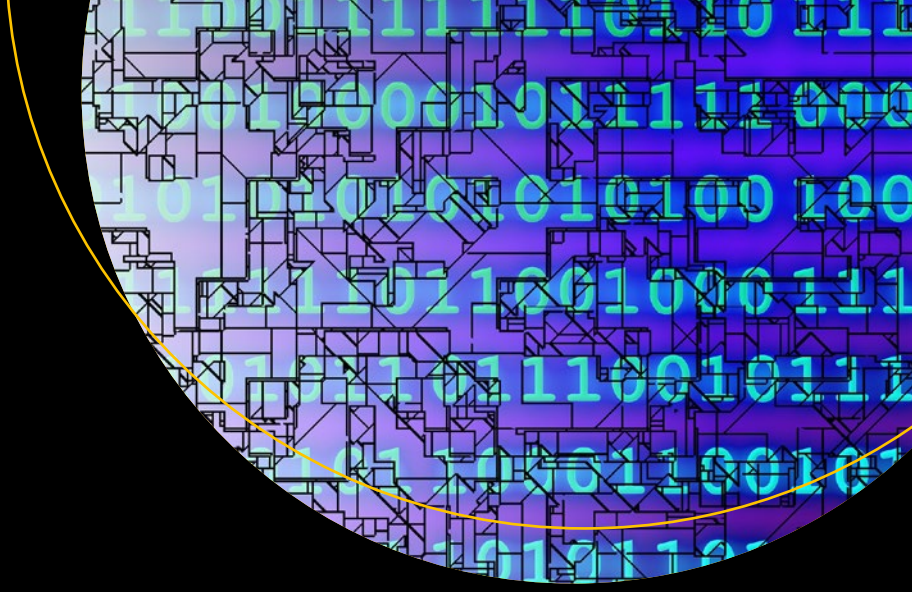

Introduction to Infrastructure as Code

A Brief Guide to the Future of DevOps

Sneh Pandya
Riya Guha Thakurta

Apress®

Introduction to Infrastructure as Code

A Brief Guide to the Future of DevOps

Sneh Pandya
Riya Guha Thakurta

Apress®

Introduction to Infrastructure as Code: A Brief Guide to the Future of DevOps

Sneh Pandya
Vadodara, Gujarat, India

Riya Guha Thakurta
Boston, Massachusetts, USA

ISBN-13 (pbk): 978-1-4842-8688-3
ISBN-13 (electronic): 978-1-4842-8689-0
https://doi.org/10.1007/978-1-4842-8689-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Spandana Chatterjee
Development Editor: James Markham
Coordinating Editor: Shrikant Vishwakarma
Copy Editor: Kim Wimpsett

Cover designed by eStudioCalamar

Cover image by Pixabay (www.pixabay.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, email orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

For our parents, who have given everything and always seem to find a way to give more.

Table of Contents

About the Authors

Sneh Pandya is an emerging Product Management leader with specialization in strategic leadership. He advocates for product strategy, digital transformation, and sustainable innovation.

His qualifications and certifications include a bachelor's degree in computer science and engineering and further studies in strategy management and leadership at the Wharton School, University of Pennsylvania.

Sneh is also a developer community leader at Google Developers Group, Baroda, and has given public talks at several worldwide developer conferences. He is a cofounder of the *NinjaTalks* podcast, which brings together experiences from the world's leaders, changemakers, and innovators to make knowledge accessible to all.

With extensive experience in the field of technology, including mobile and web software applications, DevOps, the cloud, infrastructure automation, and software architecture, he has written several articles for various technology publications.

Riya Guha Thakurta is a graduate student in Computer Information Systems at Boston University. Her undergraduate education includes a bachelor's degree in computer science application from the Institute of Engineering and Management.

She was formerly a Scrum Master in the realm of technology management, and her previous experiences with Johnson Controls include product development across several business products and software verticals. Riya also leads Women Techmakers, Kolkata, a diversity, equality, and inclusion community that encourages women in technology. She is also an Intel Software Innovator for the Internet of Things.

She is a cofounder of the *NinjaTalks* podcast, which seeks to share experiences from the world's most prominent leaders, changemakers, and innovators to make knowledge accessible to all.

Her diverse interests include technology and project management, sustainability, public speaking, and research and development.

About the Technical Reviewer

Joaquin Gonzalez is a senior reliability engineer working in the finance sector developing and implementing observability and automation software to improve infrastructure robustness. He has a telecommunications engineering degree from Universidad Nacional de San Martín (Argentina). He completed a master's program in analytics at Universidad Torcuato Di Tella (with a final dissertation in progress). In his free time, he likes amateur radio, loves to read about science, and is lucky enough to share his life with his wife and daughter.

Acknowledgments

We are forever grateful to our parents; they are our eternal source of happiness and inspiration. We are also thankful to each other for the constant support and encouragement while writing this book.

Our special thanks to all the wonderful people who were part of this journey. We would like to thank our friends and allies in the industry for sharing their insights and knowledge with us, which helped us produce this meaningful book.

We want to sincerely thank the people at Apress who made this book possible: James Markham, Spandana Chatterjee, Mark Powers, Clement Wilson Kamalesh, Silembarasan Panneerselvam, Shrikant Vishwakarma, Joaquin Gonzalez, and everyone involved.

We hope you will have as much as fun learning from this book as we did writing it.

Introduction

The management of information technology systems using a declarative programming language is known as *infrastructure as code*. Tools like source code repositories, build servers, automated testing, DevOps pipelines, etc., are used to improve the quality of software engineering. A common practice is to handle infrastructure code in the same manner as application code. In today's rapidly transforming world, infrastructure as code (IaC) has emerged as a futuristic approach to maintain, scale, and deploy software systems. From small-scale products to enterprise-grade automation, the field has enormous numbers of opportunities for DevOps-focused solutions that can transform the industry.

This book is a balanced mix of fundamental understanding and hands-on experience. You will understand the importance of DevOps culture as well as how to adapt to IaC in your workplace. The beginning of the book helps conceptualize the futuristic goals that DevOps and particularly IaC has to offer for the adoption and growth in different verticals. With the solid base prepared, you will then learn the importance, processes, and outcome of building infrastructure solutions.

This book is aimed at beginners interested in building a career in DevOps as well as existing professionals looking to gain expertise and advance in their careers with a mastery of IaC. This book simplifies jargon for businesspeople and decision-makers, including technical product managers, architects, and anyone who aims to shape and scale their infrastructure to provide better products and services to customers.

You will learn the following:

- The fundamentals of DevOps and IaC
- The ever-evolving ecosystem of modular infrastructure and the needs of the future
- How to avoid potential pitfalls and breakdowns while working with infrastructure
- How to build scalable and efficient IaC solutions that work on a small, medium, and large scale in a real-life environment
- How to understand and be responsibly aware of the security concerns related to the domain and how to address them

Thank you for reading *Introduction to Infrastructure as Code: A Brief Guide to the Future of DevOps*.

PART 1

Concepts

CHAPTER 1

Introduction to Infrastructure as Code

As a consequence of the incredible advancement of new technology, both large and small companies are more reliant on the digital world for survival and success in the global market. Businesses that are not online are unable to reap the many advantages of the digital world, and as a result, they are losing out on a plethora of possibilities. In today's fast-paced and competitive corporate environment, there is an extraordinary demand for a quick development process, rapid deployment and delivery, bug fixes with little downtime, and prompt release of new features.

All of these many duties are handled either by individual subject-matter experts or by a specialized team of subject-matter experts. However, their seemingly lack of direct contact with each other renders them incapable of completing the task on time or producing the required product efficiently. Fortunately, DevOps provides a solution to each of these significant issues. The term has gained a lot of attention in recent years, and many businesses are incorporating its ideas into their operations.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_1

The Culture of DevOps

DevOps is more than just a technological accomplishment; it is a paradigm shift in software engineering that seeks to integrate development and operations. DevOps aims to shorten development life cycles, increase frequency of deployment for the end users, and provide robust releases that are closely aligned with business objectives. It is a vocal proponent of automation and monitoring across the software development life cycle, from integration and testing through deployment and infrastructure management.

It's all about system-level thinking in DevOps, which means focusing on the end-to-end value delivery process rather than the individual effort silos that comprise the process. As a result, good communication and alignment around agreed-upon, measurable objectives are required for successful cooperation.

DevOps is a combination of the phrases *development* and *operations*.

- *Cross-functional teams*: Because there is no distinction between teams, fast communication is required. This is particularly true for the development and operations teams. The information technology department of the company will be streamlined and well-coordinated. Creating, deploying, and maintaining software all require a multiskilled workforce.
- *Automation*: Thanks to container-based technologies coupled with microservices architecture, it is now feasible to completely automate all the phases of software development cycles. This includes development, testing, deployment, and monitoring. Monolithic programs and microservices-based applications are both difficult systems, and microservices do not make deployment any easier; in fact, they may make things much more complex during the implementation phase.

With the recent global trends and demands from the market, many giant organizations opt for DevOps practices, generating greater reliability, allowing them to experiment more, and allowing organizations to launch new products and features on a significantly faster timetable, which was possible only with the deployment of DevOps. Fundamentally, DevOps allows developers to own, run, and control an applications or piece of softwares end-to-end delivery. It reduces ownership ambiguity and encourages developers toward a single automated, developer-managed infrastructure.

> *It is important to realize that DevOps is much more than simply spinning up software. Rather than that, the primary objective is to alter an organization's culture, which, of course, is no simple task.*

Many process methods are now available to assist with DevOps. Perhaps the most common approach is *Agile*, which entails small teams developing apps in chunks, with each piece including user stories. The goal is to get continuous input and feedback from users and avoid being mired down in the process of developing a big-bang program, as development is an iterative process.

The Evolution from DevOps to Infrastructure as Code

DevOps refers to a collection of terminology, procedures, techniques, and ideas aimed at improving the efficiency, security, and speed of software development. The idea of automation, which involves replacing better technology for human processes to allow for a more frequent delivery pipeline, is central to the DevOps philosophy.

Immutable infrastructure is a novel concept in the realm of DevOps. Rather than requiring updates, immutable infrastructure begins with each deployment from the ground up. Cloud deployments in general, and containerization in particular, benefit from immutable infrastructure.

Maintaining servers and guaranteeing their availability around the clock needed dedicated full-time workers almost a decade ago. System administrators were needed to guarantee server availability at the time to operate mission-critical applications effectively.

Back-end administration has come a long way. The advancements in cloud technologies and cloud providers have changed infrastructure management, while DevOps has affected software development and delivery. These two game-changing concepts had a major impact on the field of software development.

Using a multicloud environment may be extremely beneficial for some businesses depending on their use cases, since it enables them to harness the advances of several cloud providers. While a multicloud strategy allows for rapid digitization of operations in response to customer demands, infrastructure management across public and private clouds—including edge environments—may be challenging.

What Is Infrastructure as Code?

Infrastructure as code (IaC) refers to the management of existing infrastructure components such as storage devices, networking, and communication channels as well as load balancers and servers. It is also a method for automating the provisioning of IT infrastructure that uses a high-level descriptive coding language. Developers are freed of the burden of manually provisioning and maintaining servers, operating systems, database connections, storage, and other infrastructure components while creating, testing, and delivering software applications. Alternatively, they may go with automation.

Infrastructure management has been a time-consuming full-time profession for decades. However, critical infrastructure management has evolved considerably during the last decade. Microsoft Azure, Google, Oracle, and Amazon Web Services are just a few of the public cloud providers that provide essential infrastructure management services to its thousands of customers. You may have heard the term *infrastructure as code* as the popularity of public cloud platforms has grown, as has access to and consumption of the infrastructure the platforms offer.

The Perspectives

One of the most essential components of software development, the system infrastructure, serves as the system's backbone. IaC is a technique for easing the delivery and management of IT infrastructure via the use of templates that contain configuration files that look and behave like the source code used by DevOps teams. This means you may treat device or system configurations similarly to how you would treat software source code. By using software development concepts, infrastructure setup may be made more reliable, repeatable, and observable to save time and efforts while reducing human errors. As a result, many IaC tools have been developed, the most notable of which being Kubernetes, Terraform, and Ansible. IaC is critical in hybrid multicloud environments—or when companies use a variety of cloud computing services from a variety of providers—where organizations want to distribute computing resources to prevent or mitigate the risk of data loss and unavailability.

It is often heard from IT leaders that their companies are having difficulty effectively adopting DevOps. Instead of utilizing their finest people to create mission-critical business apps, they have them construct and manage the DevOps infrastructure. The critical element is to have a constant beginning point. Creating a common picture that is independent of any tool or process may be a good place to start. A successful DevOps

team leverages immutable infrastructure and automation to generate shared images and components that serve as the foundation for environment provisioning.

Benefits Adapting Infrastructure as Code

According to Statista,[1] the public cloud's IaaS sales will grow from about $50 billion in 2020 to more than $80 billion in 2022, representing nearly a quarter of the total cloud computing industry.

Because of the time and expense involved in physically installing hardware, installing and configuring operating system software, and connecting to other systems such as middleware, networks, and storage, upgrading firmware and systems, etc., is both time-consuming and expensive manner.

Because virtualization and advancements in the cloud do away with the need for conventional hardware management, developers may immediately build their own servers, saving both time and resources. While virtual infrastructure is deployed, development activities are paused. Provisioning must still be done for each new deployment, and there is no easy method to keep track of changes in the environment or prevent inconsistent behavior that may result in deployment failures.

Infrastructure as code refers to technologies that enable it to treat the infrastructure as code, which allows facilitation of documentation, versioning, and deployment, as well as automation of the infrastructure using continuous integration (CI) and continuous delivery (CD) tools. Infrastructure as code is the last step in allowing developers to purchase fully documented, versioned infrastructure by just running appropriate scripts. The following sections discuss the advantages of IaC.

[1] Infrastructure as a service (IaaS), Statistics & Facts, Statista: https://www.statista.com/topics/2739/cloud-infrastructure-as-a-service/

Improved Time to Production

As a consequence of IaC automation, production/market time is substantially reduced, as is the time necessary to provide infrastructure based on the use case and scale up or down the infrastructure as necessary. IaC is able to automate the provisioning of outdated infrastructure that would otherwise be controlled by tedious procedures since it codifies and records everything.

Reduction in Drifting Configurations

When you make ad-hoc configuration changes and updates, you may experience configuration drift, which appears as mismatched development, test, and deployment settings. This may lead to difficulties in deployment, with infrastructure being more vulnerable and open to risks when building apps and products that must adhere to stringent compliance requirements, among other consequences. When IaC is used, the same environment is generated each time, avoiding drift. This ensures compliance of infrastructure configuration, many times related to security, standardization of environments, and also regulations that are required for various industries in the real world.

Faster and Efficient Development Life Cycle

IaC can securely expedite every step of the software delivery life cycle by simplifying provisioning and guaranteeing infrastructure consistency across all environments. CI/CD environment pipelines along with sandboxes may be rapidly created by developers. QA can provide full-fidelity test environments in a short period of time. Operations may provide infrastructure for security and usability testing in a short period of time. Furthermore, after the code has been verified, the infrastructure and application on which it performs can be deployed in minimal steps.

Maximizing the Scope of Provisioning

To optimize and improve efficiency, provisioning is often outsourced to a small number of highly qualified engineers or IT employees in companies without an IaC. When one of these domain experts leaves the business, oftentimes the team is left to rebuild the infrastructure as well as processes. Furthermore, IaC guarantees that provisioning intelligence is always kept inside the company.

Lowered Costs and Increase in ROI

Infrastructure as a service (IaaS) allows businesses to fully utilize cloud computing's pay-as-you-go pricing structure by substantially lowering the amount of time, effort, and specialized expertise needed to provide and expand infrastructure. Furthermore, it allows the teams to dedicate less effort and more time on creating creative, highly critical solutions, resulting in significant time savings.

Adapting Tools of Infrastructure as Code

There are a number of well-known IaC solutions available on the market that assist in automating infrastructure configuration and setup (such as Ansible, Terraform, and Helm). An IaC platform is in charge of administering and supporting a wide range of IaC frameworks, public cloud platforms, and cloud-native environments on a single infrastructure. Also supported are version control systems such as GitHub, GitLab, and BitBucket, to name a few.

Infrastructure automation is now a must-have for every team in a company. There is an urgent need to transition from manually maintained and configured infrastructures to IT or automated infrastructures for easier and more efficient system operation and administration. Many tools and

techniques are available for this automation. One tool will not be able to meet the needs of an organization or team. For example, some business use cases that can be solved by Terraform may not be solved by Ansible, and vice versa. As a result, tools should be chosen based on the needs of the business as well as a variety of other criteria such as cost, skill set, functionality, etc.

Factors Deciding Adaption of Infrastructure as Code

It is critical to determine whether to create a mutable or immutable infrastructure environment initially when implementing IaC and choosing an IaC solution. This decision must be made early in the process, or else it can hamper the processes in the next stages, and the team may need to put extra efforts into fixing or starting over entirely.

Mutable infrastructure is defined as infrastructure that is changeable or modified even after it has been constructed. With a changeable infrastructure, development teams may make server changes on the fly to better fit the needs of the product or service or to address a critical security vulnerability. However, it undermines a fundamental IaC benefit—the capacity to maintain consistency between deployments or inside versions and may significantly complicate infrastructure version monitoring.

As a consequence of these considerations, the vast majority of IaC infrastructure is built using immutable technology—that is, technology that cannot be changed once it is deployed. If immutable infrastructure needs to be modified, new infrastructure must be built in its place. Given the ease with which cloud-based infrastructure, especially IaC-based infrastructure, may be built, immutable infrastructure is more feasible and practical than it may seem at first sight.

Immutable infrastructure logically develops IaC, basically hardening it to guarantee that the advantages it provides continue to be realized.

Furthermore, it virtually eliminates configuration drift, making it considerably simpler to maintain consistency across test and production systems. Furthermore, the management and monitoring of multiple infrastructure versions are streamlined, as is the safe rollback to any previous version as required.

Approaches for Infrastructure as Code

The infrastructure specs are defined and written by the developers in a domain-specific language (DSL). The generated files are subsequently transmitted to a management API, master server, or code repository. The platform then performs all of the steps required to generate and configure the computing resources.

The imperative method and the declarative method are the two primary approaches to infrastructure as code. Both techniques base IaC setups on a template, in which the user defines the resources required for each server in the infrastructure. Let us learn more about these techniques and grasp the differences between them.

The *imperative* method specifies the specific actions or instructions required to accomplish the desired configuration. It also specifies the sequence in which these instructions must be executed. Essentially, it is concerned with the "how," i.e., how we arrived at our intended configuration. When choosing a viable solution, infrastructure automation should be approached from both a declarative and an imperative perspective, since both have their advantages and disadvantages.

This method defines the system's desired state. Other than the state, only extra information such as needed resources and attributes must be specified, and the IaC tool will configure everything for you. The declarative technique, otherwise known as the *functional* approach, is suited best to the vast majority of businesses. You specify the desired infrastructure state, and the IaC platform handles the remaining tasks—spinning up the virtual machine, installing and configuring the required

software, resolving system and application dependencies with each other, and maintaining versions—all while you focus on other tasks.

Using the *imperative* method, also known as the *procedural* approach, the solution guides you step-by-step through building automation scripts that supply your infrastructure one step at a time. As your business expands, this may take additional effort, but it will be simpler for current administrative personnel to comprehend and utilize preexisting scripts for setup.

With the imperative method, newly embarking infrastructure teams on the DevOps path may lack the necessary depth of expertise. Moreover, the IaC scripts prepared through imperative methods are often less idempotent, leading to results that differ based on factors such as environments and customizations. Hence, the imperative method's major drawback is that it often requires the employment of an experienced professional or technical administrator to implement and maintain it, who are experts in the respective areas and solutions for which they are responsible.

On the other hand, the major drawback of the declarative approach is the loss of control over the individual steps in various delivery and provisioning mechanisms. As a result, small adjustments that can be handled using a simple command-line interface script may not be a good candidate for declarative programming.

Best Practices of Infrastructure as Code

IaC may provide your team with a variety of advantages, including increased speed, security, and dependability. Implementing IaC may significantly increase the productivity of your teams, since they will inevitably need cloud infrastructure to fulfill the company's technological requirements. It's important to remember that selecting the appropriate tool from the start may help you keep the doors open for future third-party solutions and APIs that will benefit you. Figure 1-1 shows some of the best practices for making the most of IaC.

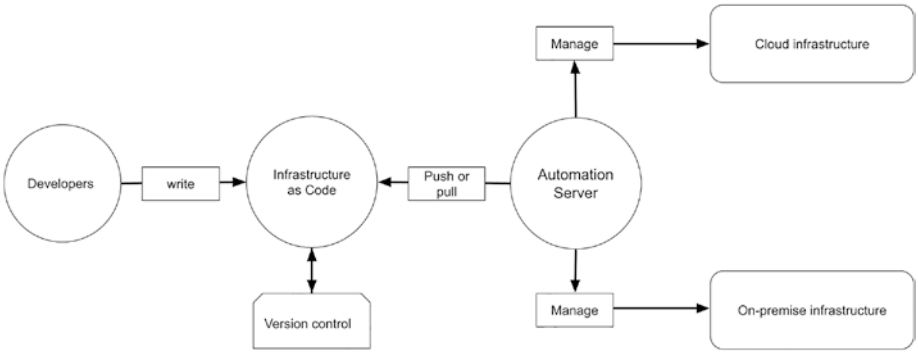

Figure 1-1. *Infrastructure-as-code best practices*

- *Continuous integration, testing, and deployment*: Continuous testing is critical for the long-term viability of DevOps, and the same is true for your IaC. Perhaps it is even more important to verify your infrastructure configuration on a regular basis to avoid problems once the deployment is complete. A continuous development cycle may involve unit and regression testing, as well as functional and regression testing, as well as automated experiments that evaluate infrastructure configuration for each code update.
- *Single source of truth*: To avoid misunderstandings, make sure that each infrastructure specification is stated clearly in the configuration files. You can guarantee that your configuration files are always accurate by making them the single source of truth for all of your infrastructure management problems.
- *Use immutable infrastructure when possible*: An immutable infrastructure is one in which components are replaced rather than changed while still in place. You can write code for the module once, use it

numerous times, and never change it. If you need to modify, just deconstruct and re-create the module. Because of the absence of configuration changes, immutable infrastructure improves security, prevents configuration drifts, and simplifies troubleshooting. Moreover, when it's not possible to do so, tools like Ansible help maintain desired states of infrastructure configuration.

- *Version control*: It's also a best practice to keep all of your configuration files under source control and is especially helpful in scenarios when scaling your infrastructure. Version control is essential for keeping track of changes and ensuring that every team member is working on the most recent version. It should be utilized for any programs and assets on which several team members are working together. It not only helps with file management and tracking but aids in the development and distribution of products. This is feasible because it improves visibility, improves communication among team members, and speeds up product delivery.
- *Everything should be coded*: The goal of IaC is to automate everything and eliminate as much human labor as possible, therefore attempting to set up the whole infrastructure by coding. All infrastructure specs should be explicitly coded in configuration files. There should ideally be no need for documentation, and these configuration files should include everything relevant to infrastructure administration.

- *Adapting to modular infrastructure code*: When you deconstruct your infrastructure into various stacks or modules and then combine them in an automated way, you have a brilliant concept. The modular architecture of this method has the significant benefit of reducing the number of configuration modifications that may be done without affecting the remainder of the system. Smaller-scale changes make it simpler to identify problems, enabling your team to operate with more agility.
- *Fewer variations in document specifications*: Essentially, this is about the logical consequences. Since your configuration files should be the only source of truth, any extra documentation is superfluous and should be avoided at all costs. External configurations may quickly get out of sync with the real settings, while configuration files, as previously mentioned, do not.
- *Regulations and compliance*: Defects in IaC files may pose a major liability if they are not detected and corrected in a timely manner prior to the deployment of IaC definitions. As a result, scanning all IaC files on a regular basis is considered a best practice to guarantee that validating code is run whenever an IaC definition is generated or modified. If everything went as planned, IaC scanning would be readily incorporated into the CI/CD process. You should test and scan IaC rules before deploying them in the same way that you test and scan application versions before deploying them.

The Way Ahead

IaC can provide your team with increased speed, security, and dependability, to name a few advantages. Because they will ultimately need cloud infrastructure to fulfill the company's technical requirements, adopting IaC may result in a significant increase in your team's productivity. It's important to remember that selecting the correct tools from the start may enable you to keep the doors open for future third-party solutions and APIs that will benefit you in the long term.

In the next chapter, you will learn about various principles and patterns that are fundamental to IaC.

CHAPTER 2

Patterns and Principles of Infrastructure as Code

It's fairly common for IT architects building large-scale systems to have several objectives in mind. Naturally, their success is dependent on meeting business requirements, producing money, and guaranteeing internal and external customer satisfaction. Building blocks, architectural patterns, and design patterns may assist less experienced architects. It is considered that a certain level of organizational maturity is needed to use the principles, patterns, and practices.

The use of infrastructure as code (IaC) to build and manage large platform infrastructure installations may soon become complicated. In this chapter, you will learn about the principles and patterns of IaC and how they build a fundamental platform for the concept to be utilized at scale.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_2

The Emergence of Infrastructure as Code

IaC is a method of automating infrastructure that is based on software development methods. It offers a framework that allows you to manage the technological stack for an application without manually processing and configuring individual hardware devices and operating systems. To overcome the issue of environment drift in the release pipeline, IaC is capable of installing technologies such as version control systems (VCSs) and experimenting with development approaches such as test-driven development (TDD), continuous integration (CI), and continuous delivery (CD). Before moving into patterns and practices, let's take a look at some fundamental concepts for successful IaC.

The Focus with Infrastructure as Code

The notion is that contemporary tools can address infrastructure in the same way that software and data are treated. To manage infrastructure, users may use software development tools such as version control systems, automated testing libraries, and deployment orchestration to manage their infrastructure. Also possible are development approaches such as test-driven development, continuous integration, and continuous delivery, which may be taken advantage of in this way.

