packt

1ST EDITION

DevSecOps for Azure

End-to-end supply chain security for GitHub, Azure DevOps, and the Azure cloud

DAVID OKEYODE | JOYLYNN KIRUI

Foreword by Scott Hanselman, VP Developer Community, Microsoft

DevSecOps for Azure

End-to-end supply chain security for GitHub, Azure DevOps, and the Azure cloud

David Okeyode

Joylynn Kirui

DevSecOps for Azure

Group Product Manager: Pavan Ramchandani

Publishing Product Manager: Prachi Sawant

Senior Editor: Sayali Pingale

Technical Editor: Rajat Sharma

Copy Editor: Safis Editing

Book Project Manager: Uma Devi

Indexer: Manju Arasan

Production Designer: Vijay Kamble

Senior Marketing Executive: Marylou De Mello

First published: August 2024

Production reference: 1310724

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK

ISBN 978-1-83763-111-7

www.packtpub.com

This book is dedicated to the incredible individuals who continue to lovingly nurture me in the complex art of life: my dad, Oluwagbenga; my mum, Oluwayemisi; my three sisters, Oluwapemi, Oluwagbolade, and Ajulooluwa; and my wife, Po Kei.

– David Okeyode

For the audacity to dream and the courage to pursue those dreams. I dedicate this book to my parents (Mr. John Kirui and Dr. Janet Kirui, OGW), who gave me the gift of dreams and the ability to realize them. Also, to my amazing son – hi, Zani! 👏
Thank you to my family and friends.

– Joylynn Kirui

Foreword

You have in your hands the key to beginning of your journey to mastering Azure DevOps workflows for cloud infrastructure. I've had the absolute pleasure of teaming up with Joylynn in person on projects here at Microsoft where we work together, and have found her to be an enthusiastic and detail-oriented technologist. I've also had the pleasure of virtually enjoying David's technical content and consistent advocacy online for many years, so when I heard that the two of them were teaming up on a book covering DevSecOps for Azure, I jumped at the opportunity to write the foreword! A book that would teach me not only how to set up my CI/CD pipelines but also how to implement best practices for security? Sign me up!

As you know, security is having a moment right now and it's a top priority for many businesses. Attacks are constantly evolving, and the Azure cloud is rising to meet the challenge. That said, many organizations are finding it difficult to keep up with the security environment and maintain the compliance of their own services. The software supply chain matters now more than ever, and I appreciate how this book assembles all the knowledge that we need to effectively and responsibly create and deliver software, understand source control systems, build systems, and put them all together on CI/CD platforms that create, ship, and deploy artifacts reliably and securely.

You'll come out of this experience with a solid understanding of the relationship between the philosophy of Agile workflows, DevOps, and practical cloud management. I found the chapter on continuous and automated threat modeling to be particularly useful as my team needs to secure our development toolchains with tools such as GitHub Codespaces and Microsoft Dev Box.

Is this book for you? If you're a developer, DevOps engineer, or security professional – really anyone who wants practical and pragmatic tips on how to implement DevSecOps in a small, medium, or large organization – then absolutely! So many organizations are transitioning to the public cloud, and during this process, they're seeking to understand where security and testing fit into a continuous delivery pipeline.

I hope you enjoy reading this book as much as I did, and I can tell you that David and Joylynn very much enjoyed writing it!

Scott Hanselman

VP Developer Community – Microsoft

Contributors

About the authors

David Okeyode is the EMEA chief technology officer for Azure at Palo Alto Networks. Before that, he was an independent consultant helping companies secure their Azure environments through private expert-level training and assessments. He has authored three books on Azure security – *Penetration Testing Azure for Ethical Hackers*, *Microsoft Azure Security Technologies Certification and Beyond*, and *Designing and Implementing Microsoft Azure Networking Solutions*. He has also authored multiple cloud computing courses for the popular training platform LinkedIn Learning. He holds over 15 cloud certifications across Azure and AWS platforms, including the Azure Security Engineer, Azure DevOps, and AWS Security Specialist certifications. David is married to a lovely girl who makes the best banana cake in the world. They love traveling the world together!

I would like to first and foremost thank my loving and patient wife and son for their continued support, patience, and encouragement throughout the long process of writing this book. Thanks also to the Masters of Pie and Method teams for their generosity with their equipment—obviously a critical component for this book.

Joylynn Kirui is a senior cloud security advocate at Microsoft. She focuses on DevSecOps on GitHub and Azure, which includes application security. She is an infosec evangelist who believes in empowering developers and users in general on security best practices. She has vast experience in web and mobile app security testing, DevSecOps, and GSM security, having previously worked in the telco industry. She has a passion for mentorship and training students and empowering them. She has spoken at several conferences, where she shares her knowledge in cybersecurity and software development. She was among the Top 50 Women in Cyber Security Africa 2020 finalists, Woman Hacker of the Year Africa 2020 finalists, and Young CISO Vanguard 2022, among others. When not hacking, she enjoys farming, traveling, and adrenaline-filled activities.

About the reviewers

James Duncan is a security cloud solutions architect at Microsoft with a background in security operations, architecture, and development security. He assists customers worldwide in architecting and building cloud solutions with a core focus on ensuring applications are secure by design rather than addressing security retrospectively.

