

DevOps Design Patterns

Implementing DevOps best practices for secure and reliable CI/CD pipeline



Pradeep Chintale



DevOps Design Patterns

Implementing DevOps best practices for secure and reliable CI/CD pipeline



Pradeep Chintale



DevOps Design Patterns

*Implementing DevOps best practices
for secure and reliable CI/CD
pipeline*

Pradeep Chintale



www.bpbonline.com

Copyright © 2024 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

First published: 2024

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55519-924

www.bpbonline.com

Dedicated to

My beloved Parents:

Shankar Chintale

Kamal Chintale

&

*My wife **Prachi**, my daughter **Prisha** and son **Pransh***

About the Author

Pradeep Chintale is a Lead Site Reliability Engineer at SEI Investment Company with over 17 years of extensive experience. He has completed his bachelor's degree in Computer Science at Mumbai University. Throughout his career, he has served as a System Analyst, specializing in Infrastructure automation, Cloud and DevOps Architecture, and providing support for Application and Enterprise Infrastructure Management.

His primary expertise lies in Cloud and DevOps architecture, where he excels in designing and implementing automated and continuous build, deploy, and release processes using integrated tools within the CI/CD pipeline. He is proud to hold certification as a Cloud Solution Professional Architect.

In his current role at SEI, he continues to contribute to the field as a Lead Site Reliability Engineer. Previously, he served as a Senior Cloud Solution Architect at Microsoft, where his responsibilities included transforming technical project requirements into comprehensive architecture and design solutions. His focus has always been on establishing robust, highly available, and scalable CI/CD platforms for the fully automated deployment and operation of workloads.

When it comes to expertise in DevOps and Cloud technologies, he is recognized as a top professional in the field.

About the Reviewers

- ❖ **Nikhil Kumar** is a seasoned software engineer based in India, boasting over six years of industry experience. With a versatile skill set spanning development, architecture, and consultancy, Nikhil has worked across diverse domains for various clients and employers.

His expertise includes DevOps, Ansible, Python, CI/CD, Kubernetes, and Shell Scripting. Nikhil holds multiple prestigious certifications from RedHat, including Red Hat Certified Engineer (EX294) and Red Hat Certified System Administrator (RHCSA). As a prolific technical blogger, Nikhil shares his wealth of knowledge with the broader software engineering community.

- ❖ **Anupam Singh**, SRE practitioner leading a major project for a financial technology organisation. He has worked as DevOps and SRE across various domain industries and successfully delivered solutions. He is an experienced software developer with deep understanding of SLDC from different facets. Author to various articles on [Medium.com](#). Passionate about technology and solving people's lives through technology. He has a master's degree in MIS from Arizona State University and a Bachelors of Technology in Information Technology from GBPUAT Pantnagar.

Presently working with American Express as SRE Director for the last two years and is responsible for global payments.

Outside work, he also volunteers for a global nonprofit organization, [AnitaB.Org](#) India Delhi Chapter and organised events for women in technology.

On personal front, he loves travelling. In his free time, he is mostly with his garden plants and spending time gardening. He is also a health enthusiast and loves to explore new adventures and activities.

Acknowledgement

There are a few people I want to thank for the continued and ongoing support they have given me during the writing of this book. First and foremost, I would like to thank my wife, Prachi, for continuously encouraging me to write the book, I could have never completed this book without her support.

I am grateful to the course and the companies which gave me support throughout the learning process of DevOps Design Pattern and as it is very important to learn the tools related to DevOps Design Pattern.

Thank you for the all hidden support provided. I gratefully acknowledge Mr. Vibhu Bansal for his kind technical scrutiny of this book.

My gratitude also goes to the team at BPB Publication for being supportive enough to provide me quite a long time to finish the first part of the book.

Preface

This book covers many different aspects of **DevOps Design Patterns**, and the importance of DevOps best practices. This book also introduces the importance of DevOps Design Patterns in the field of real-time industry. It shows how the DevOps is important for the industries. This book is intended for individuals who already possess a fundamental understanding of DevOps but aspire to advance their knowledge and acquire expertise in DevOps best practices. It also gives importance to Software automation, **Infrastructure as a Code (IaC)**, **continuous integration (CI)** and **continuous deployment (CD)**.

In this book, you will learn about DevOps best practices. An important aspect of DevOps includes the implementation of CI and CD.

This book will cover a deep dive into DevOps architecture, design, and implementation of continuous build, continuous deploy, and continuous testing processes using integrated tools in the CI/CD pipeline like Github, Jenkins, Kubernetes, and AWS/Azure Cloud.

This book will provide an interdisciplinary skill set to cultivate a continuous deployment capability in your organization. The reader will be an expert in implementing DevOps best practices.

After completing this course, you will be able to:

- Explain the skill sets and roles involved in DevOps and how they contribute toward a continuous delivery capability.
- Review and deliver automation tests across the development stack.
- Explain the key jobs of system operations and how today's leading techniques and tools apply to them.
- Explain how high-functioning teams use DevOps and related methods to reach a continuous delivery capability.

- Facilitate prioritized, iterative team progress on improving a delivery pipeline

This book is divided into 11 chapters. After reading this book, readers are going to have competitive knowledge of DevOps design, architecture, and its pros and cons. This book has complete guidelines on how we can implement a DevOps best practice.

Chapter 1: Why DevOps – This chapter delves into the fundamentals of DevOps for the sake of comprehension. It explores the evolution of DevOps development and elucidates how it contributes to achieving cutting-edge technology, shortening the life cycle of system development, streamlining processes, enhancing business efficiency, lowering operational costs, and maintaining a competitive market position.

Chapter 2: Implement Version Control and Tracking – This chapter provides an in-depth exploration of version control, also known as source control. Teams contribute changes in the form of revisions, allowing the seamless merging of work at precise moments. The diverse operational possibilities within version control systems empower teams to choose methods aligned with their utilization of branching and merging features.

This section provides a comprehensive and detailed explanation of version control, outlining its benefits and the mechanisms for exercising control over it.

Chapter 3: Dynamic Developer Environment – In this study, we're exploring ways to make things easier for developers. We're focusing on flexible and on-demand environments. Software companies are interested in technologies that can be easily adjusted and expanded to make deploying software faster and more efficient. When developers are working on a new feature, it's really helpful to have a safe space to test and develop it without causing issues for the rest of the team. This way, we can be more confident about making changes and have a spot to test them before adding them to the main part of the project.

Chapter 4: Build Once, Deploy Many – In this part of the book, we will talk more about the Build Once, Deploy Many way of doing things. We will look at why it is good, what problems it might have, and the best ways to use it. This information is for developers, project managers, and others who

work on making software. We'll give a complete guide with the tools and tricks you need to use this approach well. By the end of the book, reader will have a thorough understanding of the Build Once, Deploy Many strategy and they will be ready to use it in their own software projects.

Chapter 5: Frequently Merge Code: Continuous Integration – In this chapter, we are going to cover continuous integration (CI), which is a software development practice that involves regularly and automatically building, testing, and integrating changes made to a codebase. The goal of CI is to catch and fix problems in the codebase as early as possible in the development process to prevent those problems from causing larger issues down the line.

Chapter 6: Software Packaging and Continuous Delivery – In this chapter, we will take a deeper look into continuous delivery (CD), and describe how this phase of the process is the key to achieving greater efficiency in your software development life cycle.

CD may sound daunting to teams already stretched to the limit. But once established, these game-changing practices and the automation that comes with them can take your software delivery practices to the next level. These deliveries should be frequent, carrying incremental changes to the code, which makes releases low-risk, low-stress events for DevOps teams and seamless for end-users with little or no downtime.

Chapter 7: Automated Testing – In this chapter, we will learn about how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks, along with minimizing the impact of issues and improving the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Chapter 8: Rapid Detection of Compliance Issues and Security Risks – In this section, we will learn how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks. To minimize the impact of issues and improve the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Chapter 9: Rollback Strategy – In this chapter, we will learn the importance of having a DevOps rollback strategy to minimize the impact of issues arising during or after deployment. We will delve into the key components of a successful rollback plan, including automated rollback processes, version control, testing, communication plans, and post-rollback analysis. Additionally, we will provide the best practices and tips for implementing a DevOps rollback strategy that is efficient, reliable, and minimizes downtime.

Chapter 10: Automated Infrastructure – In this chapter, we will learn how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks. We will also delve into how to minimize the impact of issues and improve the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Chapter 11: Focus on Security: DevSecOps – In this chapter, we will learn about DevSecOps. It is an approach to software development that integrates security practices into the Development Operations (DevOps) process. It emphasizes the importance of security early in the software development lifecycle rather than treating it as an afterthought.

Code Bundle and Coloured Images

Please follow the link to download the *Code Bundle* and the *Coloured Images* of the book:

<https://rebrand.ly/neoyh4f>

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/DevOps-Design-Patterns>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at <https://github.com/bpbpublications>. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at :

business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

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 business@bpbonline.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 www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Table of Contents

1. Why DevOps

Introduction

Structure

Objectives

What is DevOps

Classification

Usage of DevOps in the modern world

Advantages of DevOps

Disadvantages of DevOps

Agile and DevOps go hand-in-hand

DevOps best practices

Involvement of stakeholders in the method

Automated testing and building environments

Integrated configuration and change management

Continuous integration and continuous deployment

Continuous delivery and product support

Application monitoring and automation of dashboards

Best tools to use for CI/CD pipeline

Jenkins

TeamCity

CircleCI

Travis CI

GitLab CI

Bamboo

Semaphore

GoCD

Azure DevOps

CodeShip

Bitrise

Drone CI

Challenges that DevOps helps overcome

Overcoming the dev versus ops mentality

Common understanding of CD practices

Moving from legacy infrastructure and architecture to microservices

Implementing a test automation strategy

Too much focus on tools

Team ownership for deployments and releases

Resistance to change

Key metrics are being acted upon

Dev and Ops toolset clashes

Getting started with continuous learning

Measuring the success of DevOps implementation

Evaluate the need to implement DevOps practice

Break the organizational silos and encourage collaboration

Put customer/end-user satisfaction at the center

Do not jump start, instead, start small and then scale up

Automate wherever possible

Select tools that are compatible with each other

Define performance reviews for team and an individual

Ensure real-time visibility into the project

Integrate and deliver continuously

Achieve better results with monitoring and feedback

Conclusion

2. Implement Version Control and Tracking

Introduction

Structure

Objectives

What is Version Control or Source Control

What is source code management

Benefits of using Version Control

Version Control tools in DevOps

GitHub

GitLab

Bitbucket

Perforce

Apache subversion

Hg

How does version control work

Why is Version Control necessary

Simpler and better bug suppressing

Concurrent growth

Greater dependability of the final result

Decompose by business capability

Problem: How to decompose an application into services

Problems: How do you establish a business capability

Guidelines for using Git

Create clear, focused commitments

Commit early, commit often

Conclusion

3. Dynamic Developer Environment

Introduction

Structure

Objectives

Why feature environment

Accelerating the installation process

Information processes automation and orchestration

Coordination of operations and production organizations

What is agile in software development

Agile methodologies

Last impressions

Faster delivery

Creating a shorter route between points a and b

Send out little medals

What are our pain points

Ideal solution

Atmosphere for entrepreneurship shortcomings

Infrastructure changes

Routing of feature environments

Robotics to the rescue

Fundamental concept is a three-step procedure

Accelerated shipping

Our machining might differ

Automate deployment to feature environments

What exactly is a features atmosphere

What is the point of creating features climates

Presence of characteristic elements in all habitats

Removal of obsolete feature environments

Conclusion

4. Build Once, Deploy Many

Introduction

Structure

Objectives

Build Once Deploy Many

Library dependencies

Testing limitations

Wrapping up

Building docker image

Docker image use cases

Docker container vs. Docker image

Anatomy of a Docker image

Docker image repositories

Creating a Docker image

Interactive method

Dockerfile method

Docker image commands

Environment overlays

How do overlays affect application deployment

Continuous deployment

Continuous deployment vs. continuous delivery

Continuous deployment tools

Pros and cons

Conclusion

5. Frequently Merge Code: Continuous Integration

Introduction

Structure

Objectives

Traditional software development

Commit early, commit often

Build only once

Clean your environments

Monitor and measure your pipeline

Keep the builds green
Streamline your tests
Make it the only way to deploy to production
Make it a team effort
Best tools to build robust CI process
Step-by-step building robust and fully automatic CI process
Conclusion

6. Software Packaging and Continuous Delivery

Introduction
Structure
Exploring the benefits of CD
Increasing developer productivity
Simplifying implementation through automation
Accelerating feedback delivery
Enhancing testing quality
Expediting market introduction of new capabilities
Selecting the tools for a robust CD process
AWS Code Deploy
Octopus deploy
Jenkins
TeamCity
Deploy Bot
GitLab
Bamboo
Circle
Code ship
Google Cloud Deployment Manager
Building a robust and fully automatic CD process
Measuring considerations for calibrating a CD pipeline
Analyzing the lead time in a CD pipeline

Evaluating the cycle time in a CD pipeline

Assessing the Mean Time to Recovery

Examining defect resolution time

Understanding the test pass rate

Best practices for adopting continuous delivery

Develop Service Level Objectives

Automating SLO evaluation with quality gates

Automating every redundant process

Keeping everything in version control

Providing fast, useful feedback

Deploying the same way to every environment

Avoiding direct changes in the production environment

Deploying to every environment the same way

Deploying in a copy of production

Including the database

Eliminating complexity

Establishing observability and continuous monitoring

Conclusion

7. Automated Testing

Introduction

Structure

Objectives

Adopting test automation

Introducing tool integration

Tracking metrics

Leverage containerization

Keeping communication transparent

Saving time with headless execution

Exploring multi-layer tests

Integrating performance testing into delivery cycle

Building robust continuous testing with tools

Continuous testing

How test management tools work

Step-by-step build robust and fully automatic continuous testing

Conclusion

8. Rapid Detection of Compliance Issues and Security Risks

Introduction

Structure

Objectives

Types of continuous monitoring

Network monitoring

Why network monitoring

Benefits of network monitoring

How to use network monitoring

Configuring network monitoring

Infrastructure monitoring

Benefits of infrastructure monitoring

Configuring infrastructure monitoring

Application Performance Monitoring

Benefits of Application Performance Monitoring

When to use Application Performance Monitoring

Web application monitoring on Azure

Application Insights

Azure Container Insights

Azure VM Insights

Setup alerts

Visualize monitoring data

Dashboards

Azure workbooks
Power BI
Configuring automatic continuous monitoring
Conclusion

9. Rollback Strategy

Introduction
Structure
Objectives
Introducing rollback strategies
Manual rollback procedures
Automated rollback scripts
Snapshot backups
Version control
Hotfix rollback strategy
Feature toggles rollback strategy
Immutable infrastructure rollback strategy
Canary deployment
A/B testing deployment strategy
Blue-green deployment
Shadow deployment

Conclusion

10. Automated Infrastructure

Introduction
Structure
Objectives
Infrastructure as Code
How to choose the best IaC tool
Introducing Terraform
Pipeline as a Code

Platform as Code
Configuration as Code
Policy as Code
GitOps methodology
 GitOps WorkFlow
 Use cases of GitOps
Best tools for IaC
Conclusion

11. Focus on Security: DevSecOps

Introduction
Structure
Objectives
Principles of DevSecOps
 Automate security processes
 Collaborate across teams
Implementing security by design
 Using secure coding practices
Integrating compliance and governance
Implementing DevSecOps training and education
Integrating security into the deployment pipeline
Maintaining visibility and control
Collaboration between DevOps and security
 Threat modelling
 Compliance as code
 Container security
 DevSecOps metrics
 Securing the public endpoints
 Define policy and governance
 User right network tools to filter traffic

Define and implement IAM, RBAC and 2FA

Implementing least privilege model

Segregating DevOps network

Using password manager

Conclusion

Index

CHAPTER 1

Why DevOps

Introduction

In this chapter, the basics of DevOps are discussed for understanding purposes. The background behind the DevOps development is addressed. How this delivers the highest technology, shortens the system development life cycle, optimizes processes, improves business time, minimizes operating expenses, and retains market position is explained in this chapter.

Structure

In this chapter, we will learn:

- What is DevOps?
- Agile and DevOps go hand-in-hand
- DevOps best practices
- Best tools to use for CI/CD pipeline
- Challenges that DevOps helps overcome
- How do we measure the success of DevOps implementation?

Objectives

In this chapter, we will look into what DevOps is and what are the advantages and disadvantages of DevOps. DevOps best practices in terms of execution and implementation. We are going to cover what are the best suitable tools for making a robust CICD pipeline in this book based on real time scenarios.

What is DevOps

DevOps combines **development (Dev)** and **operations (Ops)** to integrate people, processes, and technologies in application design, development, and operations. Development, IT operations, quality engineering, and security can now coordinate and collaborate thanks to DevOps.

Organizations use DevOps culture, methods, and technologies to build quality applications on time, improve their responsiveness to user requests, and accelerate the achievement of business objectives. DevOps enables teams to continuously give value to consumers by creating more dependable and superior products.

Classification

Academicians and practitioners have not come up with a common meaning for the term *DevOps* other than that it is a cross-functional combining (and a portmanteau) of terms and notions for *developing* and *procedures*. Shared ownership, transaction processing, and efficient communication are some of the most common characteristics of DevOps. Software development and (telecom) operational processes were brought together in 1993 by the Telecommunication services based on organizational infrastructure collaboration. The inaugural DevOps days conference took place in *Ghent, Belgium*, in 2009. Belgian professional, project coordinator and agile operator *Patrick Debois* started the convention. Other countries have joined the meeting.

DevOps is a set of methods that enable enterprises to speed up innovations, deliver the highest technology, shorten the system development lifecycle, optimize processes, improve business time, minimize operating expenses, and retain market position. Automated provisioning, automation testing, automation build and deployment, and continuous evaluation are all part of the DevOps system. Development and IT operations are brought together in

the term *DevOps (Ops)*. It seeks to reduce the time it takes to design a system and produce high-quality software on a regular basis. Agile software development and DevOps go hand in hand, and numerous components of DevOps are derived from Agile.

When it comes to the management of an organization's products, DevOps is critical. DevOps supports companies in a wide range of industries, from start-ups to large multinationals, by providing everything from business strategy to operational oversight.

In the following *Figure 1.1*, the various steps for DevOps are presented:

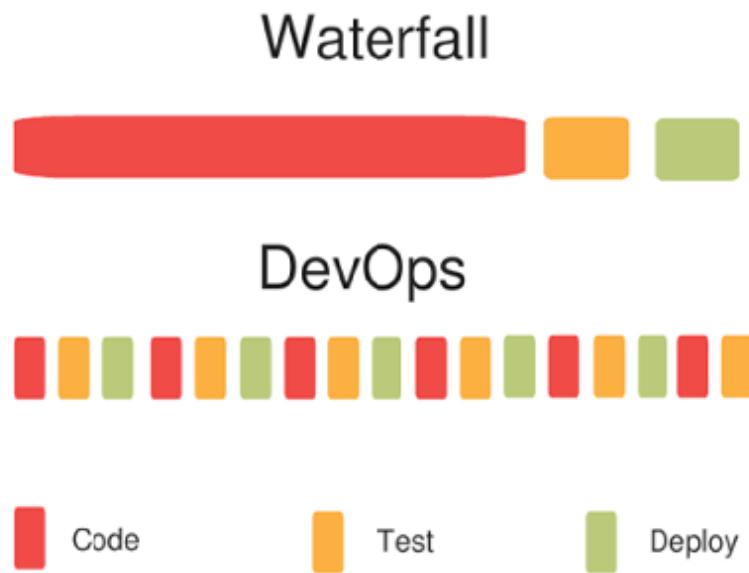


Figure 1.1: Traditional (Waterfall) s/w model vs DevOps model

The following figure shows the steps of DevOps:

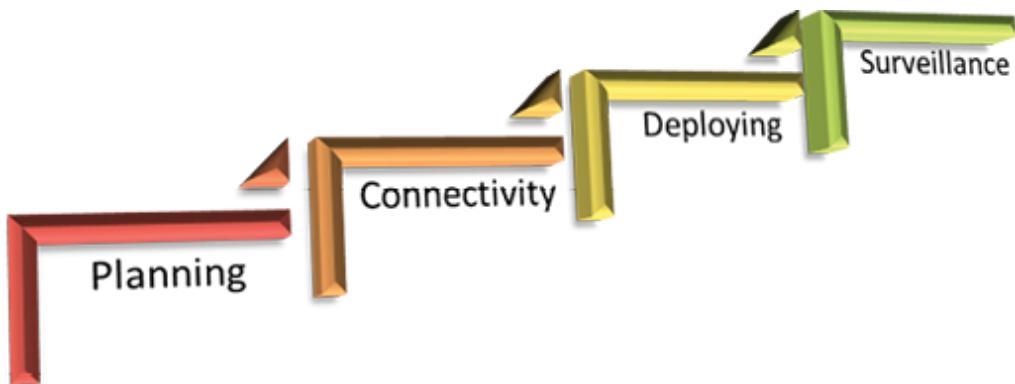


Figure 1.2: Steps for DevOps

- **Planning:** Organizing work into smaller cycles is an important first step in implementing DevOps in your firm.
- **Connectivity:** The new functions are integrated into the current programming language during this step. System testing and constant improvement are essentially the same thing.
- **Deploying:** At this point, the implementation process is ongoing. Developers will be able to make changes to the software without affecting its performance.
- **Surveillance:** As part of this phase, the production team must maintain a close eye on structural response and look for bugs.

Usage of DevOps in the modern world

Before the adoption of DevOps, teams operated in isolated units, lacking communication and collaboration between IT operations and developers. It was common for programmers to hand off their programming to the operator, who was responsible for maintaining it. Supervisors and programmers both lacked sufficient knowledge of operating processes and software packages. Consequently, as software developers prioritized the efficient release of new features, system administrators placed a higher emphasis on ensuring the long-term reliability of the system. This divergence led to miscommunication, delayed software deployments, and increased operational costs.

DevOps emerged to address the issues of software development, testing, deployment and delivery. It is a combination of strategies to remove the obstacles that inhibit designers, administrators, and other company segments from functioning together more effectively and profitably. In DevOps, the creation and deployment teams can work together more effectively because of the integration of the two groups. Regular testing of app efficiency and durability, together with automation of architectures, might help achieve these goals.

Once we understand DevOps as a Service, let us look into DevOps as a Service in more detail. At its most basic level, DaaS is a delivery model that is severe enough to require the common use of a toolkit to maintain records of all their operations on the cloud infrastructure. When you use DaaS, you

get the possibility of automating the distribution of your project's technology. DevOps engineers are in charge of overseeing the whole product development process from start to finish, making sure that nothing goes wrong along the way. For your projects, they know which technologies and tools work best and how to use them to their full potential.

Advantages of DevOps

The use of cloud services has the capability to assist a business in developing tremendously. In the following *Figure 1.3*, the advantages of DevOps are presented. Let us have a look at the benefits of DevOps in the corporate world:



Figure 1.3: Advantages of DevOps

- **Improves Business Acumen:** In an organization, corporate success depends on its ability to adapt quickly to changing conditions. Businesses can reach new heights of success thanks to DevOps. As everything is more data-driven with DevOps services, everyone on the team is using the same information. By using these technologies, businesses are able to organize and control their operations in a far more comprehensive manner.
- **Workplace that encourages teamwork:** Using the DevOps method, you may create a more stable working environment. When using a cloud-based DevOps platform, communication is significantly easier because all tools deployed in the virtualized environment can be accessed from almost anywhere.
- **Quality enhancement:** With DevOps, difficulties may be found and fixed much more quickly than with the previous way of doing things. Automated build, deployment and problem solving help early identification of issues that help deliver the product on time. The quality of a product can be greatly improved by working together efficiently between the developmental and operational teams and collecting feedback from customers on a regular basis. DevOps as a Service enables rapid testing and deployment of fresh features. The regularity of releases tends to increase when cloud services are used. Developers have access to additional computing power and data capacity.
- **Enhancing efficiency by simplifying operational procedures:** This method simplifies the understanding of information and data movement, but it can also result in team members lacking awareness of the holistic operation of the complete toolchain. IT managers can use managed services tools to make changes, while software developers can utilize source code administration solutions to test their work. DevOps experts are immediately available to construct projects using DaaS, allowing them to go to work on your assignment right away.

Disadvantages of DevOps

Testing of operational processes in the cloud's manufacturing process is more difficult and raises questions about connectivity. Interoperability, technology, and workflow coordination require a deep understanding of the subject matter. In DevOps, efficiency takes precedence over security.

- A defect can be found in any technology or system, and there is no one-size-fits-all solution to our problems. Even though it is an excellent idea, DevOps as a Service is far from perfect.
- Companies will need specialized personnel to help them implement and operate new approaches during this stage.
- A detailed understanding of DaaS products' interaction with the business framework, infrastructure, and operating methods is essential.
- DevOps places an emphasis on speed over security while developing software. Many different types of safety issues might arise as a result of utilizing cloud solutions.
- Keep an eye out for security lapses as your company implements DevOps.

Agile and DevOps go hand-in-hand

Agile and DevOps both aim to improve the efficiency with which they deliver value to the end user, but they do so in various manners to each other. DevOps brings the technical team into play to allow continuous development and continuous integration to production, while Agile focuses on making programmers and release schedules more efficient. Is it possible for Agile and DevOps to collaborate in order to improve any process? They can, of course. It's easy to mistake Agile and DevOps as rival approaches that cannot coexist. It is still possible to improve your manufacturer's maintenance and growth by integrating these two ways.

Irrespective of the purpose of the code, agile approaches such as continuous integration are an excellent way to disseminate information and enhance code quality. Using these techniques can result in brand-new functionalities, bug fixes, or infrastructures as code in your distribution pipelines. Command group for and management of the production pipeline and other associated equipment is an important part of DevOps.

There are many advantages to using pair programming as a means of exposing and educating developers. Most software projects can benefit from continuous delivery approaches, such as progressing through a requirements document or developing internal tools. All agile methodologies benefit from faster turnaround times and quicker feedback from operations. This is essential for an agile project's success. Iterative software development emphasizes feedback from the consumers gleaned through actual use. Software that is released to customers more quickly leads to faster change cycles, which in turn speeds things up. Your organization's velocity and technology deployment frequency will be slowed by adopting agile without including DevOps techniques helps in the following areas:

- DevOps and Agile connectivity.
- Increases the quality of the products on the market and simplifies the release procedure.
- Makes more effective teamwork possible.
- There is more benefit and fewer dangers with every release.
- Fixes that are more rapid and less prone to error.
- Arising from exposure to the public.
- Quality products lead to higher customer contentment.

Smaller teams working together to respond to ever-changing customer requirements is the focus of Agile, while DevOps brings together two huge, segregated groups for rapid application deliveries. In all methods, the idea of shifting to the left is a common theme. In order to improve the quality of software, it is necessary to discover and resolve flaws as immediately as possible in the software creation process.

Both DevOps and Agile have their strengths and weaknesses. The focus of DevOps is on the automation of the concurrent engineering and production pipelines in order to enable more frequent releases, whereas Agile promotes flexibility in response to shifting needs and contractual relationships among smaller clubs. The key difference is identified in *Figure 1.4*:

DEVOPS VS. AGILE

DevOps	Parameter	Agile Methodology
Emphasis on collaboration & productivity	Philosophy	Emphasis on incremental changes through iterative development & testing
Continuous testing & integration, providing end-to-end business solutions	Focal Point & Purpose	Incremental deployments in complex projects
Looks after secure deployment	Delivery & Deployment	Looks after developing & launching software
Large team with different skill-set	Team Size & Skills	Smaller team with advanced skill-sets
Extensive documentation	Documentation	Light on documentation
Received via customers	Feedback	Received internally
Specs & design documents	Communication	Daily scrum meetings

Figure 1.4: Differences between DevOps and Agile

In the following figure, the DevOps process is shown:

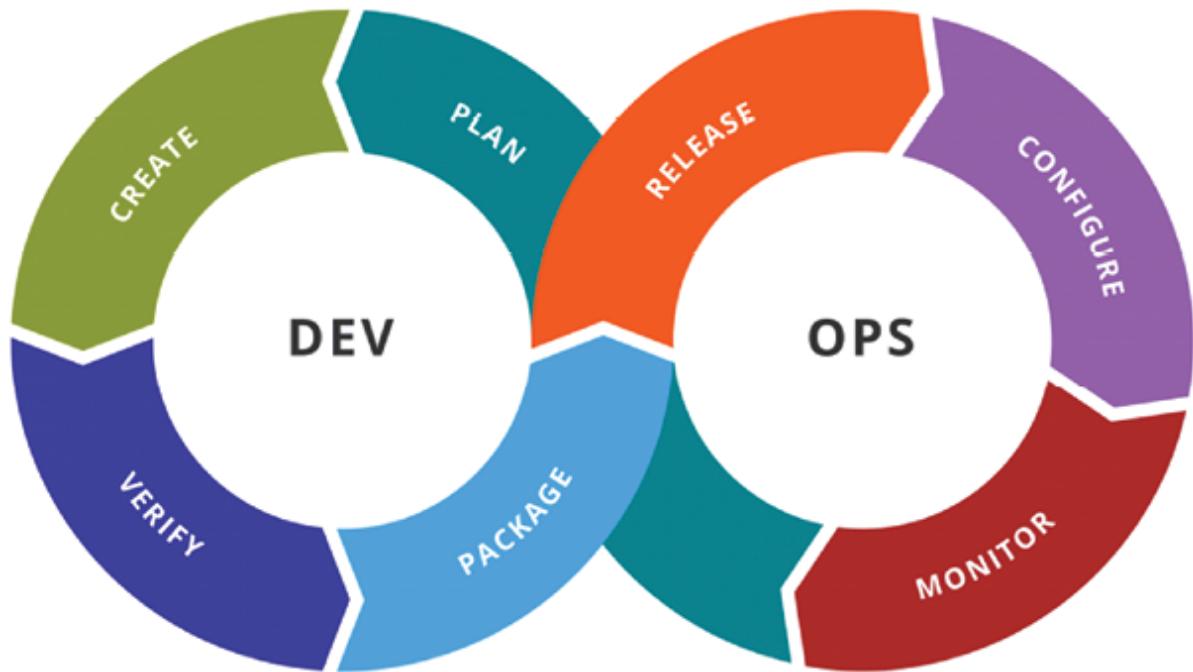


Figure 1.5: DevOps

The agile process is showcased in the following figure:

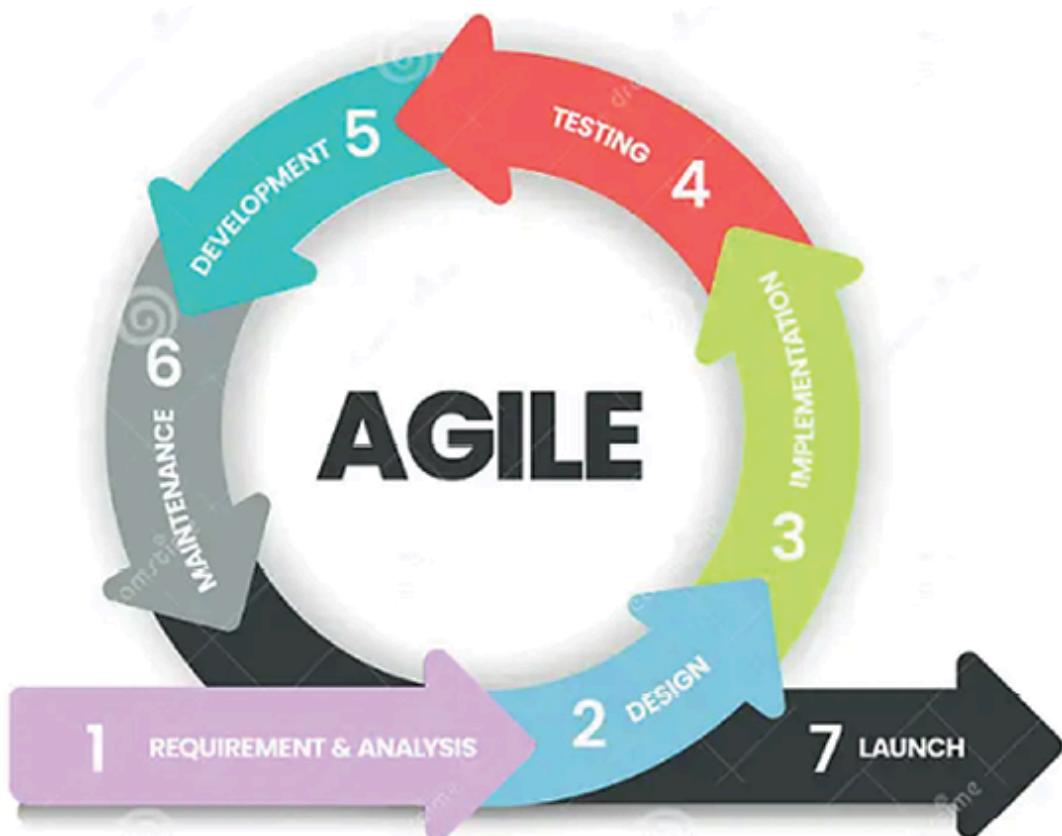


Figure 1.6: Agile process

DevOps best practices

DevOps best practices give a variety of proposals and implementations to adopt DevOps successfully across enterprises and software telecommunications companies.