IaC has been successfully implemented in the most demanding contexts for years. IT systems are not just business vital, but they are the business for organizations such as Amazon, Netflix, and Google, among others. Every day, such organizations' systems process hundreds of millions of data points. As a result, it should come as no surprise that enterprises like these are pioneering new approaches for large-scale, highly dependable information technology infrastructure.

The focus of such organizations is to use IaC to achieve the following goals:

- Instead of acting as an impediment or a limitation, information technology infrastructure could be seen as a facilitator and enabler of change.
- In contrast to normal, repetitive jobs, IaC prepares information technology professionals to devote their time to beneficial activities that challenge their talents.
- IaC provides little to no intervention of IT personnel to help customers identify, provide, and maintain their infrastructure on their own.
- Organizations want the ability to recover quickly and readily from failures, rather than assuming that failure can be totally avoided, allowing teams to perform better.
- Organizations want to achieve constant improvements rather than implement costly and dangerous mammoth initiatives.
- Organizations want to implement, test, and assess solutions to issues rather than debating them in meetings and documentation and want to demonstrate their effectiveness.

The Challenges with Infrastructure as Code

Consider the following examples of issues that teams often encounter while using dynamic infrastructure and automated configuration tools. These are the issues that infrastructure as code seeks to solve, and understanding them creates the framework for the ideas and concepts that will be discussed in further depth later.

Sprawling Servers

Cloud computing and virtualization may make provisioning additional servers from a pool of resources a simple process. If the number of servers grows faster than the team's capacity to manage them as effectively as they would want, the situation might become untenable.

When this occurs, teams find it difficult to keep servers patched and up-to-date, leaving systems open to known exploits and vulnerabilities. Occasionally, when issues are detected, remedies are not immediately applied to all of the systems that might be impacted by them.

The fact that different servers have different versions and settings means that applications and scripts that function on one computer may not work on another. This results in inconsistency across all of the servers, which is referred to as *configuration drift*.

Configuration Drift

Configuration drift occurs when servers are originally established and configured in a consistent manner but over time variations begin to manifest themselves. Drifts in configuration can be caused by a number of factors. Configuration drift can occur if administrators make changes to server settings that are not consistent with the IaC template that has been established. It is critical to completely incorporate IaC into systems administration, IT operations, and DevOps techniques, with rules and procedures that are well-documented and followed.

As soon as adherence to an IaC process is accomplished to build anything, any external intervention will cause the server environment to be altered. A machine developed using an IaC process should not be subjected to any intervention other than that required by an automated, aligned, and compliant preventative maintenance workflow. Manual or external updates, even the security patches, may result in configuration drifting, which has the potential to result in widespread noncompliance

or even service failure over time if not addressed. Snowflake servers and automation apprehension are the result of unmanaged diversity among servers.

Snowflake Server

A snowflake server differs from any other server on your network in a number of ways. It's unique in that it can't be re-created anywhere else. Most operations teams have comparable anecdotes about specific servers that couldn't be handled, much alone replicated. In other cases, this is not always due to an unexplained fragility; for example, there may be a vital software package that operates on a completely different operating system than the rest of the infrastructure.

Being different isn't always a negative thing. It becomes an issue when the team in charge of the server is unable to comprehend how and why it is different and therefore is unable to re-create it. Any server in an organization's infrastructure should be able to be rebuilt with confidence and speed by the operations team. Creating a new, repeatable process that can build a server to replace any server should be a top priority for the team.

Fragility of Infrastructure

A weak infrastructure is susceptible to disruption and is difficult to repair. This is the snowflake server issue, but it has been extended to include a whole portfolio of computer systems. Step-by-step, everything in the infrastructure must be converted to an extremely dependable and repeatable infrastructure to get the desired result.

Fear of Automation

Many IT professionals are apprehensive about allowing the automation technologies to operate on their own. IT employees often utilize automation in a selected manner, depending on their faith in the tools or the environment, for example, to assist in the construction of new servers or the implementation of a particular configuration update. This is generally because the IT employees do not completely believe in the automation tools or process, and they will often alter the settings each time they run the process to fit the specific job at hand.

IT workers are apprehensive about using automation appropriately, either because they lack trust in what they would do as a result of inconsistent server performance or because they lack thorough knowledge of automation. Servers are not continuously reliable because automation is not performed on a regular and/or consistent basis. Many teams are plagued by the dread of automation, which is understandable. Although automation saves time, building faith in the system may be a challenging undertaking, particularly when the code is responsible for filling a cloud with server instances and supporting infrastructure setup and infrastructure configuration.

Some IaC tools that keep track of state have the capability of automatically destroying resources if the code reflects that the resources have been destroyed. When using IaC in an automation pipeline, it is possible to get unexpected results, considered as *accidental destruction*.

Erosion of Infrastructure

The ideal world would be one in which you would never need to touch an automated infrastructure after it has been constructed, other than to support something new or to correct something that has gone wrong. Unfortunately, the principles of entropy dictate that infrastructure decays over time even in the absence of a new necessity.

Erosion is the concept that issues will gradually seep into an operating system over time, even if no problems are present. In an ideal world, once an automated infrastructure is in place, it should not need any human adjustments other than to support new features or to repair things that have gone wrong with it. Unfortunately, the principles of entropy dictate that infrastructure decays over time even in the absence of a new necessity.

> *The concept of erosion refers to the assumption that issues will gradually seep into an operating system over time.*
>
> —Heroku Team

Examples of erosion include the following:

- Security vulnerabilities and old-school operating systems, which are not upgraded for a long period of time
- The log files on the server that take a lot of space on the server's hard drive
- One or more of the application's processes has crashed or been stuck, necessitating the need for someone to log in and restart the processes
- Failure of the underlying hardware results in the failure of one or more whole servers, resulting in the loss of the application

Considerations for Quality Infrastructure

When integrating IaC across an organization, several issues should be taken into account to prevent service failure. The following are some considerations: take into account the organization's workflow, deploy code as often as possible, version everything, consult with the DevOps teams, and pilot small and scale success are all important considerations.

In-Depth Knowledge

A major investment in knowledge and abilities on the part of the IT staff is required for IaC. The organizational leaders should take into account the expenditure necessary to retrain existing staff or acquire new personnel. Hiring fresh talent would need to be a big factor, but it would need to be combined with other experience in DevOps and programming to get the greatest applicant recruitment results. Another part of training workers is to help them overcome their apprehension about automation. It will be necessary for the business to take into consideration the employee's fear of employing automation, as well as their faith in the automation technologies, while developing training needs.

Organizational Workflow

For IaC to be effective, it must be carried out inside a closed process system with the required level of automation. Before implementing IaC across the organization, think about how it will affect the workflow. Manual additions and/or modifications often cause the whole system to crash; therefore, it is reasonable to question whether the present condition of the IT ecosystem is suitable for such a limitation to be implemented.

Before deploying IaC in its entirety, it is important to carefully consider which parts of the IT ecosystem can be implemented and managed with IaC and which parts cannot. Piloting and small case studies are highly recommended to assist an organization in achieving a level of understanding and maturity on the subject matter. If your organization relies heavily on manual processes, identify areas where you can experiment with IaC. However, do not rely on IaC for mission-critical business applications until you have gained some experience with it.

Perpetual Steps

As a first step, teams should evaluate the present service catalog to identify areas where IaC might be used to increase efficiency, save costs, and alleviate manual administrative constraints associated with existing services. Determining how IaC will connect with current services in a regular way is another important consideration. For example, general cloud providers should make any new purchases of devices or platforms that have a high market value and can be quickly integrated into the general cloud network, such as Google Cloud Platform, Microsoft Azure, or Amazon Web Services.

Teams should refrain from implementing IaC for mission-critical production applications at the outset. To scale success, the development teams should pilot and develop IaC test clusters and then replicate their performance. Piloting is recommended for all new cloud-based systems, including those under development. To build up a test IaC process cluster, it is necessary to first reduce the scope of the goals and the application scenario to be tested. Tests of IaC in an IT environment are carried out by the DevOps teams in the first instance.

The Principles of Infrastructure as Code

Because it is impossible to operate servers in the cloud effectively without it, infrastructure as code has found its niche in the cloud environment. However, the concepts and practices of infrastructure as code may be applied to any infrastructure, regardless of whether it operates in the cloud, on virtualized systems, or even on physical hardware directly.

Idempotency

No matter how many times you run your IaC or what state you start with, the end outcome will always be the same no matter how many times you run it. In this way, infrastructure provisioning may be simplified, and the chance of inconsistent results is reduced. You may accomplish immutability by utilizing a stateful tool that uses a declarative programming language like Hashicorp Configuration Language used by Terraform. In this case, it is specified what kind of infrastructure is wanted at the conclusion of the process, and it is the responsibility of the tool to bring you to that end state. If it is unable to achieve the target condition, it will fail.

Business executives who are meticulous in their approach are understandably suspicious about automated technologies and their capacity to handle difficult jobs. As a result, no matter how many times IaC is conducted, it must maintain consistency. It is required, for example, that new servers be equal, or nearly equivalent, in terms of capacity, performance, and dependability to the existing servers when they are introduced. When new infrastructure parts are installed, all choices, from configuration to hosting name selection, are automated and decided in advance. It is inevitable that some amount of configuration drift may seep into the system over time; however, that drift must be documented and controlled.

Immutability

Immutable infrastructure refers to infrastructure that cannot be changed and must be replaced with new infrastructure. By creating fresh infrastructure every time, one can ensure that it is replicable and that it does not allow for configuration drift throughout the course of a project.

When it comes to infrastructure, configuration drift is a major issue. Over a period of time, modifications are made to infrastructure that are

not documented, and your multiple environments begin to drift apart from one another in ways that are difficult to reproduce. This is more likely to occur if you have a changeable infrastructure that has a lengthy life expectancy. Generally speaking, the system is more fragile for long-lived infrastructure owing to the possibility of difficulties such as gradual memory leaks, disk space exhaustion due to log piling, and so on, occurring over time. As a consequence, you won't be able to provide the infrastructure as regularly as you would your apps or configuration, and you will be less confident in your ability to do so. It is possible to alleviate these difficulties by using immutable infrastructure. When providing infrastructure in cloud settings, immutable infrastructure allows for greater scalability as well as greater reliability.

Easily Reproducible Systems

The implementation of an IaC strategy should make it simple and quick to create and rebuild any aspect of your IT infrastructure. They should not need a considerable amount of human work or sophisticated decision-making on your part.

Any part of an infrastructure should be able to be rebuilt quickly and reliably with little effort. Simply said, it indicates that there is no need to make any substantial judgments regarding how to reassemble the item. Software and version selection, hostname selection, and other decisions about how to provide a server should be reflected in the scripts and tools that are used to supply it.

The capacity to quickly and easily construct and demolish any portion of the infrastructure is quite useful. It takes away a lot of the uncertainty and worry that comes with making changes. Failures may be dealt with in a timely and confident manner. Provisioning new services and environments is a simple process that requires minimal effort.

Defining the activities that must be completed—from selecting the software to be installed to configuring it—is essential to the success of the

project. The scripts and tools that handle resource provisioning should be equipped with the necessary information to carry out their job without the need for human interaction.

Easily Disposable Systems

When it comes to system operations, IaC is completely reliant on dependable and resilient software, which makes hardware reliability irrelevant to the system. When it comes to cloud computing, where the underlying hardware may or may not be trustworthy, companies cannot afford to have hardware problems impair their operations. So software-level resource allocation guarantees that hardware breakdown scenarios are instantly reacted to with alternative hardware allocations, allowing IT processes to continue uninterrupted.

One of the advantages of dynamic infrastructure is that resources may be readily generated, destroyed, replaced, scaled, and relocated, which makes it more flexible. As a result, to take advantage of this, systems must be built with the assumption that the infrastructure will constantly change. When servers vanish, reappear, or are resized, software should be able to continue to function.

Infrastructure as a service is built on dynamic infrastructure that may be produced, destroyed, scaled, transferred, and rebuilt. It should be able to seamlessly manage infrastructure changes such as resizing and expansions. The ability to gracefully manage changes makes it simpler to make upgrades and corrections to operating infrastructure as the system matures. It also increases the resilience of services against failure. This is particularly crucial when sharing large-scale cloud infrastructure, since the dependability of the underlying hardware cannot be guaranteed.

Easily Repeatable Processes

The reproducibility principle states that each action you take on your infrastructure should be able to be repeated in the future. There are several advantages to utilizing scripts and configuration management tools rather than doing manual modifications, but it may be difficult to maintain a consistent approach, particularly for experienced system administrators, to doing things this way.

For example, if an administrator is faced with a seemingly one-time activity such as partitioning a hard disk, they may find it more convenient to just log in and do the work rather than writing and testing a script. To select how large to create each partition, what filesystem to use, and other such considerations, the administrator may take a look at the system disk, evaluate what the server on which they're working requires, and use their expertise and knowledge to make these decisions.

The difficulty is that someone else on my team may partition a drive on a separate workstation and make somewhat different judgments later. It seems that we are failing to follow the consistency principle, which will ultimately compromise our capacity to automate processes.

System administrators have a natural inclination for jobs that are simple and straightforward. As soon as a need for resource allocation arises, they like to approach it in the most natural manner possible: by assessing the resource needs, determining the best methods, and allocating resources.

Despite its effectiveness, this procedure is counterproductive to the automation process. IaC necessitates the use of scripts in the minds of system administrators. Their duties must be decomposed or grouped together into repeatable procedures that may be defined in scripts to be effective.

Infrastructure teams that are successful have a strong scripting culture. If a task can be written, you should do so. Drill down and see whether there is a method or tool that may assist, or whether the issue the job is addressing can be dealt with in a different manner, if a task is difficult to script.

Consistent Systems

By allowing inconsistent elements to creep into an infrastructure, you undermine your ability to put your confidence in automated processes. When two infrastructure parts provide a comparable service, for example, two application servers in a cluster, the servers should be substantially equal in terms of performance and functionality. Their system software and configuration should be identical, with the exception of those configuration elements that distinguish them from one another, such as their IP addresses.

If one file server has a 100GB partition, another has a 150GB partition, and a third has a 200GB partition, you can't depend on a single operation to operate the same way on all three. When servers don't exactly match, it is more likely that specific activities will be done for them, resulting in unstable automation.

Teams that adhere to the reproducibility principle may quickly and easily construct many identical infrastructure components. It is possible to maintain consistency by changing one or more of these pieces. For example, if one of the file servers requires a bigger disk partition, there are two options. One option is to update the specification such that all file servers are constructed with a partition that is big enough to accommodate the demand. Other options include creating a new class or role so that there is now a file server with a bigger disk than the ordinary file server and modifying the existing class or role. Either sort of server may be constructed in a repeatable and consistent manner.

The ability to establish and restore consistent infrastructure is beneficial in preventing configuration drift from occurring. However, it is evident that any modifications that occur after the servers have been set up must be addressed.

Ever-Evolving Designs

Making changes to an established system is complex and costly in the age of information technology. Consequently, it makes sense to restrict the number of times the system must be modified once it is developed. As a result, detailed initial designs that take into consideration a wide range of conceivable needs and conditions are required.

The fact that it is hard to correctly forecast how a system will be utilized in reality and how its needs will vary over time leads to the development of too complicated systems. Ironically, the complexity of the system makes it more difficult to update and enhance it, making it less likely to function well in the long term.

Making changes to an existing system may be simple and inexpensive thanks to the cloud-based dynamic architecture. This, however, is predicated on the assumption that everything is structured to allow change. To satisfy current needs, software and infrastructure must be created in the most straightforward manner feasible. Change management must be capable of delivering changes in a safe and timely manner.

The most crucial precaution to take to guarantee that a system can be modified in a safe and timely manner is to make modifications on a consistent basis. All parties involved are compelled to develop good habits for managing changes as well as to develop efficient, streamlined processes and to put in place tooling to assist them in accomplishing these goals.

The architecture of the information technology infrastructure is always altering to meet the changing demands of the company. Because infrastructure modifications are costly, businesses attempt to keep them to a minimum by methodically forecasting future needs and then designing the systems to meet those requirements precisely. Future modifications to these too complicated systems will be even more difficult and costly as a result of this overcomplication.

The IaC-driven cloud architecture addresses this issue by streamlining the process of change management. However, although the existing systems are intended to fulfill the needs of the present, future adjustments must be simple to incorporate. The only way to guarantee that change management is simple and fast is to make regular adjustments so that all stakeholders are aware of the usual concerns and can design scripts that efficiently overcome the relevant challenges. Change management should be simple and rapid.

Self-Reliant Documentation

Teams struggle to keep their documentation relevant, useful, and correct. Even if someone creates a thorough document for a new method, it is rare for such documentation to be keep up-to-date when modifications and adjustments are made. In addition, records have holes in them on a frequent basis. Folks tend to come up with their own shortcuts and modifications. Some people build their scripts to make certain components of the process go more smoothly.

Despite that documentation is often utilized as a way of preserving continuity, conventions, and even legal enforcement, it is a dramatized depiction of what really occurs in reality. It is the scripts, definition files, and resources that execute the strategy with the infrastructure as code, which include the stages for carrying out a process. Individuals are simply asked to provide the bare minimum of additional documents to get them started. To guarantee that when individuals make changes, the current documentation is handy and in their minds, it is recommended that the current documentation be maintained close to the concept that it records.

These principles, patterns, and practices need a specific degree of organizational maturity before they can be implemented. It is expected that these concepts will be used throughout the infrastructure development process, from provisioning to backup and DNS, as well as the process of building code that encapsulates and implements the services that you need.

The Patterns of Infrastructure as Code

As the popularity of public cloud platforms has grown, so has access to and consumption of the infrastructure they offer. It gets more difficult to accomplish the advantages of infrastructure as code as the infrastructure develops in size, complexity, and the number of people who utilize it.

Because of the wider range of things that might be impacted by a single modification, it is more difficult to make changes on a regular basis, swiftly, and securely. People spend much more time on fighting problems than they do on the more significant task of improving services. Also, allowing users to provide and manage their own resources may result in service interruptions for other users and services, if done incorrectly. When this happens, the standard response is to centralize authority over infrastructure. Because of this, more time is spent on meetings and paperwork, as well as the change management process, and less time is spent on activities that offer value to the business.

However, designing infrastructure in such a way that the extent of the effect of a particular change is minimized provides an alternative to centralizing control. When infrastructure is defined, provisioned, and managed effectively, it will be possible to make changes on a regular basis and with confidence, even as the infrastructure expands in size. The application's infrastructure orchestrates the processes required to satisfy the users' requests. Despite that these services shouldn't contain business logic, they may end up doing a substantial portion of the tasks required by an application. To complete a job, the application infrastructure layer will often interact with the service infrastructure. It is then possible to securely assign ownership of application and service infrastructure as a result. To achieve this, you need to understand the patterns that help develop, maintain, and scale your infrastructure in a proper manner.

Updates in Documentation

Documentation is time-consuming. Maintaining the appropriate amount of documentation to communicate the desired message is critical, just as it is with code. The presence of more documentation does not imply that it is better. Documentation that is out-of-date is much worse. Because everything is codified, you should not need significant documentation while using IaC; yet, some documentation is still required in rare cases. Better-quality documentation is beneficial not just to the team responsible for maintaining IaC, but also to the people that use the infrastructure.

Making sure that documentation is readily accessible when it is required is essential. For example, while showing an error message, it's a good idea to add a link to the documentation so that users can fix the problem themselves. Additionally, having a runbook for common situations might aid in debugging when there is a production problem.

The documentation should be kept as near to the code as possible, which ensures a better probability that you will keep it up-to-date. This way you will remember to update the documentation in conjunction with the code changes, and it may also serve as a reminder throughout the pull request process. Ideally, you should be able to produce documentation from your code or utilize tests to document your software.

Using GitOps

Another pattern of executing IaC is made possible by GitOps. GitOps is an extension of IaC that includes a mechanism for applying changes to the production environment, or any environment for that matter. The infrastructure might alternatively be controlled by a control loop that checks on a frequent basis to ensure that the actual state of infrastructure is the same as the planned state. For example, it will ensure that if any modifications are made directly to infrastructure, the infrastructure will be restored to the intended condition as defined by the source control system if necessary.

Securing Your Infrastructure

A critical, but frequently ignored, part of IT security and compliance is ensuring that your infrastructure and the apps that operate on it are safe and compliant. Traditionally, many organizations have manual checks and gates for this that are time-consuming and usually occur at a later stage in the deployment cycle; however, with IaC, you can automate them to provide better security as well as compliance and run them more frequently and earlier in the cycle, reducing overall costs.

Make certain that your IaC and the infrastructure it provides are protected by a comprehensive identity and access management system. The use of role-based access control (RBAC) for infrastructure provisioning in IaC helps to reduce the total attack surface of the infrastructure. With RBAC, you only provide your IaC with the permissions necessary for it to complete the action for which it was created.

When it comes to provisioning any infrastructure, IaC often requires secrets. When deploying resources in AWS, for example, you will want AWS credentials to access the service. Always utilize a trusted secrets management platform, such as Hashicorp Vault or AWS Secrets Manager, to keep your secrets safe. If you must output or keep any secrets in the state file (though you should strive to avoid doing so), make sure they are encrypted so that if someone has access to the state file, they are unable to extract the secret from it. Figure 2-1 illustrates this concept.

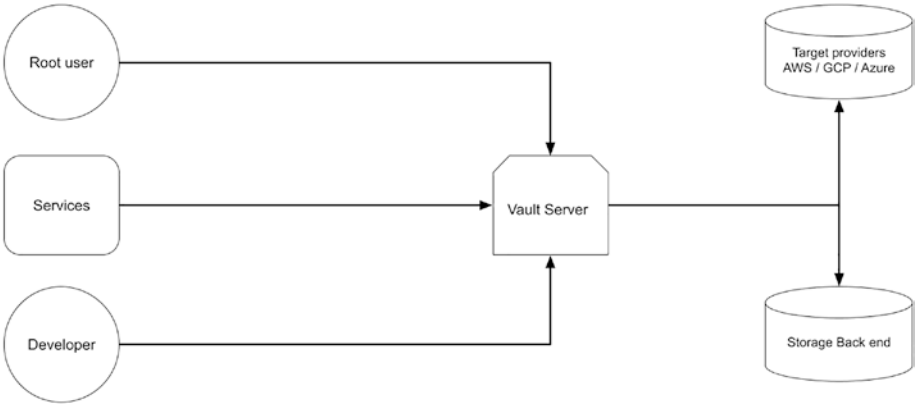

Figure 2-1. *Infrastructure secrets flow*

Many firms have compliance obligations, but if you work in the healthcare or financial industry, the regulations are much more stringent. I'm sure you're familiar with some, if not all, of the following: HIPAA, PCI, GDPR, and SOX are just a few of the regulations. As previously stated, historically, compliance teams were responsible for doing all inspections and documentation by hand. Using different tools, such as Chef Inspec or Hashicorp Sentinel, to automate these compliance requirements allows you to run them more regularly and detect the errors much more quickly, resulting in lower costs. For example, you may perform these compliance tests every time you alter your IaC by providing an ephemeral environment, which will allow you to determine whether there are any problems with the new code before it is sent to production.

Securing Secrets

To keep up with the evolving trends via IaC, however, there are several inclusions. IaC tool vulnerabilities might provide a point of entry into the core infrastructure, for example. It is possible for hackers to circumvent security measures by exploiting flaws in the system. Misconfigured IaC templates may potentially reveal critical information or open the door to assaults.

With infrastructure as code, a single program may manage many environments indicated in configuration files and execute code within target environments. The common element is the configuration itself, its storage, and notably the secrets, which are required for connecting to controlled infrastructure. The secrets storage holds sensitive data such as application tokens, Secure Shell (SSH) keys, and passwords. Storing such data in source code management systems, like Git, or plain-text files is risky and irresponsible from a security standpoint. It is highly advisable to keep secrets in vaults and reference them in configuration files.

Adapting Security Standards

There are several issues with using IaC to keep CI/CD cycles going swiftly. An unpatched hole in an IaC tool, for example, might be exploited by a malicious individual to get access to a network's critical infrastructure. Attackers may be able to exploit weaknesses to circumvent restrictions, install malware on susceptible computers, or even start up cryptocurrency mining on compromised devices or infrastructure. Incorrectly built IaC templates have the capability of exposing sensitive data or establishing attack pathways. To fulfill consumer expectations, regulatory and compliance standards, and technical security needs, the code must undergo significant modifications. This implies that security will be able to keep up with growth since it will be able to employ some of the same technologies as expansion.

Restricting User Privileges

Another component of security to consider is user privileges. If an IaC application is used to manage application deployment, it is unlikely that root credentials on the destination machine would be needed. While adhering to the notion of least privilege may be challenging, it should minimize the potential of compromise. The same concept applies to deploying apps in public clouds such as Amazon AWS. It is preferable to

use a dedicated account or role for a dedicated task such as producing prepared virtual machines with a limited set of abilities. It is risky to exchange cloud provider credentials with administrator access in order to do less privileged tasks.

Relying on Trusted Sources

The delivery and maintenance of cloud infrastructure is made possible via IaC templates, which are machine-readable specification files that construct environments for the deployment and execution of code from external sources. However, they are not without risk. An untrusted operating system or container image might be used as part of an IaC process using these templates. These channels might be used to insert back doors and cryptocurrency miners.

Data may be exposed due to vulnerabilities and unsafe defaults in IaC templates. IaC-deployed cloud infrastructures and stored data may be at risk because of the weaknesses, especially if they are accessible to the public Internet. Handlers should check IaC templates early in the development process for unsafe setups and other potential problems. In addition to detecting and correcting misconfigurations, a cloud security posture monitoring service may also aid with regular scanning.

Security Measures

Infrastructure as code delivers and maintains infrastructures such as databases, network servers, services, and virtual machines. It is an important DevOps approach that supports Agile software development. IaC aids in the acceleration of growth and the effective management of infrastructures. Developers and IT teams manage the settings of individual deployment environments in the absence of IaC. This manual procedure may result in deployment discrepancies and security risks. The following procedures should be implemented to safeguard hybrid and public cloud infrastructures.