James received a BSc in computer science from the University of Hull and has previously worked in cybersecurity at Fujitsu and Rolls-Royce.

I would like to thank both Joylynn and David for selecting me to technically review this book. Their commitment to writing and releasing comprehensive security guidance in an ever-changing space is remarkable.

Chris Boehm, a seasoned cybersecurity professional and global field CISO at SentinelOne, is dedicated to empowering organizations worldwide to overcome cybersecurity challenges. With a diverse skill set in corporate strategy, marketing, sales, and engineering, Chris plays a pivotal role in driving SentinelOne's global initiatives. As a thought leader in cybersecurity, Chris leads a dynamic global team of cybersecurity professionals. Beginning in SLED, Chris swiftly transitioned into an enterprise role, and during his tenure at Microsoft, he was part of the cybersecurity engineering organization, collaborating with MSTIC and CDOC teams to develop a CI/CD pipeline demonstrating Microsoft Security's efficacy against real-world threats.

I extend heartfelt gratitude to my wife for her unwavering support and collaboration, which have been instrumental in my professional growth. To my dedicated colleagues and friends, your camaraderie and encouragement have been invaluable. I am also profoundly grateful for the enduring support from my family, whose unwavering love and understanding have been the bedrock of my journey.

April Yoho is a senior developer advocate at GitHub and specializes in DevOps. While also having previously worked at Microsoft, she specializes in application transformation and DevOps ways of working. Her focus is on taking customers on a journey from legacy technology to serverless and containers, where code comes first, while enabling them to take full advantage of DevOps practices. April is also an international speaker who has given talks on various topics at community events, large conferences, and customer events.

April spends her spare time outdoors hiking, skiing, or scuba diving. She is also a triathlete having competed in the Ironman and Half Ironman triathlons.

Table of Contents

2

Security Challenges of the DevOps Workflow 33

Part 2: Securing the Plan and Code Phases of DevOps

3

Implementing Security in the Plan Phase of DevOps 51

4

Implementing Pre-commit Security Controls 83

5

Implementing Source Control Security 125

Part 3: Securing the Build, Test, Release, and Operate Phases of DevOps

6

Implementing Security in the Build Phase of DevOps 165

7

Implementing Security in the Test and Release Phases of DevOps 221

8

Continuous Security Monitoring on Azure 271

Preface

Security is a major concern for businesses. Sixty percent of organizations report that their DevOps initiatives face security challenges due to the increased speed, automation, and decentralization of the development process. Our goal in writing this book is to help you (the reader) gain a clear understanding of how to implement continuous security into every phase of the DevOps workflow for organizations that are adopting the Azure cloud, its services, and the DevOps toolchain.

Complete with hands-on labs, this book will take you beyond foundational knowledge to having a clear understanding of integrating security early in the DevOps workflow. By the end of this book, you will be fully equipped with information on how to harden the entire DevOps workflow, from software planning to coding to source control to continuous integration and running Azure cloud workloads.

Who this book is for

This book is tailored for developers and security professionals who are transitioning to a public cloud environment or moving towards a DevSecOps paradigm. It's also designed for DevOps engineers, or anyone keen on mastering the implementation of DevSecOps in a practical manner. Also, individuals seeking to understand how to integrate security checks, testing, and other controls into Azure cloud continuous delivery pipelines will find this book invaluable. Prior knowledge of DevOps principles and practices and an understanding of security fundamentals will be beneficial.

What this book covers

Chapter 1, *Agile, DevOps, and Azure Overview*, will introduce the working definition of DevOps that we will use for the rest of the book. We will discuss the stages in a DevOps workflow and the five core DevOps implementation practices. We will also explain the relationship between Agile, DevOps, and cloud; the security challenges of implementing DevOps; and how organizations can start to address those challenges.

Chapter 2, *Security Challenges of the DevOps Workflow*, will explore the unique security risks and threats that arise from implementing DevOps practices. We will examine how organizations can begin to address these challenges effectively.

Chapter 3, *Implementing Security in the Plan Phase of DevOps*, covers how the PLAN phase of DevOps focuses on gathering requirements and feedback from key stakeholders and customers, producing an evolving product roadmap that prioritizes key requirements, and designing a flexible software architecture. Implementing DevSecOps for this phase should focus on security challenges that can be

addressed before the developers start writing code! Activities in this phase should include implementing an agile threat modeling process to identify design-level security issues earlier; and implementing security training for your teams. In this chapter, we will cover what works when looking to implement a continuous threat modeling process. We will also discuss the different maturity levels of a secure code-to-cloud training program.

Chapter 4, *Implementing Pre-commit Security Controls*, will focus on security measures and checks that can be implemented before code changes are committed to a version control system by developers. This includes implementing security controls to reduce development environment risks, and setting up security safeguards to identify and fix vulnerabilities and common mistakes before code is committed to the local code repository.

Chapter 5, *Implementing Source Control Security*, examines how source control in DevOps is a way to organize and track the code for a project using a **source control management (SCM)** system such as Git or **Team Foundation Version Control (TFVC)**. When implementing DevSecOps in source control, it is important to consider how the code repository is managed and secured. If access to the code repository is compromised or protections can be easily bypassed, it is hard to trust the code stored in it. To keep the code repository safe, we should implement a code-signing process to verify the authenticity of code changes. We should also protect sensitive branches and implement security controls.