Involvement of stakeholders in the method

Producers, administrative workers, support personnel, and other constituencies all play an important role in DevOps processes. Developers are encouraged to collaborate closely with the organization and its support staff and processes by the use of the term *onsite consumer*, which was coined by **Extreme Programming (XP)**. The success of a project is considerably enhanced if all parties are involved.

Automated testing and building environments

The software must be tested by programmers on a regular basis in order to provide high-quality code. Using DevOps, developers have the chance to detect and resolve difficulties in the early stages of the development process. Automated testing, in contrast to penetration processes, allows the software development lifecycle to be completed more quickly and is an important part of code development, middleware settings, and data and communications modifications. When virtualized infrastructures are deployed, automating the environment-building process is vital; manual environment-building runs the danger of necessary conditions being overlooked or returned to legacy solutions.

Integrated configuration and change management

Using integrated continuous integration, software development teams can make use of pre-existing technologies instead of having to keep inventing new ones. Strong and durable are monitored and maintained across all of these components, as well as extra professional services such as storage and server services. During change control, agile methodologies come into play whenever there is a requirement for a modification. In order to effectively manage change, it's critical to consider the potential implications and

possibilities the transformation may present, as well as how it might affect other organizations.

Continuous integration and continuous deployment

Continuous integration refers to the process of updating the repository on a regular basis. With this tool, programmers may identify integrating difficulties early on and take proactive measures to address them, resulting in better teamwork and higher-quality software. Automated introduction of code in the production environment is a component of ongoing installation. In order to reduce the time seen between the discovery and development of new functionalities and their implementation in operation, there are a number of solutions accessible that can be used for system implementation.

Continuous delivery and product support

DevOps uses continuous delivery to reduce the risk of product failure, boost system throughput, and reduce the amount of administrative effort required. The QA team uses both manual and automatic ways to evaluate the newly developed code, and after it passes all of the tests, it is certified for deployment. Consequently, software is created in short iterations that are iterated frequently to ensure high quality. Change has been a successful strategy used by developers to assist them in comprehending and improving their software conceptual design.

Application monitoring and automation of dashboards

Maintaining the app's infrastructure, which comprises network services, web applications, and telecom services, is a requirement for enhancing the app's performance.

As a result, proactively addressing and testing the development of applications using various technological tools is critical for the operations and creation teams. Autonomous Dashboards provide authentic information and information for each operation, as well as a knowledge of which automated software techniques are most suited for the job. Network operators now have a complete picture of the program's improvements.

In order to deliver products of the highest possible quality, software engineering businesses adopt DevOps principles to provide visibility and

efficient communication between development and operations teams. DevOps Best Standards can be used by enterprises and IT service providers to design and produce robust development services.

Best tools to use for CI/CD pipeline

Software development and DevOps testing have both benefited greatly from the adoption of CI/CD, also known as **continuous integration and delivery (CI/CD)**. As a result, it gives programmers the tools they need to continually release their code. Furthermore, the continuous submission of source code prevents structural problems by detecting defects early on.

If you are looking for the finest CI/CD tools for your budget and production needs, there are a lot of options. We believe that this list will help you choose the finest CI/CD tool for your company. Because there are numerous options available, it can be tough to pick the best Continuous Deployment technology for your project.

Following is a real time example of how the CICD workflow works:

- Infrastructure Engineers employ tools such as Jenkins and Ansible for constructing infrastructure and platforms, which encompass the CI/CD pipeline.
- Subsequently, developers utilize the CI/CD pipeline to construct code and execute deployments within the development environment.
- Through automation, once developers create code within the development environment, it is automatically deployed to the testing environment.
- Following this, the QA team conducts further code testing using automated tools.
- Upon successful testing, the code is deployed to the production environment, where it is maintained by the support team.
- This entire process is orchestrated by a team of engineers who are primarily developers.

Thanks to this automation, developers engage in regular code building and deployment activities. The entire process is built upon a platform that

provides visibility to development, QA, and operations teams, allowing them insight into the timing and content of code as it progresses into production.

Jenkins

Jenkins is an open-source programming language CI/CD tool. It allows for both agile methodology and delivery because of its built-in support for both. Real-time monitoring and monitoring are also possible, thanks to this.

Installing Jenkins is as simple as downloading the `.war` file and running it from the console.

Modeling delivery pipelines *as code* is possible with Jenkins Pipeline's suite of computational methods. DSL is used to build the pipeline in Jenkins (Domain Specific Language). Because it is open-source and has been around for so long, it is one of the best CI/CD tools. This Jenkins Tutorial might assist you in discovering more about the Jenkins utility.

AWS, Google Cloud, Azure, and Virtual Ocean are just a few of the cloud systems it may connect to. It can be used to handle multiple tasks at once and meet even the most demanding CD specifications.

Because it is a standalone Software component, the war installation works right out of the box. Visit the Best Jenkins Pipeline Tutorial for Beginners post if you're a newbie and are looking to learn more about the Jenkins pipeline. Using Jenkins, one of the greatest CI/CD systems in DevOps, is the goal of this tutorial for both novices and experts alike.

TeamCity

Server-based CI/CD pipeline tool written in Java, TeamCity. In addition to PyCharm and IntelliJ Idea, JetBrains is responsible for this system's life cycle and upkeep. It can be installed on a Windows or Linux server and can be accessed remotely. With TeamCity, good groups may easily link with Azure DevOps and Jira Software Cloud, which are both free to use for open-source development.

Following are the features of TeamCity tool:

- Salient features of TeamCity.
- Sub-project settings can be reused from the parent project thanks to a system that is very customizable and adaptable.

- In addition to being able to run several builds continuously, it also provides the ability to run builds on different builds and surroundings instantaneously.
- Kotlin-based DSL is used to define pipelines in TeamCity (Domain Specific Language). All of these tools can be used in conjunction with one other.
- There's an on-premise version that works with Google Cloud, **Amazon Web Services (AWS)**, and also more. It has sophisticated capabilities that allow you to run history, check test progress (and history) reports on the fly, and add versions to your list of favorites.

CircleCI

Fully accessible and large-scale projects can benefit greatly from CircleCI, a CI/CD solution. If you're looking for an on-premise (or self-hosted) CircleCI solution, CircleCI Server is your best option. Its sensitivity and awareness can be created on Windows, Linux, and macOS. A unique YAML format is used for pipelines, making it simple to set up and use. CircleCI was recognized as a Cloud Native Project Collaboration Leader by Forrester Wave in 2019.

Splitting and balancing builds over many containers helps speed up build times.

CircleCI's parallel screening makes it possible to execute tests on many executors at the same time. Timing data can be used to split tests, which further cuts down on test processing time. There are a number of prominent third-party technologies that can be linked with CircleCI Server, including GitHub Corporate, Lambda Test, and Protective Gear. AWS, Google Cloud, Azure, and more are all supported by CircleCI Servers.

Travis CI

CI/CD pipeline tools like Jenkins and Travis CI have been around for a while. Open-source projects were the first to benefit from it, but closed-source ones were added subsequently. For accessibility and builders, Travis CI is one of the top CI/CD systems. The accessible communities can use Travis CI for free, whereas businesses who want to utilize Travis CI on their

own cloud environment can use it for a fee (or self-hosted platform). The blog CircleCI versus Travis CI examines the differences between the two best CI/CD tools.

GitLab CI

A built-in tool in GitLab is GitLab CI. As one of the most comprehensive dashboards for CI/CD technologies, it includes a wide range of capabilities like technical reviews, CI/CD, agile methodologies, and more. A YAML file called `gitlab-ci.yml`, that needs to be included in the admin area, specifies the scripts needed to do a build, test, and distribution of the application using GitLab CI/CD. It is referred to as a pipeline since the programs are bundled together into jobs. The GitLab runner tool is able to identify and run various programs.

When jobs are ongoing, the GitLab CI cache method saves time. Caches can be shared between branches, as well as between separate sections, and they can be disabled for certain jobs as needed. Leverage these options to your advantage, taking into account the various GitLab CI choices at your disposal, based on your specific needs.. It is possible to execute many GitLab CI jobs at the same time. In addition, a custom pipeline can be defined. GitLab CI is a simple replacement for Jenkins or CircleCI. Using GitLab CI's shell operator, you may start a build from the command line. Automating Testing Using GitLab CI/CD and Selenium Grid is an easy process.

Bamboo

The **continuous integration (CI)** tool Bamboo is a widely used CI. Build, test, and deployment are all done in a single window with the help of *Atlassian*, the company behind the development of the popular Jira project management tool.

Using Jira and popular SCM solutions like Bitbucket is a breeze. This is also accessible on mainstream operating systems like Windows, Linux, and Mac OS X. There are different programming technologies and technology supported by the Bamboo CI/CD pipeline, including AWS, SVN, and Git:Atlassian Bamboo is a continuous integration and delivery (CI/CD)

server that provides various features to streamline the software development and release process.

Here are some key features of Bamboo CI/CD:

- Build Plans: Bamboo allows you to define build plans for your applications. Build plans specify how your source code should be retrieved, built, and tested.
- Integration with Source Code Repositories: Bamboo integrates with popular version control systems such as Git, Mercurial, and Subversion, allowing you to automatically trigger builds based on changes to your source code.
- Artifact Management: Bamboo can manage and store build artifacts, making it easier to track and deploy specific versions of your software.
- Build Agents: Bamboo supports the concept of build agents, which are responsible for executing the build plans. This allows for distributed builds across multiple machines.
- Deployment Projects: Bamboo facilitates the deployment of applications by allowing you to define deployment projects. These projects specify how artifacts should be deployed to various environments.
- Continuous Deployment: Bamboo supports continuous deployment, allowing you to automate the deployment process and deliver software to production or staging environments with minimal manual intervention.

By conducting builds on distant build agents, it is possible to achieve simultaneous builds. Remote build agents and simultaneous test batching can be supported on up to 100 agents. Self-hosted and cloud-hosted versions of Bamboo are both commonly accessible.

It is possible to send push alerts from Bitbucket after a build is triggered by modifications in the repositories.

Semaphore

On the Cloud, Semaphore is one of the top relationship and Agile methodologies technologies. Only Semaphore's CI/CD solution offers robust support for mono-repo applications right out of the box. With GitHub integration, this is yet another excellent CI/CD solution.

software engineers may speed up the testing and deployment of their apps with Semaphore's CI/CD pipelines. Both parallel and sequential builds are supported with Semaphore's customizable pipelines, as they are with other top CI/CD systems. The Semaphore CI and Selenium Grid CI/CD pipeline for automation testing can be built with the help of this article on developing an automation testing pipeline.

Salient features of Semaphore:

- Semaphore can automate Linux, Android, and macOS development, test, and installation processes.
- To deploy to Kubernetes in the cloud quickly, you may use Semaphore to conduct continuous integration and agile methodologies on any Docker image.
- C#, Python, Java, PHP, Ruby, Rust, iOS and Android apps are all supported.
- GitHub and Bitbucket are seamlessly integrated into the platform.
- Semaphore's CLI for analyzing logs can debug errors in a matter of minutes. With SSH, it is possible to see what jobs are currently running and how long they have been operating.
- Using Inspection Reports, you can keep track of how your team is performing the tests as they progress through their software development assignments. The information your team needs will be readily available if tests are done simultaneously.
- Using personality interfaces, you have complete control over the way your jobs are executed and the ecosystem in which they run. With identity interfaces, you have total command over the equipment, desktop, and utility programs in your pipeline.

GoCD

Thought works' GoCD is an open-source Stream Processing system. The Continuous Improvement Map (VSM) feature distinguishes it from other CI/CD DevOps solutions. The *Implementation Pipeline* or *Cumulative Delivery Pipeline* idea is satisfactorily mapped to VSM's end-to-end view spanning infrastructures.

- To reduce dependencies across organizations, pipelines can be chained.
- Salient features of GoCD.
- GoCD's dependency configuration is simple.

Like other CI/CD pipeline technologies, GoCD enables you to create *pipeline as code* in both YAML and JSON forms. VSM in GoCD provides real-time visualization of the workflow's implementations from beginning to end. Even though GoCD's plugin community is not as comprehensive as Jenkins', it is continuously growing.

- GoCD takes care of user identification and consent.
- If you are looking for the best CI/CD tool, go no further than this one.
- Similar to other CI/CD technologies, GoCD's requirements may be easily set up. Parallel processing is essential for DevOps development, and this tool has it.

Azure DevOps

Microsoft Azure's Azure DevOps provides an easy way to create a CI and CD pipeline to Azure. DevOps services like Azure Pipelines, VSTS and Software Transportations are all included in this set of capabilities. Your Selenium tests can now run in the Azure DevOps infrastructure. The following are additional features that are designed to speed up the software creation process:

- With the help of this blog, you can get acquainted with the Azure DevOps CI/CD pipeline.
- Salient features of Azure DevOps.
- Continuous Integration/Continuous Delivery is a breeze with this system.

- Offering capabilities such as a versioning system, code repository administration, build automation and interaction with Microsoft Visual Studio Team Solutions.
- It is also compatible with a wide range of scripting languages and applications kinds.
- It is possible to deploy the program to many platforms, including Virtualization Technology and Capsules.

CodeShip

As a hosted Continuous Integration, CodeShip is the perfect choice for developers. When it comes to version control, CodeShip might be a great asset if your company uses GitHub. For the most part, this is because GitHub repositories may be used to test, create, and deploy applications.

You can create an endless number of versions for free each month thanks to the freeware pricing system. The CodeShip UI and preconfigured infrastructure make it simple to have builds and migrations up and running in a matter of seconds.

Salient features of CodeShip:

- Control over the design of CI/CD systems can be tightened up using CodeShip by programmers. Additionally, users can tailor the surroundings and process to their needs. Vulnerability screening tools, on-premise SCMs, installation, and alerting tools are all supported.
- A simple online interface enables establishing CI/CD incredibly simple. Assigning credentials to various groups inside an organization is made much easier thanks to this feature (or a set of team members). SSH can be used to debug the builds right in the CI framework. Because the Docker image does not have to be rebuilt every time, the CI/CD system goes more quickly when Caching is used.
- Builder processes can be easily created using multi-stage builds. It also helps reduce the final Docker image's screen resolution.

Bitrise

Bitrise is one of the best CI/CD solutions for designing, testing, and releasing smartphone applications quickly since it offers calls or messages. If you do not want to purchase overpriced infrastructure, you can use this cloud-based CI/CD tool. The software is also free and open-sourced to use for independent mobile app creators and open-source organizations.

It is compatible with both Mac OS X and Linux, so you may use it on either platform. As an added bonus, Bitrise supports a diverse set of computer languages, including popular mobile app development technologies like Kotlin and Swift as well as more traditional desktop cultures like Objective C, Cordova, and Ionic.

Salient features of Bitrise:

- GitLab, GitHub, GitLab Enterprise, GitLab Enterprise, and Bitbucket are all supported by Bitrise's integration with the various Git services, including Bitbucket.
- The Bitrise CLI facilitates the execution of Bitrise processes on a local system. The new infrastructure can be automated using a single instruction at the terminals.
- If you are seeking a CI/CD platform that is secure, flexible, and highly available, Bitrise has a solution for you.

Drone CI

In the world of CI/CD pipeline tools, Drone CI is one of the greatest and most cutting-edge. With it, not only is Scrum Methodology provided, but so is a CI/CD pipeline that is disseminated. Other accessible best CI/CD tools, including Jenkins do not have this functionality. You may use Drone's streaming server pipelines engine to automate build, test and deployment procedures.

When it comes to CI/CD, Drone is one of the greatest CI/CD tools out there. When using Drone CI, each pipeline step is completed in a separate, isolated Virtualized environment. Windows, Linux, and macOS are all supported platforms. The ARM microprocessor is also supported (predominantly used in mobile phones). Drone Professional can be implemented on major platforms such as Cloud, Open Stack, Amazon EC2, and more, while the Cloud (or open-source) version is free to use.

Salient features of Drone CI:

- Docker instances segregate pipelines, ensuring that builds do not interfere with each other.
- GitHub, Bitbucket, and GitLab are just a few of the prominent SCM solutions that Drone CI (Cloud and Enterprise) can easily interface with.
- It is possible to run Drone CI in a virtualized environment with any computer language, database or service.
- Integration with Lambda Test allows for concurrent development and testing as well as cross-browser experimenting.
- Installing Drone CI is as simple as grabbing and installing the official Virtual machine. For accessible applications, Drone Cloud is completely free.

Challenges that DevOps helps overcome

DevOps methods have been used by many IT Ops and software teams to assist them in becoming more flexible and innovative than they were before. Firms who have not yet adopted Methodologies are feeling the heat from the competitors to do so. Development can be difficult at first, but it's worth it in the long run if you stick with it. We're going to look at the top ten obstacles that organizations experience when attempting to implement DevOps.

Overcoming the dev versus ops mentality

When it comes to software development, the classic adage goes something like this: developers throw code over for a hypothetical wall to a centralized engineering team, and the operations center scrambles to keep up with rapid innovation and change.

The goals of these two groups are often at odds, generating changeovers and greater costs, as well as lengthier chain reactions, as a consequence of the conflict. When it comes to implementing DevOps, the goal is to integrate teams and eliminate organizational silos. The first step is to lay up a plan for the way this will benefit your company. The first step in implementing Methodologies is for a company to identify where development traditionally

ends and operations begin, and then to figure out how to connect the two functions as effectively as possible.

Common understanding of CD practices

As soon as you have figured out that a phased approach is necessary to reduce feedback, you must consider the following question: Is it possible to do this with pipelines and CI tools? Whenever uploading to the manufacturing environment, you can use this method to ensure that your construction piece is in a usable and clean state.

A consistent knowledge of what it means to constantly provide and how to attain sustainability can be established by clearly defining Functionalities within your organization.

Moving from legacy infrastructure and architecture to microservices

By moving from a legacy monolithic architecture to microservices, organizations can achieve greater agility, scalability, and resilience in their applications, while also benefiting from a more efficient development and deployment process. Using a modular approach in a cloud-native ecosystem can lead to faster development and quicker innovation. Furthermore, the increasing process effort that cryptocurrencies offer needs a stable platform around automating, configuration management, and constant delivery methods.

Implementing a test automation strategy

Automated tests are critical to DevOps processes, particularly Continuous Integration and Continuous Delivery (CI/CD). Not only should the test approach be stated, but it should also be demonstrated in practice to serve as a north star for the development teams.

Streamlining feedback loops and speeding up product release are both benefits of a well-thought-out test automation approach that can be implemented across an organization.

Too much focus on tools

New DevOps technologies may appear to address all of a company's problems, especially in light of the enticing potential of implementing them. To ensure that new tools are secure and well-integrated with the power network, you must first train your employees on how to use them.

Your team may suffer as a result of being distracted by all of this.

DevOps relies heavily on your team and organizational architecture. The team's procedures will fall into place once you've established the proper framework. Afterwards, you will be able to figure out what tools are needed to satisfy the requirements. When making the switch to DevOps, it's crucial to have the right individuals on your team. Confusion about the newly deployed methods and tools will impede the adoption of DevOps principles if they are not trained correctly.

Team ownership for deployments and releases

DevOps principles have been introduced in many organizations, yet we still see teams lacking full control over their software's development and production cycles. Deployment and publishing are two different things, although many people confuse the two. If the team starts working closely with any ops people and assessment and management for installations, upgrades, and operating, then there will be a common context between the two. This gives developers a better understanding of what it takes to effectively launch and deliver their code to operational, for example, alongside ops teams. DevOps principles may be adopted by the rest of the team with the support of this context, allowing the team to take possession of not just installations, but also launches!!

Resistance to change

Some members of the team and important stakeholders may find the transition to DevOps intimidating. If you package it as an evolutionary rather than a revolutionary, you'll be able to address that issue better. In some minds, telling someone that they need to improve their situation is an attack on the one who is listening. To be clear, a DevOps transition cannot happen suddenly; it must be steady and fluid. As people get used to the DevOps way of working and see how they can make a difference, they are better able to accept it. It's only when teams witness the advantages of fresh working

practices in action that other teams desire to embrace them as well. DevOps is a brand new environment, and it is going to take some time for everybody to get used to it.

Key metrics are being acted upon

Many organizations have begun compiling various metrics since statistics and facts are difficult to refute. However, organizations can focus too much on collecting! A whirlwind of analytics and monitors might be easy to fall into. Collecting basic data could be tedious and time-consuming even when undertaken in good faith.

The DORA metrics can be collected and made available to teams, with agreed-upon (and achievable!) actions on how to enhance these metrics for that specific team, as one method of achieving this goal(s). This focused and concentrated strategy can help these organizations to begin adopting technologies and systems that can subsequently actually facilitate and embed DevOps culture throughout the organization.

Dev and Ops toolset clashes

Problems might arise when the development and operations teams use different sets of tools and KPIs. Despite how simple it may appear, it is essential to meet down with both teams and attempt to determine which tools they use and the statistics they observe make complete sense to combine. Sometimes, teams may be reluctant to give up on outmoded tools, even if they are inferior in terms of technology and cause infrastructural slowdowns owing to compliance difficulties. Do not let the implementation of new tools detract from your main purpose by making sure they are aligned with your organizational objectives. DevOps adoption will be a lot easier if you can overcome these initial hurdles. Teams can adapt to the feeling of perpetual evolution and innovation over time. It is only when the various teams learn to work together that they will discover new ways to aid themselves and collaborate even closer.

Getting started with continuous learning

Curiosity is a powerful drive for many people to begin their educational journey! Curiosity to learn, adapt, and develop one's abilities and

understanding is by far the most critical enabler for a team to begin implementing a DevOps attitude. One approach to accomplish this is to make sure that teams have access to a platform that allows them to learn and collaborate. There are a number of ways to do this, such as through monthly community of practice, or lunch and learn events, or the adoption of guilds across organizations.

If your company is large or small, you can still use a train-the-trainer methodology to guarantee that appropriate procedures are passed down to all teams involved. It does not matter which strategy you pick to create a culture of constant learning; the initial little step can lead to a great jump in the adoption of DevOps techniques. Apart from the obstacles mentioned above, there are a number of levers that may be used by organizations to guarantee that new methods of operation and a DevOps mindset are implemented, whether it is from the top down or from the bottom up.

Measuring the success of DevOps implementation

In this section, we will assess the success of DevOps implementations. A foundation's ability to produce high-quality, rapid, and trustworthy software relies on the culture of teamwork and collaboration that exists throughout its many departments when using DevOps principles. Cultural shifts and modifications in historical program patterns are required. As a result of these standard practices, a company may ensure collaborative efforts, seamless procedures, and code that is free of defects. The following are the ten most important DevOps best practices:

Evaluate the need to implement DevOps practice

Align your IT objectives with those of the organization. Businesses should be the ones driving the deployment of DevOps. Your corporate objectives should necessitate this adjustment in your project lifecycle, not due to the recent fashion.

Break the organizational silos and encourage collaboration

The practice of DevOps necessitates the dismantling of traditional IT silos. Development, administration, and other departments must work together closely in DevOps, according to the underlying principle of the term.

Breaking down organizational silos and fostering collaboration is a pivotal strategy for enhancing overall efficiency and productivity within a company. Organizational silos refer to isolated and compartmentalized divisions within an organization that hinder the free flow of information and collaboration across different departments or teams. Encouraging collaboration involves creating an environment where individuals from various parts of the organization can work together seamlessly.

Put customer/end-user satisfaction at the center

To stay up to the needs of their customers, businesses must adapt and provide offerings that not only meet but also surpass their requirements in terms of speed, effectiveness, and quality. In order to do this, the company must adopt a new culture that prioritizes teamwork, openness, and a focus on the needs of the client. DevOps cannot succeed even without backing all of the company's stakeholders involved. All should be participating in every step of the process, from developing the specifications, prototyping, component testing, and implementation.

Do not jump start, instead, start small and then scale up

DevOps should be implemented at first at a limited scale in order to achieve short development cycles. The new approach's credibility is bolstered by a few great accomplishments. The new IT approach must be trusted and accepted in order to move IT culture away from silos. But instead of bringing in new employees in from the outside, companies ought to concentrate on training their own employees and enhancing their current workforce. As a result, current staff are more inclined to feel confident about implementing DevOps in the future.

Automate wherever possible

In order to stay up with the speed of DevOps, automated is necessary all throughout SDLC. Code construction, application administration, database and communication modifications, and key testing like as debugging and load testing can all be automated. Using automating, programmers, testing, and operators can save energy and time while also lowering their overall expenses.

Select tools that are compatible with each other

DevOps automated testing tools should be chosen based on how they interact with one another. Choosing a tool set that is suitable for your IT environment is suggested is essential. Make sure you're using tools that are compatible with the remainder of the existing toolchain. The overall tool interoperability of your establishment should indeed be taken into account when making design and manufacturing decision making. The best results can be achieved if the tools you employ are all from the same vendor, as these tools are likely to be tightly connected. Choosing the right tools can minimize the potential for friction between and operations.

Define performance reviews for team and an individual

The monitoring of both group and individual effectiveness in the team is required when IT culture must be cooperative. DevOps emphasizes teamwork and cooperation. Therefore programmers and technical teams should be evaluated primarily on their capability to achieve their organizations' creation and deployment objectives.

Ensure real-time visibility into the project

Project management software that preparatory stage transparency into a project or app is critical for a bridge IT organization. It simplifies the project collaboration among several departments. All parties involved in the project must be aware of exactly where the program is in its progression from conception through implementation. By identifying who and what the most critical project components are, automated project solutions make it easier to locate the data you need.

Integrate and deliver continuously

A DevOps strategy that does not include system integration and software integration is a waste of time and money. System testing is a critical element of agile procedures, allowing developers to develop technology in small, regular stages by identifying flaws and delivering prompt feedback.

There is an extension to agile development, which is continuous deployment. Every new or amended requirement may be promptly and safely delivered to commercial with assurance by providing each and every modification to a

manufacturing environment and by ensuring that the software or app performs as expected through thorough automated testing. Automation testing ensures that the software does what it is supposed to. DevOps installation can only be accomplished if Integrated Software and Configuration Management are not overlooked.

Achieve better results with monitoring and feedback

Ongoing monitoring is necessary to ensure that the software or app is working as expected in an unchanging atmosphere. To guarantee that the apps are working at their best, the operations team must be involved. Analysis and monitoring technologies may be incorporated directly into apps being developed by working with the developers.

To summarize, the ideas and methods of DevOps empower a company to keep its software delivery lean and efficient, while taking advantage of comments from end-users to continually enhance the quality of their products. An application's delivery process can be improved by implementing a feedback process. Continued growth of an institution's procedures and culture through DevOps is not just an effort, but a long-term journey toward better customer service and commercial consequences. If you would like to learn more about DevOps and adopt the DevOps concept, please get in touch with us. Agile and DevOps progression solutions are geared toward assisting individuals to find major areas for improvement, benchmark and align their Agile/DevOps procedures to the side of the business solutions; design a target organizational structure to achieve maximum investment opportunities; and offer a roadmap to adopt Agile methodologies to achieve high strength of character and adoption of the techniques and procedures A DevOps organization that is agile.

Conclusion

In this chapter, we have learnt what is DevOps, what are the advantages and disadvantages of DevOps. We have discussed DevOps' best practices and implementations. We overviewed which are the best suitable tools to implement the CICD pipeline.

In the next chapter, we will learn to implement Version Control and Tracking, where we will describe how Git flow works.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 2

Implement Version Control and Tracking

Introduction

In this chapter, the version control or source control will be explained. A version control system keeps track of modifications made to the system's files. Source code, resources, and other materials that might be included in a software development project may be located in these folders. Groups of changes known as contributions or revisions are made by teams.

Additionally, it will make it possible to merge that work back at the precise moment. There are numerous operating possibilities. The team can choose the method based on how they intend to use the VCS's branching and joining features. In this section, the version control is well explained. The benefits of version control are explained along with the control of it.

Structure

In this chapter, we will learn about the following topics:

- What is version control or source control?
- Benefits of using version control system(VCS)
- Version control tools in DevOps

- Why is version control necessary?
- Decompose by business capability
- Commit early, commit often

Objectives

In this chapter, we will learn about DevOps and version control. The various changes and backup are the learning objectives. We will also learn the benefits of version control.

What is Version Control or Source Control

Consider your files to be a book. It has beginnings, medium and end, sections and sheets. When using Agile, facilitates comparisons, also known as configuration files, enables you to update anything while still preserving the entire cycle and collaborating with the crew.

A **Version Control System (VCS)** is a way to make changes to files without worrying about something that will get lost or things will fall out of the flow. Version Control also offers backup and history of any changes for any files line-by-line.

Resource Control is fundamental to the Docker process. In Agile methodology, access controls, also known as DevOps, aid in managing major changes during one program's design process. Code variants, documents, or even environment configurations may be involved.

What is source code management

System Software Organization, also known as SCM, is a DevOps automation tool that keeps track of providers and service (revisions) crafted. Every revision includes a timestamp and the change-maker. One may contrast and combine these variants. The term *SCM* also refers to source code.

Benefits of using Version Control

The process of monitoring and managing modifications to software code is recognized as version control, commonly referred to as source control. Software technologies called version control systems assist software development teams in tracking changes to source code over time:

- Provides a thorough history of each file that has been created by a person throughout time. The addition, destruction, and editing of files are among these modifications. The writer, the time, and a comment explaining why the modification was done are also shown in history. In the future, this will make it easier to identify the situation's primary cause.
- The programming language is shielded by VC against unintentional events, human error, and inadvertent modifications. The source code is the most valuable resource for any given project. Protecting it is essential because a development's source code provides all the information and modification information linked to it.
- The ability to merge and branch. A VCS branches will maintain the separation of different processes. It will also provide the ability to combine that work back at the exact time. There are several operational options available. According to how they want to employ the branches and joining features of the VCS, the team may pick the approach.
- Understanding the distinctions among different versions and the rationale behind the alterations is made easier by using source control or versioning.
- In big, dispersed teams where engineers work on several initiatives and it is challenging to keep track of changes and updates across organizations, a repository is very helpful.
- Dev and Ops are included in the source control thanks to version control. DevOps' original beauty in this. As a result, everyone can clearly and transparently see what is there in the source code and in each iteration of it.