Least Privileged Position

In cloud services, account privileges should be regulated, especially when the services are linked to public cloud providers, taking advantage of the idea of the least privileged position (LPR). Permissions and tool access should be restricted to prevent attackers from gaining a foothold in nodes and gaining access to sensitive information. Consequently, IaC configurations are securely stored, and data leakage is prevented.

Using Security Tools

Take advantage of the security plug-ins provided by infrastructure as code. A security plugin in the integrated development environment (IDE) may aid in mitigating potential vulnerabilities in IaC templates prior to deployment.

Infrastructure Updates

Upgrading infrastructure software to the most current version is a common practice. It is important to apply security patches as soon as they become available as visible in Figure 2-2 below. It is best not to expose a fundamental system. If possible, a central server should not be accessible from the Internet in order to prevent compromise from spreading to other components further downstream.

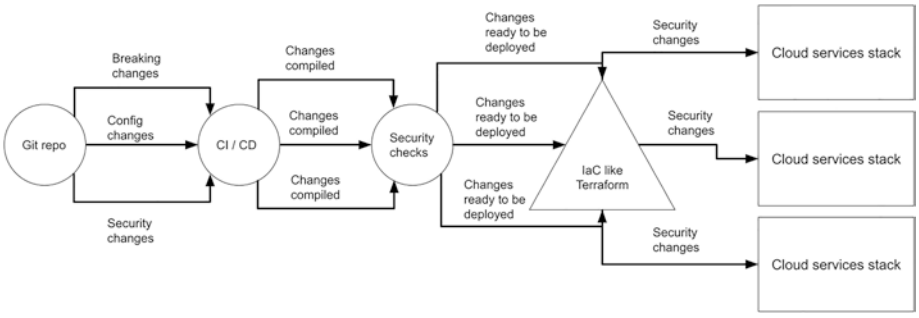

Figure 2-2. *Updates performed with IaC*

Threat Detection

Improve your organization's security and compliance posture. Cloud service providers should use real-time security that can detect and correct misconfiguration to provide pipeline protection. Solutions that have an auto-remediation function may also be useful in the rectification of faults.

Testing the Infrastructure

Testing your IaC at different stages is similar to software development in that it must be planned ahead of time. Essentially, the concept is that as you go up the test pyramid, the tests get more expensive, more fragile, and more time-consuming to perform, and need more upkeep. As a result of these considerations and to get quicker feedback, you should run tests at the bottom of the pyramid as often as feasible and run tests at the top of the pyramid less frequently.

It is recommended to perform static analysis as often as possible, even on your own system, to get the most immediate feedback. There are connectors that allow you to do this automatically when you save a file in your text editor or integrated development environment.

As a result of the declarative nature of most of the tools such as Terraform, unit testing is often not required for IaC. Unit tests, on the other hand, may be beneficial in some situations, such as when you have conditionals or loops. If you are writing bash scripts, you can use bats to do unit testing; if you are using Pulumi, which supports languages such as TypeScript, Python, Go, or C#, you can use the language test framework to perform unit testing in those languages.

The process of provisioning resources in an environment and determining whether they meet particular criteria is referred to as *integration testing*. You should refrain from creating tests for things that your tool is in charge of, particularly if you are writing declarative code. Instead of focusing on whether the rules defined in IaC were followed, you

should be building automated tests to ensure that none of your private partitions of the server are accessible to the public. Similarly, you may want to check that only certain ports are open across all of your server instances.

The last, but not the least, method of testing is to create an environment, deploy a fake application, and then run brief smoke tests to ensure that the application was successfully installed. Test those situations that your actual application might encounter, but that are not yet set up for use in production, using a fake application. In the case of an externally hosted database, for example, you should test connecting to it in your fake application to ensure that the connection is successful. Having this assurance offers you peace of mind that the infrastructure you are deploying will be capable of supporting the applications you want to run on it. Because these are slow tests, you should run them when a fresh environment has been provisioned and thereafter on a regular basis.

Concerns with Infrastructure as Code

With the growing popularity of the public cloud platforms, IaC also has access to consumption and the infrastructure these public clouds offer. IaC is used by DevOps practitioners to create and maintain an automated deployment pipeline that enables IT organizations to rapidly release software changes at scale. Low-quality IaC implementations can have substantial ramifications, such as significant system failures and service disparities. Practitioners can increase the quality of IaC scripts by recognizing anti-patterns and developing the infrastructure. New technologies can be controlled more easily with the help of infrastructure as code.

Technophiles admit that innovations such as software-defined data centers, virtualized infrastructure, cloud computing, and other software-defined technologies have simplified infrastructure provisioning. All of

this has made it more difficult to manage. First, IaC is not an ephemeral phenomenon. On the other hand, it is now possible to build better, easier, and faster cloud-native applications. IaC, which represents whole application infrastructures, has enabled developers to achieve previously unheard of speeds, allowing them to bring products to market faster than ever before due to scalable, automated deployments. Even with a holistic approach and perspective in mind, the meaning of IaC can differ at every level in the organization. The following are the anti-patterns and pitfalls that can turn out to be decisive for any IaC implementation.

The size, complexity, and changeability of infrastructure settings are all key variables to consider when it comes to the effectiveness of IaC deployments. Constructing and configuring IaC environments is a time-consuming and costly endeavor. When the infrastructure is large enough, it will begin to pay off sooner rather than later. The money and necessary work put in if the organization wants to gain the advantages are considerably high.

If the primary motivation for participating in an IaC project is to save money or improve operations, the organization most likely will be disappointed with the results. In the long run, the advantages of adopting IaC exceed the short-term profits. When determining the worth of an IaC project, it is important to include both the total cost of ownership and the operational savings.

There are currently no industry standards or numbers on quantifiable benefits available to help management understand the value of IaC while creating a business case for it. When it comes to putting the IaC concept into action, there are no best practices that have been created. Suppose you want to know what the best practices are for performing testing. Software development tools and processes are easily accessible, but this is not the case when it comes to infrastructure development tools and methods. There are only a selected few tools accessible in the industry. Because an organization may lack the requisite in-house skills, knowledge, and expertise to choose the proper tool stack for their particular

environment, and integrate the tools as part of the IaC implementation, this might result in the failure of an IaC endeavor. Regardless of the concerns, IaC should be given significant attention. IaC is way ahead in terms of provisioning and managing infrastructure, as long as the infrastructure is built while considering these problems.

Infrastructure as Code at Scale

In addition to the ever-increasing demands on IT infrastructures, IaC comes into play, providing structure and managing capabilities. Administrators and architects may use IaC to automate the process of setting up both on-premises and cloud infrastructures. Corporations have used IaC as a way of quickly creating and implementing scalable cloud systems as an addition to infrastructure as a service (IaaS).

IaC uses descriptive language to allow more flexible provisioning and deployments. DevOps relies heavily on IaC, which also helps with cloud security automation. The IaC toolset is divided into two categories. To deliver, organize, and manage infrastructure components, orchestration software such as Terraform is utilized. On the other hand, configuration management tools are used to manage software that is installed, updated, and maintained on infrastructure components. There are a number of new technologies that make it easier for cloud developers to build up complex systems and maintain them in a more efficient manner.

Evolving Business Requirements

During the initial stages of adoption and implementation, an infrastructure may just demonstrate the use cases and may not interact with any sensitive organizational data. When the infrastructure becomes a customer pilot and handles sensitive data, the priorities must be altered. At this point, additional security needs may be developed, and organizations need

to adhere to a variety of regulatory or internal best practices. Customer and business needs and capabilities will continue to evolve, as will the applications that support them. The infrastructure of organizations may take a variety of forms, specific to products, services, research purposes, and internal use cases. Developers may easily develop an analytics service in response to a client's requests. However, such an update would need a major rewrite of the application's architecture and capabilities. Changes in strategy, the demand for extra capabilities, or customer input may all entail modifications to the service or product, necessitating an upgrade to the application architecture resulting in changes of previous security assessment.

Infrastructure as code allows you to compare architectural modifications to your security reference architectures and design patterns automatically, enabling you to identify security and compliance concerns more quickly. Any discrepancies are then re-introduced into the process.

Evolving Security Requirements

With the rise in cloud-based security issues, new guidelines are being updated on a regular basis, requiring the need for flexibility. However, it is more than a matter of adapting infrastructure best practices. Security issues, new compliance and regulatory requirements, and changing consumer expectations all have a direct effect on how infrastructure is designed and implemented. Depending on the customer and the nature of their business, they may need more stringent security measures than those included in the product initially. Even if a security update is intended to address a particular vulnerability, it has the potential to introduce new security threats as application designs evolve. Along with automatically informing security teams of all changes, IaC enables them to monitor the repercussions of each update throughout the whole application architecture.

Evolving Provider Requirements

Cloud service providers such as Amazon AWS and Microsoft Azure are constantly increasing their capabilities. Developers and security experts are struggling to keep up with the constant stream of new features being released. Nonetheless, their importance continues. Even if a new feature or capability poses security problems at the time of delivery, developers may be willing to take the risk since AWS and Azure will handle it in the future. When new, more secure versions of cloud services become available, IaC's automation enables quick upgrades. To keep up with IaC, security mechanisms, similar to IaC or software development itself, need a more dynamic strategy. As a result, security is never slowed down or forced to be evaded. Developers and experts can move quickly and efficiently as a group.

IaC's promise lies in its capacity to unify all cloud setup operations, increasing efficiency and lowering costs. Infrastructure automation is increasingly a critical necessity for all enterprises. As more contemporary apps migrate to the cloud, IaC will become increasingly more critical.

Always keep security in mind while picking an IaC tool. Consider solutions that assist firms in securing IaC templates and conducting security audits to identify security flaws, compliance violations, and other misconfigurations. The best systems enable developers to discover, monitor, and correct misconfigurations as part of their usual process, without having to leave their code repositories to check findings or create a separate scanning procedure. A system that prioritizes security provides security administrators with the trust they need in DevOps-built and operated IaC.

The Way Ahead

The capacity of a team working on infrastructure to adapt effectively to changing needs is one way that efficiency may be measured. The highest-performing groups are able to quickly observe trends and adjust to new needs, while also promptly linking requests to a consistent supply of low-impact alternatives. The restoration of the whole system may be done quickly and easily. You take measures to guarantee that all of your systems are always up-to-date, have all of their vulnerabilities fixed, and are completely interoperable with one another. You don't need assistance from members of the IT department to set up servers and environments; you can do it in a couple of minutes.

There is often no need for predetermined maintenance windows to be arranged. Throughout the course of normal business hours, potential dangers such as software upgrades and other installations are carried out. The accomplishment of a group's goals is not dependent on the absence of mistakes in their work; instead, the average amount of time that passes without a problem can be tracked. Members of the group think that their contributions to the firm are beneficial in a measurable way.

In the next chapter, you will learn about how IaC is managed and how businesses need to equip themselves with strategies they can use to harness the true power of infrastructure as code.

CHAPTER 3

Management of Infrastructure as Code

When it comes to information technology for businesses, it is not unusual to see a mix of on-site data centers and infrastructures hosted on the public cloud. It is essential to the IT environment's capacity to function correctly that the settings and interface needs of these assets be appropriately identified and managed. Even a little adjustment to the configuration parameters might have a significant influence on the functionality, safety, and effectiveness of the code. It's likely that the infrastructure required to run tests is quite different from what's required in a production environment. This is something to keep in mind. Hence, it becomes essential for the businesses to manage their infrastructures in an appropriate manner.

Oftentimes infrastructure as code (IaC) is considered to be derived from infrastructure as a service (IaaS) and development teams attempt to apply the same management principles of IaaS to IaC. This results in an unhealthy infrastructure, since both of them are completely different concepts, and each of them requires different approaches and strategies to be able to manage them.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_3

IaaS is a cloud service that provides virtualized computer resources such as servers, networking infrastructure, storage, and so on. IaC, on the other hand, is a tool for provisioning and managing infrastructure. It is not confined to cloud-based resources exclusively.

The Emergence of Infrastructure Teams

The number of complex workloads and business applications continues to expand, and IT departments are turning to multicloud systems to manage and support the enterprise's fast expansion. It is anticipated that reliance on these solutions will increase, although the complexity and lack of compatibility of legacy infrastructure with public cloud environments continue to pose obstacles to cloud adoption. In a setting that is cloud-native, operators of conventional infrastructure must also be creators of software for that infrastructure.

In contrast to the operational responsibilities that were performed in the past, this new manner of doing things is one that is still developing. As soon as humanly possible, businesses need to get started analyzing the existing patterns and developing the recommendations, since the necessities are bespoke and may impact a large part of the infrastructure operations, as well as management.

Preparing Infrastructure as Code

Merging several clouds may be an effective technique for increasing productivity, managing complicated applications and workloads, and providing support for hybrid workforces, despite that there is no deployment architecture that is appropriate for all circumstances. If your team is in the process of building a multicloud architecture, the guidelines shown in Figure 3-1 can help you overcome common challenges effectively such as interoperability, security, and data integration.

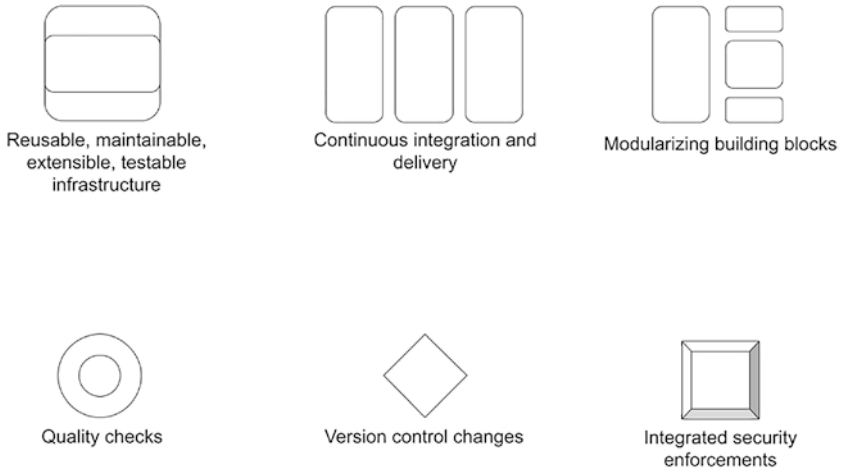

Figure 3-1. *IaC elements*

Evaluation of Infrastructure

The infrastructure that is hosted in the cloud has the potential to increase productivity, but it is not suitable for all types of workloads. On-premises and private infrastructures are often necessary for legacy programs because of the specific requirements that go along with them. If your business already manages older workloads, you should consider a few things including on-premises solutions, as the latency and delivery time originated from your deployed infrastructure to your end customers depends on multiple factors including server workloads, coverage of services provided by the public cloud in different geographic locations, Internet speeds, etc. For example, some countries do not yet have blazing-fast Internet speed; similarly, some specific geographical regions are not yet served by public cloud providers.

These solutions offer the benefits of a cloud operating model while allowing you to keep complete control over the underlying infrastructure. If your business already manages older workloads, you should look into

on-premises solutions. When building IaC for your company's specific cloud workloads, it is important to consider factors such as security, compliance, performance, and cost. The public cloud infrastructure is best suited for customer resource management, remote workstation management, and collaborative application architecture, whereas the private cloud architecture is best suited for database, human resources, and disaster recovery architecture.

Choosing the Right Security Mechanisms

When it comes to matching workloads to the right infrastructure, security is the most important consideration. The use of VPNs and firewalls to protect against current cybersecurity threats is becoming less successful for many enterprises. Identify and assess cloud-based security solutions that place a high priority on identity management, reduce attack surfaces, and are specifically designed to safeguard cloud data, apps, and service.

Structuring the Data

The fragmented nature of separate data centers and data lakes is another problem that has been overlooked in legacy data access. A lack of visibility and portability is frequently overlooked by IT executives in their efforts to secure and manage data regardless of where it's kept. Consistent governance and protection rules should be implemented to guarantee that all data repositories are accessible to authorized users, regardless of location. Data portability and enterprise-grade security are both ensured when data is unified in multicloud settings, allowing remote teams to remain productive and collaborative no matter where they are located.

Automating Workloads

Simplifying administration of IaC, boosting backup and resilience, and improving data cost and governance are all accomplished via the collaborative efforts of scaling multicloud setups, automation, and orchestration. You should make use of rules and parameters that have been established exclusively for use in such systems if you want to get the most out of automation. Suppose your organization wants to launch and orchestrate a new cloud stack that consists of a series of virtual machines, networking gateways and CDNs, storage, monitoring, and billing. Here's one approach to achieve this: you can combine these services into groups and then make use of IaC to launch them. Now every layer of the infrastructure can be orchestrated independent of the other using IaC. Moreover, once the infrastructure is deployed, you can make use of the tools, modules, plugins, and more to restrict role-based operations, deployments, monitoring, and automated checks and orchestration of new layers based on previously performed steps in the deployment process depending upon the IaC solution that you're using to maintain your infrastructure.

If manually managing workloads is slowing progress on high-value projects, the solution may lie in automating and coordinating infrastructure and IT tasks utilizing hybrid operations and solutions. This might be the case if hybrid operations solutions are being used.

Uniform Governance

The availability of data at any time and in any location is essential to the operation of a cloud-based architecture. However, the reality of settings that make use of several clouds is that each cloud has its own set of rules, governance, and tools. This leaves data rights and access vulnerable to being violated.

Before beginning to install cloud infrastructure, you should make certain that your team is aware of the significance of standardized rules and tools for the purpose of protecting multicloud systems, as well as the methods by which to acquire best practices. As the company grows, the use of consistent rules may provide new layers of insight into workloads and improve general collaboration. Take into consideration the establishment of centers of excellence to promote the dissemination of cloud-based best practices.

Hybrid Strategies

The use of IT infrastructure by enterprises is becoming an increasingly important factor. Infrastructure that makes use of many clouds has grown to encompass both private and public clouds. This hybrid method may assist companies in achieving the flexibility and agility necessary to compete in today's digital world, while also assisting organizations in the process of streamlining their operations.

These modifications are being driven by the market's understanding that there is no cloud solution that is universally applicable to all situations. The current state of affairs makes it necessary to upgrade the IT infrastructure to give greater application stack management capabilities. If you keep these tips in mind, you will be able to get the most out of your cloud investment and overcome the challenges that are most often associated with multicloud architecture.

Blue-Green Deployment Strategy

When it comes to deploying as well as managing infrastructure in a production environment in a safe and secure manner, there are a number of options from which you can choose. These patterns reduce the negative effects of a poorly executed deployment, make it possible to get inputs, and

ensure that there is no downtime throughout an implementation. The vast majority of them adhere to certain patterns designed to limit the number of newly published updates.

Continuous integration/continuous deployment (CI/CD) is a methodology that can be used to automate various stages of software development. This methodology can be used with the phase of integration and testing and continue on through the phases of delivery and deployment. Your choice of a specific deployment strategy for a CI/CD pipeline may have a direct bearing on the success or failure of your project. Every potential solution requires a balance between a number of different considerations, such as cost, timeline, risk, and user impact.

The blue-green deployment method is used so that a risk-managed release can be sent out to production servers in a controlled manner. In this section, you will learn the merits and drawbacks of this approach, as well as examine how it compares to the many other possibilities. Figure 3-2 shows the strategy.

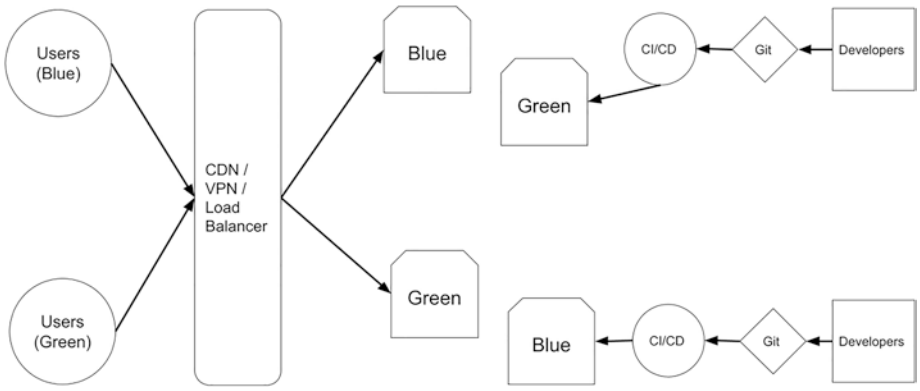

Figure 3-2. *Blue-green deployment strategy*

Process and Architecture

Blue-green deployment can ensure the quality of the infrastructure by sending it to a test environment that is analogous to the production environment but is inaccessible to the general public or the rest of the organizational teams.

The process of maintaining two almost identical versions of your infrastructure is referred to as *blue-green* maintenance. The first setting is known as the *blue environment*, while the second setting is known as the *green environment*. When you are ready to release a new update of the infrastructure, configuration, or updates, you deploy it to a new blue environment and transfer the operational capabilities on the blue environment. At the same time, you keep the previous green environment around as a backup in case you ever need to roll back to the previous state.

Your blue-green load balancers or deployers will redirect the operational traffic to the blue site, which contains the new functional update or service that has been completely coded and tested, if the deployment has been tested and if it has been validated that the deployment is operating as intended. If all goes according to plan, the green site will be kept running in case it is necessary to do a rollback. At this point, you will be ready to repeat the process and launch the next update, at which time you will develop a fresh replica of the green site using blue.

If, on the other hand, the product does not function as planned and requires some modifications to the code or debugging before it can be released, all you have to do is keep your live site in the green environment. As a result, you will have more time to enhance the product before reintroducing it to the market.

Working Mechanism

Consider an infrastructure that offers various features in a large-scale environment. A number of tasks are performed by many container-based microservices on the back end.

The initial release offers basic functionalities, which are advantageous to thousands of customers. The technology is capable of recording thousands of new transactions every single minute. Your DevOps team has strongly suggested that you release new versions of your infrastructure more often; thus, you will implement a minor modification to the mechanics that significantly improves the size and speed of processing the data.

You employ a blue-green deployment method to update the application in the middle of peak demand, as opposed to waiting until midnight to deliver the update to production, when there are fewer prospective customers online. By cloning the mechanics from the production environment to a different environment, you will finish the work faster than any previous attempt.

After the new business logic has been deployed, it must pass the quality checks, test cases, and staging processes before it can be moved to the production environment alongside the active blue environment.

Preparing Deployments

Before you can begin a new project or transfer an existing one, you need access to the infrastructure from which you can perform a blue-green deployment. You'll need to keep track of which environments are in production and which are in staging once you've set up your infrastructure. If you have many environments, you need to know which version of your code is operating in which environment. Preparation is essential when making large-scale code modifications. Without a plan, it is hard to test and monitor changes.

Infrastructure components that can be reused are kept in module repositories, which are similar to the common libraries used in application development. This might be an actual infrastructure component or module, or it could be a series of infrastructure launch templates that define a specific piece of architecture. These modules include configurable settings that allow for a wide variety of applications. They are designed to be automatically instantiated with the help of a configuration repository rather than having environment-specific prerequisites. Infrastructure components like ECS service definitions for a given application workload, shared RDS database definitions, and VPCs may all be stored in a single module repository. Module repositories should update like application libraries. A Git approach that allows releases and versions is appropriate. GitFlow, semantic versioning, and `CHANGELOG.md` are often-used technologies. Professional programmers know this works well.

Configuration repositories record deployed resource configurations. These repositories are generally organized in the same way as deployed workloads. Configuration repositories should follow a "truck-based" approach. The configuration repository prohibits configuration changes and records system state. This implies configuration repositories should save only intended changes. Automatic updates will follow. It's hard to tell whether a configuration repository's feature branches or "develop" branches represent the genuine infrastructure state. Git history removes the necessity for publishing and labeling configuration repositories.

Maintaining order may be simple if you just use one code base that isn't very complicated. This isn't always the case in the professional development industry or organizations. Many manual and automated actions may be necessary when configuring a new version of your IaC.

Adapting Simplicity

A long history of ever-increasing complexity, rather than steadily improved simplicity, has long been associated with IT enterprises. Human error is increased, time is wasted, and users are unable to concentrate on their other objectives when a system is too complicated. Instead of keeping employees busy just for the sake of it, technology should help businesses run more efficiently, continue to develop features, and expand into new markets.

It is possible to minimize needless complication when expanding capacity to an existing system by using current seamless techniques instead of a hardware update. If you have less to manage, you'll have more time to focus on the things that are really essential to you. Simplicity reduces or eliminates the risk of human mistakes in manual management processes, which may be a concern.

To get a deployment right, here's what you need to ensure:

- It is essential that you carry out all of the required checks to ensure that the update is functioning correctly.
- When operating the code in its most recent version, a blue server is the one that should be utilized.
- It is necessary to configure the router or load balancer so that traffic from valid customers will begin to be sent to the blue server.
- It is necessary to configure either the router or the load balancer in such a way that no valid user traffic is allowed to access the green servers.
- Continue to send traffic to the blue server until you are convinced that there are no problems and that there is no need to roll back any changes.

- Keep the traffic flowing to the blue server and then pass it to the green server. It is necessary to set either the router or the load balancer in such a way that users are sent to the green server while the issue is being resolved.
- The green server has to be prepared for the subsequent blue-green deployment as soon as the final checks have been carried out.

Environment Replicability

Environment replicability refers to the capability of moving code binaries and configuration files from an active and well-tested server to a new server to set up a new environment with only minimum configuration changes required. This is accomplished by using the new server.

Along with maintaining stability, this is one of the most important aspects of blue-green deployments. The procedure consists of a number of different components, such as servers and settings. Depending on the needs of your organization, environments may range from a single instance to several instances. Some examples of environments are production, staging, mirror, and development.

The more habitats there are, the higher the level of complexity that must be dealt with. To deal with this situation, you need to devise and implement automated inspections, in addition to monitoring and sending out alerts.

Configuration Management

It's possible that increasing an organization's agility may be accomplished by automating processes that would normally take a lot of time thanks to configuration management in DevOps. On the other hand, configuration management is an essential part of DevOps, and it is widely acknowledged that this component is more than just another piece of the jigsaw puzzle.