Chapter 6, *Implementing Security in the Build Phase of DevOps*, will focus on understanding the continuous build phase of DevOps, securing CI environments and processes, hardening the build process to enhance security, and integrating SAST, SCA, and secret scanning into the build process.

Chapter 7, *Implementing Security in the Test and Release Phases of DevOps*, will focus on ensuring that release artifacts are built from protected branches, implementing a code review process, selecting a secure artifact source, and validating artifact integrity. Additionally, we will cover managing secrets securely in the release phase, implementing Infrastructure-as-Code security scans, and validating and enforcing runtime security with release gates.

Chapter 8, *Continuous Security Monitoring on Azure*, will focus on understanding continuous monitoring in DevOps, implementing runtime guardrails in Azure, and preventing, detecting, and remediating application risks at runtime.

To get the most out of this book

A general understanding of the Azure cloud is necessary to get the most out of this book. To follow along with the practical exercises, you will need the following:

- A PC with an internet connection
- An active Azure subscription
- An Azure DevOps organization
- A GitHub Enterprise organization

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/DevSecOps-for-Azure`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: **Resource group: Create new | Name:** `DevSecOps-Book-RG` **| OK**.

A block of code is set as follows:

```
apiVersion: v1
kind: Pod
metadata:
  name: non-root-pod
spec:
  containers:
  - name: mycontainer
    image: myimage
    securityContext:
      runAsUser: 1000
      runAsGroup: 3000
```