Version Control tools in DevOps

The business offers a wide variety of change control technologies for Logistics. However, we've compiled a list of a few of the best systems for version control in Logistics that will reduce complexity for you and your team.

GitHub

Zip is an expansive, cost-free **Version Control System (VCS)**. Git is designed to function in projects of any size. The history of code modifications will be maintained and merged with the aid of Git. All of the source code is stored in a branch called Wikipedia by Git users. Remote updating, as well as other processes, are available on GitHub. It provides quicker operating speed and is simple to understand.

GitLab

GitLab is a number of alternatives and Ruby-based open-source versioning system. It includes features like a group's website, an interconnected application, and so on. The **continuous integration (CI)** feature of GitLab allows for the automated testing and delivery of code. The release technique GitLab is housed at GitLab.com, which offers free infrastructure. The GitLab API makes linking projects simple. It is compatible with several operating systems, including Windows, Linux, OS X, and others.

Bitbucket

This version management system is for sale. It is a component of the software lineup from Atlassian. It provides options like pull requests, in-line comments, and debate. Repair and maintenance were created primarily for professional teams. In addition to allowing individuals access code, it also lets them administer and work together on GIT initiatives. Both a local machine and a cloud may be used to install it.

Perforce

An established business change control solution is called Perforce. Users connect to a shared file repository in this instance. Between the file repository and each user's computer, files are transferred using Continuous integration software. It offers merging and splitting, interconnections,

artifacts maintenance, and screen source administration. By means of its HelixCore, it provides configuration management. The most crucial components of the project are safeguarded by this security plan.

Apache subversion

A well open-source version control system is **Apache Subversion (SVN)**, sometimes known as **SVN (Version Control System)**. There is, however, a business version as well. CollabNet first developed SVN in 2000, and the Apache Software Foundation currently oversees its project maintenance. File lockdown is supported by Apache Evolution so that users may be alerted when multiple users attempt to modify this very same file. Apache Subversion offers capabilities, including process monitoring, record recording, access controls, and stock management. In comparison to Git, SVN accepts vacant folders and also has greater PC compatibility.

Hg

Often known as **Mercury**, it is a distributed version-control tool for programmers. It is a free tool that offers exceptional speed and adaptability for distributed employees. Designers say Monarch is simpler to set up and use than Git. Python is used to create it the most.

One of the essential technologies good operational companies employ to speed up deliveries and cut down on development efforts is virtualization technology, commonly known as *versioning*. Teams of software developers can interact more quickly because of source code due to the rapidly changing IT world of today.

When it comes to DevOps achievement, robust versioning solutions that are utilized to their maximum potential are crucial. Now, let us quickly go through the fundamentals of source code for the reason of clarity. Source control, version history, and VCS are a few names for version control. Whatever name you give it, repository refers to any method or tool used to record and monitor major changes to a product over time. You have theoretically employed a version control method if you have ever used file names like `DevOpsOnesheetFinalVersionMK2.pdf`.

A current standard for change control tools, specifically when it comes to accessing public applications, is GitHub, which is a well-known instance.

Every modification that is made to a project over time is recorded using git repository platforms like GitHub, which also records the date, nature, and author of each change. Most change control platforms enable the user contributing the changes to provide annotations explaining the changes' justification, which gives anybody seeing the new changes organizational factors.

In order to test changes without compromising the original project, version control systems also offer the creation of *branches* from the *trunk* of the primary project. These branches may be *combined* with both the current topic when they have been successfully tested and confirmed. This kind of version control makes it possible for many programmers to work on the same project at once without interfering with one another's productivity or generating compatibility problems.

Sounds quite useful, doesn't it? Powerful revision control solutions provide so many advantages that the majority of people think engineering teams of all sizes must use them.

How does version control work

Here is how version control works, along with some real-time examples:

- **Repository:** In a version control system, you have a repository where all your project files and their version history are stored. There are two types of repositories: local and remote. Local repositories are on your local machine, while remote repositories are hosted on a server or in the cloud.
- **Commits:** Developers make changes to their files and commit those changes to the version control system. A commit is like a snapshot of the project at a specific point in time, and it includes a description of the changes made.

Example: Suppose you are working on a software project. You make some changes to the code, such as adding a new feature, fixing a bug, or updating documentation. You then commit these changes with a meaningful message describing the purpose of the commit.

- **Branches:** In version control, you can create branches to work on different features or bug fixes independently. Branches allow you to isolate your work from the main project until it is ready to be merged.

Example: You are working on a new feature for a website. You create a new branch called `feature-x` to develop this feature. Meanwhile, the main project continues to evolve in the `master` branch.

- **Merging:** After making changes in a branch and ensuring they work as intended, you can merge those changes back into the main branch or another branch. This integrates your changes with the rest of the project.

Example: Once you have finished implementing the `feature-x` in your branch and thoroughly tested it, you merge the changes into the `master` branch, making the new feature part of the main project.

- **Pull requests (or merge requests):** In many version control systems, such as Git, developers create pull requests or merge requests to propose changes to a project. Other team members can review the proposed changes before they are merged.

Example: You create a pull request to merge your `feature-x` branch into the `master` branch. Other team members review the code, provide feedback, and approve the merge if everything looks good.

- **Conflict resolution:** When two or more people make changes to the same part of a file or project concurrently, version control systems help identify and resolve conflicts that occur during merging.

Example: You and another developer both made changes to the same function in a code file. When you try to merge your changes, the version control system detects a conflict, and you must manually resolve it by choosing which changes to keep.

- **History and rollback:** Version control systems maintain a complete history of changes, making it possible to revert to a previous state of the project if necessary.

Example: If a critical bug is discovered after a release, you can easily roll back the project to the last stable version from the version history.

In real-world scenarios, version control systems like Git are used in software development, documentation management, collaborative writing, and more. They enable multiple people to work on a project simultaneously while keeping a well-organized history of changes and providing tools for collaboration and conflict resolution.

Why is Version Control necessary

Version control is necessary for several reasons, particularly in the context of software development, but its principles are applicable to various collaborative projects.

Version Control is crucial for maintaining the integrity, collaboration, and history of a project. It provides a systematic way to manage changes, collaborate with others, and ensure the stability of software development projects.

Simpler and better bug suppressing

Bug finding and suppressing take up what appears to be 95% of the spending time programming, as anyone who has ever done any amount of it will attest. Software developers are well aware of how much time and energy errors can consume. Scrum methodology offers quick access to possible areas of error due to the continuous & thorough monitoring of changes made to programs.

Sometimes, it takes longer to uncover defects that are worth the effort. Nevertheless, software may still be quite helpful in these scenarios. Scrum methodology allows you to quickly go back to earlier working versions of the project and provides you with a clear understanding of when and why the problem was developed. This implies that if everything fails, reverting could be employed to guarantee that there is a workable starting point from which the additional innovations can be created once again, this time sans having to abandon the whole project and without wasting far too much time.

Concurrent growth

Communicating with other programmers on the same project may sometimes be a real annoyance because of the nature of technology. Parts of

the system, which occur when some initiatives need other software modules to operate effectively, may cause enormous problems if modifications are made without the knowledge of all team members. Even while synchronization with both the new modifications is often a straightforward operation, not being aware of them frequently leads to the creation of new defects even when there are no real mistakes in the code.

To guarantee that perhaps the modifications made by one designer do not conflict with the tasks being carried out at the time with another developer, sections are formed using version control techniques. Solutions for configuration files also enable the introduction of notes to each change that is made. Another good improvement each researcher made was following posts when it was appropriate to combine the work.

Greater dependability of the final result

Every deployment's dependability and performance are immediately impacted by version control. Due to the greater visibility of changes and the context each alteration is given by software version control systems, functioning with other individuals is made easier but more enjoyable. The most crucial component of working as a team is communication and version control improves collaboration without having team members write each other notes all day.

It is much simpler for everybody working on the project to understand what occurred and when changes are visible and accompanied by background. This makes it possible for each individual to work together to remain on task and continue to deliver outcomes. Better goods are provided in less time because of this enhanced collaboration, which also boosts team morale.

Version control and excellent software engineering effectiveness go hand in hand, as is clearly apparent. When the team appropriately implements continuous integration software, adjustments may be made at a much quicker rate and with rising levels of dependability. DevOps seeks to accelerate the whole production schedule while simultaneously improving the quality of work produced. By fostering effective team communication and product innovation, the repository significantly helps to accomplish these objectives.

Decompose by business capability

The distributed system is something you will want to adopt when creating a huge, sophisticated program. A project is designed using the appropriate software as a collection of loosely connected activities. By allowing functionalities and distribution, the service layer aims to hasten software engineering.

The microservice architecture does this in two ways:

- Permits modules to be launched individually.
- Streamlines monitoring

The technical initiative was taken as a collection of small (6–10 person), independent teams, each of which is in charge of providing one or more services.

These advantages are not always certain. Therefore, only the thorough functional decomposition of the application into services may lead to their execution.

A product has to be modest in order to be created by a small team and validated without difficulty. The Relevant Ethical Model from **Object-oriented Design (OOD)** is a helpful principle **Single-responsibility Principle (SRP)**. According to the SRP, a class should only have one reason to alter, and even a function of a category is defined as a cause for modification. Applying the SRP to service design makes sense in order to create systems that are coherent and perform a limited number of closely linked functionalities.

The applications should also be broken out such that the majority of new and modified constraints only have an impact on a single service. This is due to the fact that cooperation between many sections is necessary for adjustments that affect integrated services, which slows down development. The **Consistent Occlusion Concept (CCP)**, which asserts that classes that change for the same cause should be contained in the same container, is another helpful OOD guideline. In particular, it is possible that two classes implement different parts of the exact same business rule. Programmers should only need to alter the code in a limited number of packages, preferably only one, when a company rules updates. When creating

services, this method of thinking is sensible as it will assist in guaranteeing that any change will only affect one function.

Problem: How to decompose an application into services

- Coherent services are necessary. A service may only provide a limited number of functionally closely linked tasks.
- Companies must follow the Universal Closed Guideline, which states that items that are modified at the same time should have been wrapped together, to guarantee that each adjustment only affects one operation.
- The coupling between administrations must be light; each business must have an API that wraps its implementation. Without harming customers, the architecture may be altered.

A product has to be tested:

- Each service must be modest enough to be created by a “2 pizza” crew or a group of six to ten individuals.
- Every team that controls at least one service must be independent. It must be possible for a team to create and implement its offerings with the least amount of assistance from those other team.

Solution:

The following section describes the solution of the organizational capabilities:

- Describe offerings based on organizational capabilities. Enterprise, wide modeling includes the idea of a professional industry. It is a task that a company does in order to produce value. An industry object, such as a competence, often matches a place of order.
- Deliveries are the responsibility of order placement.
- Consumers are under the control of marketing automation.
- A number of co-architecture of professional skills is often used. A corporate program could, for instance, contain top-level subcategories for manufacturer creation, delivery, demand generation, and so on.

Examples:

An electronic retailer's commercial possibilities involve:

- Product catalogue administration
- Inventory control
- Organization control
- Distribution control

Each of the following characteristics will have an equivalent functionality in the relevant distributed applications. The advantages of this design are as follows:

- Steady infrastructure as a result of the mostly stable investment skills
- Bridge, independent, and oriented on producing commercial value rather than technical improvements, production teams
- Coherent, consistent and loosely connected products

Problems: How do you establish a business capability

Analyzing the company is essential for recognizing skills and, by extension, providers. The goal, architecture, operational procedures, and specialty areas of a company are examined to determine its business potential.

Iterative processes work well for identifying circumscribed contexts.

The following are excellent places to look to uncover powerful models:

Functional chart: Depending on the business capabilities or business capability groupings, various factions within a company may correlate. Commercial capacities often correlate to business elements in a greater relational schema:

- Create clear, focused commitments
- Commit significant messages.
- Be dependable and consistent.
- Avoid changing historical records.
- Avoid committing created files

Guidelines for using Git

Git is now widely accepted as engineers' preferred VCS. Utilizing Git has significant benefits, particularly for engineering teams where numerous engineers collaborate since it is essential to have a reliable framework for merging everyone else's code.

But it is best to set up rules to be followed with any strong instrument, particularly one that requires teamwork. Otherwise, we risk hurting ourselves.

Working with a VCS like Git is made simpler by several core tenets that DeepSource has developed for their own group. Here are five basic guidelines you may adhere to:

Create clear, focused commitments

When concentrating on one issue, hackers sometimes get diverted into doing too many various tasks. For example, when attempting to solve one issue, you can see a second and feel compelled to address it as well. It quickly gets out of control, so you wind up with a whole lot of changes that are all committed at once.

This is a concern, and it is preferable to make submissions as concise and targeted as you can for a variety of reasons:

- It facilitates periodic audits by simplifying it for group members to examine your update.
- It is much simpler to undo a transaction fully if it needs to.
- With your booking process, tracking those changes is simple.
- It also helps in psychologically examining changes you have made using Git log.
- Writing stronger pledge communications: 5 Steps

Here is a summary of the recommended rules:

- **Apostrophes:** Do not use a period after the last word and emphasize the very first word. Keep in mind to apply just letters of the alphabet while employing traditional commits.

- **Feeling:** Make the subject line urgent.
- **Illustration:** Correct the toggle condition for dark mode. Using the imperious mood conveys the tone of an order or request.

The kind of commit should be specified. Having a set of terms that are consistently used to explain your modifications is advised and may even prove to be more helpful. Examples include *bugfix*, *inform*, *households or individuals*, *crash*, and so on. For further details, see the subsection here below typical commitments.

- **Duration:** The body should be limited to 72 words, with the opening sentence preferably being no more than 50 letters long.
- **Information:** Try to avoid using unnecessary phrases and terms and remain as straightforward as possible (examples: though, maybe, I think, kind of). Consider yourself a researcher.

Commit early, commit often

Git usually works in one pursuit, where you regularly push your work. It is beneficial to focus on little increments and keep committing to your work rather than expecting the publication to be flawless. Maintaining your code that includes the most recent modifications may help you prevent disagreements if you are working on a feature branch that might take a long time to complete.

Additionally, Git only fully assumes ownership of your data when you commit. Implementing source control prevents you from losing work, undoing updates, and making it easier to track everything you have done.

Conclusion

In this chapter, the implementation of version control and tracking are presented. The basics of version control and the benefits of it were covered in this chapter. The version control tool and business capabilities were explored in this chapter. Furthermore, the commit is discussed with the GIT features.

In the next chapter, we will learn about the feature environment.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 3

Dynamic Developer Environment

Introduction

In this research, we are taking a look at dynamic and on-demand environments for developers to gain flexibility. Software companies want technologies that are adaptable as well as extendable in order to speed, compress, and eliminate deployment processes. When working on a new feature, having an environment to safely test and develop the feature – without affecting the rest of the development team, is a major win.

Ultimately, this solution will help give us more confidence that we can make the changes we need to while also giving us a place to test those changes before merging back into the main branch. The innovation atmosphere provides an on-demand scenario where you may evaluate your *relatively brief* improvements individually and without requirements before deploying them to operation.

Structure

In this chapter, we will discuss why we need a feature environment. Successively, the faster delivery is well addressed, followed by our pain points and ideal solution. The infrastructure changes are then added along with the routing of feature environments. Automate deployment to feature environments is being incorporated in this chapter, consequently the removal of obsolete feature environments. Finally, the advantages and disadvantages are concluded. Following are the topics that we will cover:

- Why feature environment?
- Faster delivery
- What are our pain points?
- Ideal solution
- Infrastructure changes
- Routing of feature environments
- Automate deployment to feature environments
- Removal of obsolete feature environments

Objectives

This chapter helps reader to lean how feature branch development is helping to streamline the software development process by allowing developers to work on specific features separately, without interfering with the work of other team members. With this method, you will learn and able to perform a standalone assessment on your improvements beforehand merging those into the repository without interfering with any of our crew members' continued work. Whenever you establish a product branching form of the original in any source code repository, additional features contexts are created on-demand, and then they are removed after the product development has been completed. As a result, the company will always have a pristine, performance masters branching that could be pushed into operation anywhere at the moment.

Why feature environment

Innovative DevOps organizations take pride in their ability to respond quickly to both market and client needs. Nevertheless, obstacles such as delayed data transfer and an absence of communication hinder installations and cause a prolonged time until market. Adaptable and extensible solutions that expedite, consolidate, and compress distribution workflows are required by development studios.

For optimum adaptability and flexibility, organizations need to consider implementing platforms including continuous build, integration and

deployment whenever nurturing a modern DevOps workplace. A feature environment is a separate environment that is used to test new features before they are deployed to the production environment. It is an important part of the development process in DevOps because it allows developers to test new features in a controlled environment that is similar to the production environment. We will examine the following three aspects that a continuous production infrastructure requires to be flexible:

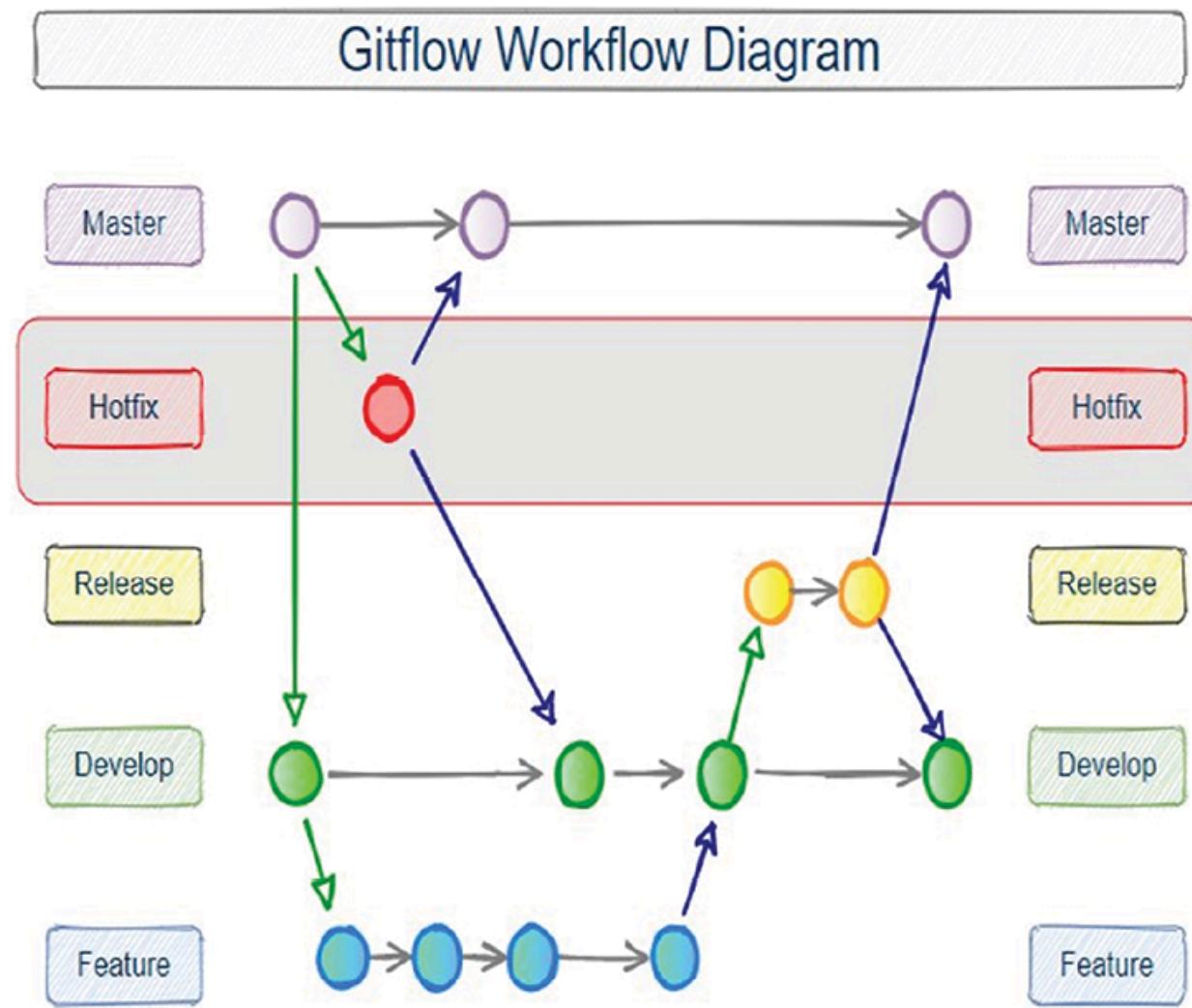


Figure: 3.1: Git branching strategies for multiple releases

Preceding referral [Figure 3.1](#) has several benefits of using feature branches in software development:

- **Isolation of work:** Feature branches allow developers to work on specific features without interfering with the work of other team

members. This reduces the risk of conflicts and makes it easier to manage the development process.

- **Faster development:** With feature branches, developers can work on multiple features simultaneously, which can speed up the overall development process.
- **Easy rollbacks:** If a feature does not work as expected, it is easy to roll back to a previous version of the code by merging an older feature branch into the main branch.
- **Improved collaboration:** Feature branches allow for better collaboration between team members, as different developers can work on different features at the same time.
- **Easier to experiment:** Feature branches make it easy to experiment with new ideas or technologies without affecting the main codebase.
- **Improved testing:** By isolating new features, it is much easier to test them before they are integrated into the main branch
- **Better maintainability:** With feature branches, the codebase is more organized and easier to maintain. This can make it easier to fix bugs and add new features in the future.

Benefits of the code review process:

- **Quality assurance:** Code reviews ensure that only high-quality, well-tested code is merged into the main codebase, reducing the likelihood of introducing bugs and defects.
- **Knowledge sharing:** Code reviews promote knowledge sharing among team members. Developers learn from one another, leading to skill development and improved coding practices.
- **Consistency and coding standards:** Code reviews enforce coding standards and consistency across the codebase, which is essential for maintainability and readability.
- **Reduced bugs and defects:** Catching and fixing issues early in the development process is cost-effective, and code reviews are an effective way to do this.

- **Improved documentation:** Review comments and discussions serve as documentation for the code, aiding future reference and troubleshooting.
- **Team collaboration:** Code reviews foster collaboration, providing a platform for discussing design decisions, architectural changes, and project direction.
- **Better code design:** Code reviews encourage discussions about code design and architecture, leading to improved code structures and design patterns.
- **Enhanced code readability:** Code that goes through a review process tends to be more readable and maintainable, thanks to feedback on organization, naming, and comments.
- **Early detection of security vulnerabilities:** Code reviews help uncover security vulnerabilities and reduce the risk of security-related issues.
- **Continuous improvement:** Code reviews can lead to process improvement as teams analyze feedback trends and address recurring issues in coding guidelines or through training.
- **Mitigation of “Bus Factor”:** Sharing code ownership through code reviews reduces the risk of a project’s failure if a critical team member becomes unavailable.

Benefits of a feature branch strategy:

- **Isolation of features:** Feature branches allow developers to work on new features or fixes without affecting the stability of the main codebase. This isolation can prevent bugs from creeping into the production code.
- **Parallel development:** Feature branches enable parallel development of multiple features by different team members. This speeds up development and promotes a more agile workflow.
- **Testing and validation:** Each feature branch can be tested and validated independently, reducing the risk of integration issues and providing a more reliable product.

- **Clear versioning:** Feature branches provide clear versioning of features. It's easy to see which features are being worked on and which have been completed.
- **Easier collaboration:** Feature branches make it easy for team members to collaborate on specific features, reviewing and merging changes in isolation from the main codebase.
- **Enhanced control:** Feature branches give you control over when new features are merged into the main codebase. You can choose the right time for integration and avoid interrupting ongoing work.

Benefits of combining code review and feature branch strategy:

- **Improved code quality:** The code review process ensures that code changes in feature branches meet quality standards before merging, maintaining a high level of code quality.
- **Risk mitigation:** Feature branches isolate changes, reducing the risk of bugs and defects in the main codebase. Code reviews further mitigate this risk by catching issues early.
- **Efficient collaboration:** Code reviews allow team members to collaborate efficiently on feature branches. They can discuss changes and ensure that each feature meets project requirements.
- **Clearer tracking:** The use of feature branches provides a clear and structured way to track the progress of each feature, and code reviews add transparency to the state of these branches.
- **Enhanced documentation:** Code review discussions can serve as documentation for the changes made in feature branches, helping in tracking and understanding the evolution of the codebase.
- **Improved project management:** The combination of code reviews and feature branches helps project managers and stakeholders understand the status and quality of each feature, making project management more effective.

In summary, the combination of a feature branch strategy and code review process offers a structured and efficient way to develop software, ensuring code quality, reducing risks, and enabling effective collaboration among

team members. It's a valuable approach for modern software development practices.

Accelerating the installation process

DevOps techniques emphasize not just speed but also value delivery. Avoiding shortcuts and hurrying operations can only result in increased inefficiency and poor product quality. Alternatively, with the right tools, a CI/CD pipeline may be sped to make it faster while improving product quality. A widespread misunderstanding is that CI/CD solutions that accelerate the process substitute personnel. In truth, the goal of CI/CD solutions is to handle administration and duplicated chores so that people may concentrate on activities that need specific talents and rational reasoning.

Capsules, in particular, make enabling CI/CD much simpler. How so? Capsules are adaptable, inexpensive, infinitely expandable, and operate in any ecosystem. Furthermore, canisters employ a community approach, allowing programmers to rapidly migrate code between canisters and Elasticsearch clusters rather than transferring code throughout monolithic virtualization in various settings. As a result, programmers get faster access to the right infrastructure without having to wait for hours shifts for information transmission.

Similarly, ionir offers the versatility and efficiency that even a continuous DevOps environment requires. For example, by reducing data impediments, the internet services technology accelerates DevOps operations and assures developer productivity. In reality, the technology reduces data shipping times between hours to minutes and eliminates 99% of unnecessary waiting period. Through fast data movement, programmers may transfer any amount of data in less than 40 seconds, permitting them to access and maintain the data where and whenever they require it.

Information processes automation and orchestration

Adopting automating and coordination platforms is the next step in achieving versatility in a changing DevOps ecosystem. Management and orchestrating may help DevOps companies ensure a quicker time for market by speeding up the CI/CD workflow. Fundamentally, automating software

distribution to commercial, research, and experimental systems is the best approach to accomplish agile methodologies. Take into account that continuous integration is more than merely adding new capabilities constantly. To be honest, CD assures that automated processes can restore output fast and securely, even in the early hours of the morning when everyone is asleep.

This data migration necessitates flexibility and adaptability in which information may be transferred to and between platforms and systems with minimal disruption to the ongoing operations. Information services platforms for Elasticsearch, such as the ionir. Information platform to enable virtualization, offers smooth data portability and seamlessly choreograph subjects performed. Researchers, for particular, may use ionir to rapidly retrieve new information after impact testing or to instantly make replicas of approaching information on faraway edge devices.

In essence, the system enables PostgreSQL information as portable as programs, allowing for more versatility in meeting business objectives. Furthermore, any permanent storage capacity, irrespective of size or location, may be duplicated between clusters or throughout the planet in 40 seconds less. As a result, software in the new destination has immediate access to an entirely operational read/write duplicate of the volumes.

Coordination of operations and production organizations

Many companies believe that by simply installing CI/CD, your firm is immediately executing a DevOps approach. This, unfortunately, is not the truth. DevOps is a societal change that dismantles lengthy barriers and redistributes power, employees, and capabilities. It is critical for a DevOps organization to welcome fundamental societal changes and reimagine how developers, management, and other organizations collaborate. With the correct tools and tools, CI/CD facilitates this cultural transformation.

A successful DevOps approach allows all departments (developer, management, cybersecurity, and quality management) to cooperate all across the distribution pipelines. Specifically, organizations may create, evaluate, and publish apps in separate closed quarters avoiding compromising with those other aspects of the production by leveraging Elasticsearch canisters.

Additionally, the ionir Information Services Gateway for OpenShift streamlines construction and deployment timelines by facilitating collaboration across dev, operations, administration, and QA teams and ensuring that now the workflow operates smoothly. Designers strive to offer answers more quickly and effectively. Conversely, plant operators have ideas for new value propositions but cannot put them into action with outside assistance. As a result, DevOps tactics guarantee that the operations manager controls the technology stack, whereas engineers may disseminate and inspect information and code wherever it is required.

Furthermore, as even more businesses relocate penetration testing and processes left in the process of being developed, DevOps teams must be able to engage with protection teams to ensure protection across the pipe system. Broadly acknowledged as a reference to movement, the Ionic Internet Services Gateway for Kubernetes provides a swift and straightforward deployment of the most recent statistics collection to ensure and maintain the security and health of the software.

What is agile in software development

Agile in software development is an approach and set of principles that prioritize flexibility, collaboration, customer feedback, and incremental progress in the process of creating software. The Agile methodology contrasts with traditional software development methods, which often involve rigid planning, extensive documentation, and fixed requirements.