Configuration management is the practice of using a configuration platform to automate, monitor, create, and manage otherwise manual configuration activities. Every controllable component of the system is re-engineered throughout the process. Configuration management relies heavily on defining the current state of each system. Using a platform, organizations can ensure that all linked systems operate in the same way and increase productivity. Thus, companies may expand more quickly without having to hire more IT management staff. A DevOps approach may help companies that might otherwise be unable to grow.

The terms *configuration management* and *change management* are often used interchangeably because of their close connection. There are two types of configuration management: one is *change management*, which involves redefining and updating settings to meet changing needs, and the other is *configuration management*.

Process and Architecture

The process of managing configurations is a key aspect of the management framework, and data architecture is a vital part of that process. A database that is known as the configuration management database is one of the features of configuration management that is considered to be among the most significant (CMDB). This database stores information about an organization's whole configuration management infrastructure, including all of the systems and applications that comprise it.

CMDB is helpful because it enables development teams to investigate the links between connected systems before making any modifications to the configuration of the system. In addition, it is an excellent provisioning tool for gaining knowledge about infrastructure elements like servers. Because it allows teams to escalate issues until they are resolved, incident management may also benefit from the use of a CMDB.

The Way Ahead

Configuration management is not something that should be considered apart from DevOps. Successful operation of an organization requires comprehensive configuration management, including DevOps strategies, as they go hand in hand. This is because configuration management builds the framework for far more automation than it directly touches. Businesses may be able to strengthen their communication and operate as more Agile development unit by using IaC in the environment in which the infrastructure is being developed. This unit will be focused on continuous integration and continuous delivery. Without the appropriate tools, configuration management is a challenging task.

In the next chapter, you will learn about the production complexity with IaC at scale and how to manage the complexity through various techniques.

CHAPTER 4

Production Complexity Management

Before the advent of the cloud, it was infamously difficult for IT teams to manage downtime during installations. Because local consumers have limited access to programs located in conventional data centers, businesses often plan application rollouts in the middle of the night. With the increasing number of businesses transferring their operations to the cloud, the traditionally required deployment durations are becoming obsolete. Every company's CEO dreams that its product will one day be accessible to each and every prospective user.

Users have traditionally suffered interruptions in service whenever programmers have had to shut down applications to install updates and fixes. Automating the application development, automated testing, and automated deployment process is helping to maintain environments. This is made possible by the use of continuous integration/continuous deployment (CI/CD) pipelines. When deploying an application or making changes to the environment, however, there is a possibility that there may be downtime as well as other problems. To choose the most suitable deployment method for your use case, it is essential to have a solid understanding of the benefits, drawbacks, and requirements associated with different deployment strategies.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_4

Modern Application Infrastructures

The cloud is becoming an increasingly common location for the storage and distribution of contemporary software. Cloud services are more tolerant of failure as a consequence of their easily accessible design, and they can evolve more swiftly to accommodate demand as a result of their accessibility. Fully managed services, such as Amazon Web Services, Google Cloud, Microsoft Azure, etc., relieve developers of some of the pressures associated with running their businesses by handling some operational tasks.

For instance, a mobile application for a smartphone or a website geared at end users may undergo several revisions during the span of only one month. Some of them go through the manufacturing process many times every single day.

They often use designs that are based on the concept of microservices, in which a number of distinct components collaborate on the completion of a single mission. It's possible that various components of a product will be launched at different periods, but they all have to be compatible with one another.

When there are more moving components, there is an increased possibility that something may go wrong. When many development teams are engaged in a project, there is a greater chance that the underlying issue may be difficult to identify and fix. A further problem is the abstraction of the infrastructure layer, which is now considered to be code. When a new application is deployed, there is a possibility that more code for the infrastructure will be necessary.

Managing Deployments Without Downtime

The majority of companies, especially those with a modern application architecture and a well-established CI/CD pipeline, want to be able to deploy new applications and features at any time with no adverse effects

on the customers who are now using the software. Because of this, it is necessary to implement changes in production settings in a quick and secure manner so as not to interrupt users who are doing essential tasks or who rely on your systems for mission-critical processes.

Your application's architecture, and the way it is deployed, plays a significant part in determining how much, if any, downtime your deployment will experience. In general, the following conditions should be met by your environment for either the canary or blue-green deployment techniques to be successful:

- Build, test, and deploy to certain environments using a deployment pipeline.
- There are a large number of application nodes or containers spread out behind the load balancer.
- An application that does not save any state and that may serve requests at any time and from any node in a cluster to be considered as a successful blue-green deployment.

In addition, any changes you make to your infrastructure should not affect the data layer (e.g., database) in any way and should be considered nondestructive. Instead of renaming or reusing columns for a variety of reasons from your existing database, the newer approach would make data columns nullable or optional. This would allow restoration of previous states without loss of any data.

The blue-green deployment is a strategy designed to solve some of the challenges of the in-place strategy. Even if you manage your dependencies and libraries in-place or via the blue-green deployment strategy, there is still one issue that has not been handled, and that is the fact that it is a tough process.

How can you be certain that a new environment, one that you have created locally or one that you have created on the cloud, is precisely the same as the one that came before it? If you forget to change one of the environments, the deployment might not go as planned.

Suppose there is a flaw in the system but it cannot be reproduced in a local environment since it is difficult to know for sure that the two environments are exactly the same. It is possible that configuration management may be of assistance, but the simplest solution would be to avoid ever updating your environment so that you can continually produce the same arrangement.

Canary Deployment Strategy

Among various strategies in infrastructure as code (IaC), canary deployment is used by organizations to determine whether a new CI/CD release will result in any challenges for the production and the users. Canary deployment is a practice that includes gradually exposing a change to a select set of users before making it accessible to everyone. This is done to lessen the risk of an update getting into production.

Canary deployments demonstrate how actual people in the real world interact with programmed improvements and how improvements are utilized in the real world. Canary deployments, which are comparable to blue-green deployments, provide very minimal downtime and rapid rollbacks in the event that an issue arises. In comparison to blue-green deployments, canary deployments operate more efficiently and have a lower rate of failures overall.

Canary release refers to an early test version of an application. The version control system branches are versioned and tagged based on the state of the deployment. The stable branch holds the code base that is currently live on the servers. The branch that consists of the code base to be deployed is generally a development branch. When the development is complete, the CI/CD checks and security checks are run before merging it to the stable branch. Once the automation is run, the canary deployment execution starts. Keeping the stable and development branches of a project distinct is an approach that is often used. Canary versions of updates and releases are often made available for use by the organizations

in the expectation that a small number of consumers consisting of advanced technical skills or early adopters would try them out. A plethora of organizations that use this approach with their application releases include Mozilla, Google, Microsoft, and many more that offer canary releases for their various products, services and applications.

Process and Architecture

Similar to blue-green deployment, canary deployment employs a slightly different approach. Canary deployments swap over a selection of servers or nodes before completing the remainder of the environment, as shown in Figure 4-1.

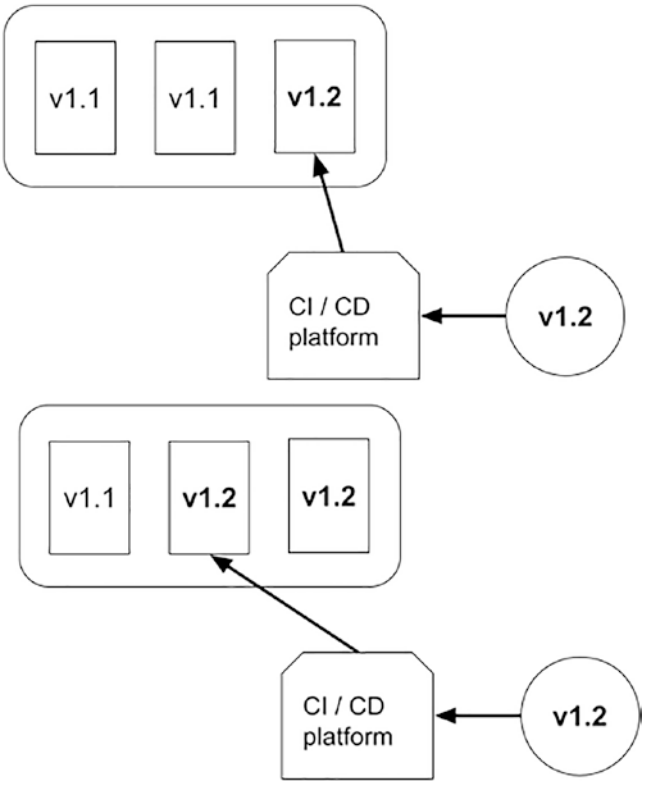

Figure 4-1. *Canary deployment*

There are several methods to design your environment for canary deployments, but the easiest is to set up your environment normally behind your load balancer but maintain an extra node or two (depending on the size of your application) as unneeded spares. This spare node or server group is your deployment target for your CI/CD process. After building, deploying, and testing this node, you add it to your load balancer for a limited duration and a restricted user population. This enables you to verify the success of modifications before applying them to the other nodes in your cluster.

Optionally, a canary deployment can be configured using a development method known as *feature toggles* or *feature flags*. Feature toggles function by developing and delivering your modifications to an application governed by a configuration that is inoperable until those changes are enabled. You remove a node from your cluster, deploy it, and then put it back without testing or controlling anything via the load balancer. Then, after all nodes have been updated, you enable the functionality for a subset of users prior to rolling it out to everyone.

Working Mechanism

The canary deployment method requires simultaneously running two different application versions. The update may be delivered in one of two ways: side-by-side or successive deployments. Let's understand how they help deployments and manage infrastructure at scale.

The first step is putting in place a new canary infrastructure, which will be used to distribute the most current update. A relatively little portion of the traffic is shifted to the canary instance, even though the vast majority of users will continue to make use of the baseline instance.

Once some traffic has been redirected to the canary instance, the team will begin gathering data, which may include metrics, logs, information from network traffic monitors, and findings from synthetic transaction monitors—anything that can assist in establishing whether the new canary

instance is functioning as expected. After that, the data is analyzed, and the results are contrasted with the baseline version.

After the team has finished the canary analysis, they decide whether to go through with the release and make it available to the rest of the users or if they should revert to the baseline situation that existed before the release.

Adapting Simplicity

A long history of ever-increasing complexity, rather than steadily improved simplicity, has long been associated with IT enterprises. Human error is increased, time is wasted, and users are unable to concentrate on their other objectives when a system is too complicated. Instead of keeping employees busy just for the sake of it, technology should help businesses run more efficiently, continue to develop, and expand into new markets.

It is possible to minimize needless complication when expanding capacity to an existing system by using current seamless techniques instead of a hardware update. If you have less to manage, you'll have more time to focus on the things that are really essential to you. Simplicity reduces or eliminates the risk of human mistake in manual management processes, which may be a concern.

To get the deployment right, here's what you need to ensure:

- It is essential that you carry out all of the required checks to ensure that the update is functioning correctly.
- When operating the code in its most recent version, a blue server is the one that should be utilized.
- It is necessary to configure the router or load balancer so that traffic from valid customers will begin to be sent to the blue server.

- It is necessary to configure either the router or the load balancer in such a way that no valid user traffic is allowed to access the green servers.
- Continue to send traffic to the blue server until you are convinced that there are no problems and that there is no need to roll back any changes.
- Keep the traffic flowing to the blue server and then pass it to the green server. It is necessary to set either the router or the load balancer in such a way that users are sent to the green server while the issue is being resolved.
- The green server has to be prepared for the subsequent blue-green deployment as soon as the final checks have been carried out.

Environment Replicability

Environment replicability refers to the capability of moving code binaries and configuration files from an active and well-tested server to a new server to set up a new environment with only minimum configuration changes required. This is accomplished by using the new server.

Along with maintaining stability, this is one of the most important aspects of blue-green deployments. The procedure consists of a number of different components, such as servers and settings. Depending on the needs of your organization, environments may range from a single instance to several instances. Some examples of environments are production, staging, mirror, and development.

The more environments there are, the higher the level of complexity that must be dealt with. To deal with this situation, you need to devise and implement automated inspections, in addition to monitoring and sending out alerts.

Rolling Release Deployment Strategy

Rolling deployment is a great way to decrease downtime in situations with a lot of static resources. It is also more cost-effective than other strategies because no new resources are required. Backward compatibility across application components is generally considered when employing a rolling deployment. Rolling deployment can be used with phases and batches. In terms of phases, rolling deployments can be used based on the scenario or case structure.

Let's look at how an e-commerce business utilizes the rolling deployment method to launch its shopping cart application to a production environment. The shopping cart application has three levels: web, app, and database. The business intends to deploy this application to the production environment in the following four stages:

1. First, deploy only to the database tier and test the database update.
2. Deploy to some of the resources in the app and web tiers to ensure that the application was successfully deployed.
3. Apply some restrictions to a few additional resources in the app layer.
4. Deploy the remaining resources in the production.

This rolling deployment technique is appropriate when predictable mapping between resources and batches is not required. Let us understand the batch process.

When resources and phases have a deterministic mapping, this rolling deployment strategy is appropriate. You can assign resources to phases manually or dynamically using expressions. Because nodes are updated in batches, rolling deployments necessitate services that handle both new and old versions of an asset.

The following actions are possible with rolling updates:

- Transfer a program from one environment to another via container image updates.
- Revert to prior versions when required.
- Apply continuous integration and continuous delivery with minimal downtime.

The advantages of a rolling deployment are that it is relatively easy to roll back, it is less dangerous than a basic deployment, and it is straightforward to implement.

We need to consider the disadvantages too since rolling deployments necessitate services that accept both new and old artifact versions. Verifying an application deployment on each incremental update also slows down the deployment.

When deciding on a deployment plan, there are several factors to consider.

- Long-running connections must be treated with care.
- Database conversions can be difficult and must be completed and rolled back in tandem with the application.
- Downtime may be required to complete the transition if the application is a combination of microservices and conventional components.
- You need the required infrastructure to accomplish this.

Process and Architecture

Rolling deployments need a production environment that has several servers that are running an application. Typically, but not always, rolling deployments also require a load balancer to be placed in front of the servers so that traffic may be distributed. When the DevOps team is ready to deploy an updated version of their program, they perform a staggered release by delivering the update to each individual server in turn.

While the upgrade is being sent out, some of the live servers will run the new application, while others will continue to run the older version. In contrast, a blue-green deployment implies that the upgraded software is either live or is not live for all users.

When starting a new session, users have the option of accessing either the old or new version of the application depending on the instructions given to them by the load balancer. As soon as the deployment is finished, each new user session will automatically have access to the most recent version of the software. In the event that an issue arises during the rollout, the DevOps team may choose to suspend the upgrades and route all traffic to the servers that is still known to be good until the problem has been resolved.

Rolling deployments are an option that should be considered by companies that are in a position to successfully manage such a vast production environment. For companies that operate in this manner, they are an excellent tool for delivering small, incremental improvements, quite similar to the approaches used in agile software development.

Steps for Managing Production Complexity

Delivering software and infrastructure is a difficult process. The implementation of certain deployment procedures and practices that will aid in the operationalization of your production environment, resulting in a better management of production complexity. The procedures and practices discussed in this chapter will assist in the operationalization of your services in a better, more efficient and manageable way.

There are a number of benefits of automating and scripting as much of the process as possible. The changes will take place more quickly, and there will be fewer opportunities for mistakes to be made by humans. If the checklist is managed by a script or a management platform, it is impossible for a developer to forget an item on the list. If everything is included in a script, the deployment may be carried out by any developer or even someone who is not a developer. Don't waste time waiting for your system professional to get back from vacation before you act. Consider putting the following procedures or standards into place.

Harnessing the Power

Teams working inside CI/CD frameworks and deployment strategies may release and manage applications in production at any time if they employ the appropriate deployment strategy with the supporting fail-safe mechanisms. In most cases, the only thing that is necessary to go live is a change in the routing. These deployments will have no negative consequences for users since there will be no downtime.

They also create less disturbance for the DevOps team. They are not required to install updates within a defined window of downtime, which increases the likelihood of deployment difficulties and adds stress. Furthermore, the success is extended to the executive teams. They won't have to keep track of the time to figure out how much money was lost during downtime.

Fail-Safe Environment Management

This further reduces the risks associated with experimenting with production processes. Your team may instantly resolve any issues by making a simple routing modification back to the stable production environment. There is a chance that cutting down resources may result in a

loss of customer transactions, which will be explored more in this section, but there are numerous alternative measures that can be done to address this issue.

During cutovers, you may make your app read-only for a limited amount of time. You may also use a load balancer to do rolling cutovers while waiting for transactions to complete in the live environment.

Monitoring Your Infrastructure

Always keep an eye out for infrastructure in your immediate surroundings. If you want your deployments to succeed, you need to be aware of everything that's going on in both your live and nonactive systems.

These systems generally need the monitoring alerts, but the order in which they are checked should be different. Let's say you want to be notified the instant a problem arises in the live system you're using. If the error is still there in the operating system at a later point in the day, it will need to be addressed.

Make sure that your code is compatible with both older and newer versions of the deployment environment. There can be occasions when a cutover will be unable to run either the new or old version of your software. In the case of changes to the database schema, for instance, it would be to your advantage to organize your updates in such a way that both the blue and green systems would continue to function normally throughout the cutover.

Compartmentalizing Releases

One strategy for dealing with these difficulties is to partition your releases into progressively more manageable bundles of code.

Already significantly dependent on rapid, incremental updates, Agile development and CI/CD are becoming even more dependent on updates. Blue-green deployments need an even stricter adherence to this criteria

than other deployment types. Reducing the number of feedback loops and integrating the knowledge gained from doing so into subsequent releases may cut deployment durations in half.

Adapting Serverless Architecture

Your applications have to be separated into separate microservices. This method works particularly well when used for more modest installations. Microservices make it possible for you to manage updates and adjustments to the code of your application in an easier manner. Because the application has been divided into more manageable portions, it's much easier to update a single feature at a time.

Feature Flagging

Utilizing feature flags is one method that may be used to accomplish further risk reduction. When blue-green deployments are used alone, they provide a single, very short window of prone to failure. You are now updating everything, but if there is a problem, you may limit the amount of work you have to do.

When it comes to deployments in scenarios such as enterprise-grade production environments, each cutover is accompanied by a consistent quantity of administrative overhead. This overhead must be paid for. Whether you are changing a single line of code or redesigning your whole e-commerce platform, you will be required to go through the same process. This is the case regardless of how significant or how little the change may be.

It is possible for feature flags to give a degree of fine-grained control over the timing and introduction of new software to consumers. Feature flags operate in a manner similar to that of powerful "if" statements, which may point to the execution of code in any one of a number of ways depending on the circumstances of the system.

As a condition, you may use anything as simple as a "yes/no" question or as complex as a decision tree. Using feature flags, which allow you to pick whether features are enabled or disabled at the feature level, makes managing software releases simpler. Feature flags allow you to choose whether features are enabled or disabled.

Because our e-commerce company uses the customizer microservice, we are able to do a blue-green deployment while concealing the newly written code behind a feature flag. After that feature has been activated, the DevOps team will have the ability to turn it on and off at their discretion.

It's possible that the manufacturing process needs another round of A/B testing. It's likely that they could ask you to undergo some further physical examinations. Alternatively, the customizer might be made available to a limited number of early adopters in the form of a canary release.

You may use feature flags in conjunction with a load balancer during a blue-green deployment to control which users see certain application and feature subsets. This is done by controlling which users see which flags. You don't have to swap over the full programs at once; instead, you may gradually activate and deactivate certain features on both the active and inactive systems until you've completely updated everything. With each new feature that is introduced, the level of risk is lowered, and it becomes simpler to identify and resolve any problems that may crop up in the future.

Compared to manual administration, the control that feature flag services give over your code base's feature flags is far more powerful. In addition to reporting and monitoring key performance indicators (KPIs), these solutions contain a variety of DevOps management functions.

If a major application release is going to be delivered in a blue-green manner, feature flags are going to need to be utilized. Because of their adaptability, they may be beneficial even in deployments with fewer parameters to manage because of the frequency with which they can be

changed. In the event that a significant problem arises, you have the option of gradually activating functionality on blue while maintaining green in standby as a hot backup. Using feature flags and blue-green deployments, continuous delivery may be achieved at any scale. This is true regardless of the size of the organization.

The Impact of Deployment Strategies

Because the blue-green deployment strategy employs two production environments, it automatically enables disaster recovery for your company's systems. A dual production environment may operate as its own warm backup.

Load balancing may also be made easy in parallel production settings. When both environments have the same set of capabilities, you may use a load balancer or feature toggle embedded into your software to divert traffic to either environment based on the conditions.

A/B testing is another application that may benefit from parallel production configurations. You may add new features to your idle environment and then use a feature toggle to split traffic between your blue and green systems.

Collect data from those segmented user sessions, watch your key performance indicators (KPIs), and then, if the results of your study of the new feature in your management system seem promising, switch traffic to the redesigned environment.

It is possible that some of these tactics will be of assistance in the event of a server or infrastructure interruption, issues with deployment, frequent feedback, or the introduction of new applications. In addition to laying the groundwork for the expedited delivery of your IaC, the concept of *continuous verification* makes it possible for us to automate some aspects of our metrics and monitoring tools. Continuous verification is a method that takes action depending on the performance and quality of application deployments by making use of data and operationalizing tool stacks.

The stage in the deployment process known as *deployment verification* produces the failure solutions of auto-rollback and manual rollback, respectively. In addition, there is the 24/7 Service Guard, which is an ongoing assessment of change effect that assesses the overall health of the service and links it to deployments.

When it comes to operationalizing deployments across a wide variety of tools, dependencies, and settings, there is the potential for a number of obstacles to arise. The next step in accelerating and streamlining the delivery of software will be to automate some of the issues that are now being faced.

Caveats While Managing Complex Production Environments

Many companies nowadays conduct the analysis phase of canary installations in a nonintegrated and compartmentalized manner. An engineer on the DevOps team is responsible for manually analyzing the canary version's monitoring data and logs. Time-consuming CI/CD techniques are not scalable for rapid deployments. When a new release is rolled back or pushed forward without proper analysis, erroneous decisions might be made.

Maintaining on-premises client applications might be difficult. In a world where apps are being used on personal devices, it's tough for a business to undertake a canary deployment strategy. Workarounds include creating an environment where end users are automatically updated. Canary is a great deployment strategy for managing many versions of an application, but handling databases requires a different set of skills. Deployment becomes much more complex when we modify the application to interface with or adjust the database structure.

It's necessary to make changes before running a canary to handle many instances of the program. As a result, both programs will be allowed to run at once. The new version may be deployed and switched over after the new database architecture has been installed.

On the other hand, blue-green deployments are quite costly. Creating an environment that is identical to one used in production may be difficult and costly, particularly when microservices are involved. There is a potential for danger in concurrently rerouting all user traffic due to the fact that quality assurance and user acceptability testing may not discover all bugs or regressions. An outage or malfunction may also have a substantial effect on business before a rollback is initiated, and depending on the implementation, user transactions that are in flight may be lost while the traffic moves.

The team adapting the blue-green deployment strategy needs to make sure that any changes made in the future are consistent with the environment that existed before. Because traffic is exchanged between the blue and green instances, using this method might be challenging if the database has to be modified to accommodate an update in the program. The usage of a database that is compatible with all of the different software updates should be required before any of them can be implemented.

The Way Ahead

Each deployment strategy is useful and has an architecture that is comparable to the others, but each has its own unique set of characteristics.

If you have the capability to run two fully functional infrastructure environments and your infrastructure does not change in a manner that is backward incompatible very often, a blue-green deployment strategy can provide you with the greatest number of secondary benefits while

requiring the fewest number of changes to your applications. In the event that performance concerns arise or a catastrophe has to be recovered, it offers an environment with minimal downtime.

Canary deployment is an option worth considering if your application is both configuration-driven and modular, as well as if you have a limited amount of additional resources at your disposal. Because you do not have an additional environment that may be used for other matters, your operating and maintenance expenditures are reduced. Functionalities may be engaged or deactivated at whim and according to any number of criteria when utilizing a canary deployment, which is an extra advantage of employing this kind of deployment.

However, prior to using any approach, you will need to do a significant amount of planning and research into the architecture of your applications and environments.

It's possible that even if you follow all of the best practices, something will slip through the strategy. Therefore, it is just as important to monitor for problems that develop after a deployment as it is to design and execute a faultless deployment.

Your team may benefit from using an application performance monitoring (APM) solution to monitor critical performance indicators such as server response times after the deployment of new software. The performance of an application may be significantly impacted by changes to the system architecture or the program itself. It's also necessary to have an error monitoring system. It will quickly notify your team of any new or reactivated defects from a deployment that may disclose serious issues that need immediate correction.

It's possible that the issues wouldn't have been discovered without the use of an error tracking application. Only a small percentage of those who are aware of problems will go out of their way to report them. Long-term dissatisfaction or even a halt in present economic activities might result from poor customer service.

Developers and operations/DevOps teams may collaborate on post-deployment issues thanks to an error monitoring system. It's possible that performance will suffer if the application and the architecture's other services interact with one another. Inefficiency may be caused by a variety of factors, including problems with database queries, failed service connections, and a lack of resources. Checking in with the metrics and logs provided by the platform is something you need to do if the teams want to monitor the performance of the services provided by your architecture. The teams can be more collaborative and responsive because of this shared understanding.

In the next chapter, you will learn about the different aspects of businesses with respect to IaC as well as approaches, perspectives, and ways to develop, manage, and deploy IaC for various business solutions.

CHAPTER 5

Business Solutions with Infrastructure as Code

Utilizing infrastructure as code (IaC) is now among the most often used approaches to automate a company's infrastructure. In terms of the distribution of innovative technologies, it is possible to compare it to a factory floor. In the field of technology, having reliable manufacturing procedures is essential to the operations of a corporation.