Any command-line input or output is written as follows:

```
pip install pre-commit
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Click on **Sign in**."

> **Tips or important notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *DevSecOps for Azure*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/9781837631117`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1: Understanding DevOps and DevSecOps

In this part, we will discuss the stages of a DevOps workflow, the security challenges of implementing DevOps, and how organizations can start addressing those challenges.

This part contains the following chapters:

- *Chapter 1*, *Agile, DevOps, and Azure Overview*
- *Chapter 2*, *Security Challenges of the DevOps Workflow*

1

Agile, DevOps, and Azure Overview

DevOps is a modern application development and delivery approach that helps organizations release quality software more quickly into production with fewer defects! However, the benefits of adopting a DevOps approach are not realized in isolation. They are best realized in conjunction with other concepts such as Agile planning and cloud computing.

Most of this book focuses on DevSecOps, but in this chapter, we will begin with an introduction to DevOps for those unfamiliar with the concept. We will introduce the working definition of DevOps, which we will use for the rest of the book. We will discuss the stages in a DevOps workflow and the five core DevOps implementation practices. We will also explain the relationship between Agile, DevOps, and cloud computing, the security challenges of implementing DevOps, and how organizations can start to address those challenges.

By the end of this chapter, you will have a good understanding of the following:

- What DevOps is
- The five core practices of DevOps
- The stages in a DevOps workflow
- The importance of a collaborative culture in DevOps
- The DevOps anti-types to watch out for
- The DevOps toolchain (Azure DevOps, GitHub Actions, and GitLab)
- The why of DevOps
- The relationship between Agile, DevOps, and cloud computing

These topics will equip you with the essential foundational knowledge to understand and contextualize the discussions presented throughout the remainder of this book. Now, let's dive in and begin our journey!

Technical requirements

To follow along with the instructions in this chapter, you will need the following:

- A PC with an internet connection
- A valid email address

Defining DevOps – Understanding its concepts and practices

If you ask 10 people what DevOps is, you will probably get 10 different answers, depending on these people's backgrounds and probably the books they have read. Therefore, it is important for us to establish a working definition that we will use for DevOps for the rest of this book. Microsoft's official definition of DevOps was coined by Donavan Brown at a conference in 2018. You can still find the video on YouTube: `https://www.youtube.com/watch?v=cbFzojQOjyA`. Here is the definition:

> *DevOps is the union of people, process, and products to enable continuous delivery of value to our end users.*

From this definition, we want to highlight a few essential points. To start with, it is essential to understand that DevOps is *not* a tool, a product, or a job title. Instead, it is a collaborative approach to software development. It is a way of working/thinking, and most of all, it is a change of culture (more on this later). Another key point to note is that the primary goal of DevOps is to ensure the speedy and frequent delivery of functional software to end users. If what has been implemented does not have this impact, it is likely not DevOps, or it has not been appropriately implemented (we will discuss this in more detail in the *Staying clear of DevOps anti-types* section in this chapter). The last point that we would like to stress is that there are three aspects to DevOps. There is a **people** aspect, a **process** aspect, and a **product** aspect. In the next section, we will begin by examining the process aspect, but before we do that, let's discuss why organizations are rapidly moving towards a DevOps approach for software development and delivery.

The why of DevOps – Innovation, velocity, and speed

While we have dedicated significant time to discussing the process, people, and product aspects of DevOps, it is equally important to understand the driving factors that lead companies to embrace DevOps and the reasons for its growing significance in recent years. DevOps provides unique advantages to companies that other software delivery approaches cannot match. The following points are some of the benefits associated with DevOps adoption:

- **Accelerating time to market**: This refers to the ability to bring new products to market faster. According to research conducted by Puppet, companies that embrace the culture and practices of DevOps deploy code 46 times more frequently compared to those that do not.
- **Adapting to the market and competition**: This means being able to adapt to changes in the market and competition. For example, Etsy, an online marketplace for handmade and vintage goods, uses DevOps practices to deploy code changes 50 times per day. This allows the company to quickly test and launch new features, respond to user feedback, and stay ahead of competitors.