Agile in software development lifecycle:

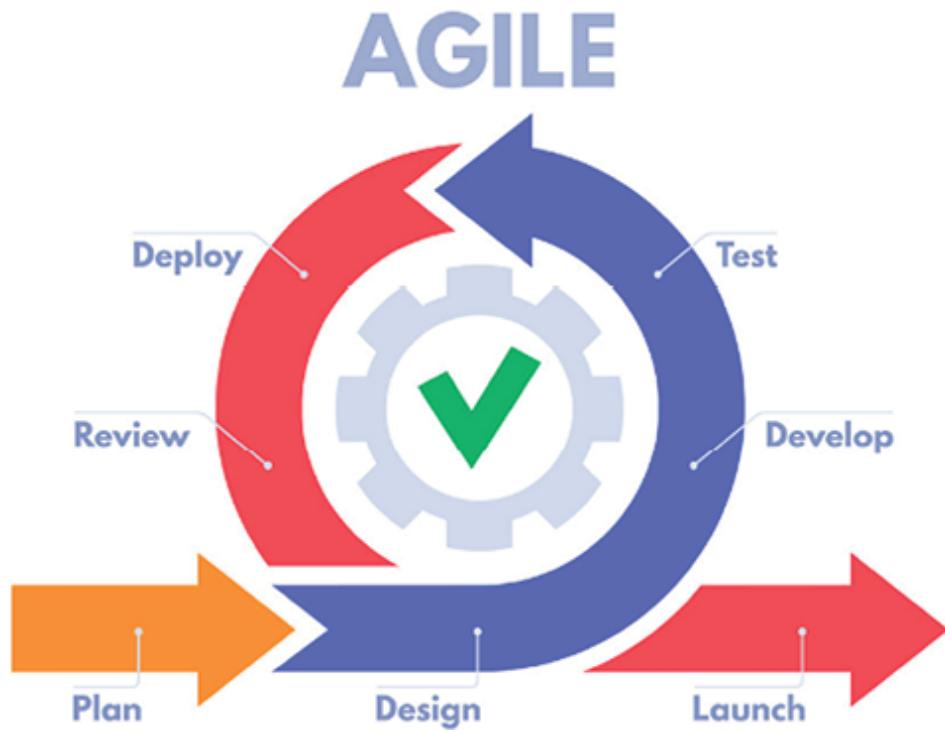


Figure: 3.2: Agile in software development lifecycle

Agile methodologies

Agile methodologies are a set of principles and practices for software development and project management that prioritize flexibility, collaboration, customer feedback, and incremental progress. Agile methods aim to deliver value to customers quickly and adapt to changing requirements efficiently. There are various Agile frameworks and methodologies, with Scrum, Kanban, and **Extreme Programming (XP)** being some of the most popular. Here are the key principles and practices associated with Agile methodologies:

AGILE METHODOLOGY

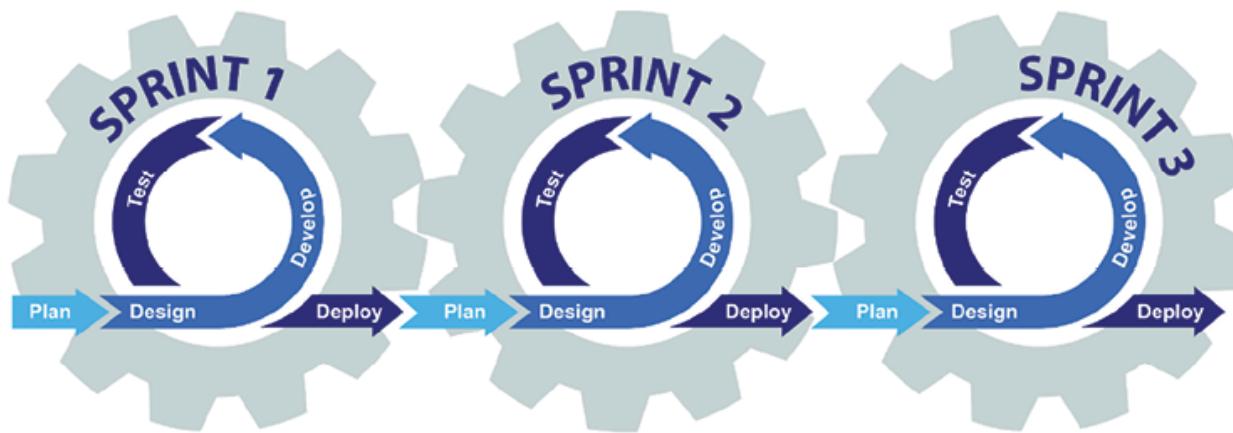


Figure: 3.3: Agile methodology

Common agile methodologies:

- **Scrum:** Scrum is a popular Agile framework that organizes work into time-boxed iterations called sprints. It includes roles like Scrum Master and Product Owner and uses ceremonies like daily stand-up meetings and sprint planning.
- **Kanban:** Kanban is a visual management method that focuses on limiting work in progress and improving workflow. It's suitable for both software development and other knowledge work processes.
- **Extreme Programming (XP):** XP is a set of practices that emphasize continuous feedback, simplicity, and technical excellence. It includes practices like pair programming, test-driven development, and frequent releases.
- **Lean software development:** Lean principles, adapted from lean manufacturing, emphasize reducing waste, optimizing the whole process, and delivering value to the customer. It shares similarities with Agile.
- **Feature-Driven Development (FDD):** FDD is an iterative and incremental approach that focuses on identifying and developing features individually.

- **Dynamic Systems Development Method (DSDM):** DSDM is an Agile framework with a strong focus on meeting user needs and maintaining a business perspective throughout development.

Agile methodologies have become increasingly popular in various industries, not just software development, due to their ability to adapt to changing requirements, deliver value quickly, and promote collaboration among team members and stakeholders.

Last impressions

If the goal of DevOps is to decompose conventional barriers among programmers and operational processes in order to create efficiency in a CI/CD workflow, ionir is the catalysis that helps accelerate deployment milestones while also providing data adaptability.

Several varieties of variables, including increased corporate competition, fast technological advances, and marketplace demands, significantly altered the ecosystem in which program businesses compete. As a consequence, numerous businesses confront a similar challenge: quicker and more frequent program development. Customers no longer have to wait decades for a service to be released before providing feedback. Users demand quick replies to their ever-changing needs. Numerous deployments are required to offer constant information to the client. Organizations such as Zuckerberg often deliver applications to consumers. Though continually supplying information to clients provides a competitive edge over competitors, it is not simple to do so since several technical and semi difficulties must be addressed simultaneously. To solve those issues, businesses should be compact and flexible across the whole technology creation lifetime cycle.

Agile software engineering is prevalent in many businesses, thanks to the many advantages it provides, such as decreased project duration, higher project overall success, substantially lower expense, and more satisfied customers.

Scrum methodology improves adaptability and ensures that customers' demands are satisfied on deadline. Agile methodologies promote cooperation between design teams. Sprint, XP, Yokohama, DSDM, Rapid, Quartz, as well as other flexible approaches, are illustrations. Agile methodology, which is focused on agile methodology, communicates better among the

advancement staff and clients. Unfortunately, the primary emphasis of agile approaches focuses on the technical side of things, with the management part being ignored. These network operators implement, administer, and ensure the functioning of technology at consumers' sites.

As a result, software companies provide upgrades considerably quicker, whereas management players struggle to stay up and get below. Something may cause the whole package to still be postponed, despite the fact that all the necessary preparations and planning have been done.

Miscommunication and disagreement were likely since innovation and so it services distinguishing feature entities. Most technology doctors believe that the separation alone between engineering and operations departments does indeed have a detrimental impact. Another new method known as Logistics has been developed to address barriers and improve interaction, cooperation, interaction among development and operational workers. "*Dev*" + "*Ops*" equals *Test automation*. Although agile processes increased the effectiveness of game developers via engagement, DevOps expands cooperation to operational teams and automation tests.

Logistics is founded on agile concepts; it promotes improvements to meet delivery processes. DevOps comprises a loosely defined set of procedures aimed at fostering collaboration between engineers and maintenance workers. The overarching goal of Agile is to improve return on investments and promote customer happiness by introducing new capabilities and improving customer service on a constant basis. Automation is being used by a number of firms, including Twitter, Microsoft, Spotify, Vimeo, and as well as Soft-pedal.

Therefore, why DevOps is useful? Logistics is a broad phrase that may signify various items in various settings. It's frequently regarded as a synthesis of operations and creation. DevOps is defined by Dyck et al. as a *managerial model that encourages compassion and understanding and bridge collaborative efforts between and among workgroups - particularly growth and Now it procedures - in application development advocacy groups in order to perform durable materials and speed it up transformation shipping.*

So according to Shaw, there is no universally recognized definition of Docker. There has been a lot of disagreement over what DevOps genuinely

implies. He claims that certain blog entries have presented DevOps as a particular career requiring combined developer and maintenance expertise, while others maintain that this is not the case entity. They feel that Docker is a modern criterion for production, monitoring, and administration, among other things. It reminds out that numerous businesses hold the previous viewpoint, citing multiple job postings for *continuous integration architects*.

Logistics is described by Ott et al. as a *theoretical model for facilitating and administration*. According to the books, Linux is a foundation of theories and concepts which enterprises could include in the business development cycle.

According to Purniima, Scada is *compact on overdrive*. It incorporates various agile approaches and certain clean concepts, as a method of controlling the lifetime of project management. Docker is not only for developers and service organizations.

Robotics involves a variety of participants, including industry professionals, engineers, engineers, quality management professionals, technical teams (computer, databases, and network engineers, designers, as well as security people).

Automation encompasses those components that contribute towards the provision of rapid, efficient, and increased programming. Docker is composed of four major components: attitude, automating, evaluation, and communication. The primary emphasis of Docker is on allowing interaction and cooperation among teams, rather than on group norms or technology. Cloud computing supports Bpm It is often associated with internet offerings such as Grid computing **Infrastructure as a Service (IaaS)**, **Software as a Service (SaaS)**, **Platform as a Service (PaaS)**, and Microservices. Manager, Maestro, BladeLogic, Salt, Amazon Workups', Docker, Nomad, and other solutions are utilized to enable Dec.

Faster delivery

Benefits are delivered more quickly because of a shortened installation phase and reduced identifiers.

When a function is meaningless until it has been published, you really would like to reduce the time between developing the customer narrative and

distributing as much as feasible. How then do people go about doing this? By shortening the trip from *a* to *b* while sending a lighter emblem between *a* and *b*.

Creating a shorter route between points a and b

When it concerns IT infrastructure, smaller and simple is best. A CI/CD workflow, in particular, will take the functionality closer to the final goal: commercialization. The first and only environment required for functional deployments is programmer surroundings, to bogus dev information as well as a testing environment. This basic design greatly simplifies administration, while we have to yet, encounter a use case that this arrangement somehow does not cover.

Organizations, on the other hand, have four ecosystems among development and production! It's no surprise that all these firms face the greatest difficulty with the review process and only publish maybe once every month. Leveraging feature domains, you may evaluate the innovation you're going to introduce in dev and pr while affecting consumers, and afterwards, publish it after it has been properly security evaluated.

Send out little medals

The more features a publication includes, the scarier it is, thus what would be the systemic change to reduce the risk of the distribution? Limited functionality insignia should be sent. Delivering a shorter emblem simplifies anything: less than that to check in QA, simpler tech evaluation, and, in the event of an issue, simpler to determine what modification triggered the issue and remediate.

What are our pain points

The pain points in software development can vary depending on the project, team, and organization. However, there are common challenges and pain points that many software development teams face. Here are some of the typical pain points in software development:

- **Changing requirements:** Requirements often change during the course of a project, leading to scope creep and additional work.

Adapting to these changes can be challenging.

- **Inadequate documentation:** Poor or outdated documentation can make it difficult for developers to understand the codebase, APIs, and configurations.
- **Lack of testing:** Insufficient or ineffective testing can lead to bugs, quality issues, and delayed releases.
- **Technical debt:** Accumulated technical debt from shortcuts and compromises can hinder development progress and decrease code quality.
- **Inconsistent development environments:** Discrepancies in developers' environments can lead to compatibility issues, bugs, and build failures.
- **Poor code quality:** Maintaining code quality can be challenging, resulting in code that's hard to maintain, debug, and extend.
- **Resource constraints:** Teams may lack the necessary resources, such as skilled developers, tools, and hardware, to meet project demands.
- **Communication issues:** Ineffective communication within the team and with stakeholders can lead to misunderstandings, delays, and misalignment of goals.
- **Time constraints:** Tight deadlines can put pressure on developers and lead to rushed code, inadequate testing, and quality issues.
- **Dependency management:** Managing dependencies and dealing with dependency updates can be challenging, especially in complex projects.
- **Security concerns:** Identifying and mitigating security vulnerabilities and ensuring compliance with security standards can be demanding.
- **Onboarding difficulties:** New team members may struggle to set up their development environment and understand the project, leading to slow onboarding.
- **Scalability challenges:** Scaling a software system to handle increased workloads can be complex and may require architectural changes.

- **Conflict resolution:** Disagreements and conflicts among team members can disrupt productivity and morale.
- **Legacy code:** Maintaining and enhancing legacy code can be a pain point, as it may lack documentation and modern best practices.
- **Technological complexity:** Managing complex technologies, frameworks, and tools can increase the learning curve and create challenges for developers.
- **Resource management:** Allocating resources effectively, including time, budget, and personnel, can be a significant challenge.
- **Regulatory compliance:** Meeting regulatory requirements and standards in certain industries, such as healthcare or finance, can be demanding.
- **User feedback integration:** Incorporating user feedback into the development process can be challenging, especially when priorities and feedback conflict.
- **Cultural and organizational challenges:** Adapting Agile or other methodologies and fostering a culture of collaboration and continuous improvement can be difficult in some organizations.

To address these pain points, development teams may implement practices such as Agile methodologies, DevOps, automated testing, code reviews, and continuous integration and delivery. Additionally, effective project management, clear communication, and proactive risk management are essential to mitigate these challenges and ensure successful software development projects.

Ideal solution

The principle of DevOps is to streamline whatever is possible. Those that go the farthest with robotics stand to win significantly by minimizing possible misunderstandings and shortening technology deployment turnaround time. Such automating occurs in the CI/CD process. Automating different testing processes in the process is now an accepted practice. Whenever software is submitted to a repository, the execution context is refreshed with the latest software and testing is executed directly. Nevertheless, there has been a

delay in introducing new capabilities into this well-oiled process. The problem arises whenever the team is focusing on many projects at the same time, and each one requires a separate infrastructure to conduct tests prior to merging them.

Atmosphere for entrepreneurship shortcomings

Using virtualized technology platforms, it is not always possible to launch a full local programming environment on a designer's notebook. While technologies like *LocalStack* may imitate cloud computing regionally, these do not necessarily provide all of the required technology platforms or are applicable to every individual business scenario. Whenever things become complicated sufficiently, relying on surrounding ecosystems that resemble genuine internet services will leave you with few choices. A fast feedback mechanism is critical for effective computer programming. Developers prefer to concentrate on their work. After modifying the code, integrating trial runs should be completed quickly. A heterogeneous environment might result in an ambiguous baseline condition for testing as well as unnecessary sample results.

Infrastructure changes

The infrastructure that underpins and supports a system or organization. The flow, storage, interpretation, and analysis of data are supported by the physical and digital equipment that make up the technology infrastructure in computing. Equipment can be centralized inside a data center or dispersed throughout a number of data centers that are either managed by the company or by a third-person, such as a coworking facility or cloud platform. The electrical, cooling and structural components required to support data center gear are frequently included in data center infrastructure. Computers, storage components, networking equipment like switches, gateways, and physical cabling, as well as specialized network equipment like network firewalls, make up the hardware backbone of data centers.

IT network security must be carefully taken into account while designing a data center architecture. This might involve the building's physical security measures, including such electronic key entrance, ongoing human and video monitoring of the space, strictly regulated access to the servers and storage areas, etc. This minimizes the risk of intentional damage or data theft by

ensuring that only authorized people access the hardware infrastructure of the data center.

A broadband network is located outside the data center and consists of routers, aggregates, repeaters, packet filtering, fiber optic cables, telescopes, microwave terminals, and other network elements that manage transmission pathways. **Internet service providers (ISPs)**, like Verizon and AT&T, are in charge of designing, constructing, and managing the internet infrastructure. A network is the framework or base upon which a system or organization is built. Both physical and virtual capabilities that facilitate data transmission, storage, transmission, and analysis make up the infrastructure of information technologies in computing. Infrastructure may be centrally located within a data center or maybe decentralized and dispersed over a number of data centers that are either managed by the company or by a third-person, such as a coworking facility or service services. The construction materials, cooling systems, and power sources required to support data center gear are frequently included in data center infrastructure. Computers, memory subsystems, networking gear, including switches, gateways, physical cabling, as well as specialized network equipment, such as networked firewalls, are common components of data center hardware platforms.

Routing of feature environments

Establishing learning surroundings starting beginning fixes the concerns outlined previously, however it has typically required passing through a lengthy number of manual processes. Of principle, this might be performed one time for each programmer such that each of them has their individual experimental settings. However, such settings have a tendency to deviate from the state indicated in infrastructural programming, causing problems in the long term.

Robotics to the rescue

A feature branch is a type of development workflow in which a team creates a separate branch of the main codebase to work on a specific feature or set of features. This allows multiple developers to work on different features simultaneously without interfering with each other's work. The *Robotics to*

the rescue is a project or feature that has been developed on that specific branch.

Fundamental concept is a three-step procedure

Whenever a pull request is received in the backend system, a pipeline process starts, that configures the branch, which represents the fundamental building block. To avoid possible confusion for every branching, we often employ a naming scheme with a prefixed such as `eature/` or `f/` for the tree title for branches that need an ecosystem.

Following the completion of the infrastructure configuration, the standard deploying process runs upon every recent initiative to the branching. Because when functionality is completed as well as the pulling application is submitted, an infrastructural deprovisioning task is started to eliminate the branching configuration.

The pulled requests serve as a hub for peer evaluation and decision-making process. The push proposal receives messages from the distribution pipeline activities, signaling if installations and the testing are successful. Certain operations, including such computer code dependencies modifications, may be completely automated using technologies including Redevelop, in addition to the evaluation feedback first from pipelines back to the original source.

Technology licensing may become complicated for a range of reasons, including interconnections among environmental components and development and automated testing stages. Splitting large distribution pipelines into phases is an effective technique to lend discipline to such complexities. Equipment is crowdsourced in stages, with further processes such as creating applications components completed at the appropriate time once prerequisites have indeed been delivered.

One issue that organizations may encounter is higher costs. Maintaining the whole infrastructural stack with each scenario might be prohibitively costly. An additional disadvantage seems to be the time required to offer complicated technologies. Datasets and some other expensive technologies may not have to be provided from the ground up for each situation. Employing models with splinter group ontologies may save your cash and

deploy quicker since you will not have to wait for the database capabilities to start during the test environment.

We divided the notion of an ecosystem into two tiers to allow for the recycling of pricey elements. A programming language is preoccupied with baseline project, although a must with is engaged with reduced infrastructures such as connectivity, storage, and communications processors. We should launch a release about the whole mix of common and applications infrastructures, perhaps we could maintain an environment for applications in many different deploys, each with a different feature development.

Accelerated shipping

Establishing based feature branching deployments is a significant step forward in DevOps competence. Until you have these functions, you will notice a huge reduction in the time it takes to deliver innovative features! You will be able to flexibly create new testing settings, such as dynamic loading as well as other situations. Blue-Green implementations in business are also a possibility. It simply involves transitioning from one application system to another through the top of a shared environmental chamber your performance data.

The higher work required to design and manage the pipeline's code results in more distribution process sophistication. It is important to consider how much automation and optimization you want to accomplish vs how much time you will end up spending on the pipelines actually. The effective usage of automatic keyword ecosystems necessitates a strategy based on the individual aims of every software system.

Our machining might differ

This design is general and may be used in any cloud infrastructure. The specifics of implementations may vary depending on the software and platforms you employ.

For DevOps management, we propose using the Azure **Cloud Development Kit (CDK)** which is a cloud-based platform that gives us a glimpse of various staged performed for various development operations. Although tooling is improving, prepare to develop a sizable proportion of

programming to build automatic image branching for your individual requirements.

Automate deployment to feature environments

The benefit of a crisis scenario is that it forces you to look outside the box. If you have read about adopting functionalities, you will understand how we advocate for the usage of featured contexts, which are contexts that include just a selected data branching and are used to test a product before it can be deployed. Let us see exactly the use of highlight contexts, it proved to be significantly more beneficial than we originally anticipated.

We started discussing how to launch a new version, which included a modification in basic features and a user identification client. We decided we needed to try things out in business, or we'd have to shut down the operation. Reflecting on the way we were previously utilizing feature branching in the dev instance, the following thought occurred to me:

What if we simply establish a featured atmosphere in productions to gain some experience, and afterwards send this to nudge when we have verified something without mistakes?

In that piece, I discuss why and how you should build up feature scenarios across all of your systems in order to transform into a delivery powerhouse.

What exactly is a features atmosphere

A unique atmosphere is a technology branching that's also maintained as a maximum separating, allowing QA to test it independently before releasing it to commercial. Very precisely, this is a development branch that has been distributed as a subdirectory to one's database server, so you may go to a component utilizing, for example, `*application-server-url*/feature-branch-name*`.

What is the point of creating features climates

Features domains allow you to create a template for each individual feature. This enables QA may examine and evaluate each function independently. Frequently, you will observe a configuration with a common production server, at which a lot of innovations are mashing combined, frequently

spurred by an unproductive review process, which causes the organization to bulk capabilities for each and every deployment.

Presence of characteristic elements in all habitats

Nowadays that we understand why component branching are advantageous and why I advocated a minimal configuration with just two organisms, what makes it helpful to just have component situations in all surroundings?

Besides maintaining featured branching in all scenarios, anyone may evaluate a functionality that has been packaged in a git repository within both the dev and operational sections prior to it being ever launched! This is much simpler to set up versus shadows deploys with functionality toggle switches since no modifications to the software application are required.

Removal of obsolete feature environments

Numerous businesses that publish webpages or products deploy from before the platforms to evaluate new modifications before they're made available to consumers. While this has numerous advantages in terms of establishing layers of protection to detect issues and errors, it may also raise expenses, causing the opposite outcome. Considering strategies like Continuous Integration urging companies to ensure that product is constantly transportable, there has been a gradual shift toward working in diverse architecture and assessment contexts further towards simplified configurations.

Noisy, a firm that helps companies discover how people used your website or app sans infringing their privacy, has chosen a unique perspective, explaining why they do not employ a template. Companies feel that something like this allows them to deliver more quickly and reduces the amount of manufacturing concerns.

Squeak's *Thomas Cardiff* discusses various issues in development platforms in a medium post detailing this methodology. Also, before the infrastructures were hardly equal to production.

A sustainable platform software will almost always demand more capabilities in power generation to handle the load, yet the expense of constructing this same configuration before it is prohibitively expensive. Qu's causes configuration deviations as well as calibrated design in or before

settings that might make research in any of these circumstances ineffective. There is frequently a wait, which increases the size of distributions and therefore decreases authority: If numerous employees or companies wish to deploy code during the same timeframe, also before the conditions might become a hindrance. Waiting for such direction of propagation development inefficiencies and additional constraints, particularly if experiments break and everyone must watch for the issues to be rectified. Maintaining a deployment list promotes division separation, therefore produces writers' anguish even if a significant number of revisions have to be combined afterwards.

Shortening the backlog, upgrades are packaged collectively, thereby increasing the likelihood of defects becoming generated and making it harder to follow precisely that adjustment or where modification contributed to the issue since vulnerabilities become compartmentalized because employees might not even be aware of improvements having moved to operation. As deployments are typically handled by operations-focused personnel long before they reach the infrastructure, initiating a deployment to a previously managed infrastructure may suggest an implicit shift of responsibilities from development to operations workgroups. Squeak's alternate approach seeks to fix or prevent those challenges, relying on four major beliefs to do so. Properly merging content that also is ready to be released. the above is supported by ensuring that sufficient benchmarks are in place and that any improvements have already been verified in progress. Only those sections are bifurcated first from the new headquarters, therefore modifications are never reconciled and returned there. Regional smoking testing is done on a creator's PC. Technology labeling for increased updates: if management is concerned about productivity under pressure or how people will respond to development, they could deploy modifications with a product signal.

This is something companies can do on a yearly basis for every premise. Practical learning installations such as monitors, archiving, and notifications have been extensively used to verify that no difficulties arise. Companies often use turquoise implementations to release updates to a group of users until the company is certain that everything is working correctly in order. Squeak's decision to forego a template in lieu of various continuous integration concepts has altered the way people think about releasing programs. Eliminating the interval for modifications when they go public

necessitates giving us a feeling that perhaps the improvements are ready for deployment. This is what has resulted in lower complexities and costs as well as a fast production lifetime.

Conclusion

With this procedure, you can test your features separately before they are merged into the master branch without interfering with any of your team members' ongoing projects. When you establish such functionality, a separate branch will be set up specifically for the new feature, separate from the master branch. This branch can be deleted after the feature development is over. As a result, you will always have a clean and production-ready master branch that is available for deployment at any time.

In the next chapter, we are going to learn **Build Once, Deploy Many**, which is a software development practice that aims to create a single version of an application that can be deployed to multiple environments without modification. This approach allows for a consistent, repeatable and efficient process of deploying software across different environments, such as development, testing, staging, and production. It is also known as *immutable infrastructure* or *immutable deployment* which means that once an application is deployed, it should not be changed in place. Instead, any changes should be made to a new version of the application and deployed again. This approach helps to ensure that the deployed software is always in a known and consistent state, which can reduce the risk of errors and improve the overall reliability of the system.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 4

Build Once, Deploy Many

Introduction

In this chapter, we will delve deeper into the principles and practices of the **Build Once, Deploy Many** approaches, exploring its benefits, challenges, and best practices. We will provide a comprehensive guide for developers, project managers, and other stakeholders involved in the software development process, highlighting the tools and techniques required to successfully implement this approach. By the end of this book, readers will have a thorough understanding of the Build Once, Deploy Many approaches and be well-equipped to apply them to their software development projects. *Figure 4.1* shows the CICD pipeline, where it builds once in the CI stage and deploys to multiple environments (Dev, Test and Prod) using the same binary:

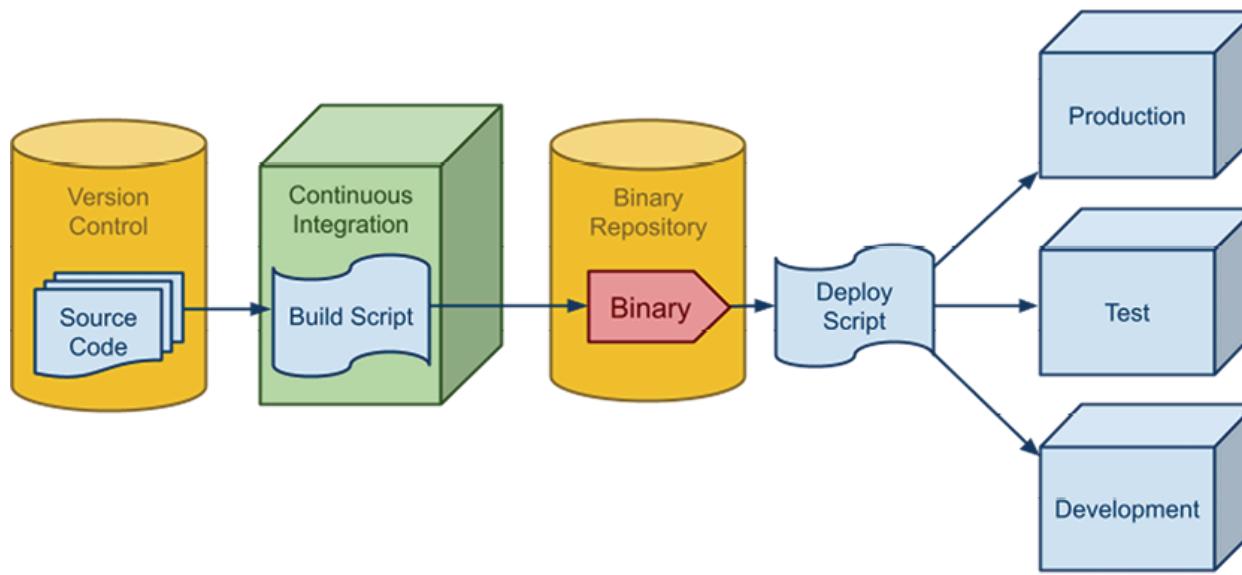


Figure 4.1: CICD pipeline

Structure

This chapter covers the fundamental principles of the Build Once, Deploy Many approaches, including how they differ from traditional development methods, the benefits of using this approach, and the challenges that developers may encounter.

We will cover the following topics in this chapter:

- How to build a docker image?
- Environment overlays
- Continuous deployment
- Pros and cons

Objectives

The objective of the book Build Once, Deploy Many is to provide a comprehensive guide to the principles, practices, and tools involved in the Build Once, Deploy Many approaches to software development. By the end of the book, readers should have a thorough understanding of the Build Once, Deploy Many approaches, its benefits and challenges, and the tools and techniques required to successfully implement it.

Build Once Deploy Many

The Build Once, Deploy Many approach is a modern software development strategy that has gained significant popularity in recent years. The notion of Build Deploy Many is fundamental in application development. The basic concept is to utilize the very same package for any and all settings, whether development or manufacturing. This strategy facilitates distribution and testing and, therefore is regarded as a core component of agile development. Following figure shows build and promote flow for web apps:

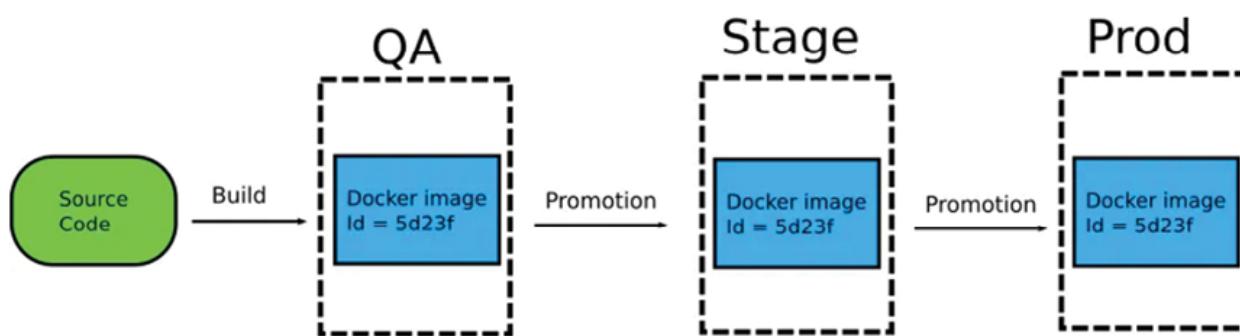


Figure 4.2 Build and promote flow for web apps.

One goal is to generate the distribution packages once and then set configuration files at the distribution moment. This component of a piece may be saved in one or even more places; typical ones include startup credentials, registry entries, configuration data, and decentralized systems such as etcd, ZooKeeper, or Consul.

That is the ideal case, however, some rely on creating a single package for all contexts, which is more difficult than necessary.

Build once, deploy many is a software development strategy where a single codebase is developed and then deployed across multiple platforms or environments. While this approach has several benefits, it also comes with its own set of challenges.

Here are some of the challenges that may arise:

- **Platform-specific requirements:** Different platforms have unique requirements that must be considered when deploying software. For example, a web application deployed on AWS may have different requirements than one deployed on Google Cloud Platform. These

differences can cause issues when deploying the same codebase across multiple platforms.

Example: A software development team builds a web application using Node.js and deploys it on AWS. Later, they decide to deploy the same codebase on Google Cloud Platform. However, the Google Cloud Platform environment has different networking requirements than AWS, which results in the application not functioning correctly.

- **Configuration management:** Managing configuration files across multiple platforms can be challenging. Each platform may have its own configuration settings, which must be managed separately. Changes to the configuration on one platform may not be reflected on other platforms, which can lead to inconsistent behavior.

Example: A software development team deploys an application to multiple cloud platforms, each with its own set of configuration settings. Later, they need to make changes to the configuration file, but must manually update each configuration file on each platform. This can be time-consuming and error-prone.

- **Platform-specific bugs:** Different platforms may have bugs or compatibility issues that are unique to that platform. A codebase that works flawlessly on one platform may not function correctly on another, leading to platform-specific bugs.

Example: A software development team deploys an application to multiple platforms, including Windows, macOS, and Linux. The application works perfectly on Windows and macOS, but crashes on Linux. After investigation, the team discovered that the issue is related to a library that is only available on Windows and macOS, and not Linux.

- **Testing:** Testing software across multiple platforms can be time-consuming and expensive. Each platform may have its own testing requirements, which must be considered when deploying the same codebase across multiple platforms.

Example: A software development team builds an application using React Native and wants to deploy it to iOS and Android platforms.

The team must test the application on both platforms, which requires separate testing environments and may require additional resources.

- **Resource scaling:** Resource scaling is a key component of a Build Once, Deploy Many strategies in software development. It refers to the ability to dynamically adjust the allocation of computing resources (such as servers, memory, and processing power) to accommodate changes in workload, user demand, and application performance. Effective resource scaling is particularly valuable in this strategy because it allows you to optimize resource utilization, enhance flexibility, and reduce costs.
- **Cost management:** Cost management is crucial in a ‘build once’ strategy to ensure that the approach is cost-effective and aligns with budget constraints. While the strategy focuses on building a single codebase for deployment across various platforms or environments, it’s essential to manage costs efficiently.