Not only does IaC enable you to have a greater understanding of and control over your surroundings, but it also provides you with extra benefits. For example, you can easily deploy your code on a remote server, launch various database instances, define network and security rules, and customize your cloud-specific configuration, as well as manage states, executional logs, and failure strategies all from the script maintained as IaC. One definition of a flexible infrastructure automation chain is one that is able to adjust to various new circumstances. IaC allows you to set abstraction boundaries, which makes it easier for you to choose which technologies to use and which providers to work with. You are able to make speedy updates, repairs, or changes to your technological automation chains when you have the backing of IaC. For example, you can orchestrate your IaC to set up an EC2 virtual remote server on AWS

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_5

for server-specific operations, you can provide a content delivery solution using Microsoft Azure's CDN, and the network logging can happen on a platform like Sentry, Datadog, or somewhere else. All of this is possible through abstraction provided by IaC, which keeps things clean and easy to manage.

When a business embraces IaC, it opens up new doors for the rest of the organization to see technology as a commodity and capitalize on those possibilities. In the realm of commercial technology, IaC is seen as a potentially transformative innovation.

Managing Modern Infrastructures

Even in the highly automated era of cloud computing and the Internet, information technology infrastructure does not evolve on its own. It is still the responsibility of an organization's IT department to architect, establish, provision, create, deploy, and manage the chosen infrastructure, at least initially, before DevOps teams may fully take the ownership. This is true even if the majority of the ongoing maintenance is handled by a third-party technology provider that is taking care of the infrastructure as a service (IaaS) or IaC for an organization.

IaC has a descriptive model for defining and supplying an IT network's data storage capacities and capabilities, server topologies, and other fundamental pieces like load balancers that are comparable to software application code. IaC files, written in languages like Terraform (the language) as well as Terraform (the technology), CloudFormation, etc., are used to build IT infrastructure rather than office server hardware. IaC lacks connections and cables. Instead, model source code files define connection topologies.

Because of the ever-evolving nature of these infrastructures, the growth of the information technology infrastructure may be carried out accurately and provided in a manner that is more streamlined and efficient. It is

also possible for it to be performed by what computer purists refer to as a misconfiguration of resources, which is simply a technical euphemism for a mistake.

Enabling Business Possibilities

The software development life cycle is often referred to as IaC, and some of the most successful organizations in the world have already used cutting-edge strategies for the process. Even though these skills build on top of one another, the ability to automate tasks is essential to all of them.

The capability of IaC to automate the process of installing and deploying systems and software is the most notable aspect of this component of its functionality. The use of automation might result in a technique that is both more effective and more easily replicated.

The source code that makes up the IaC has to be stored in a repository and put under version control at all times. As a consequence, it is possible to securely version the process of installing infrastructure: utilizing the IaC code for the version that is given makes it easy to create any infrastructure version. It's possible that the automation and versioning of the installation process will make things simpler and more trustworthy.

It is helpful for troubleshooting and debugging to be able to undo changes made at any stage in the development process, including integration and production. This functionality enables you to reverse changes made at any point in the development process.

It is feasible to apply code written in IaC to all phases of a project, including the stages of production, integration, and development as visible in Figure 5-1. This increases environment parity and may help to prevent instances in which one developer's environment works but another's does not or in which the environment works in development but does not function in production.

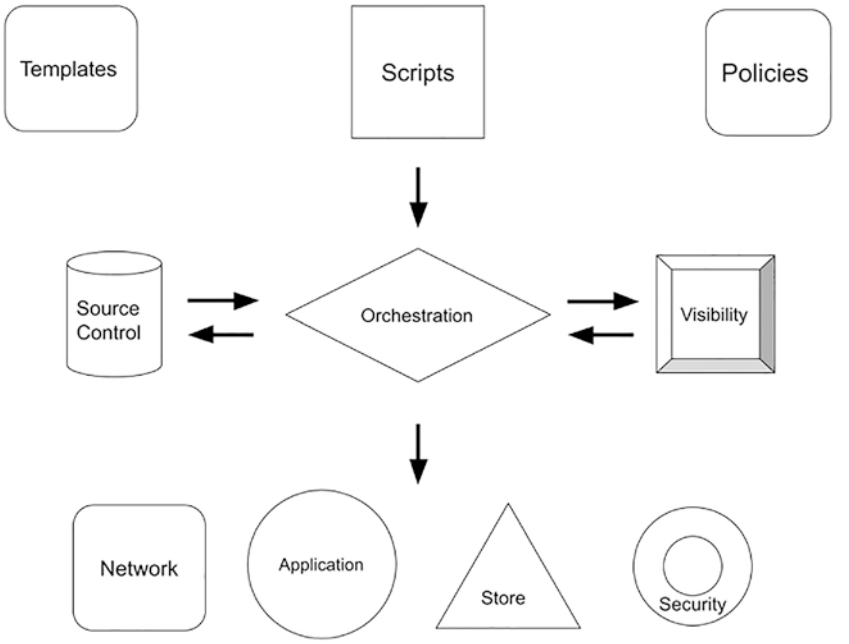

Figure 5-1. *IaC applicability*

IaC may also make it possible for information technology to be run in an immutable manner, which is a notion referred to as *immutable infrastructure*. It is common practice to deploy application management software and infrastructure management software on the systems that make up an operating system (OS). Patches are installed on each node individually, software is kept up-to-date, and network settings and other configuration elements are modified as necessary throughout the course of time. A configuration drift may take place, for instance, if there is a disparity in the patch level between different nodes. There are certain instances in which the configuration of a node cannot be re-created from scratch, and the only option for restoring the node is to use its backup. In an immutable architecture, it is not possible to apply patches, updates, or make configuration changes to nodes that have already been deployed. This is because it is not practicable. Instead, a new version of the code that

makes up IaC is constructed. This new version includes all of the required modifications to the application as well as the infrastructure. It is possible for a new version to undergo testing in the development and integration environments prior to being sent to the production environment. While this is going on, environment versioning may be utilized to undo any modifications that were made to the environment after it was deployed.

Numerous cloud service providers are used by businesses to increase their operations' adaptability, productivity, and error-proofing. The efficiency with which several cloud environments collaborate is crucial to the success of multicloud architecture. Switching between several cloud service providers is simplified by code that supports multicloud operations, setup, and execution. For instance, in a completely remote cloud architecture, you may switch between data centers based on their proximity to peak use periods. This would provide you with more options and access to better prices than a single supplier could offer.

An active business ecosystem has developed cloud-independent managed services built on open source technologies. These public cloud providers are prepared to capitalize on the growing demand for their services as more and more businesses transition to the cloud.

It is possible that in the future investigations, evidence for assurance methodologies such as the Risk Management Framework, and the IaC artifacts, will be used. IaC artifacts open up a new attack surface, which might be beneficial to some forms of cybersecurity, such as moving target protection.

Enabling Domain Sustainability

IaC is a novel methodology for the creation of infrastructure. It is more directly connected to DevOps than it is to Agile software development, although the link is still indirect.

The Manifesto for Agile Software Development provides the basis for a broad variety of Agile software development practices. Creating software that functions as intended and being receptive to change are two of the best examples of Agile principles. The use of IaC and automation tools, such as CI/CD platforms or workflow engines, in addition to versioning systems like Git, may help automate infrastructure in accordance with the ideas of the Manifesto for Agile Software Development.

By merging the efforts of development and operations, it is possible to shorten the amount of time required to make a change to a system while simultaneously improving the system's ability to function normally once the change is implemented.

IaC is an essential component to the success of DevOps processes like continuous integration and continuous delivery. IaC may also be of assistance to DevOps. When deployments are automated, both the delivery mechanism for deployment and the cycle time are improved. To improve the quality of the software as a whole that is sent out, the testing that occurs in the environments used for development and integration must be equivalent to the testing that occurs in the settings used for production.

Maintainers may find IaC useful since it functions as a safety net that allows them to carry out controlled tests and, as a result, get additional information about the system. Because of the automation and versioning capabilities of the IaC technology, maintainers are able to roll back unsuccessful tests in a progressive and Agile manner. Additionally, maintainers are able to document and apply successful upgrades.

With the help of IaC, site reliability engineers (SREs) may design and efficiently deploy infrastructure configurations that boost reliability. With IaC, there is less of a chance that dependability recommendations would be misunderstood or overlooked. It simplifies interactions between SREs and other teams.

If an SRE is investigating an outage or other reliability issue, they may use this tool to determine whether a modification to the infrastructure configuration was to blame. Problems may be solved, and events can be understood with the use of this data. If an engineer makes a mistake while manually configuring infrastructure, such as opening the incorrect port or installing the wrong container image, it might have serious consequences for availability. IaC eliminates these dangers by facilitating the automated application of configurations by teams. This eliminates the possibility of typing errors or other blunders on the part of engineers. As long as your IaC files are accessible, your infrastructure can be properly configured.

SREs may quickly become proficient with IaC tools since they are widely available and simple to implement. These methods are efficient in terms of output, and it doesn't take long to master them. Many SREs already have coding skills, making it simple for them to pick up IaC.

SREs must also evaluate the configuration languages available in the tools to see whether they are a good fit for their needs. Since the scanning of IaC files varies depending on the IaC tool being used, SREs should investigate its scanning and validation procedures before settling on an IaC platform.

IaC artifacts also give the actual deployment structure of the system, which may be compared to the architectural documentation of the system in order to validate that the system adheres to the design in the manner that was intended.

There may be a future edition of the software in which the configuration changes are either permanently remedied or, at the very least, detected automatically. This would be the next era of self-sustainable IaC.

Supporting Evolving Strategies

Consider a situational criticality such as the excessive latency caused by the corporation's internal servers; software development and training activities located in the region being targeted are unable to take advantage of global assets. Users may connect only to a virtual private network (VPN) as a result. The objective is to provide them with an infrastructure that meets all of the necessary requirements, including being highly compliant, dependable, and highly secure, while also taking into consideration cost, licensing, and performance. Figure 5-2 illustrates this.

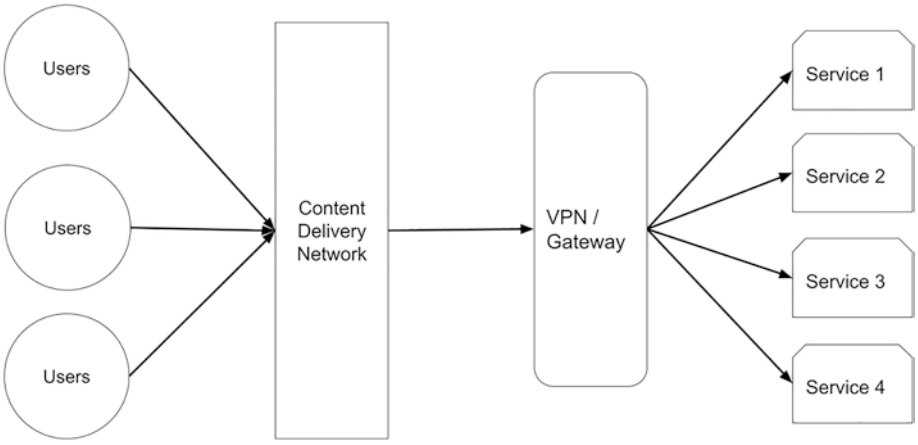

Figure 5-2. *Infrastructure supporting different use cases*

Another example is an e-commerce business that utilizes the rolling deployment method to launch its shopping cart application to a production environment. Architecture teams develop and deploy to the cloud, using the Cloud Platform Automation Framework that supercharges infrastructure management. IaC satisfies the criteria pertaining to network and security while also being transportable, scalable, and trustworthy.

The use of cloud resources allows for the development and testing of infrastructure to take place at a more rapid pace. Because of this, businesses are able to continue their evolution as well as their other development operations with very few disruptions. The automated solution not only helps businesses and customers save resources but also frees up the time of the IT administration and support employees by automatically complying with all regulations and best practices.

The deployed IaC, despite its magnitude, results in reliability, consistency, and automation across the organization. Teams can use a number of innovative approaches, including IaC. However, the infrastructure is created over time by a large number of people. Businesses can adhere to a number of industry standards, including an infrastructure code tree with deviations described by environment-specific variables.

Suppose the organization's project goes into complete anarchy because it did not make use of the techniques and tools that are associated with DevOps. It is not unusual for the infrastructure to be unorganized, unclear, and unrecorded in any way whatsoever. It is difficult to make adjustments or add new components without causing damage to the components that were already present in the system. There is no way to guarantee that a feature that had been tested and found to work properly in one environment will also work correctly in the other environment. Maintaining such projects is both expensive and risky. Hence, implementing appropriate IaC strategies is essential as well as beneficial for organizations.

The time it takes to release the software gets cut drastically thanks to certain best practices that DevOps teams adapt with infrastructure automation and deployment. The infrastructure gets upgraded continuously to be more intelligent and simpler to utilize. It is less difficult to make adjustments and add new elements. Code developers working on infrastructure no longer have to monitor all of the dependencies that exist across different components. The structure of the code has been clarified,

and there are now more individual components. Unified coding is used to assist in the establishment of a wide variety of situations. As a result of all of these enhancements, the product will have increased uptime, fewer release risks, and reduced operational expenses.

Decision-Making for Businesses

A growing number of companies and industry leaders in technology are starting to adopt infrastructure as code. Because modern companies take part in the near-constant development, testing, and deployment of software applications, they demand a flexible infrastructure that can quickly and securely adapt to the ever-changing requirements of their customers. IaC is analogous to a technological solution for the management of the resources necessary to develop technological solutions; it does this by using high-level code to automatically offer IT infrastructure.

Before implementing IaC, businesses and executives in charge of IT must have a thorough understanding of both the possible advantages, some of which may be very significant, and the feasible challenges.

Cloud computing makes it possible to implement IaC. Setting up, modifying, or removing services associated with virtual infrastructure is made noticeably faster and less difficult as a result. IaC gives teams the ability to use infrastructure in a manner that can be controlled by a computer, which automates the administration of life-cycle processes. Command-line interfaces, which do not need any programming, are used by certain automation systems to handle the devices they control.

Using IaC will probably be highly useful to corporations. Coding documentation and setup instructions removes the potential for errors, which is very helpful for newcomers. If you are able to provide IaC, then the risk of errors caused by humans and of breaches in security is significantly minimized.

Prior to the introduction of IaC, cloud service deployment and configuration were treated as separate tasks. After the cloud infrastructure was established, each option could be independently implemented and maintained. Setting up and securing virtualized infrastructure at the code level is managed by IaC security.

For IaC to work, every virtualized infrastructure must be installed and configured in the same manner. Serious security issues may arise if the IaC is not accurate and secure. When IaC is broken or not secure, all newly deployed server instances may have security holes.

When a business begins looking at IaC, its security paradigm might shift from one of detection to one of prevention. Before starting construction, an IaC scan is performed. By doing so, security shifts toward reducing the cost and impact of a misconfiguration.

IaC must have its settings adjusted correctly for it to be of considerable use to a corporation. Securing, reusing, and administering governance over an IaC pipeline are all key components. It is possible that a thorough continuous integration and continuous deployment pipeline that includes IaC might significantly cut down on the amount of time needed to bring an application to market and save money in the process.

When transitioning from an established IT organization to an IaC model, the present IT workforce of a firm is something that has to be taken into account. IaC systems need a degree of skill in system development that your present technical people may not have. You run the risk of lowering employee morale and alienating existing staff if they do not have experience in coding.

The necessary infrastructure needs to be defined using computer code. When it comes to automation, the infrastructure of a firm that was built just once is not an ideal solution; however, infrastructure that is often utilized for the development of new applications or services is acceptable. It's not always the greatest approach to get the most out of your IaC endeavors to automate everything there is to automate.

The most essential step is to ensure that the design receives the appropriate amount of time and focus. Everything must be designed to be modular, and the configuration must be the primary focus of all efforts. When developing the infrastructure, you should do it with an awareness of the apps that will be operating on it. Concerns related to IaC architecture might vary depending on a number of factors, such as whether you want to construct a transactional application or a reporting application using the database you want to install.

As a general rule of thumb, jobs that involve repeated tasks should be mechanized, whereas exceptions should be dealt with manually. As a consequence, internal stakeholders and departments could benefit more from using this strategy. It is essential to examine the return on investment that automation may provide for a company since if that company automates every infrastructure process, the expenses may be inflated (ROI).

The infrastructure of any organization is, by definition, a mission-critical component, and the code should be managed exceptionally well. This includes having the appropriate processes and backup methods in place in the event that the infrastructure fails. As a consequence of unforeseen transformations, a significant number of well-known firms have declared bankruptcy lately. Virtual networks, data centers, and servers are much like their physical counterparts in that they need to be fine-tuned and put through intensive testing.

In addition, it is essential that engineers and software developers collaborate in order to advance the infrastructure. Many infrastructure specialists are not up-to-date on the most recent developments in software development, despite the high levels of skill they possess in this field. Utilize your most talented software engineers so that you can push your infrastructure staff out of their comfort zone and get optimum results. It is important that the process of creating a team be transparent and equitable.

Concentrating on the adoption of IaC may result in a lack of flexibility for startups while they are still in the early stages of their growth, despite that this strategy offers enormous advantages for larger organizations. In the field of information technology, a significant emphasis is placed on originality and creativity. Therefore, you need to find a middle ground between finishing a significant milestone and encouraging your teams to think creatively outside the box. Consider an example of an organization that is developing business solutions using the latest technology where their competitors are already big giants in the industry. This organization wants to create their own space. In this case, many times developing a solution would not be sufficient because the organization would be required to showcase unique features and offerings that are original and creative from the previous offerings provided in the industry. Hence, this organization would have to balance both innovation and creativity in their core work.

When considering whether to use IaC, businesses need to consider both the benefits and the potential downsides. The acceptance, security, and scalability of IaC provide some of the most significant challenges when it comes to the process of integrating new frameworks with existing infrastructures. Implementing an IaC solution requires not just time but also collaboration with other departments, such as those responsible for security and compliance.

Your preparation for the transformation to IaC should include establishing your goals in detail. The organization's technology and people may be placed under unnecessary strain as a consequence of inexperienced engineers and outdated network automation, which may lead to instability. Incorporate IaC into modernization initiatives and connect it to retraining in order to free up engineers to focus on more difficult tasks.

When deploying IaC, it is necessary to take into account a number of aspects, including the frequency of maintenance, potential security issues, and the total amount of time spent developing. Get ready for the

worst-case scenario, and keep in mind that there are many paths to get there: what steps will you take in the event that option A or B is no longer applicable? If you have a plan laid out, you should be able to recoup your initial investment and turn a profit.

When it comes to the discovery of pitfalls and solutions, it is simply too easy to identify severe vulnerabilities that were not anticipated. Standard processes for deploying infrastructure take security precautions into account, which makes it simple to introduce new vulnerabilities. The implementation of an IaC approach comes with a number of distinct benefits. This indicates that you need to have a comprehensive strategy in place, which should include procedures for quality control and safety precautions.

The Way Ahead

Markets for IaC are in continual motion; hence, fresh techniques to tackle these difficulties are regularly tried. For example, Pulumi makes use of Open Policy Agents (OPAs) by default; these agents are a helpful answer for the problem that Terraform does not have a uniform model.

The most important question that has to be answered is whether each and every member of the engineering organization needs to be proficient in IaC language, ideas, etc., for the method to be effective. If organizations don't put effort into understanding how it works and how it is going to help them, IaC is going to be the biggest black box. Going forward, the tension between the development team and the operations team is reduced because both teams want to optimize their infrastructure and are adapting the scripts that control it using IaC.

In the next chapter, you will learn about another IaC automation and delivery tool named Terraform, including how it works, why organizations should consider it, details of the Terraform architecture, and how you can implement Terraform in your projects. You'll also see use cases of Terraform.

PART 2

Hands-on Experience

CHAPTER 6

Hands-on Infrastructure as Code with Hashicorp Terraform

Cloud-native systems are characterized by increased speed and agility because of the utilization of microservices, containers, and a contemporary system design. They automate the build and release stages so that they can guarantee the quality and integrity of the code. Cloud technologies allow for quick and scalable deployment by layering abstractions on top of complex infrastructure services like Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (S3), and Kubernetes. With Terraform's user-friendly interface, automating the rollout of these services is a breeze. But there is more to the story than meets the eye. You will now understand how the virtual cloud environments with these technologies are deployed and provisioned.

Infrastructure as code (IaC) is extensively used in contemporary cloud-native project development. IaC is used to automate the process of platform provisioning. The software engineering practices of testing and versioning are included in the DevOps toolkit of approaches.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_6

Your deployments and infrastructure are fully automated, reliable, and consistent with one another. IaC is completely changing the way that application environments are handled in the same manner that continuous delivery has automated the manual deployment method.

Tools like Hashicorp Terraform provide you with the ability to write the cloud infrastructure you need using declarative language. In this chapter, you will learn about Hashicorp Terraform in detail. This chapter talks about why you should consider Terraform for your organization and includes a step-by-step guide to implementing Terraform from scratch in detail. Then it talks about the different use cases and nitty-gritty details from complex scenarios to security and policy compliance in the real world.

Introduction to Terraform

Terraform is an IaC solution developed by HashiCorp. It enables users to specify cloud and on-premises resources in configuration files that are simple to understand and can be used, shared, and modified. After that, you are free to continue providing and maintaining your infrastructure using the same strategy for as long as it is in existence. With Terraform's assistance, activities such as constructing, upgrading, and maintaining infrastructure are simplified. Compared to YAML or JSON, HashiCorp Configuration Language (HCL) is a user-friendly option for writing Terraform configuration files.

You can manage your whole infrastructure with Terraform, from end to end. However, it does not replace the tools that you use to manage the configuration of your virtual machines (VMs). If you have a configuration management system, you may use it to apply your configuration, recipes, or playbooks once your infrastructure is ready. For Terraform to be such a useful tool, it has a number of key qualities, some of which are as follows:

- It is not only configuration management; it also provides orchestration.
- It offers infrastructure that is immutable and enables simple configuration adjustments at the same time.
- HCL is easy to understand and not complicated.
- It is easy to switch to a different provider, and there is the option of client-only design, which removes the need to retain settings on a server.
- Support is offered for a wide variety of cloud service providers, including Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), DigitalOcean, and others.

Why Choose Terraform?

Utilizing application programming interfaces (APIs), Terraform is able to build and manage resources on cloud platforms and other services. In its current state, Terraform is compatible with the vast majority of API-supported platforms and services.

Terraform allows environments that are disposable and may be repeated, as well as infrastructure that can be reused and perhaps shared. If, for example, the production environment is codified and shared with the staging environment, quality assurance environment, and development environment, etc., utilizing these parameters allows for the rapid creation of new testing conditions, after which those environments may be retired.

Terraform Cloud,[1] like other forms of code, is designed to interact directly with the provider of your version control system (VCS), making it simpler for several people to work together. Using a version control system

[1] https://www.terraform.io/cloud-docs

such as Git, it is feasible for teams to collaborate on the development of infrastructure and for members of the team to have their own, individualized copies of the code and to establish their own testing and other environments for themselves.

Because infrastructure management solutions are often cloud-specific, deploying them across several clouds may be challenging. Terraform is a tool that may assist you if you are required to utilize several cloud providers and have cross-cloud dependencies. As a result of a decrease in the complexity of administration and orchestration, operators now have the ability to design large-scale multicloud systems.

Before starting the web servers, Terraform will ensure that the database layer is ready to go, and it will also make sure that the load balancers are aware of the web nodes. Through the use of Terraform, each tier's scale may be readily adjusted by modifying a one configuration option. It is not difficult to keep up with demand as long as the supply and demand for resources can be clearly defined and accounted for via automation.

Understanding Terraform

The number of providers that can handle tens of thousands of different resources and services has already reached more than 1,700 thanks to the combined efforts of HashiCorp and the Terraform community as of mid-2022, and this number is still growing. The Terraform Registry compiles a list of all publicly accessible providers, including but not limited to Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, and DataDog. The Terraform workflow is broken up into three stages.

Write: It is possible to establish resources that are shared across several cloud providers and services. For instance, you may construct a configuration to

deploy an application on virtual machines that are part of a VPC network that also has security groups and a load balancer.

Plan: Terraform gives an execution plan that outlines the infrastructure that it will construct, update, or delete depending on the current infrastructure and your configuration settings. This plan is generated based on the existing infrastructure.

Apply: After receiving permission, Terraform will next carry out the predetermined operations in the appropriate sequence, taking into account the interdependence between the resources. For example, if you change the number of virtual machines that are allowed in the settings of a VPC, Terraform will rebuild the VPC before scaling the number of allowed virtual machines.

Core Concepts

The most important ideas and terminology associated with Terraform are as follows:

Variables: These are the keys and values that are used by Terraform modules to allow customization of the environment. Inputs are often referred to as the variables that are entered.

Providers: Plug-ins enable access to service APIs and the resources associated with them.

Module: Each Terraform template has its own folder, which is known as the module. This folder is stored in the Terraform root directory.

State: The state is where Terraform stores information and settings pertinent to the infrastructure that it administers. This information includes the infrastructure itself.

Resources: A resource is a collection of infrastructure items, and in the context of infrastructure configuration and management, it refers to that collection (compute instances, virtual networks, etc.).

Data source: Through the data sources that are offered by service providers, Terraform is able to have access to the information that is contained inside external objects.

Output values: These are the values that are returned by the Terraform module in all other circumstances.

Directory Structure

To make it easier to grasp the code, the Terraform configuration files (`.tf`, `.tf.json`, and `.tfvars`) are organized in a tree-like fashion. Modules of Terraform will load these configuration files when they are loaded.

The Terraform modules reside at the absolute pinnacle of the structure of the configuration files. A child module may be invoked from a Terraform module's local directory, from anywhere on the disc, or from the Terraform Registry.

There are five files that comprise Terraform: `main.tf`, `vars.tf`, `providers.tf`, `output.tf`, and `terraform.tfvars`.

The Terraform `main.tf` file contains the basic code that specifies the resources that need to be generated, updated, or maintained. This code can be found at each module's root directory.

`vars.tf` is a file that is part of the Terraform configuration system that stores the user-customizable input variables that are specified in the `main.tf` file.