- **Maintaining system stability and reliability**: DevOps practices can help organizations maintain system stability and reliability by improving communication and collaboration between development and operations teams. For example, Netflix uses a DevOps approach to ensure that its streaming service remains available and responsive at all times. The company achieves this by automating its infrastructure deployment and using a "chaos monkey" tool to intentionally introduce failures in its systems, which helps identify and address weaknesses before they cause problems.
- **Improving mean time to recovery**: By adopting DevOps practices, organizations can improve their ability to recover from incidents and outages more efficiently. For instance, Target, a leading retail company in the US, reduced its overall **mean time to recovery (MTTR)** by 90% after implementing DevOps practices. This allowed the company to minimize the impact of outages and maintain high levels of customer satisfaction.

With the basics covered, let's delve into the process used in DevOps to create workflows.

Understanding the process aspect of DevOps

Whenever DevOps is discussed, it is tempting to make technology or tooling the main focus. However, without well-defined processes in place, any benefits or results achieved from adopting DevOps will be limited at best, and it may even create additional challenges and complexities!

In the DevOps methodology, the process aspect refers to the creation of an efficient and streamlined workflow for software development, testing, and deployment. The goal is to optimize the development process to ensure that software is delivered quickly and reliably to end users while maintaining a high level of quality.

This involves the use of agile development methodologies and **continuous integration and continuous delivery (CI/CD)** practices. These practices involve automating various aspects of the software development lifecycle, such as code testing, building, and deployment. Generally, when an organization adopts a DevOps approach, it must implement five essential practices: **Agile Planning**, **Version Control**, **Continuous Integration** (**CI**), **Continuous Delivery** (**CD**), and **Continuous Monitoring** (see *Figure 1.1*):

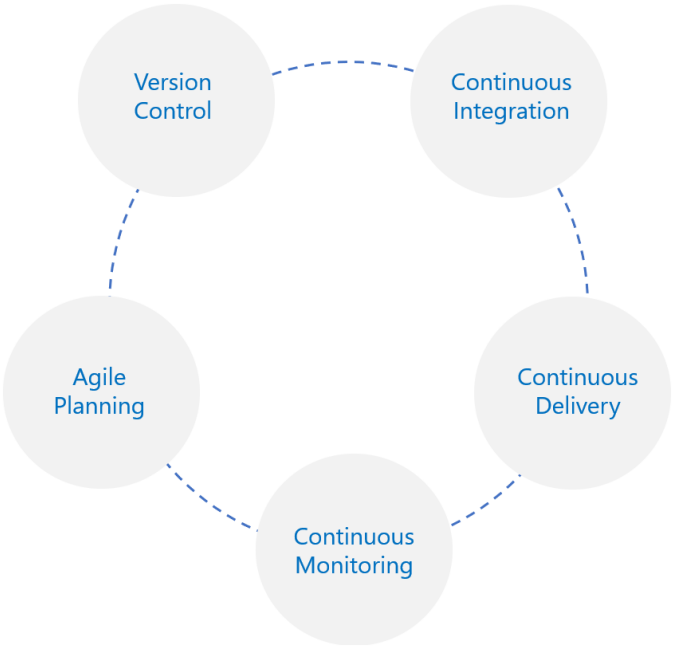

Figure 1.1 – The five essential practices of DevOps

It is worth noting that these are not the only practices in DevOps, but they are considered to be crucial ones. In the next section, we will describe these five core practices in more detail.

> **Important note**
>
> For those keen on exploring other definitions and models related to DevOps, the DevOps **Competence Model** by the **DevOps Agile Skills Association** (**DASA**) is a valuable resource. You can find more information about it here: `https://www.dasa.org/products/guidance-products/team-competence-model/`.

Understanding the five core practices of DevOps

In this section, we will examine the five fundamental practices of DevOps, beginning with agile planning.

Agile planning is a broad reference to techniques used to plan and track our software projects in DevOps. It is a project management approach that involves breaking down a project into small, manageable pieces and working on them iteratively. The agile methodology was formally launched in 2001 through the Agile Manifesto, covering the main principles of Agile project management. To get more information on the Agile Manifesto, you can go to `https://agilemanifesto.org/`.

The goal is to deliver a functional product incrementally and continuously while taking feedback from the stakeholders.

A simple example of agile planning can be seen in the development of a mobile app. Let's say a company wants to develop a mobile application that can be used to order food from local restaurants. The development team would first identify the key features that the app should have, such as a menu, ordering system, payment system, and user profiles. With these requirements in hand, they would then design the architecture of the app. Following this, the team would break down these features into smaller, more manageable tasks, such as designing the user interface, creating a database to store orders, and integrating the payment system. The team would then prioritize these tasks based on the business value they add and the level of effort required to complete them. Once the tasks are prioritized, the team would estimate the time required to complete each task and create a sprint plan. A sprint is a short, time-boxed period (usually 1–2 weeks) during which the team works on a set of tasks.

During each sprint, the team would work on the tasks in priority order, complete them, and get feedback from stakeholders. The feedback would then be used to make adjustments to the product and the plan for the next sprint. This process of breaking down tasks, prioritizing them, estimating time, and working iteratively with feedback is the core of agile planning.

> **Important note**
>
> To understand the guiding values of agile development, we recommend reviewing the twelve principles of agile development that are highlighted here: `https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/`.

The second practice, **version control**, allows developers to manage changes to code efficiently, collaborate effectively, and keep track of all changes made to the code. *Figure 1.2* shows a simple example of how version control works in DevOps. Suppose a team of developers is working on a software application. They create a repository (a central location to store code) using a **version control system** (**VCS**) such as Git. Each developer can clone the repository to their local computer, or they might work directly in a controlled development environment, eliminating the need to copy code to a local PC. It is worth noting that some companies have strict policies regarding this workflow and do not allow code to be cloned locally.