In summary, Build Once, Deploy Many can be a useful strategy for software development, but it also presents its own set of challenges, including platform-specific requirements, configuration management, platform-specific bugs, and testing. These challenges must be carefully considered when deploying a single codebase across multiple platforms.

Library dependencies

While the answers might change in certain unique cases, everyone will accept that the least vulnerable level in a building architecture is the collection of libraries on which the application programming relies.

Maintaining interdependent packages is so critical in contemporary development that every language platform released in the previous 20 years has some type of de-facto basic package management in its environment to address this issue – npm, Yarn, Maven, NuGet, pip, gem, and so on. Recent languages even have *built-in* package management (dep for Go, Cargo for Rust, Swift Package Manager).

If the design team is not stringent regarding requirements, executing the very same build script based on the same software again may provide opposite findings: higher current versions may be brought in, the building might

unexpectedly crash owing to conflicting requirements, and Semantic Revision control principles might not be followed. Many changes in individuals and social additionally, uniquely identify binaries (for example, via key milestones), making it very hard to get duplicate files.

Two techniques may be used to tighten the hold on dependent libraries:

- Specify the precise library version to be used (often called pinning)
- Draw the library from a safe place, such as a shared folder or a full-fledged artifact server (Nexus, Artifactory, and so on.)

Indicate the precise library edition to be used (often called **pinning**). Draw library from a secure place, like a shared network or an artifact server (Nexus, Artifactory, and so on.)

Testing limitations

What is the ramification of this incapacity to replicate the same aberrations from different sources? These artifacts should indeed be regarded as a new release of the system, implying that tests would have to be rerun.

Individuals have complained about a variety of runtime difficulties as a result of upgraded libraries, including code failing on particular versions of operating systems, aesthetic defects that were undetected by automation testing, and so on. Most significantly, you will be unable to attribute changes in surroundings to settings. One spends a great deal of time seeking down the cause while pounding one's head against the wall since the program has not been updated.

Wrapping up

In the initial section, we discussed several basic challenges related to building vulnerability and present in solution lockin, emphasizing the need to deploy the very same resources across all configurations. In the following chapter, *Frequently Merge Code (CI)*, we will look at some real-world issues and discuss how to address them.

Building docker image

A Docker image is a lightweight, stand-alone, executable package that contains all the necessary code, libraries, and dependencies required to run a software application. Docker is a containerization platform that allows you to create and distribute these images, making it easier to develop, deploy, and run applications consistently across different environments. Docker images are a fundamental component of the Docker container ecosystem. In virtual machine settings, a picture is similar to a snapshot.

Docker is a tool for creating, running, and deploying programmers in containers. A Docker image includes the entire application, frameworks, tooling, requirements, as well as other items required to operate an application. Whenever an agent uses a picture, it may generate one or more representations of containers.

Docker images feature numerous layers, every one of which is derived from but distinct first from the preceding layer. The layering accelerates Docker deployments by improving reusability as well as reducing disc usage. Picture layers encompass interpreted documents as well. After creating a container, a writing layer is built on top of the immutable pictures, enabling the user to make adjustments.

Storage capacity allocations in Docker engine & applications might be perplexing. It is critical to differentiate between actual and virtual size. Dimension refers to the volume of disc space required by a bottle's write element, whereas virtualized dimension refers to the quantity of disc space needed by both the containers and the rewritable overlay. The noticed levels of such a photograph may be accessed by any containers that were created with the same picture.

There are different types of building Docker images, depending on the application requirements and use cases. Here are some of the most common types:

- **Single-stage build:** This is the simplest way to build a Docker image. It involves using a single Dockerfile to define the base image and copy the application code into it. This method is suitable for small applications with minimal dependencies.
- **Multi-stage build:** This is a more advanced technique that involves creating multiple stages in the Dockerfile to build different layers of the image. For example, you can have one stage to build the

application code and another stage to package it in a lightweight runtime image. This method is suitable for large applications with complex dependencies.

- **Cache optimization:** Docker images can be slow to build because each layer of the image is rebuilt from scratch when changes are made. One way to speed up the build process is to use cache optimization techniques, such as caching dependencies, to avoid unnecessary rebuilding of layers.
- **Buildkit:** Buildkit is a new feature of Docker that offers improved performance and security for building images. It uses a parallelized build process to speed up image creation and supports the use of declarative syntax for defining the build process.
- **Docker Compose:** Docker Compose is a tool that simplifies the process of building and deploying multi-container applications. It allows you to define the Docker image build process in a YAML file and run multiple containers in a single command.
- **Third-party tools:** There are many third-party tools available for building Docker images, such as Jenkins, Travis CI, Kaniko, and CircleCI. These tools provide a CI/CD pipeline for building, testing, and deploying Docker images.

In conclusion, building Docker images can be done in many ways, depending on the complexity of the application and the requirements of the development and deployment process. By choosing the right approach, you can optimize the build process, improve performance, and streamline the deployment of Docker images.

Docker image use cases

A Docker image contains all of the components required to operate a container - based applications, including software, configuration files, registry entries, frameworks, & virtual machines. Whenever an application is published in a Docker ecosystem, it may be run as a Docker container. The docker start function builds a container based on a picture.

Docker images are disposable assets that may be deployed on any server. Designers may reuse static picture elements through one application in the other. So, because the customer does not have to generate a picture starting from the beginning, they are saving effort.

Docker container vs. Docker image

A Docker container is a virtualized development kit that is used in the development of applications. It is employed to develop, operate, and deliver software that is separate from physical infrastructure. To execute multiple separate activities, a Docker container may utilize a single computer, sharing its kernels as well as virtualization technology with the operating system. Consequently, Docker containers are small.

In other kinds of VM settings, a Docker image is similar to a snapshot. It is a snapshot of a Docker container at a certain moment in time. Docker images are indeed unchangeable. They cannot be modified, but they may be replicated, distributed, and removed. The capability is handy for trying different applications or setups since the picture stays unaltered regardless of what occurs.

Containers cannot exist without an executable picture. Containers rely on pictures since they are employed to build execution contexts, which are required to execute a programmer.

Anatomy of a Docker image

A Docker image is composed of multiple levels, so each image contains anything required to set up a container atmosphere, including intimately linked, tools, prerequisites, as well as other documents. A picture's components comprise the following:

- **Base image:** With the create instruction, the operator may create this initial layer starting at the beginning.
- **Parent image:** A parent picture might be the initial level in a Docker image instead of a foundation picture. It is a repurposed picture that acts as the basis for all additional layers.
- **Layers:** Layers are attached to the underlying picture with code that allows it to execute in a container. Every level of a Docker image may

be seen using `/var/lib/docker/aufs/diff` or the Docker historical function in the (CLI). Docker's typical statuses display all top-layer pictures, along with the repositories, labels, as well as file format. The intermediary levels are stored, making the top levels more visible. Docker has storage systems that positively impact document management.

- **Container layer:** A Docker image generates in addition to a unique container as well as a readable or application surface. This layer maintains rewritten as well as destroyed documents, in addition to modifications to existing documents, and serves adjustments made to the operating container. This component is also utilized for container customization.
- **Docker manifest:** This is an extra file to the Docker image. It describes the picture employing JSON language, including the person, in this case, as well as a cryptographically signed.

Here is a sample Dockerfile:

```
FROM microsoft/dotnet:sdk AS build-env
WORKDIR /Docker

# Copy csproj and restore as distinct layers
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Build runtime image
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Figure 4.3: Dockerfile structure

Docker image repositories

Docker images are kept in both private and public repositories, such as the Docker Hub internet registration services, out of which users may deploy containers as well as test and distribute pictures. Docker Hub's Docker Verified Registry additionally allows for picture and access control.

Approved pictures are those created by Docker, while communal pictures are those created by Docker customers. The Docker image CoScale client is used to supervise Docker operations. A communal Virtual machine is Datadog/docker-dd-agent, a Docker container for an agent in the Service that enables a log management framework.

Users may also spawn additional pictures using current systems and publish vector graphics to the Docker Hub using the docker push function. Docker gives comments to writers prior to publication to ensure the integrity of communal pictures. The writer is charged with maintaining the picture after it has been uploaded. When getting a photograph from that other company, publishers must exercise caution since adversaries might acquire access to a computer using imitation pictures meant to fool a consumer into believing they originate from a trustworthy source.

The idea of the most recent photograph may sometimes be perplexing. Docker pictures with the tag :**latest** are not always the most recent. The newest tag is merely a default tag; it does not correspond with the most frequently published versions of a picture.

Creating a Docker image

Docker images may be built interactively or using Dockerfile

Commands to build a docker image

- To build the image, run the following command:

```
$ docker build -t my-app-image .
```

The **-t** option is used to specify the image name and tag.

- Once the Docker image is built, you can use the **docker images** command to list all available Docker images on your system:

```
$ docker images
```

- To run a container from the Docker image, use the `docker run` command:

```
$ docker run -it image-name
```

These are the basic Docker commands to build a Docker image. You can customize the Dockerfile to include additional instructions, such as installing packages, copying files, and exposing ports. You can also use Docker Compose to define a multi-container application and run it in a single command.

Interactive method

Users are using this approach to start a container from an established Docker image as well as interactively adjust the atmosphere after exporting the picture. The dynamic technique is the most user-friendly way to generate Docker images. The first step is to start a command connection and run Docker. Then launch Docker using the image `name:tag` name instruction. This launches a terminal connection with the containers started by the picture. Docker utilizes the much more current version of the file if the tag description is not specified. Following that, the picture must be displayed in the result.

Docker file method

This method necessitates the creation of a plain text Docker file. The Docker file specifies the parameters for building an image. This is a more complicated technique, but it works well in continuous deployment setups. The procedure entails generating the Docker file and implementing the picture's instructions. When the Docker file is launched, the user creates a `docker ignore` file to exclude all files that are not required for the final build. The `docker ignore` file may be found in the root folder. The Docker building program is then used to generate a Docker container, as well as a photograph identifier as well as tag are assigned. Finally, the Docker photograph's function is employed to inspect the newly produced images.

Docker image commands

As per Docker, there are various sets of main Docker image functions, which are grouped as children functions; such examples involve:

- `docker image build`: Creates an image based on a Docker file.
- `docker image inspect`: This command shows data about one or even more pictures.
- `docker image load`: Loads a picture from a tar archive or streaming in preparation for accepting or retrieving inputs (STDIN).
- `docker image prune`: Get rid of unneeded pictures.
- `docker image pull`: This command retrieves an image or a source from a registrar.
- `docker image push`: This command sends a picture or a directory to a registrar.
- `docker image rm`: This command deletes one or even more pictures.
- `docker image save`: This command publishes one or even more pictures to a tar package (streamed to `STDOUT` by default).
- `docker image tag`: Generates the tag `TARGET IMAGE`, which corresponds to the tag `SOURCE IMAGE`.

The Docker CLI includes commands for customizing Docker images. Following are some Docker image functions:

- `docker image history`: Displays an image's background, containing modifications made to it and its components.
- `docker update`: Allows a user to change the container settings.
- `docker tag`: Generates a tag, such as `TARGET IMAGE`, which allows people to manage and arrange container images.
- `docker search`: Searches Docker Hub for anything the customer requires.
- `docker save`: Allows a person to save some images to a directory.
- `docker compose`: This command is utilized to manage a configuration file.

Docker images is an essential topic and technology to understand while developing apps in a container - based environment using Docker. Understand further about container management and safeguarding Docker images.

Following figure shows the differences between Docker images, containers, and files:

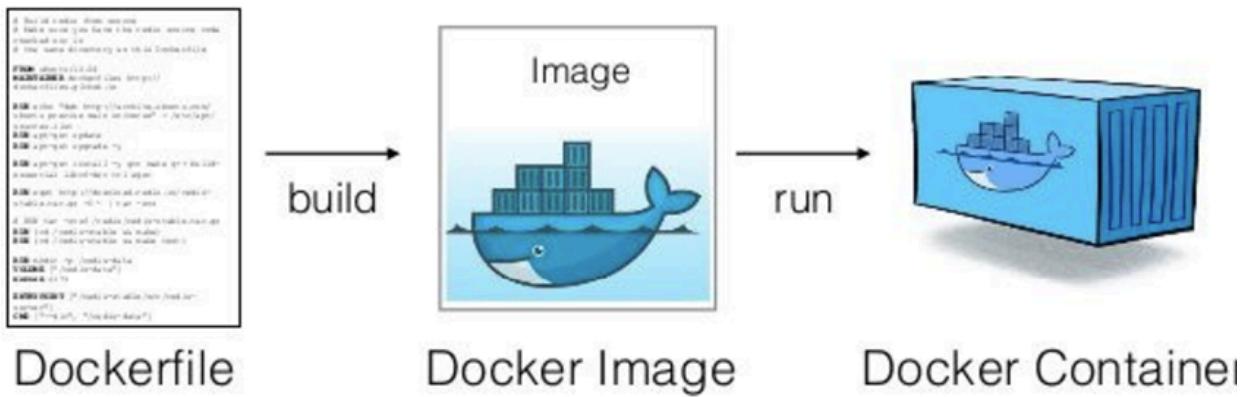


Figure 4.4: Difference between Docker file, Docker image and Docker container

Environment overlays

Environment overlays, in the context of application deployment and containerization, typically refer to a technique used to manage and apply environment-specific configurations and settings during the deployment of an application or service. This concept is often used with container orchestration platforms like Kubernetes to handle differences between environments (for example, development, testing, and production) without modifying the application code.

Environment overlays allow you to overlay or apply environment-specific settings and configurations on top of a base configuration or application image. This approach is valuable because it enables you to maintain a single, consistent codebase while adapting it to different environments.

Here is how environment overlays work in the context of application deployment:

- **Base configuration:** Start with a base configuration, which may be a configuration file, a Docker image, or a set of environment-agnostic

settings that are common across all environments. This base configuration represents the default behavior of your application.

- **Environment-specific Overlays:** Create separate overlays or configuration files for each environment in which you intend to deploy the application (e.g., development, testing, production). These overlays contain settings and configurations specific to each environment, such as database connection strings, API endpoints, logging levels, or security parameters.
- **Application deployment:** When deploying your application, you apply the appropriate environment overlay for the target environment. The overlay is combined with the base configuration to create a customized configuration specific to that environment.
- **Container orchestration:** In containerized environments, container orchestration platforms like Kubernetes often provide mechanisms to manage and apply environment overlays. For example, Kubernetes ConfigMaps or Secrets can be used to store environment-specific configurations, which are then mounted into containers as volumes or exposed as environment variables.

By using environment overlays, we can achieve the following benefits:

- **Code reusability:** Maintain a single codebase that remains consistent across all environments, reducing code duplication and complexity.
- **Configuration management:** Centralize the management of environment-specific configurations, making it easier to update and maintain settings.
- **Security and compliance:** Sensitive data (e.g., secret keys, passwords) can be kept separate from the code and secured in environment-specific overlays.
- **Scalability:** Easily replicate and scale the application across different environments while adapting to their specific requirements.
- **Testing and validation:** Ensure that configurations are validated and tested for each environment, reducing the risk of configuration errors.

- **Version control:** Apply version control to environment overlays, allowing you to track changes and manage configurations using a version control system.

The exact implementation of environment overlays can vary depending on the tools and platforms you are using. In containerized environments like Kubernetes, you may leverage ConfigMaps, Secrets, or Helm charts to manage and apply these overlays. For non-containerized applications, you can use environment-specific configuration files or settings stored in a centralized location.

Overall, environment overlays provide a valuable mechanism for managing and adapting applications to different deployment environments while keeping the codebase consistent and maintainable.

How do overlays affect application deployment

Here are some ways overlays can affect app development:

- **Customization and theming:** Overlays can be used to customize the appearance and behavior of an application, such as changing the color scheme, fonts, or user interface elements. This is often used in the development of user-facing applications to provide a branded or personalized user experience.
- **Localization and internationalization:** Overlays are valuable for making applications suitable for different languages and regions. By applying language-specific or region-specific overlays, you can translate text, adjust date formats, and adapt the app's content to different locales.
- **Feature toggles:** Overlays enable the implementation of feature flags or feature toggles, allowing developers to enable or disable specific features or functionalities in the app. This can be useful for controlled feature rollouts, A/B testing, or temporarily disabling features without changing the core code.
- **Modular and extensible architecture:** Overlays can be used to extend an application's functionality by adding or removing modules

or plugins. This approach can make the app more modular, allowing developers to add new features without altering the core codebase.

- **Configuration management:** Overlays are commonly used to manage application configurations. By applying configuration overlays, you can set different settings for various environments (for example, development, testing, production) without modifying the application's source code. This approach enhances the portability of the application across different deployment environments.
- **Security and permissions:** Overlays can be employed to enforce security policies and permissions. For example, you can apply access control overlays to restrict or grant access to certain parts of the app based on user roles or permissions.
- **Bug fixes and patches:** Overlays can be used to apply hotfixes or patches to an application without altering the primary codebase. This approach is valuable for addressing critical issues without waiting for a full software release.
- **White labeling:** Overlays are commonly used in white-label applications where the same core application is customized and branded differently for different clients or partners.
- **Legacy systems integration:** Overlays can help integrate new technologies or features into legacy systems without disrupting the existing codebase. This is particularly important in maintaining backward compatibility.
- **Testing and experimentation:** Overlays are used in testing environments to create variations of an app for different testing scenarios, such as load testing, performance testing, and user acceptance testing.
- **Versioning and releases:** Overlays can play a role in version management. For example, you can apply version-specific overlays to ensure that users are running a compatible version of the app.

The specific implementation of overlays varies based on the technology stack, programming languages, and development tools being used. Overlays are often applied through configuration files, settings files, or scripts that are

loaded at runtime. In some cases, they may be managed and deployed as part of a larger configuration management or CI/CD process.

Continuous deployment

Continuous deployment is a shorter development approach in which each code contribution that succeeds the test automation process is continuously published into the manufacturing process, resulting in observable changes to the project's customers.

Continuous integration removes artificial barriers to untested coding in live applications. It should only be utilized when the programming and IT teams strictly adhere to manufacturing methodological approaches and comprehensive validation, and when comprehensive, genuine surveillance in operations is used to detect any difficulties with big releases.

Continuous deployment vs. continuous delivery

Ongoing computer programming encompasses agile method, production, deployments and, therefore, is connected with the DevOps and Agile techniques. Constant delivery and distribution are derived from agile development, a process of quickly developing, building, and testing new code using automating such that only command verified to be excellent is included in a software package.

And though the phrases are sometimes conflated and contain the abbreviation CD, test and deploy and continuous integration are not the exact same thing.

When programmers often give over unique script to the **quality analyst (QA)** and operational processes for evaluation, this is referred to as continuous delivery. Continuous delivery often includes a manufacturing staging ground, as well as frequently a significant delay such as when a version is evaluated, when modifications are physically approved, and the moment the new software is delivered to production.

Continuous deployment, on the other hand, does not need a staging ground for physically reviewing and verifying application code since system testing is included at an early stage of development and continues across all the stages of the distribution. Another primary advantage of system

implementation is the absence of a time lag between the moment a code update passes application development services and marketplace validation and the moment it goes live.

These CDs depend on massive infrastructure procurement and software surveillance systems to identify any issues that were missed during development feedback mechanisms prior to deployment. Due to the lack of person-checking functionality, testing and oversight are more important in continuous deployment.

Regulatory requirements or other constraints may hinder an IT company from implementing continuous deployment. Additional factors, including the maturation of DevOps procedures and best - practices within in the IT organization, may affect the choice to release software on a continuous delivery, continuous deployment, or a hybrid of the two techniques dependent on the program and customers.

Continuous deployment tools

Continuous deployment streams use technologies comparable to those used in continuous delivery, but with a greater focus on software validation both before and after deployment into operation.

During development, change control and building automated, as well as specific technologies like the project management program Apache Maven, guarantee that code is delivered smoothly utilizing configuration management process software like Jenkins.

Unit and component testing involve running programmed scenarios through various execution situations to anticipate their behavior in operations. NUnit, TestNG, and RSpec are examples of test automation framework.

IT automated and system integration solutions, like **Puppet and Puppet**, automate software release and hosted resources setup for continuous deployment. Grasshopper, Selenium, JUnit, TestNG, and Cucumber known are tools that can be utilized for developing integration and acceptability tests.

Actively looking like **AppDynamics and Splunk** detect and notify any variations in applications or infrastructural speed caused by the new coding, and they can trigger IT incident management solutions and services like

PagerDuty. Management and problem management for production-ready setups need to be as near to instantaneously as feasible in order to reduce time towards restoration in the event of code errors.

Rollback features must be included in the distribution tool set so that any unanticipated or undesirable consequences of program software in circulation may be swiftly identified and handled. To protect against user disturbance from system implementation, organizations may utilize bright yellow distribution and partitioning, blue/green implementation, functionality flags or toggle switches, and other deployment controls.

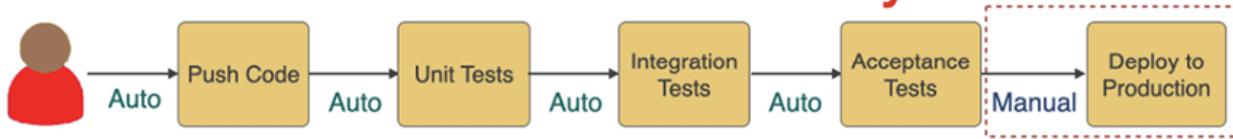
Certain software, like Docker, may be deployed in a container to separate changes from the underlying architecture.

The standards and specifications in your business will benefit greatly from continuous integration and ongoing deployment. Your code is simply pushed to a source by your engineers. This code will be combined, verified, deployed, and checked again before being combined with architecture and ready to be deployed with absolute confidence.

When CI/CD is utilized, code quality improves, and new apps are provided fast and with great assurance that no catastrophic change will be introduced. Any update's effect may be connected with information from operations and manufacturing. It can also be used to plan the very next cycle, which is an important DevOps technique in your firm's online transition.

Following the difference between continuous delivery and continuous deployment is shown:

Continuous Delivery



Continuous Deployment

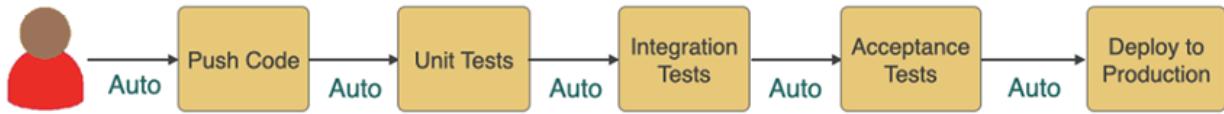


Figure 4.5: Difference between continuous delivery and continuous deployment.

Pros and cons

Build Once and Deploy Many is a software development approach in which a single codebase is created and then deployed to multiple platforms or environments, such as different operating systems, devices, or servers. Here are some pros and cons of this approach:

Pros:

- **Saves time and effort:** By building the software only once, developers can save a significant amount of time and effort that would have been required to create and maintain separate codebases for each platform.
- **Increases consistency:** Since the same codebase is used across all platforms, it ensures a consistent user experience and reduces the risk of bugs and inconsistencies between platforms.
- **Simplifies maintenance:** When there is only one codebase to maintain, it is easier for developers to update and fix issues in the software.
- **Reduces costs:** Building and maintaining separate codebases for each platform can be expensive. The Build Once, Deploy Many approaches

reduce these costs by streamlining the development and deployment process.

- With the Build Once, Deploy Many approaches, you can streamline development, improve release velocity, and get your product to market faster while maintaining code quality and consistency across different platforms and environments.

Cons:

- Limited platform-specific optimization:** Since the same codebase is used across all platforms, it may not be possible to optimize the software for a specific platform. This can result in suboptimal performance on some platforms.
- Compatibility issues:** There may be compatibility issues between the software and certain platforms, which may require additional development and testing efforts.
- Lack of platform-specific features:** Certain platforms may have unique features that cannot be incorporated into the software using the Build Once, Deploy Many approaches.
- Complexity:** The process of building a software that works across multiple platforms can be complex, requiring developers to have a deep understanding of the nuances of each platform.
- The Build Once, Deploy Many strategies can have a significant negative impact on application performance.

Overall, the Build Once, Deploy Many approaches can be a powerful tool for software development, but it requires careful consideration of the specific needs and requirements of each platform.

Conclusion

In conclusion, the Build Once, Deploy Many approaches can be a useful strategy for software development, offering significant benefits such as reduced development time, consistent user experience, and simplified maintenance. However, this approach also has its drawbacks, including limited platform-specific optimization and the potential for compatibility

issues. Ultimately, the decision to use this approach will depend on the specific needs and requirements of the software and the platforms it will be deployed on. Developers must carefully weigh the pros and cons of this approach before deciding whether it is the best option for their project.

In the next chapter we are going to learn Frequently merge code. Frequently merging code, is a software development practice also known as **continuous integration (CI)**, which involves frequently integrating code changes from multiple developers into a single shared code repository. The goal of continuous integration is to ensure that changes made to the codebase by individual developers are regularly tested and integrated into the larger project as quickly and efficiently as possible.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 5

Frequently Merge Code: Continuous Integration

Introduction

Formulation, project planning, software implementation, creation, installation, coding, and testing, as well as maintenance and upkeep, are all parts of the **Software Development Life Cycle (SDLC)** process. Planning, analysis, design, development, testing, implementation, and maintenance are the modern SDLC stages:

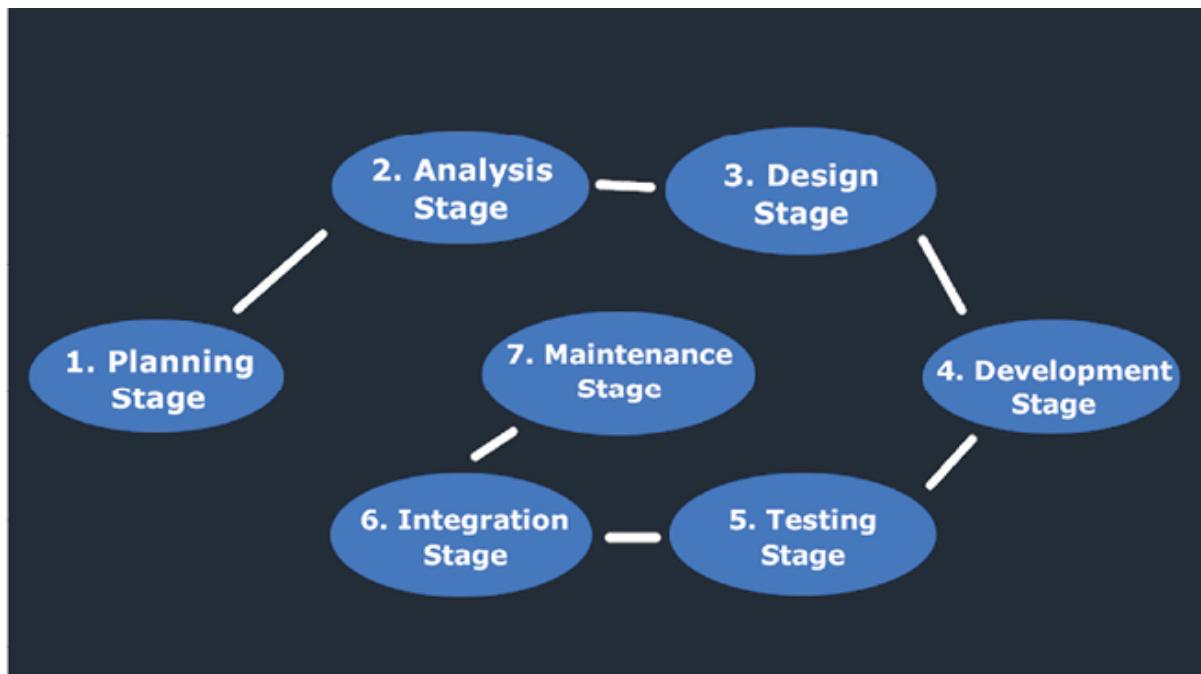


Figure 5.1: Stages of assessment

In the creation of new technologies, the operational life technique is applied. It is divided into four steps: needs assessment, design, execution, and assessment. The first stage of this procedure is to assess your requirements.

Structure

In this chapter, we are going to cover **continuous integration (CI)**, which is a software development practice that involves regularly and automatically building, testing, and integrating changes made to a codebase. The goal of CI is to catch and fix problems in the codebase as early as possible in the development process, to prevent those problems from causing larger issues down the line.

The following topics will be discussed in this chapter:

- Commit early, commit often
- Build only once
- Clean your environments
- Monitor and measure your pipeline
- Keep the builds green
- Streamline your tests
- Make it the only way to deploy to production
- Make it a team effort
- Best tools to build robust CI process.
- Step-by-step build robust and fully automatic CI process.

Objectives

The objective of this chapter is to explain the concept of CI and its role in modern software development practices. The topic aims to provide a clear definition of what CI is, how it works, and its benefits. It seeks to highlight the importance of implementing CI in a software development team, and how it can help to improve code quality, reduce risks, and streamline the development process. By the end of this topic, readers should have a solid understanding of CI and its significance in the software development industry.

Traditional software development

Conventional software development entails the process of designing and developing basic software. It is utilized whenever privacy, and other aspects of something like the programmer become unimportant. It is employed by newcomers to construct programming.

Traditional software development approaches are centered around the production lifecycle's pre-defined stages of grief. The developmental process is continuous throughout, from objectives to architecture, then it is to implementation, validation, and management:

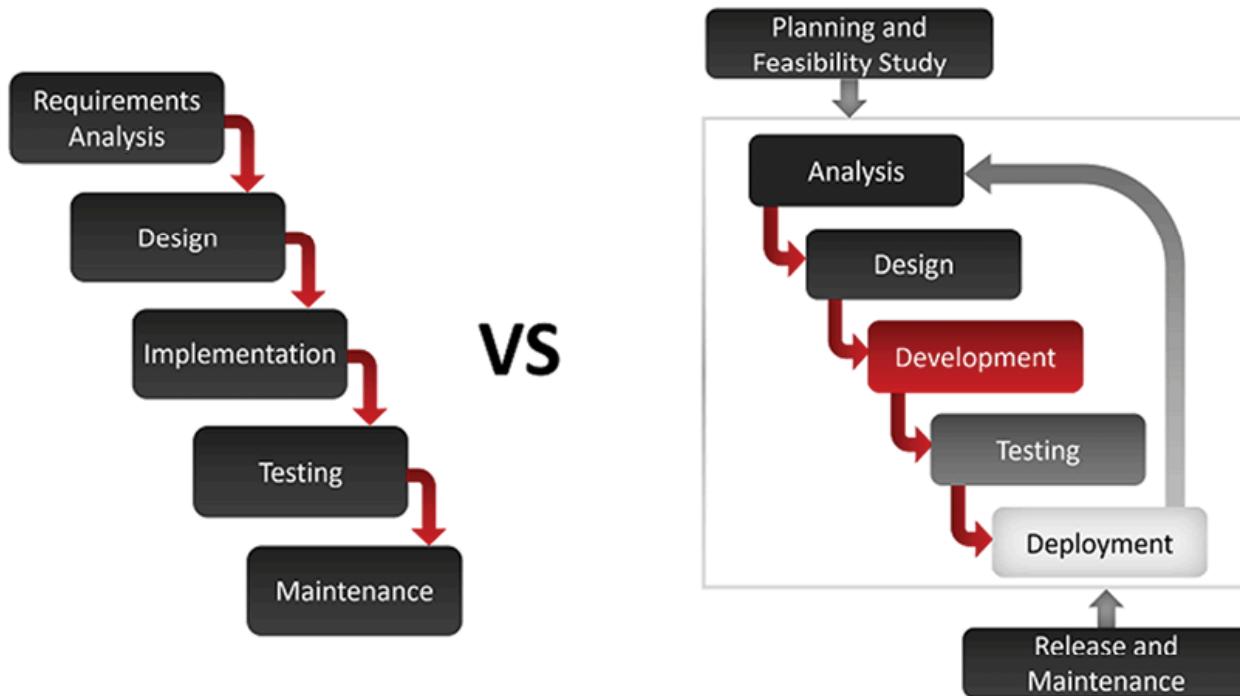


Figure 5.2: Various testing

Traditional systems development approaches may be used inside the company in several ways. Cascade, Spiral, V-Model, **Rational Unified Process (RUP)**, and Rapid Software Engineering, are some of them.

The production process of a commodity is often classified into four types: introduction, growth, maturity, and decline:

PRODUCT LIFECYCLE

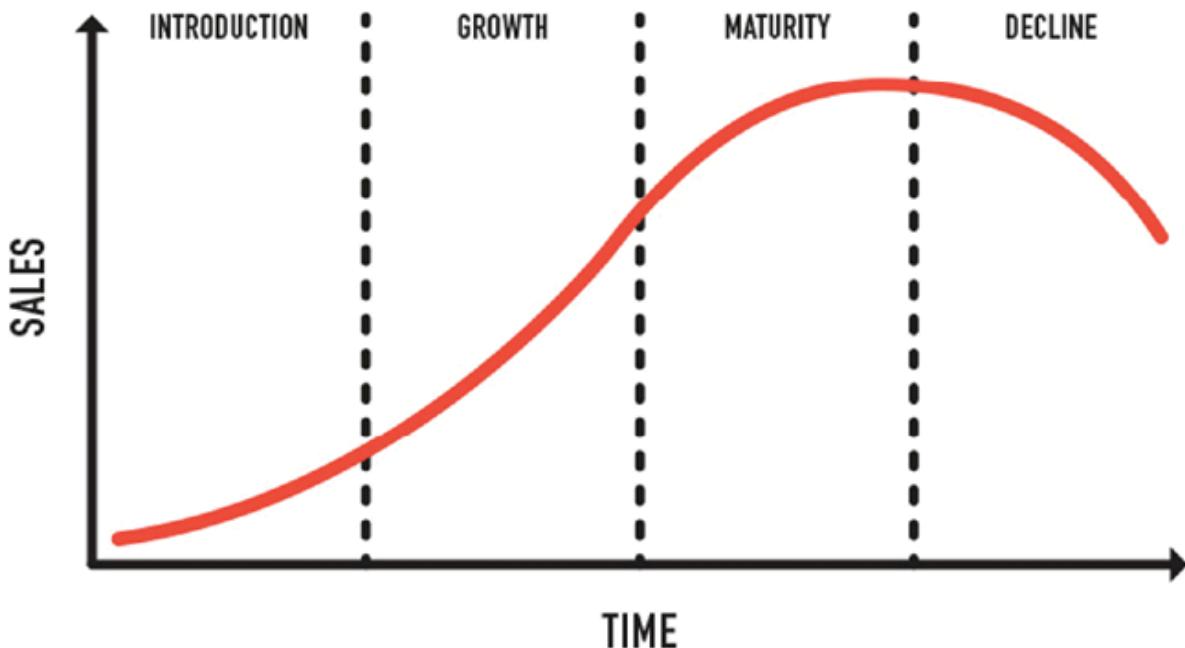


Figure 5.3: Product lifecycle

The study of organizations. Simulation of difficult structures or predictive modeling. Approximation of soft systems System modeling depending on processes.

Nevertheless, Rainfall is typically regarded as the most traditional of the hierarchical SDLC techniques. It is also a simple methodology, complete one step before moving toward the next:



Figure 5.4: SDLC technique

The production process describes how a company progresses through five different stages: conception, introduction, growth, maturity, and collapse. *Theodore Levitt*, a German mathematician, invented the notion and publicized his Manufacturing Process framework in the *Harvard Review of Business* in 1965. This generation continues to be in use nowadays.

During the conception, creation, testing, and ultimate implementation of the project, there is still extremely little possibility of encountering defects, mistakes, or difficult challenges. For shorter but instead consecutive initiatives, businesses choose the Traditional Research methods. They are completed in a shorter time, with more programmers and with less expenditures.

Anything is readily available and accessible with agile technique. Customers and judgment calls are actively engaged in the development, evaluation, and implementation phases. In the conventional model, the project coordinator controls the program, for those who do not get to make important choices.

The underlying principles of a software platform, and the profession of building similar architectures and systems, are referred to as software design. Software components, relationships between them, and attributes of both components and relationships constitute the building blocks of every structure.

The difference between SAAS and conventional source code:

SAAS solutions are generally internet application programs that can be accessible through a browser window and do not need to be downloaded. In comparison, to

utilize regular computer mobile applications, individuals must first download them on a personal machine.

Software as a Service (SaaS) solutions are prevalent in various industries, offering cloud-based software applications that cater to specific business needs. Here are some examples of SaaS solutions in the real world:

- **Salesforce:** Salesforce is a **customer relationship management (CRM)** SaaS solution that helps businesses manage their sales, marketing, and customer support processes.
- **Microsoft 365 (formerly Office 365):** Microsoft 365 offers a suite of cloud-based productivity tools, including Word, Excel, PowerPoint, and Outlook, facilitating collaborative work and communication.
- **Google Workspace (formerly G Suite):** Similar to Microsoft 365, Google Workspace provides cloud-based productivity apps like Gmail, Google Docs, Google Sheets, and Google Slides.
- **Zoom:** Zoom is a popular video conferencing SaaS solution that enables online meetings, webinars, and video collaboration for remote teams and individuals.
- **HubSpot:** HubSpot offers a SaaS platform for inbound marketing, sales, and customer service, helping businesses manage and automate their marketing and customer relationship activities.
- **Slack:** Slack is a team collaboration tool that provides real-time messaging, file sharing, and integrations with various other SaaS applications.
- **Dropbox:** Dropbox is a cloud storage and file-sharing SaaS solution that allows users to store, access, and share files from any device with an internet connection.
- **Zendesk:** Zendesk is a SaaS solution for customer service and support that helps organizations manage customer inquiries and issues efficiently.
- **Shopify:** Shopify is a SaaS e-commerce platform that enables businesses to set up and manage their online stores, offering features for inventory management, payment processing, and website customization.
- **Trello:** Trello is a visual project management tool that uses boards, lists, and cards to help teams organize tasks and projects collaboratively.
- **QuickBooks Online:** QuickBooks Online is a cloud-based accounting and financial management SaaS solution for small and medium-sized businesses.

- **Stripe:** Stripe is a payment processing SaaS platform that allows businesses to accept online payments, manage subscriptions, and handle financial transactions.
- **AWS (Amazon Web Services):** AWS is a comprehensive cloud computing platform offering a wide range of SaaS solutions, such as Amazon S3 (cloud storage) and Amazon EC2 (virtual servers).
- **Adobe Creative Cloud:** Adobe Creative Cloud provides access to various creative software applications, including Photoshop, Illustrator, and InDesign, through a SaaS subscription model.
- **Mailchimp:** Mailchimp is a marketing automation platform that offers email marketing, audience segmentation, and campaign management for businesses.

These are just a few examples of the many SaaS solutions available across different industries, catering to a wide range of business needs and functions. SaaS is known for its scalability, cost-effectiveness, and accessibility, making it a popular choice for businesses of all sizes.

Conventional approaches

Traditional approaches are blueprints, involving work starting with the elaboration and documenting a comprehensive set of specifications, then moving on to engineering, quality construction evaluation and development. Because of these weighty elements, this system becomes characterized as heavyweights.

Typical creative process

Conventional architectural, sometimes known as *sequential mechatronics*, is the advertising, structural engineering, manufacturing, running tests, and manufacturing method in which each step of the project development is completed independently and the next level does not begin again until the preceding one is completed.

Examples of conventional growth

Traditional methods of human progress have stressed change from infancy through adolescence, adulthood steadiness, and deterioration in old life. Roebuck, and Grossman, has caught the essence of various most significant adult transitions. During one of these latter decades, there may be major changes in physique, character, and talents:

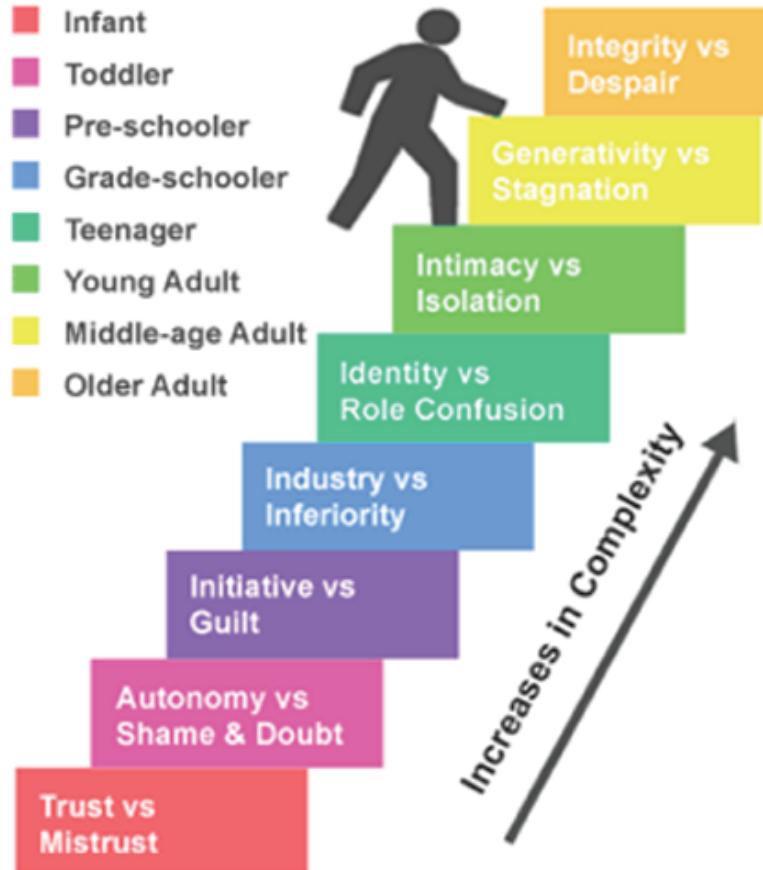


Figure 5.5: Conventional growth

What distinguishes traditional architecture

Though the origins of these forms vary, classic dwellings have some characteristics. Wide, spacious galleries featuring overlapping rafters, trusses, skylights, and a tall, pointy roof including one or maybe more flying buttresses are among these traits. Construction project elements like bricks, timber, cement, and granite are used.

Significance of architectural styles

The architectural arts, and the stunning architecture they permit, give a crucial connection to the past and contribute to a strong feeling of community culture. In addition to other cultural manifestations, they serve as a basis for both the common humanity of big and small populations.

Conventional software advancement

Conventional Software Innovation: Conventional software development refers to the process of designing and developing basic software. It is employed whenever

cybersecurity and the other aspects of the program are unimportant. It is employed by newcomers to construct applications.

Some popular uses

Conventional is defined as *anything that adheres to a lengthy history, style, or practice*. The tradition of consuming turkeys as the standard or acceptable Thanksgiving dinner is an illustration of convention. A formal design of upholstery in which it doesn't alter with trends or summers is indeed an illustration of conventional.

Why is contemporary architecture superior to classical?

Resource as well as moderate picture: Infrastructure such as stone borders, for illustration, may last tens of thousands of years more than requiring minimal or no upkeep. New architectural concepts often survive barely a generation less than prior to requiring significant upkeep.

Traditional architecture

The most prevalent type of house in the United States today is a traditional model. The style combines historic characteristics, from old residences with current components of interior decoration. This design is distinguished by uniformly arranged openings, basic sloped ceilings, and minimal decorations.

A typical life cycle model includes a certain number of steps:

Image outcome: This cycle typically consists of six stages: project planning, design, engineering and testing, implementations, marketing, and assessment:

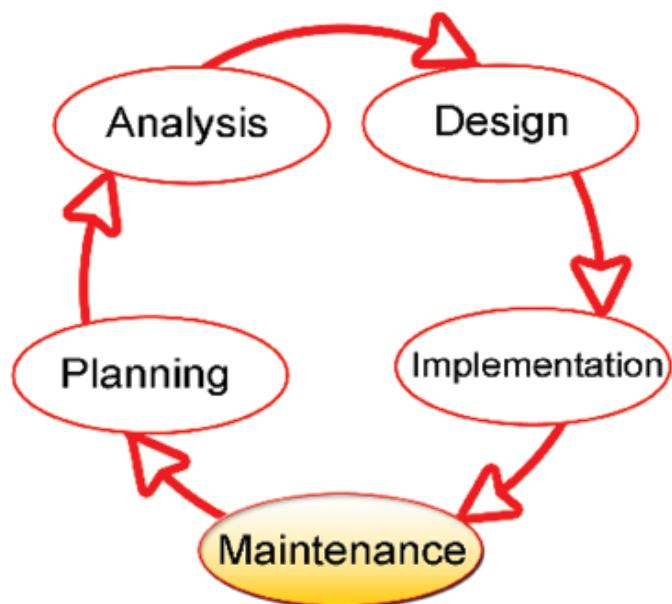


Figure 5.6: Traditional software development life-cycle model

Commit early, commit often

Rather than waiting for the commitment to be flawless, work in tiny pieces and mass-commit your progress. If you are operating on a development branching that may take considerable time to complete, it is beneficial to maintain existing code snippets to current, with the most recent modifications to prevent clashes.

Git has become the de facto **version control system (VCS)** of choice for programmers. Utilizing Git offers enormous value, particularly for engineers and designers with several engineers, because it becomes vital to have a framework for consistently merging everyone else's content. With any strong weapon, particularly one that requires cooperation with each other, it is necessary to set ground rules to avoid shooting themselves in the leg. The resource has developed certain basic guidelines for our personal organization that make using a VCS like Git simpler.

Here are the five easy guidelines to pursue:

- **Make clean, single-purpose commits**

Whenever concentrating on almost anything, developers are often diverted into doing a plethora of activities, for example, when you are attempting to solve one defect and choose someone else, you cannot seem to stop the impulse to address both. And yet another. It quickly gets out of control, or otherwise you risk ending up with a slew of improvements everything entering into one submit. This is troublesome, and it would be preferable to keep commitments short concentrated and practicable for a variety of reasons, which include:

- It makes it simpler for other team members to evaluate your update, enabling programmers extremely productive.
- If the commitment must be totally reversed, it is significantly simpler to accomplish this task.
- These modifications are simple to monitor using their online booking system
- It also assists you in mentally parsing amendments made using git log.
- Write meaningful commit messages
- Frequent branching merging, and rebasing are essential for maintaining an organized and collaborative development process.

Arguably, this is the most straightforward task, with insightful and intricately detailed commitment comments precisely outlining the modifications intended as part of a commitment. This aims to simplify life not only for others but also for your future self. This is a great criterion for crafting decent submitted communications.

The sentence “feat: add beta sequence” appears to be a commit message commonly used in version control systems like Git. This type of commit message is a convention often employed by developers to signify the addition of a new feature, in this case, a “beta sequence.”