`output.tf` is the file in which you explain the output parameters that you want to receive from Terraform, specifically after the `terraform apply` command.

The `main.tf` file and the `vars.tf` file both include references to variables, and the `terraform.tfvars` files contain the values for those variables.

`providers.tf` is the file in which you declare your Terraform providers to authenticate with the cloud provider. Examples of Terraform providers include the `terraform aws` provider and the `terraform azure` provider.

The `.terraform` directory is created at the root level of the project which stores cached versions of the provider, modules, and plugins, in addition to the most recent version of the backend configuration. This is handled by Terraform, which also creates it whenever the `terraform init` command is executed at the root level of the project directory.

How Terraform Works

Terraform's architecture focuses on the two main components, namely, Terraform core and providers. The following sections give you a deeper understanding of them.

Terraform Core

The core of Terraform receives information from two distinct sources, both of which are external. Terraform core asks the plugin of the Terraform provider for configuration to perform any operation. The second input is provided by Terraform, which is responsible for maintaining the infrastructure configuration in the most up-to-date manner feasible.

This indicates that the information has been acknowledged and a strategy is being devised to finish the current assignment. In the end, a comparison is made between the present state, the configuration that was wanted, and the state that existed before. This rule will identify what actions need to be taken in order to reach the configuration file state that has been specified. The system is able to keep track of anything that is added to, modified within, and removed from the infrastructure.

Providers

The provision of technology-specific services is the second component of the architecture. Any infrastructure as a service platform, such as AWS, Microsoft Azure, Google Cloud Platform, or any comparable option, may be included. These include the Kubernetes orchestration system as well as other high-level components like platforms and some software services.

It makes the expansion of the infrastructure easier to achieve across a variety of fronts, such as setting Kubernetes atop an Amazon Web Services (AWS) architecture and then constructing the Kubernetes services and components.

Customers of Terraform have access to all of the resources provided by the various Terraform providers. Terraform comprises more than 100 different technology suppliers. When you use an AWS provider, you have access to many of the AWS resources that are accessible, such as EC2 instances, other users of AWS, and so on. This is just one of the numerous advantages of utilizing an AWS provider.

In its most basic form, this is how Terraform operates. It is made to assist you in building up the whole application stack, from the infrastructure to the application itself, and it is meant to do it in a streamlined manner.

Implementing Terraform in Real Projects

You learned about Terraform files as well as the directory structure of Terraform in the previous section. As we go on, it is essential to have a solid understanding of how to specify Terraform variables inside the Terraform configuration file (`var.tf`).

Declaring variables enables you to switch modules across various settings of Terraform, which ultimately results in your module being reusable. Variables in Terraform may be of many different kinds, including Boolean, list, string, and map, to name a few of the available options.

You need to understand the definitions of the various sorts of Terraform variables first. Each input variable for the module has to be stated using a `variable` block. A unique variable name is represented by the label that comes after the `variable` keyword inside the same module.

Inside the variable block, you are allowed to use the following arguments:

- `default`: If you provide a default value for a variable, you may offer its value in this block on its own, thereby making the variable optional.
- `type`: This parameter is used to identify the kinds of the value.
- `description`: This parameter includes a description of the value that was entered for the variable.
- `validation`: This parameter establishes any necessary validation rules.
- `sensitive`: If you provide this value, Terraform will not publish the supplied values while it is being executed if you choose the Sensitive option.

- `nullable`: If you do not need a value for the variable, provide the value `null`.

The following code snippet offers an example Terraform configuration for your reference. The type is used to specify a data type for the variable. `default` specifies the default value of the variable when initializing the variable. `description` specifies the description or provides short documentation for the variable.

```
variable "config1" {
    type        = bool
    default     = false
    description = "boolean type variable"
}
variable  "config2" {
    type    = map
    default = {
        us-east-1 = "image-1"
        us-east-2 = "imagev2"
    }
    description = "map type variable with default values"
}

variable "config3" {
    type    = list(string)
    default = []
    description = "list type variable"
}

variable "config4" {
    type    = string
    default = "hello"
    description = "String type variable"
}
```

```
variable "config5" {
    type =  list(object({
        instancetype        = string
        minsize             = number
        maxsize             = number
        private_subnets     = list(string)
        elb_private_subnets = list(string)
    }))
    description = "List(Object) type variable"
}

variable "config6" {
    type = map(object({
        instancetype        = string
        minsize             = number
        maxsize             = number
        private_subnets     = list(string)
        elb_private_subnets = list(string)
    }))
    description = "Map(object) type variable"
}
```

Priority Order for Terraform Variables

Terraform variables follow a higher-to-lower priority order as mentioned here:

1. Variables defined as the environment variables
2. Variables defined in the file `terraform.tfvars`
3. Variables defined in the file `terraform.tfvars.json`

4. Variables defined in the `*.auto.tfvars` or `*.auto.tfvars.json` file
5. Variables defined using the `-var` and `-var-file` arguments on the command line

Declaring Output Variables

Terraform modules include the `outputs.tf` file, which is responsible for supplying Terraform output variables.

Once the `terraform apply` command has been executed, the `output.tf` file that is described next contains not one but two unique Terraform output variables named `output1` that will store and display the instance's ARN, respectively. The public IP address of the instance is saved and shown in the `output2` variable. Following the execution of `terraform apply` with the `sensitive` argument, `output3` is generated, which stores the instance's private IP address but does not display it. The following code illustrates this:

```
output "output1" {
  value = cloud_azure_instance.machine.arn
}

output "output2" {
  value = cloud_azure_instance.machine.public_ip
  description = "Public IP address of the instance"
}

output "output3" {
  value = cloud_azure_instance.server.private_ip
  sensitive = true
}
```

Declaring Terraform Resources

HCL, developed by HashiCorp, was created with simplicity in mind. Its goal was to make setup easier to understand compared with other popular languages like YAML. The various Terraform settings are accessible through stanzas or blocks in the HCL syntax. Identical key-value pairs are what make up lines or blocks.

`resource r1` creates an AWS EC2 instance using HCL,[2] while `resource r2` installs Apache on the AWS EC2 instance with the Terraform provisioner, as illustrated in the following code. Timeouts can be used to restrict the amount of time spent on certain activities.

```
resource "aws_instance" "r1" {
      instance_type = "t2.micro"
      ami           = "ami-1234"
      timeouts {
            create = "30m"
            delete = "1h"
      }
}

resource "aws_instance" "r2" {
      provisioner "local-exec" {
            command = "echo 'Terraform Deployment' >
            terraform.txt"
      }
      provisioner "file" {
            source      = "terraform.txt"
            destination = "/tmp/terraform.txt"
      }
```

[2] https://www.terraform.io/language

```
      provisioner "remote-exec" {
            inline = [
                  "apt install apache2 -f /tmp/terraform.txt",
            ]
      }
}
```

Another way to present a Terraform block is via the Terraform JSON format, as shown in the following code snippet using the same example mentioned earlier. Both ways can be used while developing or deploying your Terraform implementation as per your convenience. The JSON syntax is a bit harder for humans to read and edit but easier to generate and parse programmatically.

```
{
  "resource": {
    "aws_instance": {
      "r1": {
        "instance_type": "t2.micro",
        "ami": "ami-1234"
      }
    }
  }
}
```

```
{
  "resource": {
    "aws_instance": {
      "r2": {
        "provisioner": [
          {
            "local-exec": {
```

```
            "command": "echo 'Terraform Deployment' >
            terraform.txt"
            }
          },
          {
            "file": {
              "source": "terraform.txt",
              "destination": "/tmp/terraform.txt"
            }
          },
          {
            "remote-exec": {
              "inline": ["apt install apache2 -f tmp/
              terraform.txt"]
            }
          }
        ]
      }
    }
  }
}
```

Terraform Provider

For Terraform to interface with cloud providers or API services through plugins, it is necessary to have a Terraform provider. The registry may include entries for more than one kind of Terraform provider; examples of these entries are the AWS Terraform provider and the Azure Terraformation provider.

Configurations of Terraform need to define which providers must be downloaded and installed in order for Terraform to make use of them. Before they can be utilized, some service providers need an initial configuration, such as endpoint URLs or cloud regions. In addition, the supplier of the service makes use of its very own local utilities, such as those designed specifically for the generation of passwords and random strings. You are free to establish as many or as few agreements as you need for a single provider without facing any kind of restriction. It's possible that your code includes more than one provider.

The Terraform registry includes several service providers. Some of these were developed by the organizations. It is also possible to write your own providers. The following code offers an example of Terraform provider declaration.

`terraform {}` is the root block, as defined next, which enlists the required providers to be used in the infrastructure configuration and deployment:

```
terraform {
      required_providers {
            azure = {
                        source = "hashicorp/azure"
            }
            mysql = {
                        source = "microsoft/mysql"
            }
      }
      required_version = ">= 0.29"
}
```

Each of the provider blocks consists of the properties that are needed for that specific provider's configuration parameters, such as secret keys, endpoints, regions, and so on.

```
provider "azure" {
      assume_role {
            role_arn = var.role_arn
      }
      region = var.region
}

provider "random" {}

provider "mysql" {
      host        = azure_mysql_cluster.main.endpoint
      username    = username
      password    = password
}
```

To include additional options that are not the default, use the `alias` meta-argument, as shown in the following example for several provider blocks with the same name:

```
provider "azure" {
      region = us-east-1
}

provider "azure" {
      alias = "west"
      region = us-west-1
}
```

Resources are the default provider configurations inherited from the declared providers. The standard resource will generate, launch, and/or deploy the resources as per the providers specified in the double quotes. To make use of alternative provider configuration, you need to use an alias, which is declared by the keyword `provider` inside the `resource` block. In the following example, `provider = azure.west` is an alias for the provided configuration for the Azure instance:

```
# Defining default provider is not required here
resource "azure_instance" "resource-us-east-1" {}

# Defining alias provider here is required to use us-
west-1 region
resource "azure_instance" "resource-us-west-1" {
      provider = azure.west
}
```

Terraform Modules

Terraform modules each come with their own set of configuration files, which may be used to manage a single resource or a collection of resources collectively. Keeping tabs on a single resource inside a single Terraform configuration file is an illustration of the functionality that may be provided by a Terraform module. A Terraform module or a root module may also be used if you want to manage a large number of resources that are detailed in a number of separate files but are combined into a single file at the end of the process.

It is possible for a Terraform root module to contain several child modules, data blocks, resource blocks, and so on. When invoking the child module, use the source parameter to provide the path of the module's executable file, as illustrated here:

```
module "efs" {
      source             = "./modules/EFS"
      subnets     = var.subnet_ids
      efs_file_system_name = var.efs_file_system_name
      security_groups    = [module.SG.efs_sg_id]
      role_arn           = var.role_arn
}
```

Terraform Provisioner

Terraform gives you the ability to run commands on your local or remote system, transfer files, transport data to virtual machines, and do a variety of other tasks. This allows you to perform a variety of operations via the Terraform provisioner to handle such operations.

With the use of a Terraform provisioner, it is possible to provide any material that cannot be moved from one location to another via buildings. Terraform allows for the definition of several provisioners inside a single resource block, and these provisioners will be executed in the order that they are mentioned in the configuration file.

When it comes to connecting to remote servers, Terraform provisioners have the choice of using either SSH or WinRM. When a new instance of a cloud computing system is created, it is immediately able to access data using the various delivery mechanisms given by the majority of cloud platforms. You are free to continue using Terraform provisioners to send data once the resource has been generated successfully.

Terraform State File

The primary responsibility of the Terraform state file is to serve as a repository for the Terraform state, which consists of links between objects located on various systems. These connections are outlined in the configuration files for your Terraform installation. Typically, Terraform state files are stored locally on the machine that is being used to execute Terraform instructions. These files have been given the name `terraform.tfstate`.

JSON is the format that Terraform uses to store its state information. The state file is utilized whenever the `terraform show` or `terraform output` command is executed. The output is also generated in JSON format. A Terraform state file gives you the option to include infrastructure that was created using other methods, such as by hand or using scripts.

When you are working on your own, it is perfectly fine to maintain the Terraform state file on your local workstation. On the other hand, if you are part of a group effort, you may want to consider storing the file in a repository, such as AWS S3 or something similar. The Terraform state file is considered to be in a locked state if anything is written to the resource of the Terraform configuration file. Because of this, it will not be feasible for several users to view the file or make changes to it at the same time.

There are primarily two categories of back ends, the first of which are those that are housed locally, and the second of which are hosted remotely. A local back end is required for Terraform to function properly. This back end might be a computer operating under Linux, a computer operating under Windows, or any other form of computer. For a remote back end, you can use a URL that is based on a SaaS platform or a storage location such as a bucket from Amazon Simple Storage Service.

Example Terraform Configuration

You've learned the basics of Terraform, and now it's time to take a look at a complete example that will give you a better idea of how Terraform's configuration is performed and how it actually works.

Let's consider an example where you want to launch your own automated infrastructure deployment using Terraform and AWS. In this example, you will perform the following steps to achieve desired results:

1. Set up a Terraform configuration.
2. Set up an AWS configuration.
3. Set up an EC2 instance on AWS.
4. Set up an S3 bucket on AWS.

Let's get started. The first step is to set up a Terraform provider, which will facilitate the creation and deployment of new resources on AWS.

```
// Declaring Terraform Provider for AWS
provider "aws" {
  region = "us-east-1"
  profile = "default"
}
```

Now, you need to tell Terraform to create a new SSH key-value pair for your AWS instance. This will be done through a Terraform resource. Here, the key name is defined, and the value of the key is read from the `id_rsa.pub` file.

```
// Declaring Terraform Resource for AWS Key-value pair
ressource "aws_ssh_key_value_pair" "sshkey" {
  key = "awsTerraformDemoProjectKey"
  public_key = ("~/.ssh/id_rsa.pub")
}
```

Next, you now need to create a network security group in AWS to allow network traffic and communicate with the resources within the AWS network. For this, you need to define another Terraform resource that will do the work for you. The `ingress` keyword is used to create an inbound network rule that specifies the port, protocol, and other network configuration details. Similarly, the `egress` keyword is used to create outbound network rules.

```
// Declaring Terraform Resource for AWS Network Security Group
resource "aws_network_security_group" "awssecurity" {
  description = "Allow HTTP Incoming Traffic"
  ingress {
    description = "HTTP Traffic for AWS EC2 web server"
    from_port = 80
    to_port = 80
    protocol = "tcp"
```

```
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "HTTPs Traffic for AWS EC2 web server"
    from_port = 443
    to_port = 443
    protocol = "tcp/tls"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "SSH Traffic for AWS EC2 web server"
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    description = "HTTP Traffic for outgoing from AWS EC2
    web server"
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Once the network security group is created, it's time to create and launch an AWS EC2 instance. The `connection` block will hold the necessary configuration of your EC2 instance. `provisioner` is used to model specific action, in this case installing Git on the EC2 instance you just created.

```
resource "aws_instance" "awsec2" {
  ami = "<ID of the AWS AMI that you want to use>"
  instance_type = "t2.micro"
  key_name = aws_ssh_key_value_pair.sshkey.key
  security_groups = [aws_network_security_group.awssecurity]

  connection {
    type = "ssh"
    user = "<Name of the user accessing the AWS instance>"
    private_key = file("~/aws_key.pem")
    host = aws_instance.awsec2.public_ip
  }

  provisioner "remote-exec" {
    inline = "sudo apt-get install git -y"
  }
}
```

Now that the EC2 instance is created, you need to attach the AWS EBS volume for storage to the newly created EC2 instance. The first resource creates an independent external AWS EBS volume in the specified zone with 25GB of space. Then, the next resource attaches the AWS EBS volume to the EC2 instance.

```
resource "aws_instance" "awsebs" {
  availability_zone = aws_instance.awsebs.availability_zone
  size = 25
}

resource "aws_ebs_volume" "awsebs_storage" {
  volume_id = aws_ebs_volume.awbsebs_storage.id
  instance_id = aws_instance.awsec2.id
  device_name = "/dev/xvdf"
  force_detach = true
}
```

Finally, mount the drive to the AWS EC2 instance. And *voilà*, your AWS instance is ready for use, which was launched completely using IaC and Terraform.

```
resource "aws_ebs_volume" "awsebs_mount" {
  depeneds_on = [
    aws_instance.awsec2,
    aws_ebs_volume.awsebs_storage
  ]

  connection {
    type = "ssh"
    user = "<Username>"
    private_key = file("~/aws_key.pem")
    host = aws_instance.awsec2.public_ip
  }