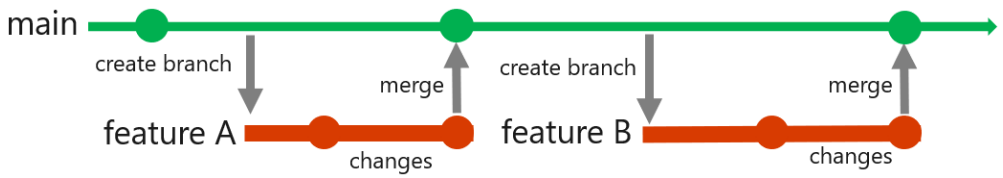

Figure 1.2 – Version control and branching example

Let's say Developer A is assigned to work on feature A; they create a new branch in the repository called **feature A** and start making changes to the code. Meanwhile, Developer B is working on a different feature in the application. They create a new branch called **feature B** and start making changes to the code. Both developers can work on their features independently without affecting each other's work. Once they have completed their changes, they can merge their branches back into the main branch (also called the **trunk** branch) in the repository.

If there are any conflicts between the changes made by the two developers, the VCS will highlight them, and the developers can resolve them before merging the branches. The VCS also keeps a record of all changes made to the code, including who made them, when they were made, and why they were made. If there is a problem with the new code, the team can use the VCS to roll back to a previous version of the code quickly. This rollback feature is useful if a bug is introduced into the code or if the new changes cause unexpected problems.

The third practice, **continuous integration** (**CI**), refers to the ongoing validation of code quality whenever developers contribute or modify code. Suppose a team of developers is working on a software project; each time a developer finishes making changes to their code and commits those changes to the shared repository, an automatic process is triggered on a CI server, such as Jenkins or Travis CI, to build the software, run unit tests, and check for code quality issues using various tools. If the build and tests pass successfully, the CI server will notify the team that the changes are ready for review and integration. If any errors or issues are detected, the CI server will alert the team, and they can then work together to fix the issues before merging the code into the shared repository. This allows the team to catch and fix issues early in the development cycle, reducing the risk of bugs and errors in the final product:

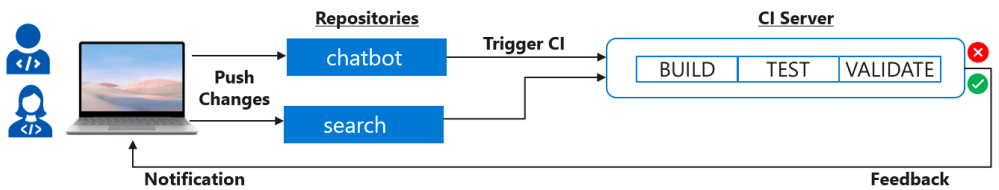

Figure 1.3 – Sample CI flow

The fourth practice, **continuous delivery** (**CD**), refers to the ongoing testing and deployment of validated software using an automated process. It allows teams to release new features and bug fixes quickly using a continuous process. The goal of CD is to enable development teams to deliver software changes to production quickly and with confidence while maintaining a high level of quality and reliability.

Suppose a team of developers is working on a web application; when the team writes code for a new feature, it is committed to a version control system and is automatically tested by a series of automated tests, including unit tests, integration tests, and acceptance tests. Once the code passes all the tests, it's automatically deployed to a staging environment where it undergoes additional testing and review by the product owner. If everything looks good, the code is then automatically deployed to production, where it's made available to all users.

The fifth practice of **continuous monitoring** involves gathering feedback from users and collecting telemetry data from running applications in real time. The goal is to ensure that software systems are meeting the needs of users and delivering value to the organization. It requires gathering continuous insights into the performance and behavior of software systems and using that information to make data-driven decisions that improve the overall quality and user experience. To understand this practice better, let's break it down into two components:

- **Gathering feedback from users**: User feedback is an essential component of continuous monitoring because it helps to identify issues and areas for improvement in the software system from the user's perspective. Feedback can be collected through various channels, such as surveys, user reviews, and support tickets. By analyzing this feedback, development teams can identify patterns and trends that highlight areas for improvement and prioritize these improvements based on their impact on the user experience.
- **Collecting telemetry data from running applications**: Telemetry data refers to a broad range of information collected from various sources as the software system operates in real time. These sources can include application logs, server metrics, network traffic, user interactions, error reports, and more. Metrics such as response times, error rates, and server load, as well as insights into user behavior, can be derived from these data. By collecting and analyzing telemetry data, development teams can gain a comprehensive understanding of the software's performance and user interactions. This data is invaluable for detecting anomalies and potential issues before they escalate into critical problems.

By combining user feedback with telemetry data, development teams can gain a comprehensive understanding of how the software system is performing and how it is being used. This information can then be used to make data-driven decisions about how to improve the system and prioritize future development efforts. Overall, the fifth practice of continuous monitoring is a crucial part of DevOps that helps to ensure that software systems meet the needs of users and deliver value to the organization.