```
feat: add beta sequence
```

If you are employing an online booking system, provide the complaint id inside the descriptions as well. A variety of newer VCS storage platforms, such as GitHub, automatically connect contributions to problems.

- **Commit early, commit often**

Git operates efficiently, especially to one's benefit, when consistently publishing current progress. So, rather than waiting for the commitment to be flawless, work in tiny pieces and end up committing all progress. When you are developing on a development branching that may take considerable time to complete, it is beneficial to maintain your git repository to date with the newest modifications to prevent problems. Furthermore, Git simply assumes complete responsibility regarding your information whenever you submit it. When utilizing git-reflog, it prevents you from getting fired, undoing modifications, and tracing your actions.

- **Do not alter published history**

It is highly encouraged not to change history when one contribution has already been incorporated into a downstream standard branch (and therefore is accessible to everybody else). Git, and other VCS technologies, can be used to alter branching histories, but this is difficult because everyone has access to the source. Although source control is a great function, it should only be used on projects when you are the only one functioning. Despite this, there could be times when a historical update on a publicized branches is required. While doing it, one should be extra careful.

- **Do not commit generated files**

Essentially, documents that rely on human effort for their creation should ideally be submitted. It is possible to submit files at any time, so paragraph diff monitoring may not always work effectively. It is recommended to

include an arginine document at the root of your repositories to immediately notify Git of whatever documents or URLs users do not want to monitor.

- **Making regular commitments**

The basic guideline (for all cases) is to submit as frequently as feasible. If you believe “it is just not prepared yet” (either because it will disrupt software building or because it is not finished yet), establish a clone and contribute towards that version and make sure to submit.

When should someone press the committed button?

Pressing the commit button a few times a day is generally sufficient. As @earlonrails mentioned, more frequent commits reduce the risk of conflicting changes, but it is usually not a major concern.

Consider this: by committing to the local repository, you are essentially stating, ‘I trust the above code. It is complete.’

Advantages of investing frequently

Being willing to commit regularly allows you to debug changes that also have damaged an aspect of the design phase and reverse without sacrificing much more effort. You also receive the extra benefit of being able to view modifications by having to navigate a file containing 1000-line modifications.

Must someone make a commitment with each transition?

Usually, must publish once you have made a little progress on your current project. It is advisable to do it at periodic intervals, ensuring that you constantly have such a duplicate of their efforts.

Must you press each time you make a commitment?

For this reason, you would contribute often enough and continuously publish. Pushing if you publish. Preferably, you would submit each time you accomplish a task. Always publish a roughly twice program without already posting comments and some dummy language.

Significance of committing

The relationship of mutual understanding is formed through dedication. It establishes a mutual understanding and duty among 2 people. It gives meaning and a vow to keep.

Secret to committing

Passion is powerful. You must have a strong and particularly convincing motivation to completely dedicate to anything. Without a deep desire, anyone will

suffer when the going gets tough. With such strong ambition, overwhelming hurdles are perceived as obstacles that must be overcome.

Build only once

The conventional procurement process is often known as ‘design-bid-build,’ ‘bid and construct,’ or ‘contractor sponsored’. It is still the most popular form of acquiring construction works. The customer initially hires specialists to thoroughly develop the project before preparing the contract documents such as drawings, working time, and bills of quantity. Manufacturers are then requested to submit bids for the development’s development, which is normally done in a straight, simultaneous step. This is known as a **conventional agreement**. Except for site preparation, the bidder is not accountable for design, however some conventional arrangements may allow the project team to construct sections of the tends to work. Even during building a project, the customer often employs consultants to generate whatever extra project documentation that would be necessary, to evaluate any blueprints designed by the contractor, and to check the activities. Typically, 1 advisor (usually, but not exclusively the architects) is selected to manage the agreement.

Mostly during construction activities, the customer often employs external experts to generate any extra construction documents that would be necessary, to evaluate whatever drawings are made by the contractor, and to check the works. Typically, one specialist (often, but not always, the architect) is selected to manage the arrangement. Traditional production arrangements are typically lump-sum arrangements; nevertheless, types of contracts and deferred payment contracts may also be used for ‘conventional’ undertakings in which design and implementation are distinct, consecutive operations. This purchasing method is appropriate, for both knowledgeable and beginner customers. Developing the architecture completely before tendering provides the client with assurance regarding the quality of the architecture and its cost. Although it can be extremely slow than most other configurations of collective bargaining, and because the construction company is not assigned until the design process, they are unable to dramatically develop the construction process and packaged foods of recommendations as they establish. It is a minimal technique of procurement for the customer since the subcontractor bears the financial burden for implementation. Nevertheless, if plans and specifications are missing at the time of bidding, or if major changes are necessary after the constructor is hired, the price to the customer might be high. A policy usually might be viewed as combative because of this, the segregation of design and development. Basic contract – advantages and disadvantages The JCT (National Company law Commission) Conventional Prime Contractor, Interim

Buildings Arrangement, and Substantial Works Development Collective bargaining agreement continue to be the most prevalent types of conventional service agreements.

When is it permissible to use it?

Suppose the customer has already had the system developed. If the concept is almost finished at the moment of competitive tendering, the customer prefers to maintain the design's oversight and specifications. Price assurance at the commencement of construction is critical. The customer's key objective is not the smallest cumulative project.

The benefits of conventional procurement

The following are some benefits of conventional procurement:

- Equality in competition and a straightforward procedure.
- Architecture - can mean quality.
- Pricing uncertainty prior to start-up.
- Treatments that are well-acknowledged.
- Improvements are very simple to implement and evaluate.

Drawbacks

The following are some drawbacks:

- The overall system length may be greater than that of other acquisition channels due to the rate and level.
- The contractors had no say in the construction or execution. Aggressive price techniques may result in hostile interactions.
- Simultaneous duty - engineering team comprising development and construction subcontractor.
- Time and expense predictability are decreased if the proposal is not finalized at the tendering stage.

Build Binaries Only Once, or BBOO, is a core idea of CI. This indicates that the binary artifacts should only be built once. These artifacts ought to then be saved in a repository management like Nexus Storage site. Following deployment, test, and deployment rounds, you should not ever try to create this executable anymore but should instead recycle it. This assures that the same binaries have been tested and supplied to the consumer several times. In numerous instances throughout each testing step, binary is recreated using a certain workstation identifier and regarded

as the exact same thing. But that remains distinct! This could also turn out to be the exact same, but that is simply coincidental. It is more than likely not the same due to varied environment setups. For example, the developer team may use JDK 8 on their workstation, whereas the experiment team may use JDK 7. There are several reasons why the binary artifacts might vary. As a result, it is critical to generate binaries just once, save them in a container, and put those through many developments, and production cycles. This boosts the positive customer trust in the presentation.

Setting up build binaries only once

Let us start with the setup:

- Once created, a Java EE 7 applications WAR file is stored in a Nexus source or a.m2 local library.
- For the shadow test, this very same executable is utilized.
- The same executable is applied to run the whole automated test.

In our scenario, the smoked testing should be a standardized test, whereas the entire suite would have four tests. Obviously, this is not one's average configuration in terms of the quantity of tests, but at the absolute least, you have a look at how all of that is arranged. There are just two phases of assessment, smoking and complete, but the idea may simply be expanded to include other rounds. A full-fledged distribution process will be shown in a forthcoming section:

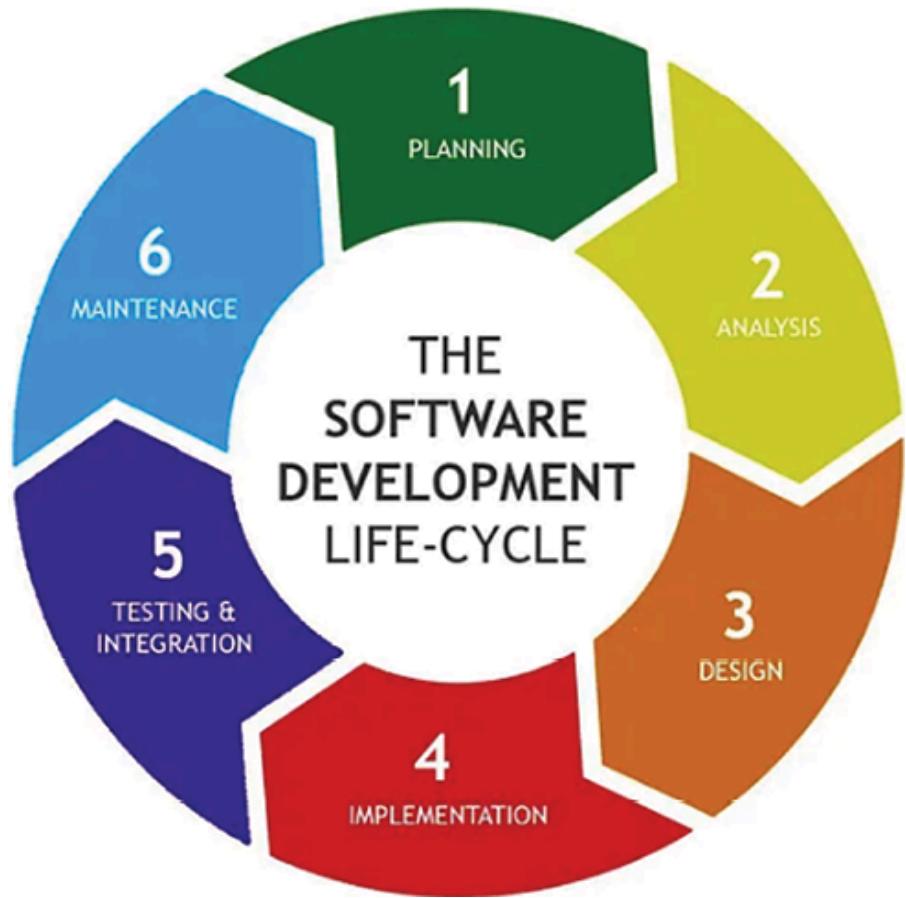


Figure 5.7: Software development life-cycle

Clean your environments

The durability, reliability, and effectiveness of a computer manufacturer depend on the quality of design, administration, and implementation of application infrastructures throughout the system development life cycle. An “ecosystem” in software engineering refers to the whole collection of physical devices (technologies, facilities, processes, and processes) that we use to design, manage, and grow the computer program. This could include, among other things confined to, a physical or digital computer running this certain version of windows, data warehouses, application frameworks (for example IDEs, processors, frameworks), information management systems, operational environments, online services, and applications for many additional reasons. Throughout my earlier years as a web developer, we often stumbled upon the relevant points:

Why do we require distinct ecosystems to develop software? Is it possible to create, evaluate, and distribute the complete product in a specific domain?

As we worked and evolved, we began to realize the necessity of having varied surroundings, and we noticed how much each atmosphere was put up whilst preserving certain aims as well as suppleness in consideration. When contemplating the intricacy of the elements that constitute the surroundings, the issue arises: how might we effectively navigate various contexts, and what techniques can we use to make this atmosphere task a breeze? The solution is to use the potential of technology and the services provided by cloud providers wherever feasible. We will discuss the Software Platform, the Testing Phase, the Intermediate Surroundings, and the Development Process. Some companies could have additional settings or call these environments by a variety of names. However, these four settings are primarily the ones that the SDLC traverses. In the event of DevOps applications, the surroundings will be part of the DevOps workflow, with the decision depending upon that integrated branch or branches tags.

Managing development environment

The **surrounding climate** refers to the specific user's personal computer, where code is created, errors are resolved, and unit tests are conducted individually by the programmer for his personality feature (in an isolated way).

Changes made locally will not affect the operation of other programmers, nor will they be influenced by modifications made by other development teams. The programmer then pushes the content from the localhost to the player's shared software platform. There, software from many other engineers will be integrated, and then it will be determined the extent to which a produced functionality in the current instalment is operating as intended in this context. Whenever maintaining a software platform, remember the following elements in mind:

Ensure that the important internal applications are installed, initialized, operational, and in the proper condition.

All prerequisites and artifacts have been appropriately implemented.

The proper directory privileges are already in place:

- Check that the recording systems have been properly configured to allow for speedy localized diagnosis of problems.
- Automated programs ought to be used wherever feasible to swiftly execute entire or partial system configuration, or to quickly take down/recreate/update an existent programming environment.
- Connection with distant computers and distributed databases is properly configured.

- Integrating tests are conducted with each code upload in the repositories to verify that the revisions do not disrupt the development.
- When compiling the whole program code contained in the repositories, make sure the setting is clean.
- Capsules should be created and used whenever feasible.
- Make as well as preserve straightforward and concise paperwork on the required configuration (for example, in the README file).
- Maintain a different file (for example, a “.env” file) with a single standard landscape software development phase and enable programmers to keep changing it as needed.
- Ensure that the software components, libraries, and dependencies used in the platform are regularly updated to address security vulnerabilities and performance enhancements. Implement a patch management process to stay current with software releases.

Managing test environment

Managing the test environment is a critical aspect of software testing:

- This is a critical ecosystem to have in the system development life cycle and it requires a great deal of care. The major goal of having any specialized atmosphere is to encounter and investigate the products via the eyes of customers, who discover and expose faults with the component. In this setting, automated tests are also done to verify that current features are functioning as planned. Once the developer team is satisfied with the function of the developer, the code is pushed to the testing process for testing by the testing. Sometimes, groups want to run tests in several settings. The following are some important considerations to make whenever administering the testing environment:
- Establish the testing environment such that parallelism tests may be executed quickly, often, and automatically. These tests should be performed at the user experience and API levels to offer feedback immediately.
- Install training dataset management to offer new / recycling / refurbished / camouflaged test data as needed.
- As needed, uninstall, compost, or renew the environments and resources (through software modules).
- Continue to monitor the management and accessibility of the experience(s).

- Incorporate communication technologies (such as Slack) to quickly alert the team of automatic test case results or just any atmosphere issues.
- Check that the endpoints, computers, supplies, networking, tools, and applications are all functioning properly.
- Ascertain that the compartments (such as Docker) and orchestration technologies (such as Elasticsearch) utilized for testing are in a ready condition.
- To construct and customize business architecture, use IaC (Roads and bridges).

Managing staging environment

The “Staging” (also known as the Pre-Prod) atmosphere is designed to be as similar to the true power generation atmosphere as possible. The primary goal of having a template is to test the software’s insertion, arrangement, but also services prior to putting it into the manufacturing process for shoppers to use. As a result, any issues discovered in the testing environment will also not influence the actual clients or customers who will use the merchandise in the manufacturing environment. While recreating the exact development team may be challenging, efforts should be made to ensure it remains as comparable as possible to identify and address issues early in the development process. Any update can also be evaluated in the production server. Before deploying improvements in and out of producers, you must first funnel out improvement concerns. The following seem to be some points to think about when helping to run the template:

- Make the web application as near to the manufacturing location as possible.
- Ensure that the environment is only available to the designated persons but should not end-users. This may be accomplished by keyword filtering IP addresses and identities.
- When necessary, update the data.
- Create automatic processes to rebuild or upgrade the ecosystem as needed.
- Check that the surveillance and reporting systems are working properly.
- Ensure that the net terminals and the database are configured in the same way as the operational but are operating with fake data.
- Whenever feasible, develop and employ container plus three types.
- Confirm the availability of the mechanism to roll back published technology swiftly and securely.

Managing production environment

The **Commercial** atmosphere (sometimes known as **Live** or even **Prod**) would be where you are actually interacting with the software application. The following were other important considerations to make while maintaining the process flow:

- Only the essential public services should really be utilized, and redundancies ought to exist.
- The infrastructure should always be operational, with little interruption.
- Procedures and remedies for emergency preparedness need to be in place.
- All spec sheets may be available in the manufacturing process, but the capability to distribute them in response to customer demand must exist.
- Updates and patches techniques that are effective should indeed be available in order to enable the upgrade producing elements with no user effect.
- Production increases should be restricted to those who really need it, and permission should indeed be set appropriately.
- Check that the recording, metering, and alerting systems are operational.
- Verify that the data center has thorough and up-to-date documents.

Building and maintaining infrastructures is a critical undertaking that requires teamwork and joint responsibility among members of the team. If the ecosystems are effectively governed and maintained, the system software will be successful:

The Waterfall Model

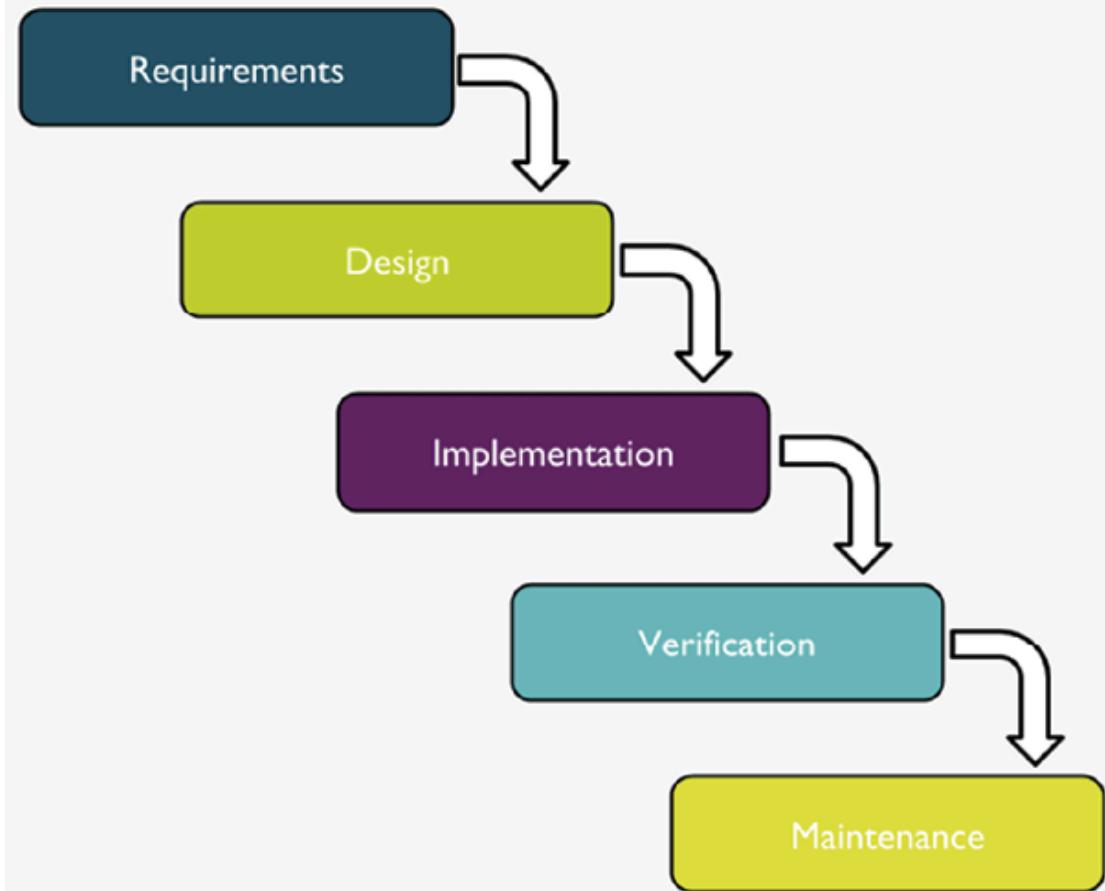


Figure 5.8: Waterfall model

Monitor and measure your pipeline

In this section, we will see how to monitor and measure your pipeline:

- You run two-phase flow pipes, whether it be on land or at water.
- With minimal or no equipment, evaluate all regions.
- Simplify and accelerate processes including duties critical to the transshipments of your goods.
- Diagnose leaks fast and consistently.
- Scrapers and groups in multi-product workflows may be tracked. Determine accumulation and monitoring problems.

- Find out something new and enhance your processes.
- PS pipelines application is your protector for the safe movement of 150+ various liquids and gasses with various compositions, such as petroleum products, coal and gas, NGL, chemical manufacturers, biocides, nitrous oxide, oxygen, chlorine, ammonia, freshwater, and many other.

Utilize cutting-edge simulation technologies to learn what really occurs everywhere and inside your pipelines. Every second, thousands of digital instruments and virtual meters are made accessible to deliver very precise data. Identify shifting situations and their core causes quickly. Immediately see all results of operator operations. Also, use the RTTM in conjunction with other apps to enhance preventive analytics and maintenance, leak detection, product monitoring, and other functions.

Protect both humans and the environment. Reduce financial and personal damages. Rapidly and precisely discover and identify network breaches and explodes. Use a single approach or combine numerous ways to create a powerful system that operates well in all operational circumstances and settings. Integrate an LDS into your safety activities to ensure that you are meeting global and national requirements.

Follow the path of various scrapers, clever pigs, and spheres from launchers to receivers. Get precise arrival timings and prevent valve accidents. Monitoring commodity batch transit in pipelines, even with extremely varied terrain and slack-line circumstances. Avoid postharvest losses or contamination by cutting your batches accurately. Determine the amount and effectiveness of DRA injection.

Continue to monitor the effectiveness of your pipes and discover deposits. Find the most expensive drinking and eating frequency and optimize your energy use. Recognize equipment drift, abnormalities, and malfunctions as soon as feasible. Maintain pipeline safety while also extending the life of your system. Get the advice you need to behave purposefully in compliance with safety and security rules.

Identify inefficiencies, enhance demand forecasting accuracy, and simplify costly operations. Gain better visibility into the functioning of your equipment and gain real-time information. Enhance proactive maintenance operations while eliminating human error. Reduce the environmental impact of activities. Build pipeline management services from a diverse set of amazing apps.

How do you quantify workflow?

Image search results for your pipelines should be monitored and measured.

Complete sales penetration is the ratio of the monetary worth of your channel to your expected revenue. A complete sales solvency ratio of three, for illustration, suggests that your overall pipeline is three times the current target. In this situation, you must execute 33% of the pipeline's volume to fulfil your sales target.

When is the quality of pipelines determined?

Frequency of Closure Date Monthly Amendments is one of the three essential network performance standards. The number of days since the most recent Stage Change. The number of days the chance has been available.

Key performance indicators for a successful piping system?

Clinch, opportunity to sell, average market value, agreement economics, lead-to-sales-qualified-lead ratio, lifetime value of customers and pipeline stage conversions are the top seven KPIs. If you can comprehend and evaluate all these KPIs, you will be well enough in your approach to building a strong sales funnel.

Method for measuring pipelines

Image outcome: Measure using a ruler or caliper at the end of the tube in which there is a visual representation. Ensure to measure from the inner edge to the border, not the exterior boundary to the outside periphery:

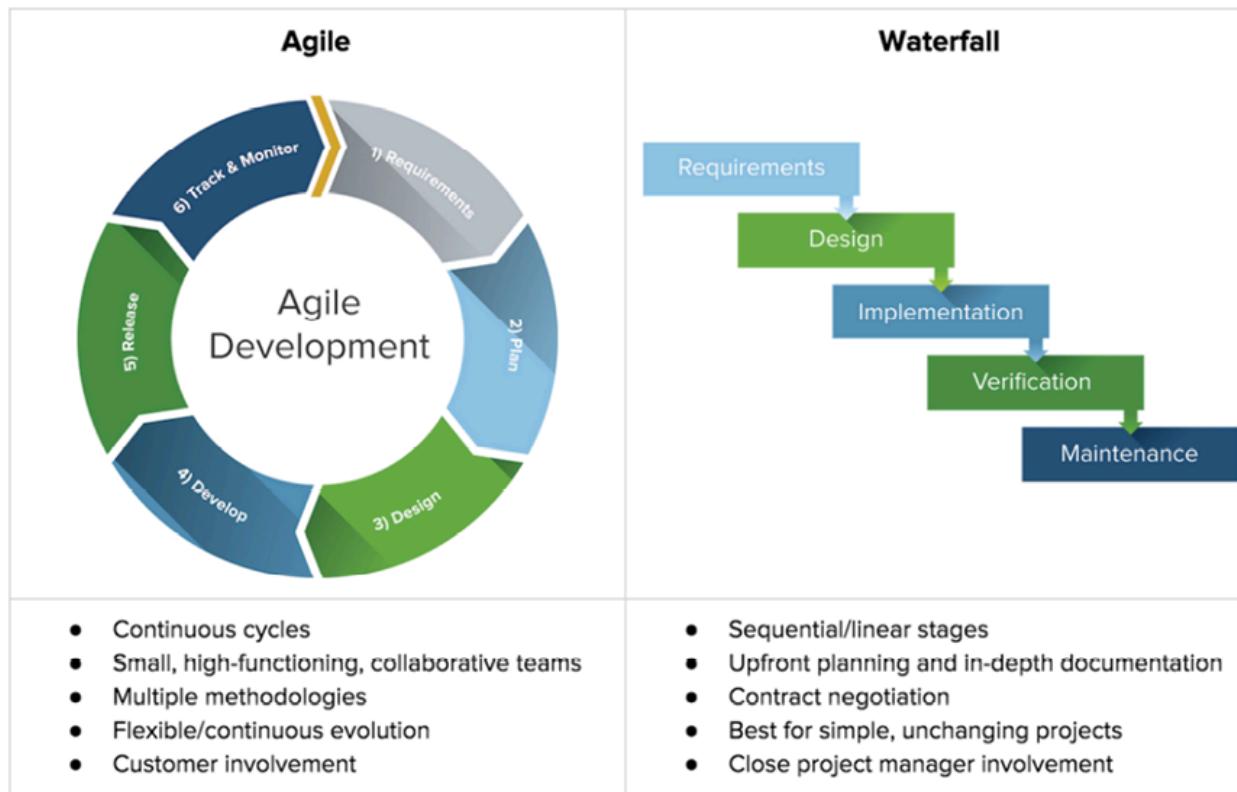


Figure 5.9: Agile vs Waterfall

Keep the builds green

Keep the build green is a practice of ensuring that automated builds of software projects are always successful and produce error-free results. This means that developers need to continually test and validate their code changes before committing them to the main codebase. By keeping the builds green, developers can detect and fix issues quickly, which helps to maintain the stability and reliability of the software:

- Keeping the builds green is a crucial aspect of **Continuous Integration and Continuous Deployment (CICD)**.
- It involves ensuring that automated builds of software projects are always successful and produce error-free results.
- By keeping the builds green, developers can detect and fix issues quickly, which helps to maintain the stability and reliability of the software.
- Writing automated tests, testing locally, using code linters, ensuring environment parity, and monitoring the build status are some ways to keep the builds green.