  provisioner "remote-exec" {
    inline = [
      "sudo mount /dev/xvdf /var/www/html",
      "sudo rm -rf /var/www/html/",
      "sudo touch index.html"
    ]
  }
}
```

Terraform Command-Line Interface

Subcommands such as `terraform init` and `terraform plan` provide access to the Terraform command-line interface (CLI). You may retrieve the Terraform command, which consists of numerous subcommands, by typing `terraform` at the command line.

- `terraform init`: This initializes the provider, module version requirements, and back-end configurations.
- `terraform init -input=true`: You need to provide the inputs on the command line or Terraform will fail.
- `terraform init -lock=false`: This disables locking the Terraform state file; this is not recommended.
- `terraform init -upgrade`: This upgrades the Terraform modules and Terraform plugins.
- `terraform get`: This command downloads and updates the modules mentioned in the root module.
- `terraform plan`: This command determines the state of all resources and compares them with real or existing infrastructure. It uses the Terraform state file data to compare and the provider API to check.
- `terraform plan -compact-warnings`: This provides a summary of warnings.
- `terraform plan -out=path`: This saves the execution plan on the specific directory.
- `terraform plan -var-file= abc.tfvars`: This uses a `terraform.tfvars` file specified in the directory.
- `terraform apply`: This applies the changes in a specific cloud such as AWS or Azure.
- `terraform apply -backup=path`: This backs up the Terraform state file.
- `terraform apply -lock=true`: This locks the state file.

- `terraform apply -state=path`: This prompts you to provide the path to save the state file or use it for later runs.
- `terraform apply -var-file= abc.tfvars`: This enters the specific `terraform.tfvars` file that contains environment-wise variables.
- `terraform apply -auto-approve`: This command will not prompt to approve the `apply` command.
- `terraform destroy`: This will destroy Terraform-managed infrastructure or the existing environment created by Terraform.
- `terraform destroy -auto-approve`: This command will not prompt to approve the `destroy` command.
- `terraform console`: This provides an interactive console to evaluate the expressions such as the `join` command or `split` command.
- `terraform console -state=path`: This is the path to the local state file.
- `terraform fmt`: The `terraform fmt` command formats the configuration files in the proper format.
- `terraform fmt -check`: This checks the input format.
- `terraform fmt – recursive`: This formats the Terraform configuration files stored in subdirectories.
- `terraform fmt – diff`: This displays the difference between the current and previous formats.
- `terraform validates`: This validates the Terraform configuration files.

- `terraform validate -json`: The output is in JSON format.
- `terraform graph`: This command generates a visual representation of the execution plan in graph form.
- `terraform output`: This command extracts the values of an output variable from the state file.
- `terraform state list`: This lists all the resources present in the state file created or imported by Terraform.
- `terraform state list – id=id`: This command will search for a particular resource using the resource ID in the Terraform state file.
- `terraform state list -state=path`: This command will prompt you to provide the path of the state file and then provide the list of all resources in the Terraform state file.
- `terraform state show`: This shows attributes of specific resources.
- `terraform state show -state=path`: This command will prompt you to provide the path and then provide the attributes of specific resources.
- `terraform import`: This command will import existing resources from infrastructure that was not created using Terraform but will be imported in the `terraform state` file and will be included in Terraform the next time we run it.

- `terraform refresh`: This will reconcile the Terraform state file. Whatever resource you created using Terraform, if they are manually or by any means modified, the refresh will sync them in the state file.
- `terraform state rm`: This command will remove the resources from the Terraform state file without actually removing the existing resources.
- `terraform state mv`: This command moves the resources within the Terraform state file from one location to another.
- `terraform state pull`: This command will manually download the Terraform state file from a remote state in your local machine.
- `terraform state push`: This command will manually upload the local state file to the remote state.

Terraform Use Cases

Your organization's IaC implementation will present more and more difficult technical and collaborative obstacles with growth. Resources and service providers may be abstracted using Terraform's flexible abstractions. Terraform's features overlap with a variety of current technologies. There are lots of tools that we compare Terraform to, but it's important to keep in mind that Terraform doesn't exclude the use of other systems. An individual application or the whole datacenter may be managed with this tool. When compared to other technologies, such as Ansible, Terraform competes in day-zero installations unlike those other technologies. When it comes to the construction of infrastructure, however, Terraform has fewer competitors. The following are some of the use cases where Terraform showcases its full potential.

Supporting Platform as a Service

Platform as a service (PaaS) solutions, such as Heroku, simplify the process of developing web applications and integrating them with many other services, such as email and database management. Heroku makes it easy to add more nodes or workers; however, the majority of complex applications need a variety of add-ons and other services. A CNAME can be configured by using a domain provider such as DNSimple, GoDaddy, BlueHost, etc. Similarly, application deployment log capturing, monitoring, and deploying an application's content delivery network (CDN) through services like Cloudflare all can be done without the need for a user-friendly online interface and by using configurable template scripts with Terraform.

Managing Self-Service Clusters

When you are part of a large organization with centralized operations, you are likely to get a large number of requests about the infrastructure that are all the same. It may be possible to empower product teams to self-manage their infrastructure by using a paradigm for infrastructure management that is based on Terraform. Teams will be able to swiftly deploy services that are in compliance with your organization's standards for delivering and managing services if you utilize Terraform modules to define and apply those standards. To facilitate the creation of new infrastructure requests, Terraform Cloud may be able to integrate with ticketing systems like ServiceNow.

Performing Multicloud Deployments

When infrastructure is split over many clouds, the ability to recover from disruptions caused by cloud providers is increased. This allows for a gentler recovery. The fact that every cloud provider has its own system

of tools and processes makes it more difficult to manage environments that make use of many clouds. You may use a single process to handle numerous cloud providers and cross-cloud dependencies when you use Terraform. This method makes it easier to administer and orchestrate large-scale cloud infrastructures by using many cloud providers.

Managing Parallel Environments

Testing new applications in staging or quality assurance environments is common practice in many companies before putting them into production. As the production environment expands and grows more complex, it becomes more difficult to keep a current environment at each step of the development process. Provisioning and decommissioning of infrastructure can be performed rapidly using Terraform, making it ideal for use in development, testing, quality assurance, and production. When compared to continuously maintaining each environment, Terraform is a more cost-effective solution in the long term.

Application Infrastructure Automation

When it comes to multitier applications, Terraform can be used to automate the deployment, release, scaling, and monitoring of the underlying infrastructure. With the help of an n-tier application architecture, you will be able to segment your concerns and develop the distinct parts of your application independently. In addition to a pool of web servers that make use of a database tier, an application may also make use of other tiers such as API servers, cache servers, and routing meshes. These are some examples of extra tiers that may be included in the application. Each layer of resources that may be managed using Terraform can be maintained as a unit, and Terraform will automatically manage the

dependencies that exist between the different layers. The deployment of a database layer comes first, followed by the possible provisioning of web servers using Terraform, as shown in Figure 6-1.

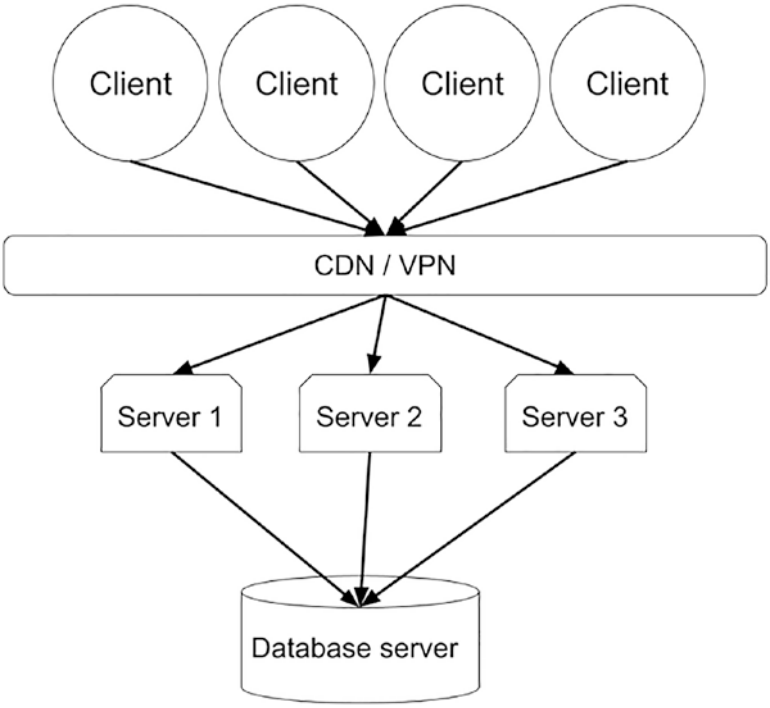

Figure 6-1. *N-tier application architecture*

Managing Software-Defined Networks

Terraform is able to connect with software-defined networks, which are APIs or software-based controllers used to handle traffic management and establish a connection to the underlying infrastructure, enabling it to autonomously create a network such that it satisfies the needs of the applications that are running on it (SDNs). This enables you to go from a deployment approach that is manual to one that is automated. When a service registers with multilayer networking tools for microservices and

cloud infrastructure to solve networking and security challenges, tools such as HashiCorp Consul and Consul-Terraform-Sync may automatically produce Terraform configurations to expose relevant ports and adjust network settings for any SDN with a linked Terraform provider. Network functions virtualization (NFV) can be used by network infrastructure automation (NIA) to partition a virtual network. This paves the way for telecoms to shift client services to less expensive server locations, even the customer's own on-premises servers.

Policy Compliance

Terraform can help enforce restrictions around resource consumption by various teams. Whenever you use a system that is dependent on tickets, there is always the chance of experiencing delays. You can make use of Sentinel policies, a paid feature provided by Terraform to enforce compliance and governance policies and processes before allowing Terraform to make any modifications to the infrastructure. Access to Sentinel may be granted to members of Terraform Cloud and the governance layer if they meet certain requirements.

After building your configuration files, you may analyze and apply infrastructure modifications using an execution plan. Once improvements are validated, the infrastructure is deployed per plan.

Continuous integration includes compliance testing to meet user-defined requirements. Delegating responsibility for remote resources using geographical naming standards is one way. Second, images are used to create virtual machines. In these and other situations, compliance testing would enforce regulations.

With IaC, you can adapt better development practices, such as submitting pull requests for other teams to review. Validating IaC has been helped by testing frameworks and IaC tools that leverage common

language programming. However, they are better suited for technical teams, and converting this to nontechnical languages may be difficult, restricting the options.

Some companies prioritize security and regulatory compliance. An application deployment pipeline verifies a workload's criteria before deployment. Here, the shift-left testing approach finds issues early in the release cycle to avoid vulnerabilities and compliance violations. Standard permissions are provided to user roles or accounts during application deployment. These administrative and reporting accounts include no customer or personal information. The following are some typical practices employed by organizations to maintain compliant infrastructures:

- If you're unsure about IT management compliances, consult an attorney or legal counsel. Understanding the legal ramifications of internal systems, down to the OS version, is difficult. Because of greater responsibility, legal departments actively advise CTOs and CIOs.
- Legal counsel will tell IT managers to write rules and procedures. Infrastructure deployment and operations management must be included in rule and process formulation for secure application development. Tasks such as providing administrative powers should be done according to policy, not habit.
- Sarbanes-Oxley compliance requires mapping processes to COBIT. Control Objectives for Information and Related Technology (COBIT) is an open standard for IT security and control operations. The Treadway Commission's Enterprise Risk Management Framework inspired COBIT. COBIT helps organizations manage compliant IT systems.

- Automate compliance monitoring. In addition to the publicly accessible COBIT frameworks and standards, third-party tools for auditing and analyzing Coso compliance levels may save time. Check for IT automation possibilities.
- Prepare to protect your data. Proactive IT organizations will be less prone to compliance-threatening security threats that need large resources to remedy. Updated virus definitions and patch management are examples.

The Way Ahead

IaC requires more careful consideration of isolation, locking, and state than conventional coding because of the various trade-offs involved. When developing a standard app, most bugs are quite minor and impact only a limited subset of the code. When issues arise in a code-managed infrastructure, they may have far-reaching effects on the availability of your apps, data, and network.

The utilization of IaC is necessary to ensure its security and dependability. Just like the value of any other kind of code, the worth of the code that represents the infrastructure relies on how effectively it is constructed and maintained. For your engineers to do as many infrastructure modifications as feasible using Terraform, you should include the aforementioned approaches. This will encourage continuous code development while decreasing the temptation to employ out-of-band approaches and the drift they introduce.

Examining the benefits and drawbacks of using Terraform at your organization can help you anticipate potential outcomes. HashiCorp provides excellent tools for managing infrastructure. Each of these options aids in the administration and protection of your infrastructure, whether

it is in a physical data center or in the cloud. Terraform allows you to construct it either interactively or programmatically via an API on several public and private cloud platforms.

Structure your infrastructure as code into modules, and you may apply many different good coding practices to it. You may securely test many versions of a module in various contexts, reverting to prior versions if a problem develops, and validate each update to a module using code reviews and automated tests. Instead of spending time manually deploying code, you may be working on modules instead.

With these characteristics, you may have an easier time putting up a solid framework rapidly. Complex infrastructure components may be obscured by using simple APIs that are applicable throughout your technological stack.

In the next chapter, you will learn about another IaC automation and delivery tool named Puppet, why businesses should consider it, details of the architecture, how Puppet works, and how you can implement Puppet in your projects.

CHAPTER 7

Hands-on Infrastructure as Code with Puppet

Scalability has become an essential necessity for modern businesses. In the field of information technology, conventional on-premises systems are losing ground to more modern alternatives that make use of virtual and distributed resources. Both the development and operations teams have had to adjust their operational practices as a result of the introduction of DevOps.

The process of ensuring that computers, servers, and software continue to be in their intended states during the course of development is referred to as *configuration management*. Even if modifications are made to a system in the future, it may be protected in this manner so that it will continue to operate as expected.

When building and maintaining information technology systems, one must have a clear idea of what the systems are meant to look like after they are completed. Configuration evaluations and drift analysis are used to determine whether systems need upgrading, patching, or reconfiguring.

Utilizing configuration management allows for the prevention of unauthorized modifications of any kind and of any size. Incorrect settings may be harmful to both the operations and security of a company

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_7

since they can cause poor performance, inconsistent behavior, or noncompliance. When unauthorized changes are made to a large number of systems and applications, there is an increase in both instability and downtime.

In big environments, it is impossible to identify manually the systems that need care, choose repair solutions, prioritize tasks, and evaluate whether they have been completed. Unless there is sufficient documentation, regular maintenance, and a change control mechanism in place, system administrators and software developers will not be able to determine what is stored on a server or which programs have been updated.

Solutions for configuration management make it possible to establish system settings in a consistent manner and also facilitate the creation and maintenance of systems that are based on these baseline settings. Users and administrators alike are granted the ability to ascertain the locations of certain services and the current state of applications thanks to configuration management.

Tools like Puppet by Perforce provide you with the ability to implement configuration management for your infrastructure. In this chapter, you will learn about Puppet in detail and how to use it in different ways.

Introduction to Puppet

Puppet is a simple modeling language that is used in the process of developing programs for automating the administration of infrastructure. You are able to specify the final state of your systems, also known as *nodes*, with relative simplicity thanks to Puppet. On the other hand, procedural scripts need extensive description of the actions that must be taken in order to get a computer system to a certain state. If you use Puppet, you won't have to worry about scripting mistakes or misaligned phases that might lead to an unwanted outcome. This is because Puppet automates the process. Puppet will handle all of the stuff for you automatically.

The Puppet programming language may be used on any platform, in contrast to procedural scripts. You are able to focus on the key parts of the system thanks to Puppet's ability to handle implementation challenges such as command names, arguments, and file formats. This gives you the ability to focus on the system. Puppet allows you to manage all of your users in the same way, regardless of the location of those users, so long as you have it installed.

Without the abstraction concept, Puppet would not be able to work. Because of this, everyone who is literate and has the ability to write may take control of the systems at the level that is required for their profession. It is possible that teams will be able to operate more effectively and that people will be able to manage resources that would otherwise be beyond their control if a culture of shared responsibility is fostered within teams.

The capability of the modeling language to be simply copied is yet another advantage offered by it. In contrast to scripts, which can be performed only once, a Puppet run may be repeated several times without affecting its results. If the system is already in the state that Puppet deems proper, it will be left alone by Puppet. Idempotency, in a nutshell, is the characteristic that guarantees that an operation's outcomes are the same, even if the same function is applied more than once after the original application. When performed once, twice, or a thousand times, an idempotent operation yields the identical outcomes each time.

The resource declaration serves as the foundation upon which the Puppet programming language is constructed. The name of a resource may be used to determine whether something is an operational service or an installation-required package.

Think of the resources you manage as building parts that may be assembled to represent the intended state. This can help you comprehend the systems you are responsible for managing much better. Puppet's capacity to mix components in a way that is both rational and effective is one of the most significant benefits of using this tool.

Why Choose Puppet?

A Puppet module is a self-contained collection of key components. This can include classes, manifests, data, and other elements. We can define specific resources with Puppet modules. Each Puppet module ultimately helps us recycle Puppet code and keep groups of code organized over time. A manifest is made up of a Puppet code class or declared type (written in the Puppet DSL). Knowing the purpose is priority number one when writing a module.

A Puppet environment can also accommodate a large number of modules, perhaps hundreds if necessary. A set of particular named classes dedicated to specific tasks serve as programs that must be constructed. These programs then generate manifests.

Puppet describes everything as data: the node's current state, the intended end state, and the activities required to transition from one to the other. Each Puppet-managed server instance gets a catalog of resources and relationships, compares it to the intended system state, and makes modifications as needed to bring the system into compliance with the ideal state.

Puppet encourages its users to keep complexity under control by developing reusable, easy-to-configure, and refactorable code. The basic way to do this is to use Puppet roles and profiles, which divide your code into three levels.

Component modules: These are standard modules that govern a single technology, such as `puppetlabs/apache`.

Profiles: These are wrapper classes that configure a tiered technology stack by utilizing various component modules. Create a profile, for example, to configure Jenkins, the continuous integration program, with its online front end and automated tasks.

Roles: These are wrapper classes that combine various profiles to provide a comprehensive system setup. In addition to the Jenkins job, the server must have conventional roles.

Above all, it allows you to create useful, business-specific interfaces for system setups. This facilitates the usage of hierarchical data, the reading of system settings, and the restructuring of code.

Understanding Puppet

To guarantee that Puppet code is deployed throughout the infrastructure, that it is updated on a regular basis with configuration changes, that unwanted modifications are fixed, and that system audits are carried out, there are a number of procedures that need to be followed. To fulfill these requirements, the vast majority of teams and Puppet users adopt a master-agent architecture that consists of multiple different components. Customers may make use of one or more Puppet masters, depending on the requirements that they have. Each node has an agent that, while communicating with the master, uses a connection that is both signed and encrypted, as shown in Figure 7-1.

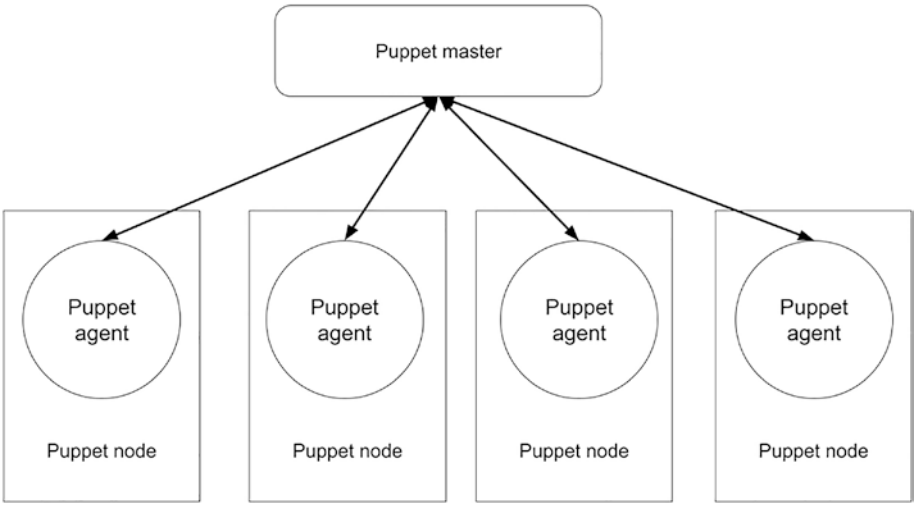

Figure 7-1. *Puppet master-node communication*

Puppet programs may be delivered to servers by using the master-agent structure, and the configuration of the servers can be managed over an extended period of time. Puppet first generates a catalog of manifests before proceeding to configure a node. Records that do not change, such as catalogs, provide insight into the connections between different resources. Depending on the task at hand and the context that is present, every catalog has the potential to be applicable to a single node. Puppet's role is to use the catalog to check if a node's configuration has been correctly applied, and it is also Puppet's responsibility to make any necessary changes to the node's configuration.

As part of the standard Puppet workflow, a node-based agent will often connect with a master server. This is considered to be best practice. After that, Puppet might do any of the following actions:

- If there has been any change, it is required to make the appropriate modifications.

- It is recommended that none of the nodes has any changes made to them.
- If there are any necessary changes to be made to your setup, you can use the orchestration tools that are provided by Puppet.
- The information collected from the events and nodes in the network can be saved and accessed at a later time.
- Since Puppet is a pull-based configuration management tool, the node-based agents will pull the configuration from the master server.
- Node-based agents will pull the configuration at regular intervals and not in real time.
- The information and configuration fetched would be compared with the current configuration at each node.
- If any mismatches arise, agents would take steps and perform modifications to match the configuration of the node with the configuration of the master server.
- It is easier to spawn new nodes with this approach. The new node is configured automatically by retrieving information from the main server.
- Since the nodes are independent of each other, configuration management and scalability are performed with ease.

Architecture

Puppet was designed with two modes in mind: client-server with a central server and agents operating on various hosts, or serverless with a single process doing all of the work. Puppet has always maintained network

transparency internally to provide consistency between both modes; thus, the two modes used the same code paths whether they flowed over the network or not. Each executable can set up local or remote service access as needed, but they otherwise act the same. It's also worth noting that you may utilize the serverless mode in a client-server setup by sending all configuration files to each client and having it interpret them directly.

Puppet Master

A Puppet master is the fundamental mechanism in charge of all configuration-related tasks. The Puppet master stores all configuration information, which is accessed by all Puppet agents when they request their configuration catalogs.

Puppet Agents

Puppet agents are the real operating computers that the Puppet master manages. The Puppet agent daemon service is operating within them.

Configuration Repository

This is the repository where all node and server configurations are kept and retrieved as needed. This is how master-agent communication works:

1. The Puppet agent process gathers information about the host on which it is executing and sends it to the server.
2. The parser compiles a configuration for that specific host using the system information and Puppet modules on local storage and provides it to the agent.

3. The agent then implements that setting locally, altering the host's local state, and files the resulting report with the server.

Puppet employs a variety of data formats for internal communication, in addition to the components that comprise this process, which we'll discuss next. These data types are crucial since they are the foundation of all communication and are public kinds that any other tool may consume or make.

Facts

Puppet uses the *Facter* tool to acquire system information before building a node and requesting a catalog for a controlled node. The information is stored as variables, and those variables may be referenced later in manifests. Furthermore, Puppet creates a set of supplemental variables known as *built-in variables* that operate similar to facts. Once the facts are available, Puppet makes changes to any target node based on the facts in the manifest. Puppet holds both predefined and custom facts.

Catalog

All Puppet manifest files or configurations are first translated to a compiled format called *catalog*, and then those catalogs are deployed to the target computer. One of the defining decisions of Puppet's application architecture is that clients should not have access to raw Puppet modules; instead, they should have access to a configuration that has been created specifically for them. This has several advantages: to begin, you adhere to the concept of least privilege, which states that each host knows only what it needs to know (how it should be configured), but it has no knowledge of how any other servers are configured.

Configuration

Puppet is used to create a representation of your infrastructure in code. Puppet and your infrastructure can be managed in the same way that you would manage any other piece of code. Code written with Puppet may be easily saved and recycled for use in other projects. It's possible that members of other teams will also be allowed to use the workplace machines. It is possible for development and operations to manage systems by utilizing manifests that are the same from the workplace machine development environment to the production environment. This will help prevent unpleasant shocks when code is transferred to production. This can potentially have a significant effect on the quality of deployments, particularly in the case of organizations that specialize in a certain field.

When system administrators treat settings as code, developers are allowed to construct their own testing environments, which helps remove the barrier that system administrators seem to represent to programmers. As a result of the widespread acceptance of Puppet manifests by auditors, Puppet code will be seen as sufficient evidence of compliance. People are able to work more effectively as a result of all of this, which also helps to lower their levels of worry.

The ability to verify Puppet code by using a conventional version control system is almost certainly the feature that is considered to be the most significant. The history of the infrastructure may then be recorded in a manner that adheres to the regulations. You may be able to enhance your configuration authoring, updating, and testing with the assistance of the colleagues of a software engineer until you feel comfortable putting them into production.

You may also try changes by using the "no-op" or "simulation" modes that are available in Puppet before actually implementing them in the real world. Because of the capability to turn back, deployments are far less stressful.

Module Structure

The majority of the code that you write for Puppet will be stored in its modules. There are several modules that are responsible for managing different components of the infrastructure, such as installing and configuring software.

It is possible to develop modules with custom data as well as customized data formats. Plug-ins are what are responsible for achieving this result. The modules need to be included on the module path in order for Puppet to work properly. Before Puppet can use this code, all of the modules that are listed on the module path must first be loaded. This is the reason why the code is valid.

Puppet Forge provides users with the ability to download and install modules. Puppet Forge is home to the hundreds of modules that have been developed for a wide range of applications by open source contributors and Puppet developers. You need to make sure that your budget allows for the creation of at least a few custom modules so that your infrastructure can meet your individual needs.

Modules are under the management of a Puppetfile whenever Code Manager[1] or r10k[2] is used, both of which are code management tools that manage environment configurations and the deployment of the code base from the code repository for you. Installing and managing modules in proof-of-concept projects or in very small infrastructures that are managed manually may be accomplished with the use of the `puppet module` command.

Puppet is able to identify and load classes, declared types, facts, custom types and providers, functions, and tasks because modules have a certain directory structure. This structure enables Puppet to manage custom data. Every module subdirectory serves a unique function and may be skipped if desired.

[1] https://puppet.com/docs/pe/2021.1/code_mgr.html
[2] https://puppet.com/docs/pe/2021.1/r10k.html

Security Mechanisms

The fundamental functionality of Puppet is based on a number of different encryption and communication protocols. Puppet and OpenSSL (a toolkit that is based on SSL and TLS) have a close connection with one another. The SSL/TLS technology as well as SSL certificates are used in the verification and authentication of masters and agents. Encrypting the communications that take place between agents and servers using SSL/TLS is another useful feature. You now have a better understanding of the many applications of encryption technology thanks to this introduction to Puppet.

- The identity of any agent in connection to the master has to be verified before it can be used.
- Confirmation of the identity of the master agent.
- The confidentiality of conversation between the master and the agents needs to be maintained.

For the purpose of providing mutual host authentication, Puppet makes use of a TLS client-side X-509 certificate. This information is supposed to be kept in the `puppet.conf` configuration file, which is located in the `/etc/Puppet/ssl` directory. It is possible to change the default location by using SMF commands, and any changes made will be reflected in the site configuration file.

In addition to requests that need to be signed, there are also directories for certificates, requests that have already been signed, and keys. These directors are present on both the master server and the agent servers. As part of this introduction, you should be familiar with the Puppet certification authority that is owned and operated by Puppet.

After the establishment of a certificate revocation list (CRL) and the creation of the server certificate, a custom CA certificate and private key are generated for the master. This is followed by the production of the server certificate. The agent will get a server certificate to facilitate SSL and TLS communications.

How Puppet Works

In addition to its functional components, the development of Puppet has been guided by two guiding principles.

- It should be as simple as is practically possible, with usability always taking precedence over capability.
- It should be built as a framework before it is built as an application so that development teams can utilize the framework to launch their own applications.

These principles are intended to allow developers to build their own applications based on Puppet's internals in different manners as required. Let's further understand how the internals of Puppet work and how Puppet configuration is performed for various scenarios, including the nuances of infrastructure, core elements such as plugins, and a control framework.

Puppet Infrastructure

Puppet is based on the notion of infrastructure as code, or treating infrastructure as if it were code. This idea underpins DevOps, the methodologies and set of practices that merge software development and operations. Treating infrastructure like code enables system administrators to adopt software developer–specific procedures such as version control, peer review, automated testing, and continuous delivery.

Puppet architecture is a master-server type of architecture, which consists of a primary node to manage information related to configuration for all the other agent nodes, which can be just one or more than one node. Secure protocol HTTPS is used by servers and agents to interact via SSL certificates. For managing certificates, Puppet also has a certificate authority provided in the framework itself. The Puppet server serves as the principal node and runs an agent to configure itself.

Plugins

One of the best aspects about Puppet is its extensibility. Puppet has at least different types of extensibility, the majority of which are intended to be used by everyone. You can, for example, develop custom plugins for the following areas:

- Custom providers and resource types
- Report handlers, such as those used to save reports to a custom database
- Plugins for interfacing with existing data storage in Indirector
- Information for learning more about your hosts

However, because Puppet is distributed, agents require a method to retrieve and load new plugins. As a result, the first thing we do at the start of every Puppet run is download all plugins that the server has accessible. New resource kinds or providers, new facts, or even new report processors might be among them.

This allows for significant upgrades to Puppet agents without ever modifying the core Puppet packages. This is particularly helpful for heavily customized Puppet systems.

Indirector

The Indirector is an inversion of control structure that is quite adaptable. Through inverting control systems, it is possible to decouple the development of features from the technique by which they are deployed. In the case of Puppet, this means we may swap between plugins with wildly different capabilities, such as communicating with the compiler via HTTP or loading it in-process, simply modifying the configuration rather than the code itself. That is to say, Puppet's Indirector is only a service locator that may be used in production.

Puppet's Indirector is essentially an implementation of a service locator. All hand-offs between classes are handled by the Indirector via a standard REST-like interface (e.g., we support find, search, save, and destroy as methods), and switching Puppet from serverless to client-server is largely a matter of configuring the agent to use an HTTP endpoint for catalog retrieval rather than a compiler endpoint.

This class can be difficult to grasp since it is an inversion of control framework in which configuration is strictly isolated from code paths. This is especially true when debugging why a specific code route was selected.

Implementing Puppet in Real Projects

You gained knowledge about Puppet files as well as the directory structure of Puppet in the previous section. Moving forward, it is essential to have a solid understanding of how Puppet configuration is applied with the help of real examples. The examples will showcase how to set up your Puppet configuration from scratch.

The central repository of an all-inclusive configuration management solution may make the deployment of any system simple and dependable. This repository explains and keeps a record of the operations that are

carried out. System upgrades may be performed easily and fast whenever they are necessary. As a consequence of this, a single administrator might be responsible for managing a large number of servers.

You can generate files known as *manifests* that have a `.pp` extension by using the language that Puppet provides. To describe what a machine, whether actual or virtual, needs to perform, either you can utilize the modules that come pre-installed with Puppet or you can create your own unique modules and put them in manifest files. In the same manner that code is executed, Puppet is driven by a collection of manifests, much like a computer. Using the `puppet apply` command, Puppet will first construct your program check to see how much it deviates from the condition you expected it to be in and then make the necessary adjustments to bring your computer up-to-date. If there are no modifications that need to be done, executing `puppet apply` on a system that has the most recent manifest should maintain its idempotency and result in no changes being applied.

Getting Started with Puppet

In this particular scenario, you will be responsible for organizing the workspaces of software engineers. `puppet-master` is the hostname of a desktop PC running Ubuntu 18.04, although the software should be compatible with any distribution of Linux. If you utilize GitHub for this reason, be sure that passwords and keys are kept separate from the platform.

Figure 7-2 shows the several ways that Puppet may be installed on the target system. During the course of the installation process, just the Puppet Labs repository, Git, and Puppet will be set up.

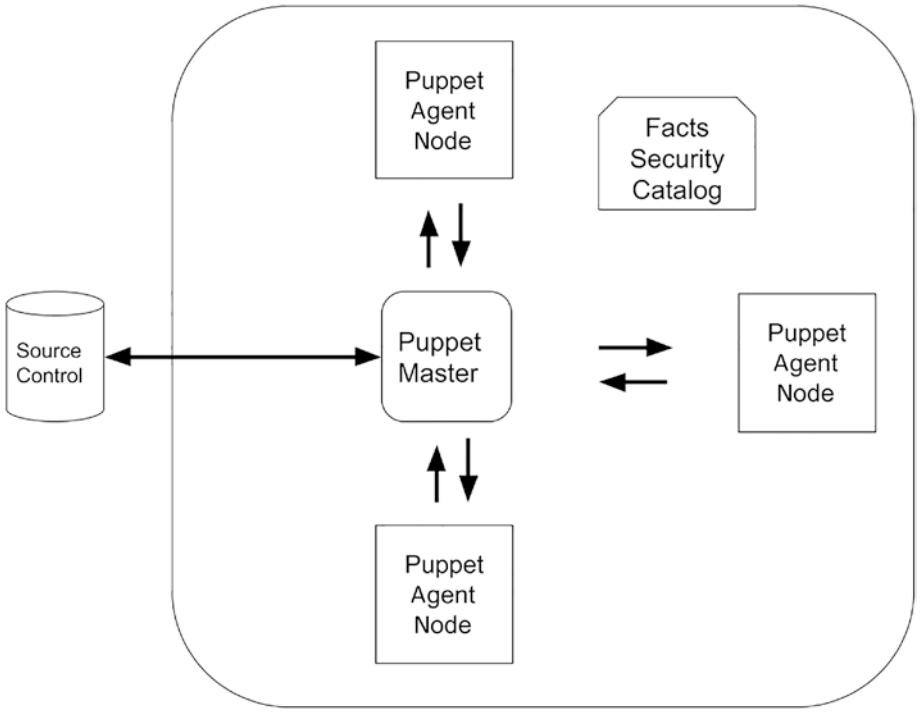

Figure 7-2. *Puppet interaction*

The precise versions of `puppet-common` and the `puppetlabs/apt` modules utilized are as follows:

```
wget https://apt.puppet.com/puppet-release-bionic.deb
dpkg -i puppet-release-bionic.deb
apt-get update
apt-get install -y man git puppet-common
apt-get install puppetserver
puppet module install puppetlabs/apt
```

Next, start the service on your machine with the following commands:

```
systemctl start puppetserver
systemctl enable puppetserver
```

By default, port 8140 is used by the Puppet master server for communication with the nodes. Once Puppet is installed on your machine, let's perform some tasks using Puppet.

Preparing the Repository

The next step is to configure a Git repository to store your edited Puppet manifest files. In your Git repository, create a file named `puppet-master.pp` inside `manifests` with the contents shown in the following code:

```
include apt
node 'puppet-master' {
    package { 'aws':
        ensure => 'present'
    }
    package { 'oracle':
        ensure => 'absent'
    }
}
```

The contents described in the file denote the following things:

- Machine's hostname matches the file name in this case, which is not mandatory; however, it helps you organize your manifest files within the directory.
- The `apt` package is imported, which allows you to manipulate installed software on your machine.
- `node` is a root-level block, which allows you to define the state of servers.
- `puppet-master` is the server node here.

- The manifest states that there should be an `aws` package installed and no the `oracle` package on the specified machine.

This Puppet configuration is now ready to be used on the system itself. The instructions listed next may be executed with your own `gitserver` if you `ssh` into the system.

Running the Repository

The following code performs two distinct actions: first, it retrieves a customized manifest file from the Git repository, and second, it applies that file on the machine. You might want to try changing the state of the machine by using the `puppet apply` command to see if you can get it to match the values that are shown in a node that has the same name. The only choices available to us here are to delete `oracle` and set up `aws`, if it isn't already set up.