Understanding the stages in a DevOps workflow

Understanding the five essential practices of DevOps is vital, but how do organizations put them into action? The implementation of DevOps practices involves a set of stages that facilitate the constant development, testing, and deployment of software. These stages may differ based on the organization and the type of software being developed, but they typically follow the pattern shown in *Figure 1.4*:

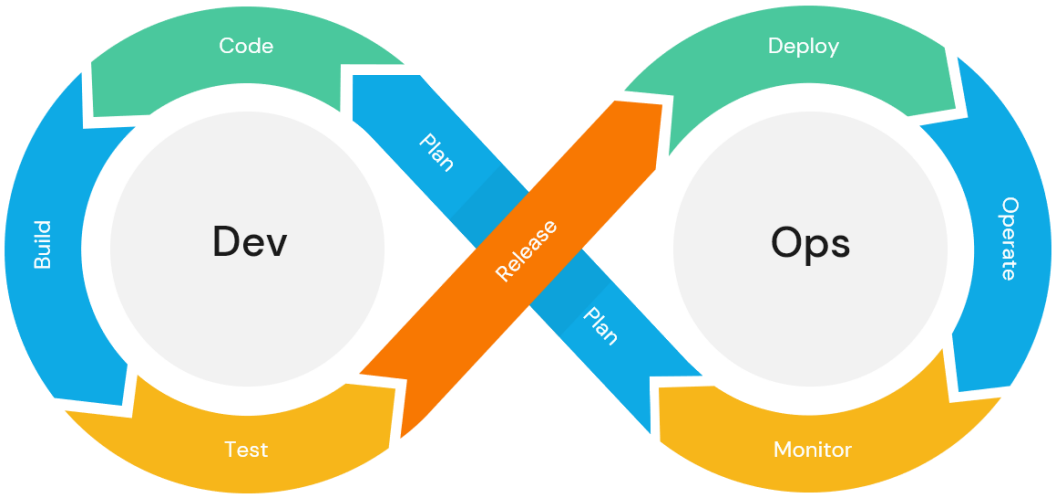

Figure 1.4 – Typical stages in a DevOps workflow

The first stage is **Plan**, where the agile planning practice is put into action. At this stage, teams plan and prioritize what needs to be accomplished based on business or customer requirements. This involves creating a project plan or roadmap, researching to understand the required architectural changes, defining the scope of work (such as feature development or bug fixing), breaking down the plan into smaller and assignable tasks, estimating the time required for each task, and setting priorities for the tasks that need to be completed first.

The second stage is **Code**, which involves the actual coding and development of software using the selected programming languages, frameworks, and tools. It is at this stage that version control practices are implemented. The team collaborates to develop the code and commit changes to a version control system.

The third stage is **Build** and **Test**, where continuous integration practices are implemented. In this stage, the code is converted into executable software and tested to guarantee that it works as intended and fulfills project requirements. A combination of automated and manual tests is employed to detect and resolve any errors, bugs, or defects.

The fourth stage is **Release** and **Deploy**, where the software is packaged and released into the production environment. This is where continuous delivery practices are implemented. This stage involves setting up the infrastructure required to run the software and configuring it to work, deploying the software into a pre-production environment to run additional validation, and deploying validated software into production.

The fifth stage is **Operate** and **Monitor**, where the software is actively monitored and maintained. The team watches for any issues or incidents after deployment, examining the application's performance, collecting and analyzing logs, and ensuring that the software complies with defined **service level agreements** (**SLAs**). In this stage, continuous monitoring tools and practices are used to track the application's performance, gather usage telemetry and performance metrics, and detect any potential issues before impacting users. The gathered information is then used to identify areas for optimization or additional features to be added. A **self-healing** approach that leverages automation is increasingly popular at this stage. This approach involves using automation to correct any failures or errors without requiring human intervention, such as terminating a problematic application instance and deploying a replacement instance or triggering failover to a passive instance in the case of unexpected events. Implementing this approach significantly improves system availability and reliability and enables faster and more efficient recovery from failures.

These stages form a continuous cycle that empowers teams to continuously deliver value to end users while enhancing their software development procedures. Keep in mind that speed is crucial to a successful DevOps workflow! It is essential that each stage is executed quickly and efficiently (we will revisit this aspect when we talk about security integrations).

Understanding the people aspect of DevOps

Simply implementing DevOps practices in a continuous workflow is insufficient to fully unlock its potential; a cultural component is also necessary. Implementing DevOps methodologies delivers better results in a culture that promotes communication, collaboration, and shared responsibility among the members of development and operations teams. However, for many organizations (particularly larger ones), this proves to be the most difficult aspect of embracing DevOps since it involves a change in mindset and company culture, which can challenge established policies and procedures that have yielded positive results thus far.

The importance of a collaborative culture

To realize the full potential of DevOps, an organization must embrace a collaborative culture! By this, we mean a culture that breaks down team silos and allows developers, operations engineers, and other stakeholders to work together to achieve the shared goal of continuously delivering high-quality software to customers. This can be achieved by creating cross-functional teams or vertical teams.

Traditionally, large organizations have organized their teams in a horizontal structure based on particular skill sets or functional areas such as development, testing, or operations (as shown in *Figure 1.5*). Each team concentrates on their area of expertise and only handles tasks within that domain. The teams are separated by a boundary (as illustrated in *Figure 1.6.*) and are measured using different performance metrics, which frequently results in conflicts.

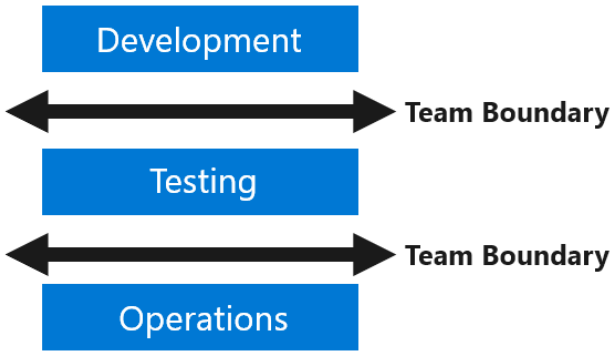

Figure 1.5 – Team boundaries in software development