- Fixing failures immediately, deploying incrementally, and monitoring logs and metrics are other critical steps to keep the builds green.

Environmental software engineering?

Environmental technology development refers to a method of developing software that emphasizes electricity structures and procedures all through the distribution lifetime.

What is the life cycle of green 1st software delivery?

The following is an answer from an experienced person. The current Emerald application development paradigm, known as the Green Reference Model, emphasizes three steps of the design: creation, usage, and the conclusion of life.

Seven elements of green building

The following are the seven green building elements:

- Weather efficiency.
- Energy efficient windows
- Green roof

- Solar power
- Water conservation
- Recycling
- Landscaping

Four sustainable and green methodologies

Murugesan proposes four approaches for dealing with the environmental implications of computer science: **green usage**, **green disposal**, **green design**, and **green production** are all important:

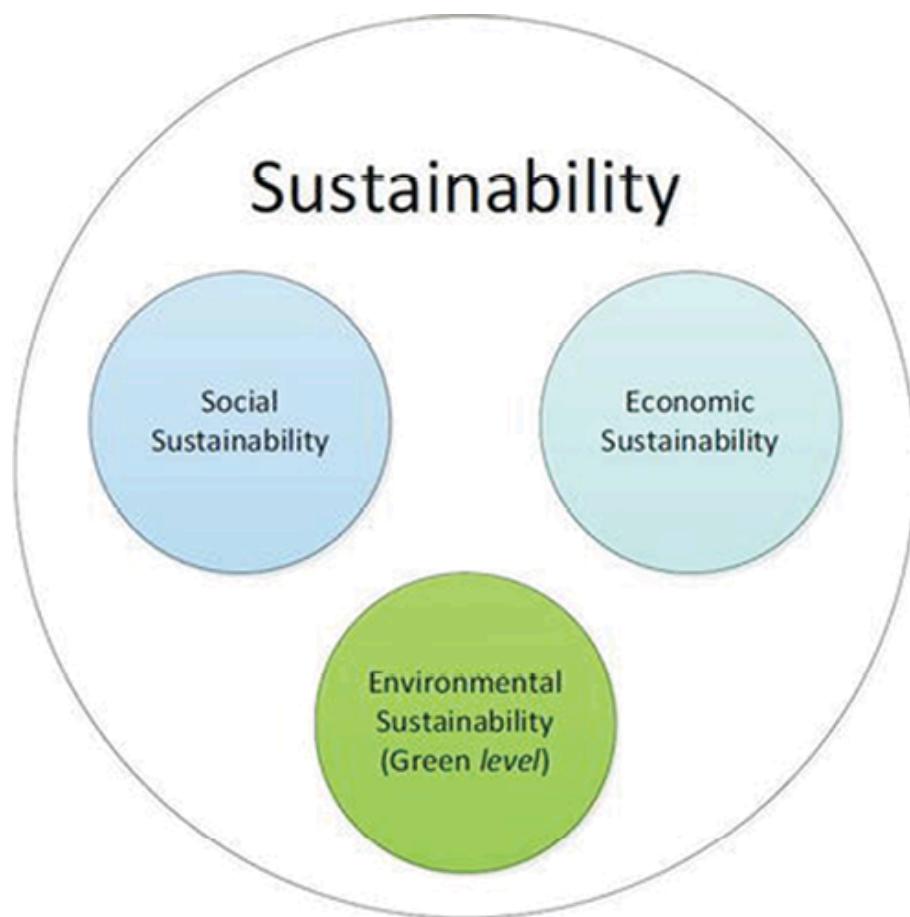


Figure 5.10: Sustainable

Green software

Green software is defined as software that emits less greenhouse emissions. Our goal is to reduce rather than to neutralize, please read the explanation.

Five green construction precepts: Built environments are planned, built, and managed to improve the very well of its residents while also ensuring a healthy

neighborhood and global ecosystems.

The five green building concepts are as follows:

- Communities that are livable
- Energy conservation
- Air quality in the home
- Conservation of natural resources
- Conservation of water

Importance of green program

Green cleaning is crucial not just for the ecology and our world, but also for the healthcare system.

Green housekeeping decreases contamination in the environment, which benefits company customers' as well as staff' health and their company's environmental impact.

Examples of environmentally friendly practices:

- Sustainable Purchasing is one of these green activities.
- Stewardship of Electronics
- Transportation.
- Pollinator conservation.
- Waste Reduction.
- Pollution avoidance.

Green IS, or Green Information Systems, promote sustainable development:

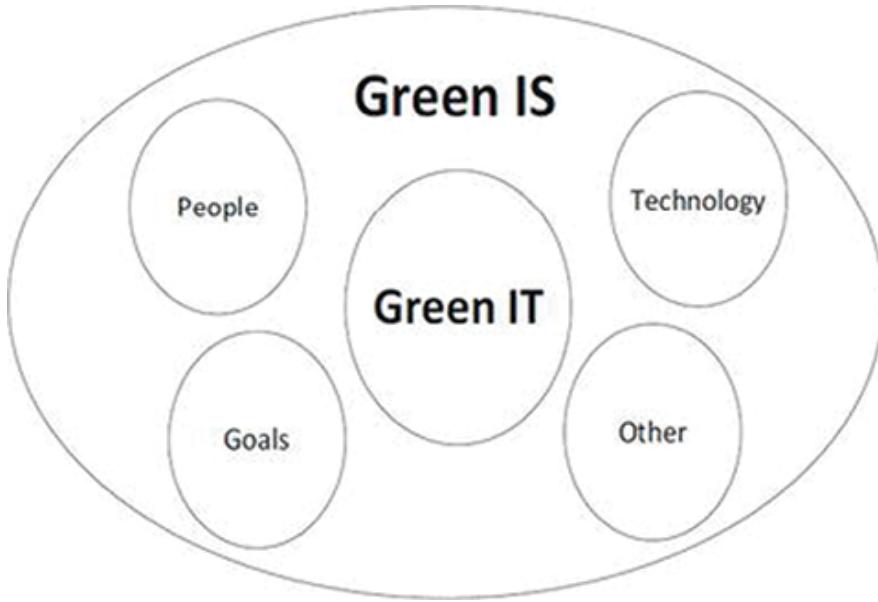


Figure 5.11: Green IS

Streamline your tests

Our modern world, where virtually everything is accessed electronically, is ruled by custom-made software development. To keep our software development processes on track, we must adhere to tight guidelines (SDLC). You will encounter sophisticated software if you are already purchasing somewhere at the mall or making vacation arrangements. Anyone can indeed complete the university documentation, which you are probably doing with your smartphone or laptop. As a result, custom app innovation has never been more important to our society or business. Methods for developing software are becoming more sophisticated. Consequently, we are struggling to innovate and develop in the face of greater global competitiveness and expanding client expectations. With this greater complexity, several businesses struggle to expedite their software architecture while remaining under budget by putting in bespoke software development processes insufficiently and exporting labour offshore. Kapsys would then address as follows:

Why do we require a unique development process?

- The capacity to map out each step of the procedure before commencing work assures that no human resource management is squandered.
- You may express your objectives to the management team if you have information describing how you want your bespoke software application

produced. So, everybody is on the same page concerning what is being created, the length this will require, and what each stage will cost.

- The game's development could serve as a reference for construction projects managing bespoke software applications or management consultants needing assistance articulating their needs. It also provides a strong platform for bespoke software development businesses to construct their products.
- The evaluation stage might be the most important aspect of any system design. That is where you find and repair issues prior to releasing the final version to clients or workers.

Methods for improving our bespoke software:

- Use a software engineering method to specify your needs; this will make the bespoke software engineering team's job simpler.
- Attend preparation sessions with industry professionals or construction managers. When programming starts, they will walk you through every step of developing your business.
- Maintain monthly progress updates with anyone and everyone involved in the development of your bespoke program. Then, you can prevent issues from occurring.
- Create a detailed estimate for each stage of the process. Also, make sure that everyone is knowledgeable of it to prevent any misunderstandings once you release your business.

Planning

The investigation and documentation of your company's needs are the focus of the project planning. Whenever you begin, make a strategy that includes the desired timeline, the benefit of spending in the project development, and the key demographic for each business. If any adjustments are necessary after finishing this step, the whole procedure will be slowed. At Kapsys, we methodically prepare to guarantee that the finished result is tailored to your interests.

Planning can help in the production of programs by lowering production costs and making sure that software companies operate within a predetermined scope.

Analysis

We may begin developing your bespoke software system after the brainstorming process has been finished. The evaluation phase is starting to break down everything you've learned from earlier activities into simple fragments such as

requirements, schematic capture, and architecture. Before commencing implementation, our Kapsys researchers will meet with customers one-on-one to determine potential necessary adjustments.

Design

Following analysis, you may design your platform's **user interface (UI)** and merge any modifications from prior processes into a comprehensive design phase for development. This is an excellent opportunity for engineers to get engaged before manufacturing starts. We may also begin creating wireframes (a set of drawings describing every display) that will eventually become UI design papers. Before proceeding, both consumers and engineers should have a thorough knowledge of the design. Design contributes to streamlining by talking to clients about their needs and outlining any adjustments that are required, hence decreasing the time waste that comes with software development.

Make it the only way to deploy to production

In this section you will get to know, how you get from development to production:

Deploy to Production: Following are the five tips for a hassle-free transition:

- Automation as often as feasible.
- Just create and package your software only.
- Always deploy in the same manner.
- Deploy Your Applications Utilizing Technology Flags.
- Deploy in limited quantities and somewhat.

When publishing a patch, a developer can mistakenly destroy a bin folder from a .NET program from one of the production machines. As a result, the site will only run occasionally, so the user must examine all the computers once more to discover the one that was down. Fortunately, there were merely four computers. That something will go amiss in an operational deployment's adjustment will go incorrect. Murphy's law applies. As a result, when it comes to missions, we have such respect for each other that borders on dread. Designers sidestep the battle because there is too much more at stake. However, releases shouldn't be the most tedious operation in the procedure.

The important part is to understand how to accomplish it. With these in mind, let us go through several strategies for easily deploying to operation while jeopardizing reliability:

- **Automate as much as possible**

Allow computers to perform the repetitive job for business; they are stronger at it than humans are. It is recommended that you rather spend some time developing a deploying script, similar to a cookbook of instructions. The software is then tested locally as well as in the development platform. In that manner, the next moment we want to make a modification, we did not have to worry about forgetting to create a directory or provide rights to specific files. We will simply have to rerun the script. Here, we are not thinking about something like a simple spade program that may be difficult to read and comprehend. New items have developed in recent years. You have technologies like Tectonic, Puppet, and Jasper at your fingertips that enable you to employ coding principles. All the business routines may be kept under configuration files, which will serve as the single truthful source. This implies that everyone must be aware of what occurred, together with the reasons and when it happened to change, not just in the code but also in infrastructural and distribution procedures. Everything above suggests that installations might be as simple as pressing a button. Reversing the distribution is as simple as clicking a button. It is possible to achieve a situation where anybody, even non-technical persons, can do distributions. Automated facilitates deploys by allowing you to progress both forward and backward using simplicity. The process becomes both reproducible and consistent. **Build and pack your application only once**

Because the building process typically takes some time, merely building once should make the situation go much more easily when it comes time to distribute. While contemplating marketing for unfinished software, you can continue writing and pushing updates. After you have finished developing your application, the subsequent phase is to package it and ensure that it stays unreachable. As a result, you are always capturing screenshots of every application. There may be moments whenever the application has mistaken or insufficient content. That is not an issue since you get to select whichever program is distributed. You will use the same bundle in all of your settings. This implies that when you produce it for research, you will upgrade the development team with the same release of the software. This is readily accomplished with proper CI. Simply substitute stand-ins with both the appropriate environment settings, such as network destinations. To make packaging simpler, practice team collaboration, of which one of its core concepts is that you construct just once. This kind of practice allows you to regularly incorporate in a central repository, validate the workflow with automation, and change problems immediately if necessary. If users adhere

to all of these elements, your migration to operation will go well since what you evaluated in developmental is precisely the same stuff you validated in those other contexts, especially commercial.

- **Deploy the same way all the time**

There is little use in following the prior advice if you do not even produce consistently in all situations. You packaged once in order to distribute this very same application throughout, which you have to provide in the same manner. You will prevent any manufacturing disappointments in this manner. You will require performance conditions because of this. This does not constantly imply that you will have the same information everywhere—this might be costly and pose a security concern. It simply implies that if you have a network interface in producing for scaling out/in, you must likewise have one in development. If users operate productions on the internet, you must also host experimentation there. In addition, you may consider ways to create more affordable surroundings. Whenever visitors upgrade an ecosystem, you only must alter the settings. This deploying advice will assist you in being daring while altering productions. You will be brave because you have rehearsed it numerous times prior to performing it for real.

- **Deploy using feature flags in your application**

If you only remember one thing from this list, let it be this one. Using feature flags can boost your deployment assurance. The value is magnified when combined with tactics such as turquoise as well as songbird deploys. These tactics will allow you to implement with your consumers to see a difference. It makes no difference whether the technique you use—all modifications must be completely compatible. It is suggested that you decide on an approach that is best for you. Blue/green migrations may be expensive, depending on your technology as well as the size of their application. They are, however, the safest choice to distribute since you are replicating the infrastructure and checking something before going live. In the context of bright yellow deploys, you are gradually introducing modifications. This can be done server by server or container by container. However, utilizing characteristic flags improves the situation. In addition, you can accomplish it so much easier that you are able to publish any modifications without enabling the functionality flag. Then, you may enable the option flag for a subset of users, observe their behavior for a time, and continue to enable the functionality for additional people once you are certain that the modification is safe to preserve. You can accomplish it all without increasing the

complexity of their software or distribution method. Cloud Bees Features Management is an invaluable tool for making change flagging simpler to apply. Then, they prepare their audiences and gradually implement these adjustments.

- **Deploy in small batches and do it often**

Incorporating the above recommendations transforms the process of delivering in small increments and maintaining consistency into second nature. It will take more time to get somewhere, particularly considering missions must be consistent and dependable, but it will be worthwhile. Unit testing is a prerequisite, and the extent of effort invested in automating your migrations matters little if you end up dedicating significant time to consistently evaluating the software. So, spend some time automating your unit tests, otherwise, making continuous deploys is pointless. But let us go back to the issue in question.

How tiny must these bunches be, and how often should they be deployed?

If you have performed the headless chicken games, you will understand what “little” and “frequent” imply. Perform this task quickly when you have anything prepared to go. It is not required to be full as provided as characteristic flags are used. Consider distributions to be the code. It is strange whenever somebody spends too much time programming sans debugging. You will not really know what is generating an error until you examine your programming as you compose it. There have been fewer places to verify if there are not numerous coding lines. Increase the number of instances you distribute to increase comments. It may seem to be perilous, but it is not. The fewer modifications you perform, the simpler it is going to be to determine what is incorrect. Do not put off deploying modifications until you have accumulated a large number of them. Make it a habit.

- **Make deployments a mundane task**

When you use one or more of the suggestions above, your installations will become monotonous, in such a situation, dull is a major gain. You will discover more entertaining things to be doing than dealing with the same issues again but over. Today’s technologies, applications, and techniques, particularly those we have highlighted above, are fantastic. They help you better manage surprises by changing those terrible evenings or extended vacations into quiet, normal working days. You should automate several of your processes, including monitoring, to make the future simpler and faster in the future. You may also package your application and make distributions more regular. Also, manufacturing settings do not deserve to be handled

differently, be sure to utilize feature indicators. They will make you less anxious while deploying since you will know you can always switch off capabilities and everything will still operate. Finally, work in tiny bunches. It is simpler to find problems, and less hazardous since you are not altering too several items at once:

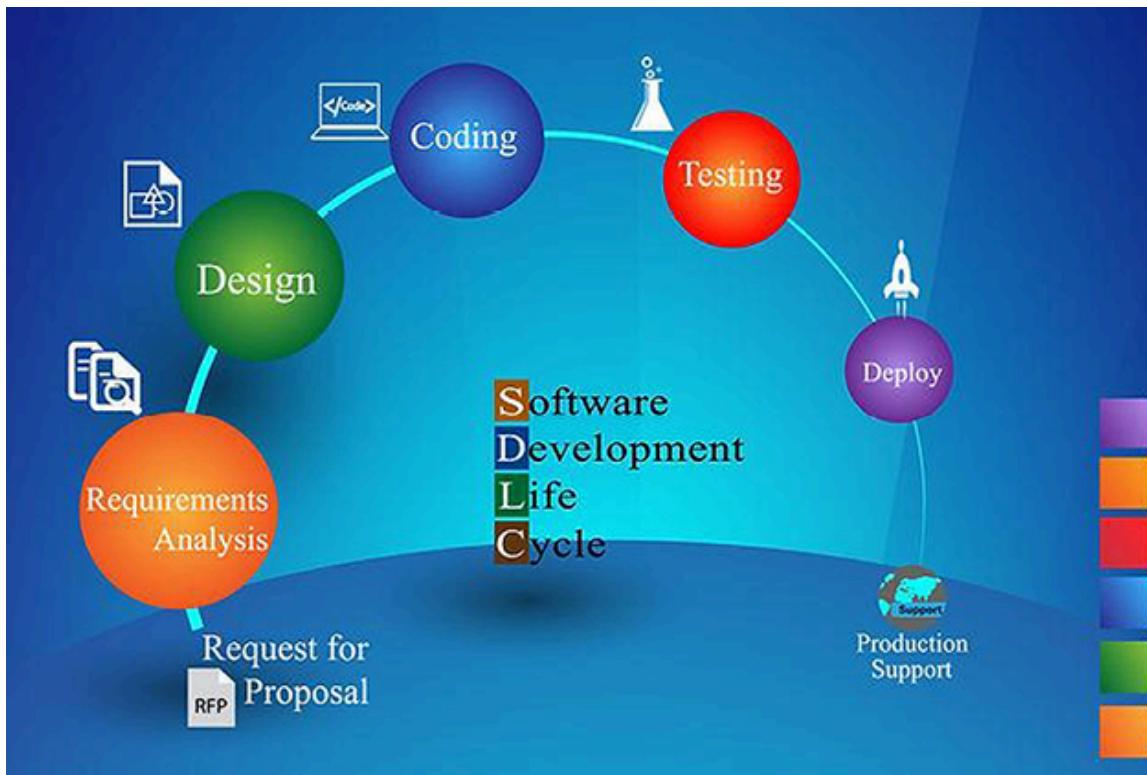


Figure 5.12: Ideal software development life cycle

Make it a team effort

A team effort is any instance of collaboration among two or more persons collaborating on the very same goal. Nevertheless, there is a more inspirational solution available. A group of individuals that work collectively and cohesively towards a single goal, fiercely helping one another to produce a pleasant working environment. That really is teamwork.

The benefits of a team effort:

Builds trust

It is critical that business staff members trust one another. A workforce that lacks trust may be challenged to execute tasks. You should encourage teamwork in your workplace to provide staff opportunities to demonstrate their skill and devotion to

each other. Nurturing this mutual trust may be a watershed moment, optimizing the process to previously unseen heights. Team members should be able to concentrate on their individual responsibilities needing to be concerned about how everyone else is contributing. Additionally, because honest conversations will be a regular practice at the business, trust may enable new concepts to arise. This is particularly true when the answer to a cooperative endeavor is not immediately evident.

Faster turnarounds

As one might imagine, numerous individuals collaborating on the same job will complete it quicker than one person carrying all of the load on their individual two hands. Functioning together educates workers that their actions have an effect on the whole team, for benefit or bad. Individuals on the team are less inclined to slack off if they understand that a loss would be shared equally by the whole squad.

Allow people to fight who need something bigger than themselves, and you will become astonished at how far sympathy can push our civilization.

Creativity

The greatest method to cultivate a creative vibe is for many individuals to work in a single group. Those few who operate alone are constrained by their fixed views and whatever inaccurate statements they may have. Organizations, on the other hand, may get around this by mixing the many viewpoints and ideas that each new employee brings to the discussion. As a consequence, answers that are devoid of prejudice and significantly more successful at the job at hand are produced. The opportunities for cooperation are boundless, whether you are a software engineer, author, or web developer, and thus, the ultimate product will be extremely worthwhile. At Chanty, we allow any individual to participate in our material as long as they want.

Collective learning

In the very same manner that teamwork allows your workers to complement their particular skills to improve the performance of the team, it also enables them to learn as a community and collaboratively expand their knowledge. Somebody operating alone will pick up new information as they advance in their careers, yet this knowledge may remain with them permanently. Groups that cooperate, on the other hand, will learn those experiences collectively and also be able to progress at the same time. This reduces the training time and serves to disseminate information throughout the whole workforce. The talents that partners share will indeed aid individuals in their solo ventures for the remainder of their lives.

Finally, teamwork fosters a desire to learn that is uncommon in isolated employment. This is attributable partly to the fact that communicating findings with collaborators has always been interesting, and the satisfying sensation of actually contributing to the accomplishment of a task pushes you to continue learning. Collaborating on a subject with others fosters a desire to learn that isolated labor does not. Individuals like being able to discuss their findings with the remainder of their organization since it builds combined personal and group expertise.

Examples of team effort:

Collaborative content

Team effort in collaborative content creation is exemplified by the collective contribution of individuals to produce content that reflects a diverse range of perspectives, expertise, and skills. Here are some real-time examples that showcase the power of collaborative content:

- Wikipedia: Wikipedia is a prime example of collaborative content creation. Thousands of contributors worldwide collaborate to create, edit, and refine articles on a vast array of topics. The collective knowledge and expertise of these contributors result in a comprehensive and ever-evolving knowledge base.
- Open-source software development: Projects like Linux, Apache, and Mozilla Firefox are developed collaboratively by a global community of developers. They work together to write, review, and improve code, ensuring the quality and security of the software.
- Google Docs: Online collaborative document platforms like Google Docs enable multiple users to work on a document simultaneously. This real-time collaboration allows teams to contribute, edit, and comment on content, enhancing efficiency in tasks such as document writing and editing.
- Social media campaigns: Marketing teams often engage in collaborative content creation for social media campaigns. Various team members contribute ideas, design elements, and copywriting to create compelling and cohesive content that resonates with the target audience.
- Crowdsourced blogs: Some blogs leverage the power of crowdsourcing for content creation. They invite contributions from diverse writers and experts, resulting in a rich variety of perspectives and insights on a particular topic.
- Academic research papers: Research collaborations among scientists and academics illustrate collaborative content creation. Multiple researchers

contribute their findings, methodologies, and analyses to produce comprehensive research papers.

In these examples, the strength of collaborative content lies in the diversity of perspectives, expertise, and skills that individuals bring to the table. Whether in the form of an encyclopedia, software, marketing campaigns, or research papers, collaborative content exemplifies the collective intelligence and creativity of teams working together.

Brainstorming

Group discussions are a great illustration of how much can be done during discussions. By concentrating the passion of any and every squad member towards that single objective, an overwhelming movement may be created. The key point to remember during discussing would be that all ideas and thoughts are legitimate. The adage “no idea is a bad idea” applies there. It is also critical that every bureaucracy exists during such a brainstorming process. There are no managers, workers, or internships around, just individuals with minds and thoughts. This not only makes use of the aggregate expertise of the whole team to solve issues or design the best course of action, but it also promotes involvement, which may develop into a trend of employee retention. In addition, if you’ve recently formed a new workforce, thinking is one of the greatest exercises to start with. This may help to break the ice amongst individuals who also do not understand one another well and get the energy going right away.

Constructive criticism

Everyone seems to have unique skills and experiences. There is absolutely no shame in looking for assistance and acknowledging that a subject is outside your area of competence. For example, if a graphic artist is producing a program’s visual identity, he or she should consult with the designers to make sure that it aesthetically complements the functionality. The opposite is also true. If programmers are aiming to build visual design technology, they might ask the design department which UI components they like and which ones they don’t. This positive feedback cycle ensures that knowledge is disseminated throughout divisions.

Encouraging team effort

Sending workers an unforgettable experience might be a brilliant team activity to start with if you are attempting to prod them into regular cooperation. As a result, they are forced to rely on excellent collaboration to discover a resolution. Furthermore, strive to maximize their production instead of their intake. It is important to realize that organizations should not be concerned with how long