```
git clone git@gitserver:yourpuppet-gitrepo.git
 ↪/etc/puppet/puppetdemoproject
puppet apply /etc/puppet/puppetdemoproject/manifests
 ↪--modulepath=/etc/puppet/puppetdemoproject/
↪modules/:/etc/puppet/modules/
```

Setting Up Users

Moving ahead, you will need to establish a `user`. An updated version of the `puppet-master.pp` code is included here, which makes use of the `user` variable to create the user:

```
include apt
node 'puppet-master' {
    $user = 'puppetuser'
```

```
    package { 'aws':
        ensure => 'present'
    }

    package { 'oracle':
        ensure => 'absent'
    }

    user { "$user":
        ensure => present,
        comment => "Developer $user",
        shell => '/bin/bash',
        managehome => true,
    }
}
```

Now, in the host machine, apply new changes by pulling the changes from the Git repository and rerunning `puppet apply`, as shown here:

```
cd /etc/puppet/puppetdemoproject
git pull
puppet apply /etc/puppet/puppetdemoproject/manifests
 ↪--modulepath=/etc/puppet/puppetdemoproject/
↪modules/:/etc/puppet/modules/
```

Creating Modules

Now, you will create a new module that is triggered by the node that you have defined earlier, which accepts a parameter to define the user. The module file `init.pp` is stored in the Git repository under the directory `modules/master_host/manifests/`.

```
class user_accounts {
  user { "${username}":
```

```
      ensure => present,
      home => '/home/${username}',
      shell => '/bin/bash',
      managehome => true,
      password => 'password_hash'
  }
}
```

Once prepared, you can use the module in your node. The updated manifest `puppet-master.pp` looks like this:

```
node 'puppet-master' {
    package { 'aws':
        ensure => 'present'
    }

    package { 'oracle':
        ensure => 'absent'
    }

    class { 'master_host': user => 'masteruser' }
}
```

Dynamic File Generation

Files containing dynamic data must be generated often. `.erb` templates, which are comparable to `.jsp` and `.php` files, may include bits of Ruby code. Using a slightly different syntax, every variable in Puppet may be accessed by code.

The name of the build root is stored in the `$HOME/PuppetProjects/props.env` file. Typically, only the user's own files and folders are included in this part. Setting this up with Puppet is as simple as using an `.erb`

template, as demonstrated next. All that differs from an ordinary `erb template file` is the need for a template folder and `@` prefixes in the names of variables.

```
# Managed by Puppet
dir.home=/home/<%= @user %>/
```

Before you can use the `props.env` file, you need to first ensure that there is a `PuppetProjects` directory in the location where it will be stored. Utilize the => operator to guarantee that the directory and file are produced in the specified sequence. The `.erb` template can be used as described here:

```
file { "/home/$user/PuppetProjects":
    ensure => 'directory',
    owner => "$user",
    group => "$user",
    require => [ User["$user"] ]
}

->

file { "/home/$user/PuppetProjects/props.env":
  content => template('master_host/props.env.erb'),
  owner => "$user",
  group => "$user",
}
```

Every time you want to make a change to more than a few systems, you need to run Puppet. This is time-consuming. Run `puppet apply` on each machine to check for changes in the source code. Installing the required functionality is as simple as naming the file `puppetautomation.sh` and scheduling a recurring run on a cron schedule, as shown in the following code:

```
class apply_puppet_automation () {
    file { "/usr/local/bin/puppetautomation.sh":
    source => "puppet:///modules/apply_puppet_automation/
    puppetautomation.sh",
    mode  => 'u=wrx,g=r,o=r'
    }
    ->
    cron { "run-puppetApply":
        ensure => 'present',
        command => "/usr/local/bin/puppetautomation.sh >
         ↪/tmp/puppetautomation.log 2>&1",
        minute => '*/60',
    }
}
```

- Create a new `puppetautomation.sh` template in `modules/apply_puppet_automation/files/puppetautomation.sh`.
- Create the `puppetautomation.sh` file and set up the crontab entry.
- Use your `apply_puppet_automation` module from your node in `puppet-master.pp`.

```
class { 'apply_puppet_automation': ; }
```

Modifying Configurations

When working with Puppet, it is essential to keep in mind that removing a rule does not result in the resource being removed entirely. Create a rule that, when applied to `developer1`, will result in a valid SSH key being generated. It is a regulation that will be adhered to, and it states that the key must be destroyed after `developer1` has left the organization.

This does not remove the item that represents unauthorized keys from the system. The state that is represented by the Puppet resources may not be the ultimate state, since alterations may be brought in from the outside. It is difficult to determine if the rule for `developer1`'s key was manually made or whether Puppet should delete it after the rule because `developer1`'s key has been withdrawn. Either way, it must be determined whether Puppet should delete the rule.

You have the ability to get rid of packages, files, directories, users, and other stuff by making use of the `ensure => 'absence'` rule. This approach was used to uninstall the `oracle` package shown earlier. You need to make certain that `oracle` is not running in the background to assure that it does not exist.

There are occasions when we need to change the SSH key associated with a developer or administrator who is leaving the company. This has an effect on each and every one of the developer's products.

Managing Repositories

Leaving all of your code only on the main branch is not a recommended practice while managing your Puppet infrastructure. You will need to make use of a specialized Git Flow model to administer our repository. The majority of the branches on each server may be followed back to the master. Only a few of the developers are now at the forefront of the development department's work. To prepare each computer for the launch of a new version, we first migrate its branch from the previous version to the most recent one. By maintaining separate branches for each server, you may stop changes from being sent in a haphazard manner to multiple servers as visible in Figure 7-3.

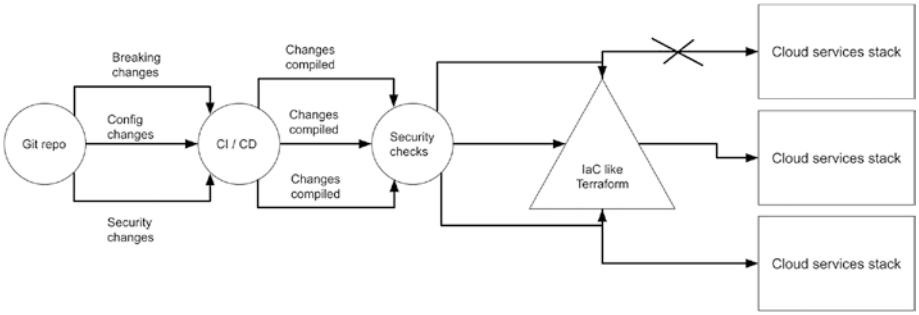

Figure 7-3. *Managing deployments*

Keeping tags on all of your branches and pushing them to new releases requires the use of scripts, which are also necessary. Roughly 100 personal computers may fit inside of it at one time. On a larger scale, it is often impossible to maintain separate branches for each individual server.

It is possible that using a single Git repository will become challenging when the number of computers in your company increases. In our opinion, the most significant problem is the commit noise that occurs between reusable modules and machine-specific parameters. Consider for a moment the possibility that you do not want every machine manifest to be sent to every machine administrator in your organization. This is just one possibility. To do this, you should make use of three different repositories: one for general modules, one for settings that are machine or client particular, and one for global data sets. Because of this, we are able to rapidly release any required global adjustments while still exercising the right amount of release control over our core modules.

The Puppet code and data should be tracked, updated, and deployed via version control. Ideally you should keep your repository in a Git-based version control model. When you make changes to the code and data in your repository, code management ensures that all of your environments are brought up-to-date immediately.

Environments are constructed and maintained by code management based on the branches in your source code repository. Ideally, your code management will generate three distinct environments, each with its own copy of your Puppet code and data, if your control repository has three distinct branches: production, development, and testing. Making environments requires access to the `/etc/puppetlabs/code/environments` folder on the primary server.

Puppet Command-Line Interface

Here are some important commands:

- `puppet-agent`: The client configuration is obtained from the Puppet master and applied to the local host. This service may be started as a daemon, on a regular basis with cron or interactively for testing.
- `puppet-apply`: Use this stand-alone Puppet execution tool to apply specific manifests. This can be used to generate a serverless Puppet site with scheduling an automated mechanism for providing manifests.
- `puppet-lookup`: This command requires access to your hierarchical data and runs it on a server node that has a replication of the same data. This generally entails logging onto a Puppet Server node and executing `sudo puppet lookup`.
- `puppet-module`: This subcommand is capable of locating, installing, and managing modules from the Puppet Forge, a repository of user-contributed Puppet code. It can also generate empty modules and prepare locally generated modules for Forge deployment.

- `puppet-resource`: This command offers basic capabilities for translating current system state into Puppet code, as well as some ability to edit the current state using Puppet's RAL.
- `puppet-config`: This subcommand can view and alter parameters in Puppet's configuration file, `puppet.conf`.
- `puppet-describe`: This printout contains information on Puppet resource types, providers, and metaparameters.
- `puppet-device`: Catalogs are retrieved from the Puppet master and applied to remote devices.
- `puppet-doc`: This is primarily intended for internal usage and is used to produce the reference document that can be seen on the Puppet website.
- `puppet-epp`: Interact with the EPP template parser/renderer directly.
- `puppet-generate`: Using Puppet code, this generates definitions for custom resource types. Puppet code types can be used to separate custom type definitions between environments.
- `puppet-node`: This subcommand interacts with node objects, which Puppet uses to create a catalog. A node object is made up of the facts, environment, node parameters, and classes of the node.
- `puppet-parser`: This operation checks the syntax of the Puppet DSL without creating a catalog or synchronizing any resources. If no manifest files are given, the default site manifest will be validated.

- `puppet-plugin`: The puppet master delivers Ruby code from its modules' `lib` folders. These plugins may be used to expand Facter and add custom types and providers on agent nodes. Plugins are typically downloaded by the Puppet agent during a run.
- `puppet-script`: This is a stand-alone Puppet script runner tool that can be used to run Puppet code without the need to compile a catalog. The Puppet script can load functions, types, tasks, and plans from modules when given a module path through the command line or config file.
- `puppet-ssl`: Manage SSL keys and certificates for SSL clients that require communication with the Puppet infrastructure.

Here are some niche subcommands:

- `puppet-catalog`: Catalogs are compiled per-node artifacts created from a collection of Puppet manifests, and this subcommand interacts with them.
- `puppet-facts`: This subcommand controls facts, which are sets of normalized system data that Puppet uses. It can read data from the local system directly.
- `puppet-filebucket`: This is a filebucket client that may be used to transfer files to a local or central filebucket.
- `puppet-key`: This subcommand is in charge of managing certificate private keys. Keys are produced automatically by the Puppet agent, and when certificate requests are made using `puppet ssl submit request`, this subcommand should not be used directly.

The Way Ahead

When it comes to advanced levels of continuous collaboration and delivery, many of the businesses still have a long way to go. The fact that Puppet can adapt to your expanding team and evolving infrastructure is a major advantage. Banking and government are two areas where Puppet is widely utilized because of their strict adherence to established norms and procedures. Businesses that don't need a high rate of continuous delivery may nevertheless benefit greatly from treating their infrastructure like code by storing and versioning it.

Puppet is useful for DevOps because it automates processes and facilitates the deployment of software more quickly without compromising on security or dependability. Create security policies and verify their legality using this handy tool. Because of this, erroneous settings and audit failures are less likely to occur. Since it considers infrastructure as code and continually adds code, Puppet makes deployments quicker and more reliable. Combining and simplifying many technologies into a unified interface for configuration saves time and effort. Puppet utilizes containers and promotes inspecting their contents closely. To have a deeper understanding of a product's use, it is possible to monitor any functional variations using the given tools.

Among Puppet's numerous applications are configuration management, test-driven software development, and the automation of repetitive chores. It is simple to adopt and incorporate the product into the software development cycle because of the product's intuitive design, security, and dependability. Less time spent on nonessentials means more time spent on revenue-generating activities such as expanding your business's product line and improving existing ones.

In the next chapter, you will learn about another IaC automation and delivery tool named Chef, why businesses should consider it, details of the infrastructure, how Chef works, and how you can implement Chef in your projects.

CHAPTER 8

Introduction to Infrastructure as Code with Chef

An infrastructure as code (IaC) tool must be chosen with care. Each IaC tool and platform has been developed to accomplish a specific goal. Over time, most IaC systems have improved to the point that they can meet the vast majority of IaC requirements and support a broad range of use cases. One of the many advantages an IaC tool brings is that depending on your needs and the expertise of your present personnel, you may also be able to develop and deploy your own custom providers and modules.

One of the most important new trends in software engineering is the DevOps movement. DevOps experts have access to a wide variety of helpful technologies. Chef is an open-source, sophisticated IaC framework developed by Opscode. Chef uses the Ruby programming language to create key building components such as recipes and cookbooks. Automating infrastructure using Chef helps save time spent on mundane, manual tasks in IT operations.

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0_8

Introduction to Chef

DevOps engineers and system administrators face a number of challenges when attempting to roll out new services and applications, distribute machines, and successfully maintain and update network packages. These challenges must be overcome before the rollout of new services and applications can be considered successful. For businesses to run properly, large amounts of human capital in addition to physical labor are required.

Under such circumstances, configuration management is often recommended as a solution. Automatic application deployment, maintenance, and updates are available utilizing configuration management systems like Chef. By analyzing its constituent parts, you can make educated decisions about the roles that Chef can play in the various DevOps environments.

Managing configurations is easy with Chef, which can be used to automate several facets of the IaC. Codification of infrastructure is made feasible by the automation tools provided by Chef.

You are free to use Chef in either a client-server or stand-alone configuration, whichever better suits your needs. A central hub that can be used for monitoring and administration is necessary for any Chef system. Agents may be deployed to workstations using Secure Shell with the use of the `knife` tool (`SSH`).

After that, the authentication process between the master and the managed nodes is carried out with the use of certificates. The establishment of antecedents is required for Chef agents to recognize when it is appropriate to speak with the "head chef." Investigating the numerous aspects that make up Chef architecture can assist you in developing a deeper comprehension of the inner workings of a "chef's kitchen."

Understanding Chef

To guarantee that Chef code is deployed throughout the infrastructure having a clear understanding of Chef building blocks is required. The following are the building blocks of Chef.

Recipe

Recipes are a set of characteristics that regulate the system. These recipe components are used to modify the configuration or status of a single node in the infrastructure. They are read by the Chef client during runtime and compared to the current node's settings (machine). When the node resource is reached, the recipe is in the final state as visible in Figure 8-1. The bulk of the cookbook is concentrated here. Complex recipes, on the other hand, need more sophisticated approaches like conditional statements to carry out the steps in the recipe. Blending vanilla Ruby with the Chef DSL yields impressive results.

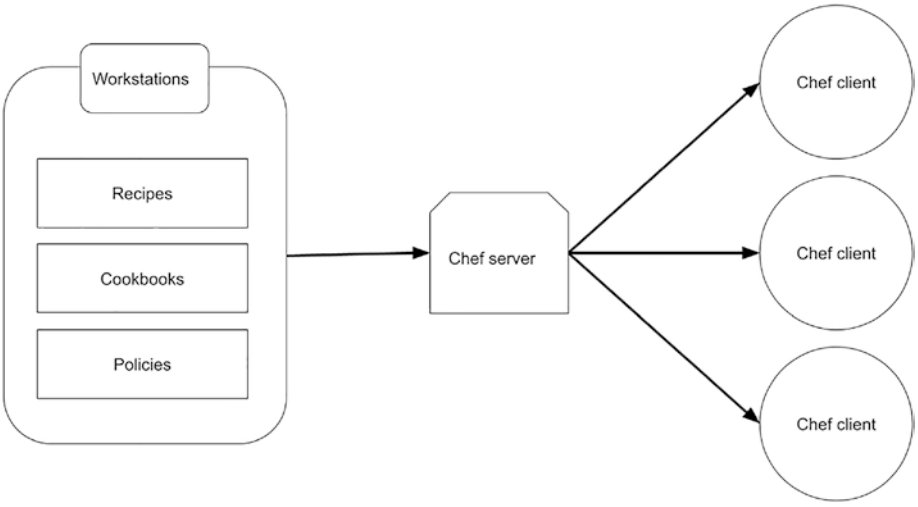

Figure 8-1. *Chef client-server*

Cookbook

The cookbook is the most basic building piece that configuration consists of. It does this by amassing resources in an effort to do what it needs to, such as recipes, templates, and files. You will initially only have access to a single folder in your cookbook that is labeled "recipes" when you first create a cookbook. If you want, you may keep your templates and other components organized in their own distinct folders.

Resource

In addition, the Chef cookbooks provide useful resources, and the Chef documentation may give trustworthy recommendations on how to make use of those resources. You can count on resources to only be precisely what the configuration policy says they are. This is the only thing you can depend on. The documents explain the configuration object's ultimate objective as well as the steps to follow to reach it. The configuration policy also defines the characteristics of what comprises a service, package, or template.

The resource block consists of four components: a name, a type, at least one property with a key-value pair, and one or more action(s). The following is a simple resource example that will install a simple package on a Linux server:

```
// Ruby
type `package` do
      attribute `name`
      action :install
      version `1.1.1`
      name `mypkg.tar.gz`
end
```

Attributes

You may consider attributes to be different locales. You may consider them to be a key-value pair, with one pair corresponding to each ingredient listed in the cookbook. There are many different kinds of attributes, and each of these forms has its own distinct effect on the configuration of a node after it has reached its ultimate state.

At every Chef client's run, the default attributes are reset and have the least precedence. On the other hand, the override attributes are also reset at every new run, but they do have a higher precedence. The following is the order of precedence from top to bottom for attributes to be applied in the Chef run, meaning applied first with least precedence:

- Default attribute in Chef cookbook attributes file
- Default attribute located in a Chef recipe
- Default attribute located in an environment
- Default attribute located in a role
- Force default attribute in Check cookbook attributes file
- Force default attribute located in a Chef recipe
- Normal attribute located in a Chef cookbook attributes file
- Normal attribute located in a Chef recipe
- Override attribute located in a Chef cookbook attributes file
- Override attribute located in a Chef recipe
- Normal attribute located in a role
- Normal attribute located in an environment

- `Force_Override` attribute located in Chef cookbook attributes file
- `Force_Override` attribute located in a Chef recipe

Metadata

The management of metadata is included as part of its functionality in this package. This provides information, such as the name of the label as well as its measurements. In addition, it provides a list of the necessary cookbooks that should be used in conjunction with this one. This ensures that the data is sent accurately and provides the Chef server with the capability to produce a suitable run list for the node. The following example showcases contents of a metadata file:

```
// Ruby
name 'chefexample'
maintainer_email 'dev@chefexample.com'
description 'Set up a new Chef instance'
version '3.0.1'
chef_version ">= 12.9"
```

Templates

The majority of time, test plans and templates may be found in Chef cookbooks with the recipes. Cookbook templates that have embedded Ruby (ERB) code make it possible to generate dynamic versions of text files that were previously static. To get better cookbooks, we need to evaluate the ones that are now available. As a consequence, syntactic testing is not the only kind of testing that is required for Chef cookbooks; moreover, tests at the unit and integration levels are required. For example, the following is an example template file that prints a “Hello World” string for the specific Chef node it is executed on:

```
// HTML
<html>
  <body>
    <h1>Hello world on <%= node[:fqdn] %></h1>
  </body>
</html>
```

The previous template then can be called into the recipe file, as shown here:

```
// Ruby
template 'home/ubuntu/ChefDemo/index.html' do
      source 'index.html.erb'
      owner '${user}'
      group '${user}'
      wheel '${user}'
end
```

Libraries

It is necessary to go to the library to get to know Chef. Because of Ruby's extensive library system, practically any chunk of Ruby code may be included in a recipe. Library-made tools are very adaptable, since they are often used in a broad range of recipes as well as materials that are manufactured to order.

Chef Infrastructure

When it comes to encapsulating infrastructure in code, the powerful automation tool Chef Infra is an excellent choice. Chef Infra is able to automate the setting up, deploying, and maintaining of your infrastructure no matter how extensive your network is or whether it is hosted in the cloud, on-premises, or in a hybrid configuration.

Chef Workstation

You are welcome to manage your infrastructure and develop recipes with the help of Chef Workstation. You are able to install Chef Workstation on a computer running either macOS, Linux, or Windows.

Many testing tools are accessible through Chef Workstation. These configurations are ideal for testing Chef Infra code before releasing it to servers that are accessible to a wider audience. Utilizing various resources while simultaneously creating code is necessary for the development of an infrastructure. A file is one example of a resource; however, resources may also contain things like templates and packages. An example of a resource is a file. In every document, the intended operational condition of a system component is laid forth, but there is no instruction given for how to actually reach that level. Your problems will be fixed by Chef Infra, rest assured. There are lots of Chef Infra's tools that you can use. You have the option of constructing resources that are tailored particularly to the requirements of your system, or you may use components derived from recipes that have been published.

A single Chef Infra recipe contains all of the instructions and components required to successfully install a web server, database server, or load balancer. The preparation of your meals will become more organized and uncomplicated if you use a cookbook written by Chef Infra.

The code that you generated and verified on your local machine will be accepted by the Chef Infra Server. On the Chef Infra Server, each and every piece of configuration data is safely stored. It is responsible for maintaining a record of the definitions of all of the systems, as well as the rules and norms that dictate how they should operate. Through the use of the `knife` command, it is possible for your computer and the Chef Infra server to connect with one another. For instance, you might add your own recipes by using this method.

Configuration of Nodes with Chef Clients

The majority of the computing power for Chef Infra comes from the nodes themselves rather than from the Chef Infra Server. Each individual server, regardless of whether it is a virtual machine, container instance, or physical machine, is represented by a node in the network. In a nutshell, Chef Infra is in charge of controlling each and every server that is part of the infrastructure. The Chef Infra Client is compatible with a broad range of operating systems, including Linux, macOS, Windows, AIX, and Solaris. It is configured on every node.

To get the most recent version of the cookbooks, the Chef Infra Client maintains continuous communication with the Chef Infra Server. The Chef Infra Client will take action to fix the situation whenever the node's present state is in a state that is inconsistent with the intended state that is stated in the cookbook. It is possible that, after sufficient back and forth, the network will finally arrive at the point that is envisioned by the firm strategy.

Chef Habitat

There is no other technique of application deployment that even comes close to matching the automatic application deployment that Chef Habitat provides. The term *application automation* refers to the automation elements that are already built into the application and may be used straightaway. The application and any relevant automation together make up the deployment unit. The functioning of the software shouldn't be altered in any way; therefore, you shouldn't notice any differences.

The program Chef Habitat has both a manager and a packaging system for its users. This approach results in the specification of individual Chef Habitat packages that are both immutable and verifiable. The manager

at Chef Habitat has a high level of expertise in the field of package management. The interdependencies of the package, as well as its upgrade methods and security rules, are all quite familiar to the package.

Chef InSpec

A language for establishing requirements such as policy, security, and regulatory compliance may be obtained via the use of the open-source testing framework Chef InSpec. When policy conformance is supplied in code, you have the option of including it in your deployment pipeline and having it check itself automatically to ensure that it conforms with the policy that has been expressed.

There are several computer setups that might be used to execute Chef InSpec code. The same battery of tests may be done locally using technologies such as the Docker API, remotely via SSH or WinRM, or even outside in the fresh air. All of these options are possible. The scope of Chef InSpec's compliance testing capabilities is far broader than those of physical servers alone.

Final Words

According to (ISC)^2 2021 Cloud Security Report,[1] the biggest threat with public clouds is the misconfiguration of resources, which can lead any organization to catastrophic failures. This introduction to Chef may be the beginning of a relationship that lasts the rest of your working life with DevOps and IaC to mitigate any such situations. Using Chef makes the process of delivering software more streamlined and improves the stability of services.

[1] https://cloud.connect.isc2.org/cloud-security-report

Because of the ease with which it may be deployed in the cloud, businesses may make better use of the features that Chef offers. When carrying out large-scale deployments in either a public or private environment, it is essential to maintain a close check on the stability, maturity, and dependability of Chef. The following are the differences and trade-offs of IaC.

Infrastructure as a service (IaaS) will streamline the most time-consuming parts of initial setup and maintenance, and it will be easy to roll out "standard" instances, assuming you anticipated the shift and your infrastructure accommodates it. Otherwise, you must start using it right away. Depending on the choices you made during implementation, the level of difficulty may vary.

The automation and IoC platform that you choose will be the center of your organization's programming efforts. This, like with any software, requires regular programming maintenance to avoid the ensuing unwieldy spaghetti code. You need a group of DevOps engineers that are well-versed in both system administration and development to handle the management of such a platform on a large scale. The challenges of IoC should not be compared to the regular management of your business.

As a result of automation, human error and missed steps are far less likely. Just remember that humans are still fallible, even if their blunders are now more conceptual. Human error from entering incorrect values will be eliminated with the use of automation. However, errors may be made whenever a new instance is launched or its configuration is altered. It's also possible that the automated system is flawed and underperforming.

Every administrative position calls for occasional upkeep and unanticipated adjustments, so keep that in mind. If you try to address an issue by reducing the degree of expertise of the team you currently have, you can end up with an unsolvable situation. Restoring service after an extended outage would incur far higher costs than laying off skilled workers.

As a result, there is no longer any reason to put off taking action. Invest some time today into learning Chef so that you may have a secure future working in DevOps. The use of Chef, an essential tool for DevOps, has lately seen an increase in popularity.

Index

A

B

C

S. Pandya and R. Guha Thakurta, *Introduction to Infrastructure as Code*,
https://doi.org/10.1007/978-1-4842-8689-0

D

E

F

G

H

I

J, K

L

M

N

O

P, Q

R

S

T

U

V

W, X, Y, Z