On the other hand, DevOps advocates for and flourishes in teams that are organized vertically around particular products or services, also known as cross-functional teams. This structure brings together individuals from diverse functional areas to collaborate on a common objective of delivering a specific product or service. Each team member possesses a wide range of skills and is responsible for contributing to the delivery of that product or service. The teams are also measured using a shared set of performance metrics, which encourages team members to leverage each other's skills and expertise to achieve shared goals. For example, a vertical team may be composed of developers, testers, and operations engineers collaborating to deliver a specific application or service, as shown in the following figure:

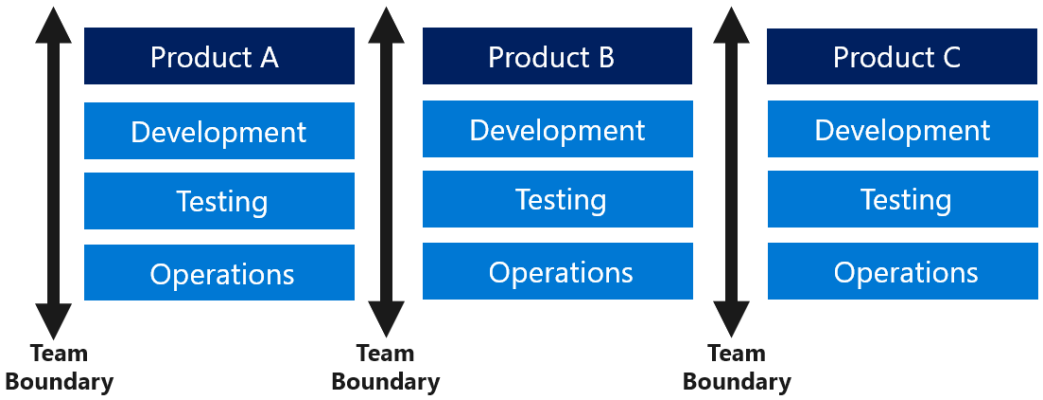

Figure 1.6 – Vertical team boundaries

It is crucial to note that while the composition of teams is vital, the presence of a guiding figure, often a servant-leader type, is equally important. Teams require clear direction and leadership to function optimally. This leader ensures that the team remains aligned with its goals, facilitates collaboration, and provides the necessary support to address challenges.

There are other cultural components of DevOps, such as fostering a culture of continuous learning and experimentation, ownership, and accountability. However, we recommend reading *The Phoenix Project* by Gene Kim for a more detailed understanding of these components.

Staying clear of DevOps anti-types

When implementing a DevOps culture, it is important to be aware of potential anti-patterns and anti-types. These are ineffective and sometimes counterproductive approaches that can hinder the successful implementation of DevOps.

For example, in an effort to implement DevOps, a manager or executive may create a separate DevOps team, which can further divide development and operations teams (*Figure 1.7*). The only time this separation may make sense is when the team is temporary, with a clear mandate to bring the teams closer together:

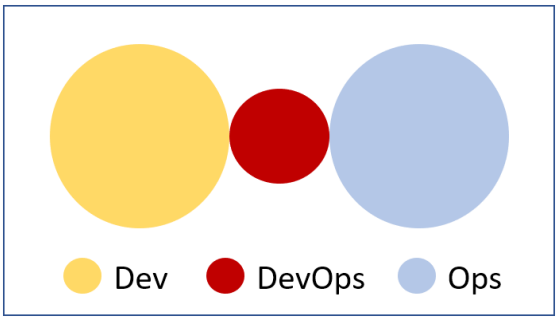

Figure 1.7 – Anti-type pattern 1

Another common anti-type is when developers or development managers assume they can do without operational skills and activities (*Figure 1.8*). This misconception is often rooted in a misguided understanding of cloud computing, which assumes that the self-service nature of cloud computing makes operational skills obsolete. However, this perspective grossly underestimates the complexities and significance of operational skills and results in avoidable operational mistakes:

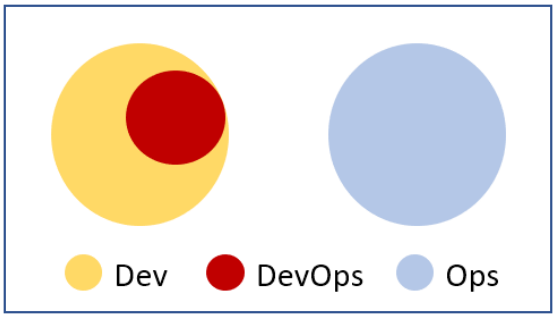

Figure 1.8 – Anti-type pattern 2

Yet another anti-type is when organizations simply rename their operations team as a DevOps or **site reliability engineering** (**SRE**) team without making any real change to their processes or silos (refer to *Figure 1.9*). This approach fails to understand or appreciate the importance of bringing individuals of different expertise together to work collaboratively towards shared goals:

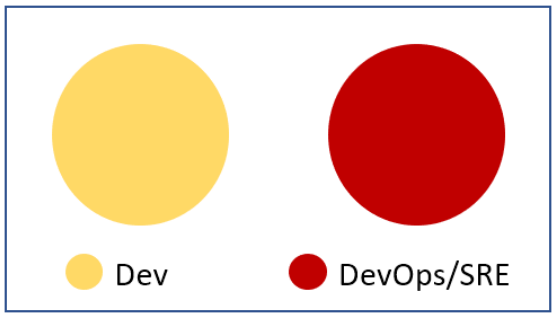

Figure 1.9 – Anti-type pattern 3

SRE is a discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems. The main goal of an SRE team is to create scalable and highly reliable software systems. While SRE aligns closely with the DevOps philosophy, merely renaming an operations team to SRE without adopting its principles or practices can be considered an anti-pattern. It is not just about the title but about embracing the methodologies, practices, and culture that both DevOps and SRE advocate for.

Important note

For a more detailed analysis of DevOps anti-types and patterns, please refer to the book *Team Topologies* by Matthew Skelton and Manuel Pais.

Understanding the product aspect of DevOps – The toolchain

While DevOps itself is not a tool or product, it requires the use of tools to effectively implement its processes and practices. Both open source and commercial tools are available to support the necessary processes for every phase of the DevOps workflow discussed earlier in this chapter (Plan, Code, Build and Test, Release and Deploy, and Operate and Monitor).

Common tools used in the **planning phase** include **Trello**, **JIRA**, **Notion**, and **Asana**. According to the latest Stack Overflow Developer Survey, professional developers prefer JIRA (49%), whereas Trello is most used by those learning to code (43%):

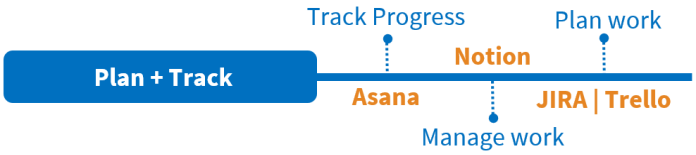

Figure 1.10 – Common tools used in the planning phase

During the **code and development phase**, developers use **integrated development environments** (**IDEs**), such as **Visual Studio Code**, **Visual Studio**, **IntelliJ**, **Notepad++,** and **Eclipse**, for coding purposes and version control tools, such as **Git** (self-hosted or cloud-hosted), **Apache Subversion** (**SVN**), **Perforce**, and **Mercurial**. It is important to note that while this list highlights some of the more common tools, it is by no means exhaustive. There are countless other tools available on the market, each with its unique features and capabilities. According to the 2022 Stack Overflow Developer Survey, professional developers overwhelmingly prefer Git as their version control tool (96%) and Visual Studio Code as their IDE (74%):

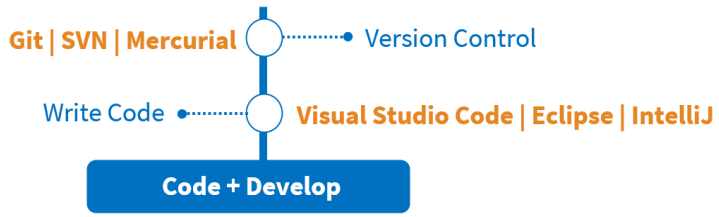

Figure 1.11 – Common code and development tools

Important note

The Stack Overflow Developer Survey is an annual survey conducted by Stack Overflow, a popular online community for developers. The survey aims to gather insights into the preferences, opinions, and demographics of the developer community. The 2022 edition can be found here: `https://survey.stackoverflow.co/2022`.