individuals work, but instead just with the amount they accomplish at the moment they do. According to scholar Cal Newport's book Deep Work, the scientific evidence regarding the human ability to undertake Deep Hard graft is roughly four times each day. This implies that making your employees be hype productive early in the day is smart practice since it may substitute again for moderate speed that they are going to fall into later in the day, which time is ideal for shallow research. Superficial employment might include tiny chores like answering emails or using a project management tool – actions that make us feel active but then do not produce anything substantial. Serious work, on the other hand, might include tasks such as content creation or study. These are typically the chores that we put off the longest because they typically seem so much more difficult. Healthy rivalry is one strategy for getting workers to be efficient, including doing serious work in the morning. Organize the workplace into numerous teams with specific objectives each afternoon. The team that completes the most number of targets before lunchtime breaks receive free refreshments (preferably nutritious foods to avoid a carbohydrate collapse in the afternoons). You may offer the team more prizes than free snacks for bigger team successes and reaching monthly targets. For instance, you may make some amusing corporate gear and offer it to the staff by utilizing a no-cost merchandise production site. If there is a tie, the squad that completed it first triumphs. Then again, recent research by New York University, psychiatrist *Gavin Kilduff* discovered that lengthy athletes can able to boost respective race timings while racing with respective competitors. This is merely one illustration from his research on how competitiveness improves human productivity.

Best tools to build robust CI process

Implementing a CI method is one of the most effective things, business organizations can do to increase productivity. Furthermore, because its performance is primarily reliant on CI technologies, you must deliberately choose the ones that most effectively meet your goals. We examined seven of the most famous CI solutions and highlighted various benefits to assist you in deciding between comparable choices. When we go into the best CI technologies, let us establish exactly what CI entails. In computer programming, **Continuous Integration (CI)** is a method of building automation and source verification that occurs whenever the developer group makes improvements to source code. Upon fulfilment of each project job, developers discuss and incorporate their contributions (software as well as unit testing) into a unified CI source even during simultaneous economic integration. If you have ever wondered why system testing is so crucial, consider the following:

- CI enables the team to identify and rectify integration problems early in the development process. This prevents the accumulation of complex and time-consuming integration challenges as the project progresses.
- CI fosters a collaborative environment by providing a shared platform for developers to continuously integrate their code. This promotes transparency, as everyone on the team has visibility into the progress of code integration
- CI aids in the avoidance of merging disputes, difficult-to-resolve issues, duplicating software, and inconsistent coding styles.
- CI reduces software testing time and produces enterprise code that is relatively homogeneous.
- CI accelerates project development and brings deployments closer together.
- Consistent communication is ensured via CI.
- CI aids in the reduction of project congestion.

While very advantageous, system testing requires a significant amount of effort. Fortunately, there seem to be resources available to assist you in succeeding.

Great CI tools offer a range of benefits.

You may now pick from a broad choice of **continuous delivery (CD)** technologies. In fact, the quantity is so big that the management is sometimes perplexed as to how to choose the finest CI technologies for their projects. If the tool you have chosen lacks the essential technical details, keep searching.

What benefits do outstanding CI solutions provide:

- A strong environment: A CI solution tries to accelerate project deployment and eliminate superfluous work from the development process. When implementing the technology, ensure that it will not cause problems in our application.
- Interoperability with the cloud: A decent CI software can enable trouble-free information transmission both to and from the cloud.
- Options for distribution: A CI tool would allow for easy installation.
- Strategies for integrating: Your CI system should be able to communicate with those other tools and resources shown on the projects.
- Ensure health and safety: A good CI tool, regardless of whether the software is open, really should not offer significant privacy issues for the information in your projects.

A CI tool must be technically proficient in addition to satisfy the demands of your project. Depending on your company plan, you may want a completely public or premium CI system. Furthermore, you should ensure that the solution you choose enables simple project administration and transfer. It is also advisable to use a device that can visualize the material. Now, let us take a look at the best seven CD technologies that might make your project's process simpler and more efficient.

CI tools

Here are the top seven recommendations for the finest CI solutions for your proposal:

- Jenkins
- TeamCity
- Bamboo
- Buddy
- Circles CI
- Travis
- GitLab

Let us take a deeper look at each one:

Jenkins: Jenkins has become one of the most extensively utilized completely public CI tools in computer programming. It is a host-machine CI system developed in Java that operates on a web server. Thousands of people all around the globe like collaborating with Jenkins because it helps them swiftly automate builds and testing.

Following are the advantages of Jenkins:

- Local implementation
- Absolutely free
- Extensive procedure modification
- Features and plugins abound.
- The pre-installed OS X, Unix, and Microsoft components make deployment a breeze.
- Developed by programmers for builders
- A well-known brand with an outstanding reputation

TeamCity: TeamCity is a robust corporate CI system that is complimentary for the first 100 build combinations. You can perform alternative building multiple times using TeamCity, label your builds, and discover the stuck ones. Major urban centers are simple to set up and have an easy-to-use interface. You will also like its environment and expert assistance.

Following are the advantages of TeamCity:

- Allow up to 100 construction combinations to be free.
- Simultaneous performing three deployments with three building agencies
- Capable of importing code base from two separate VCSs into a common build
- Capability to replace inspectors using chatbots
- Permits modifications to be tested without submitting anything to the VCS.

Bamboo: Bamboo is a Confluence-based continuous integration and deployment solution, offering a user-friendly and straightforward interface for an intuitive experience. One such tool is popular among programmers who are familiar with Confluence products. Bamboo enables you to create branch offices immediately and merge those after evaluation. CI and deployment are simple to do using this technology.

Following are the advantages of Bamboo:

- Interoperability with some other Confluence technologies is simple.
- Frictional pressure capability in a subscriber interface
- Excellent warning process
- Simple management for business CI scalability
- Test automation is possible with the use of flexible actuators.
- Each pipeline is fully automated and handles building artefacts.

Buddy: Buddy is a solution for operational robotics that enables concurrent engineering, agile methodologies, and monitoring. It was created to be used in organizations that rely on code again from Bitbucket and the GitHub repository. Buddy is a marketable company featuring an easy-to-use interface and a simple material appearance. This consumer service offers live representative help 24 hours a day, seven days a week, and its business edition enables identity on a server.

Following are the advantages of Buddy:

- Intuitive distribution cycle including Docker number of alternative presets and tips
- Provides powerful automating while needing just basic knowledge
- Capability to make changes to the produced code
- Automate, clones, parameters, connectivity, and alerts are all customizable.

CircleCI: CircleCI is a framework for agile development and distribution. It supports a variety of programming languages and may be deployed locally or on the internet. This tool simplifies automation tools, development, and distribution. Its simple user experience is jam-packed with customizable features. Designers may use CircleCI to rapidly decrease the number of problems and enhance app performance. Despite being a commercial product, CircleCI provides a free subscription for expansive applications.

Following are the advantages of CircleCI:

- Container compatibility
- Scalability and deep customization
- Options for extensive integration
- Portal for sophisticated administration
- .net framework procedure that is dependable
- enables the creation of complicated procedures
- Permits many versions to execute at the same time.

TravisCI: Tracinski is a time-tested CI system that is ideal for transparent projects. This CI program offers several possibilities for CI management. There is no requirement for a computer since it is an internet service. Tracinski also has an enterprise-focused on-premises variant. One of the nicest features of such a program, is that it saves the most recent version each time you launch a different version.

Following are the advantages of TravisCI:

- Compatibility for several languages and operating systems
- Statistical and robotic installation administration
- Enterprise-level access management
- Graceful Synchronize with GitHub
- Screening in tandem

- Capacity expansion according to demand
- Branches construction process and push proposals are supported.

GitLab: GitLab is a CI and delivery platform that is integrated with GitLab. Here are some key points about GitLab CI/CD:

- YAML-based configuration: GitLab CI/CD uses a simple YAML-based configuration file to define jobs, stages, and workflows.
- CI: GitLab CI/CD can automatically build and test code changes whenever a new commit is pushed to the GitLab repository.
- CD: GitLab CI/CD can automatically deploy code changes to a test environment whenever they pass the testing phase.
- Continuous deployment: GitLab CI/CD can automatically deploy code changes to a production environment whenever they pass the testing and review phases.
- Multi-platform support: GitLab CI/CD supports a variety of platforms and environments, including Linux, macOS, and Windows.
- Parallel testing and builds: GitLab CI/CD can run multiple jobs in parallel, allowing teams to speed up their build and testing processes.
- Docker integration: GitLab CI/CD integrates with Docker, allowing teams to easily build, test, and deploy Docker images.

Overall, GitLab CI/CD is a powerful platform that helps software development teams automate their build, test, and deployment processes and deliver high-quality software faster.

Is Jenkins the best CI tool?

Jenkins. Jasper is an open - source software, merge, Java-based program that enables users to simulate and publish in real-time. It is regarded as one of the top CICD products on the market today. Jenkins is simple to use and integrates seamlessly with major cloud applications like Microsoft Azure, Google Cloud, AWS, and Digital Ocean.

Which is preferable, Jenkins or Selenium?

Jenkins has a higher approval rating on Stack Share, with mentions in 1753 corporate heaps and 1479 programmer stacks, compared to Firefox, which itself is referenced in 770 company stacks and 425 programmer heaps. On many initiatives, system testing is a required component of the design process. Some companies, nonetheless, are still seeking a CI technology to include in their plan.

When selecting a configuration management technology, choose one that best fits your projects and business goals while also accelerating and automating the creation and execution operations. Furthermore, regardless of the CI solution you select, you can leverage Bit bar's inherent compatibility with CD technologies and its robust REST API for seamless integration.

Following figure describes continuous integration workflow:

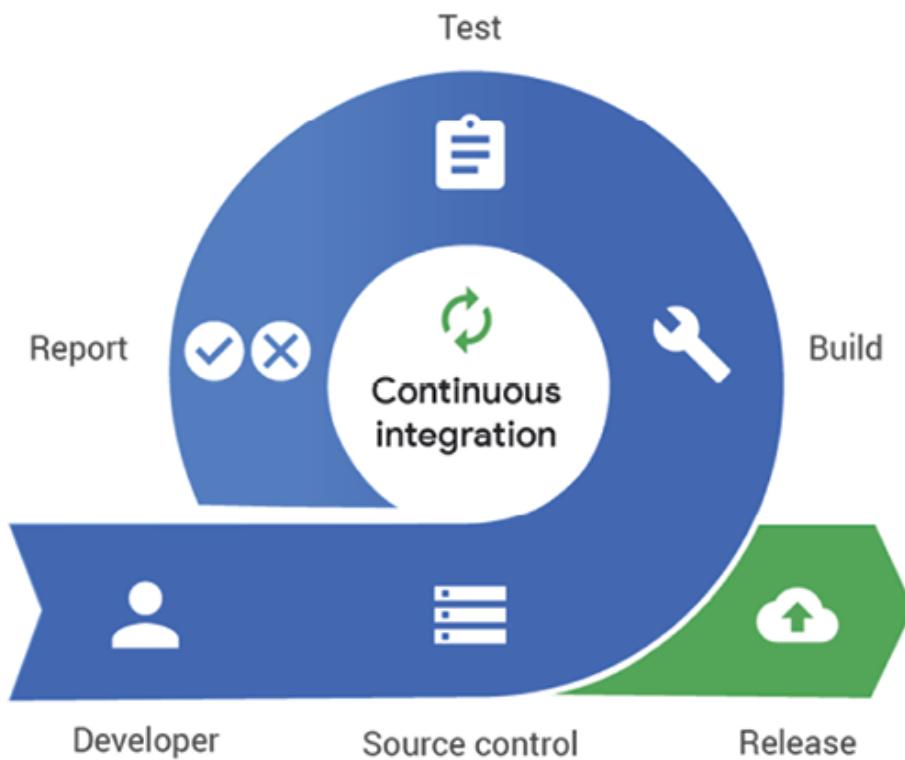


Figure 5.13: Continuous integration

Step-by-step building robust and fully automatic CI process

CI is a method in which code generated by the programmer is submitted to a common central backend system such as Git, and each contribution is constructed and tested to ensure that any bugs are identified and fixed as soon as possible. CI provides several advantages, including early integration testing, increased development production, quicker delivery, and quicker problem detection as well as resolution.

Continuous integration or installation is the process of continually and continuously preparing code updates for publication in the test environment

Following all the development, certification, and certification steps, will result in a military career artifact. To push to operation, this needs a human step/approval. CI continually pushes software to operation. Building an effective CI/CD workflow in terms of procedure and practices is critical for supporting product announcements and driving organizational performance.

CI/CD: CI/CD is a method of producing software that allows you to distribute changes at any moment in a sustainable fashion. When code changes are regular, turnaround times become more frequent, important, as well as quicker. “CI/CD” refers to the processes of CI and CD. CI is a precondition for CI/CD and necessitates the following:

- Developers must commit their modifications to the initiative stream multiple instances each day.
- Every code merging is used to initiate continuous code development and testing procedures. Developers like to obtain findings in less than ten minutes so they may concentrate on their tasks.

CI’s duty is to create an artefact that can be deployed. The purpose of test automation in CI is to ensure that the artefact for the specified iteration of code remains safe to publish.

Software updates are also continually delivered under the practice of Continuous deployment, however, the deployment is prompted remotely. Continuous distribution refers to the process of completely automating the process of transferring code first from the development location to the operation.

A litmus test for doing CI/CD

If any member in your organization can pause what they are in for right now and deploy the existing development version of code to operational in 20 seconds or less without anyone worrying about what could happen – congrats, you are performing CI/CD!

CI/CD principles

CD techniques extend CI by stating the following criteria for efficient manufacturing rollouts:

- Build the system in a manner that allows for iterative releases. Avoid excessive component coupling. Implement metrics that aid in the detection of problems in real time.
- Use test-driven development to ensure that your code is always deployable. Maintain a robust and well-maintained automated test suite. By design,

include monitoring, logging, and fault-tolerance.

- Work in modest iterations. For example, if you work in feature branches, they should have a maximum lifespan of one day. Use feature flags when you need more time to develop new features.
- The code may be pushed into production-like staging environments by developers. This assures that when the updated version of the program is sent to consumers, it will function properly.
- At the click of a button, anybody may deploy any version of the program to any environment on demand. It is game over if you need to look up how to deploy on a wiki.
- You run it if you construct it. Autonomous engineering teams should be accountable for the software's quality and stability. As they collaborate to accomplish high-level objectives, this breaks down divisions between conventional developers and operations departments.

To make CI/CD an actuality, you must automate as much of the continuous deployment method as possible and run it via a CI/CD pipeline:

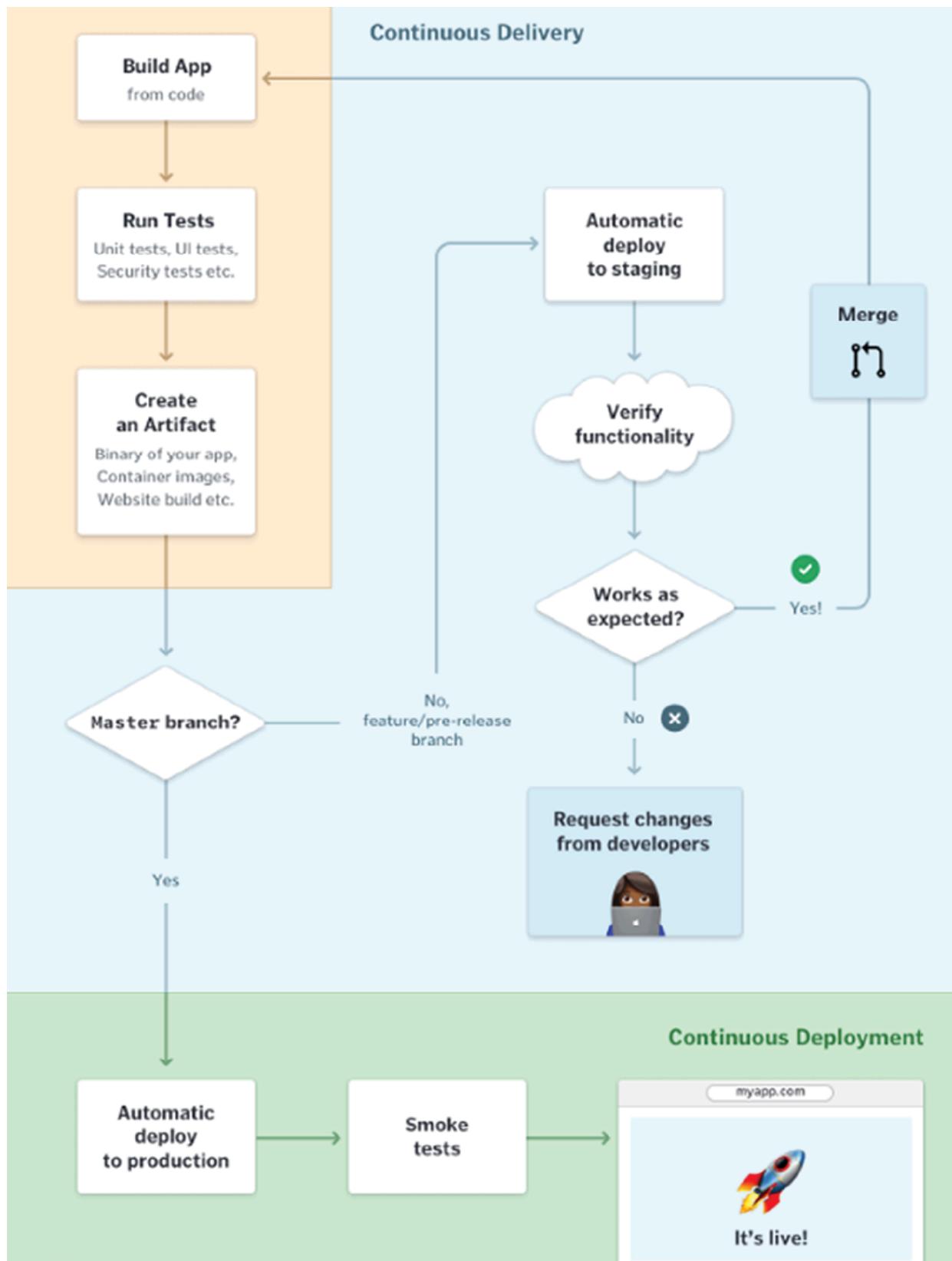


Figure 5.14: CICD pipeline

Example CI/CD workflows

Here is a straightforward example of a completely automated CI/CD (**Constant Deployment**) piping system:

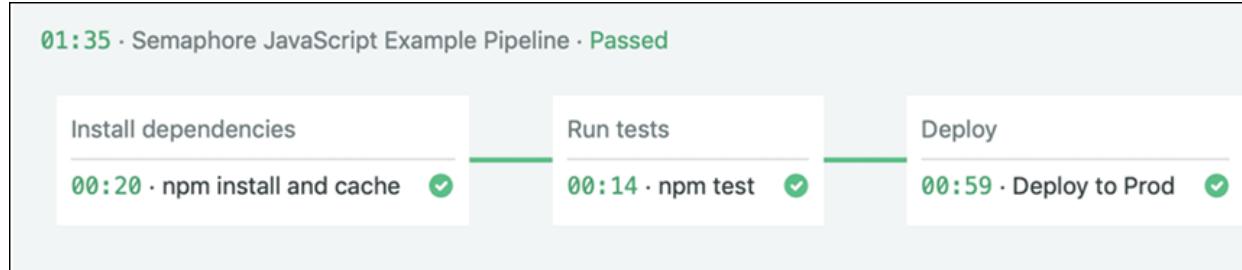


Figure 5.15: CICD workflow

Every modification to the primary Git repository causes Hand signal to undertake the following things:

- Build software and web resources while making use of the dependence caching.
- Carry out an automation test suite. Because this is a JavaScript / Node.js project, the tests are written in Jest.
- If the benchmarks pass, a Deploy step upgrades the cloud-based procedural code.

CI/CD with manual steps

Here is a more detailed CI/CD methodology for virtualization:



Figure 5.16: CICD methodology

Every update in this instance immediately conducts the following actions:

- Develop an application from the programming language and requirements.
- Carry out a test automation suite. If the tests pass, create a Docker image and submit it to a public repository. We have such a functional artifact, a

container's images, at the conclusion of something like the Kubernetes build process.

- The programmer, or the distribution management, has the option of directly triggering:
- Distribution to intermediate or operation, which might also involve smoking tests to prove no serious issues have arisen.
- Marking the folder as an item brought into service, allowing inspections and clawbacks.

Benefits of CI/CD

CI/CD is so much more than automating operations to eliminate errors. It enables us to provide innovative solutions to consumers as fast, effectively, and affordably as feasible. As Dave Farley, founder of the book Continuous deployment and a computer science specialist, notes in a Synchronization talk show:

When is CI/CD not feasible?

The concept of continuous improvement is fantastic, however it will not work without my application.

There are certain instances when CD may not even be appropriate: Your clients do not desire constant system upgrades.

Practices and related methods through which software may be upgraded. Continually performance of the computer used during the aeronautical, telecommunications, and pharmaceutical sectors, for instance, would not be an option. Even in a CD-averse setting, organizations may benefit from the simplicity of installation and the ability to keep the technology accessible. They, too, may practice and just get the full advantages of CI. Download Semaphore's introduction on CD for a more comprehensive look at CI and best practices.

Journey to CI/CD

If you are accustomed to extensive development cycles, you will need to shift your perspective. However, if you are contemplating using CI/CD, you have taken a crucial first step! Consider any impediment performing CI/CD to be technology indebtedness, and handle it as such. Estimate the amount of work that remains to be improved. Establish reasonable goals with all participants and set aside time to concentrate on them.

There seem to be two important turning points on your path to CI/CD achievement. The first step is to build CI: CI that is quick, accurate, and frequent. The following

step is to implement CD, which means automating distribution, making it a one-button process, and making it simple to test new code in manufacturing conditions.

Conclusion

CI is a critical practice in software development that has gained significant popularity in recent years. It is a process that involves integrating code changes from multiple developers into a shared repository several times a day. The code is automatically built and tested to ensure that it integrates smoothly with the existing codebase. By using CI, developers can identify and fix issues early in the development process, before they become more complicated and costly to resolve.

The importance of CI lies in its ability to catch issues before they can affect other parts of the codebase. With frequent and automated builds, teams can detect conflicts, errors, and bugs in real-time, enabling developers to make necessary changes and prevent delays in the development process. This reduces the risk of “integration hell,” a term used to describe the difficulties that arise when teams attempt to merge code changes from different developers into a shared codebase.

CI also promotes collaboration and transparency among team members. With CI, developers can work in parallel, making changes to the codebase without worrying about conflicts. Each change is tested, and the entire team is notified of any issues that arise, allowing them to work together to resolve them quickly.

Moreover, CI enables teams to deliver high-quality products more rapidly and with greater confidence. By detecting issues early, developers can make necessary changes and deliver working code more quickly. This reduces the risk of project delays and ensures that teams can deliver products that meet customer needs and expectations.

In conclusion, CI is an essential practice for modern software development. It promotes collaboration, transparency, and early detection of issues, enabling teams to deliver high-quality products more efficiently. As the software industry continues to evolve, CI will remain a crucial practice for teams to adopt and implement.

In the next chapter we are going to cover **continuous deployment and delivery (CD)**.

CD is a software development practice that focuses on ensuring that software applications can be reliably and efficiently released at any time. It involves automating the entire software delivery process, including building, testing, and deploying the software.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 6

Software Packaging and Continuous Delivery

Introduction

The software development business has been undergoing a gradual but significant shift recently. Everything these days has some kind of software component; therefore, programmers are scrambling to find ways to meet the soaring demand by expanding the use of automated systems. With such a high demand for additional technology distribution and application availability, the pipeline approach of **continuous integration (CI)** and **continuous delivery (CD)** has matured significantly. To ensure that apps are delivered rapidly and reliably, **DevOps** practices, as well as **Agile** tenets have been created. As a result of shifting to a pipeline mindset, proper planning has skyrocketed. Additional features are added to the system with each iteration deployment, as represented by agile techniques. Delivery might be impacted by either well-developed functionality or defects and failures in such techniques. This chapter presents a pipeline strategy for solving quality problems, which speeds up the deployment process, simplifies testing, and enhances comparisons. As a result of the streamlined procedure, system downtime is reduced, and reliability and on-time delivery are improved. It may reduce uncertainty as well as guesswork, assure quality, and increase efficiency by providing standardization. This essentially implies using a proven, tried-and-true procedure. This application may be easily integrated with other systems since it is written in Scratch, a

natively interpreted language. We show the value that our solution presently generates based on data from experiments. Automation pipelines are the backbone of this system, making it easy to create, monitor, configure, as well as execute CI as well as CD solutions based on Agile methodologies. The proposed solution serves as a foundation for conventional CI/CD procedures, caches Docker containers for future use, and employs a Kubernetes cluster with Helm to provide highly accessible outputs. In a later conversation, we will talk about modifying the system-provided underlying ideas as well as extending them to work on other systems (windows). The following figure shows the software release cycle:

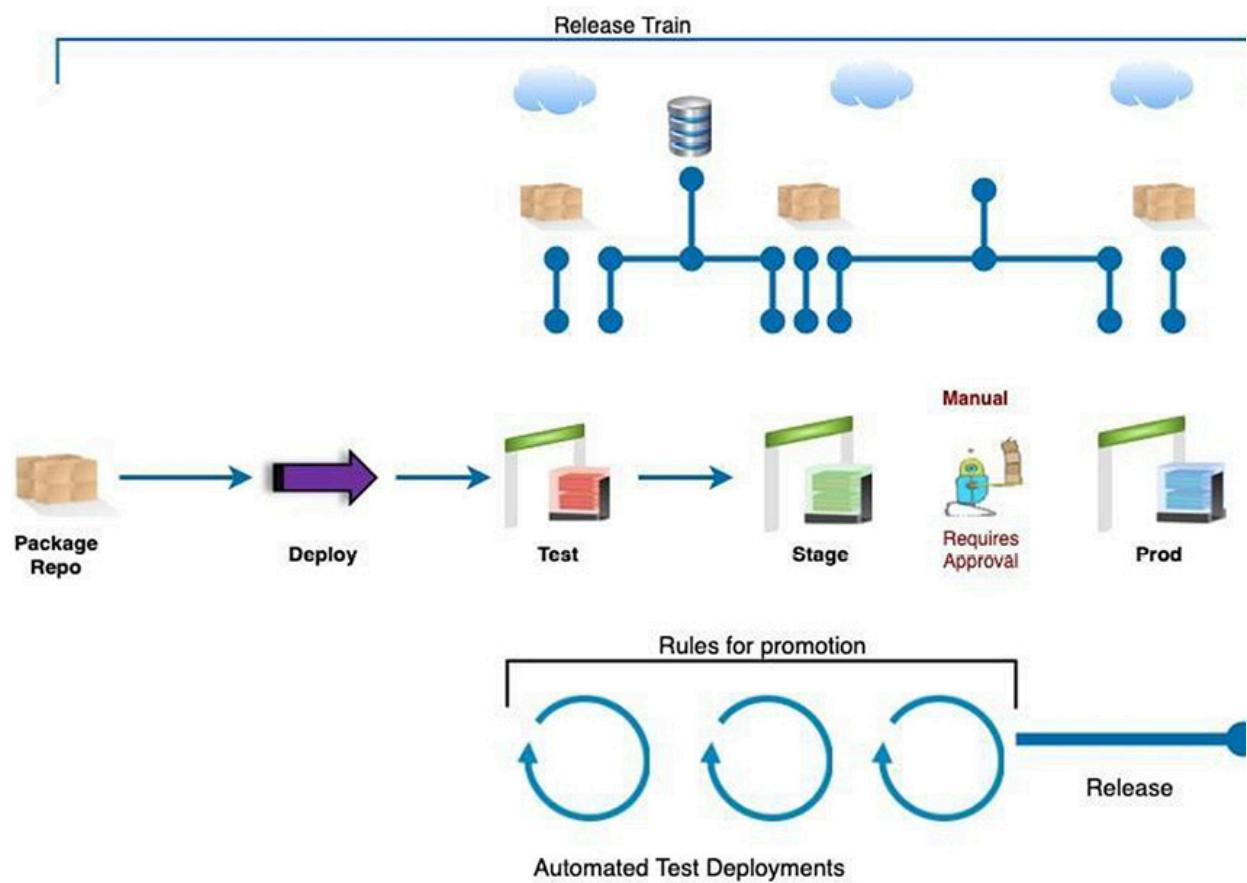


Figure 6.1: Software release cycle

Structure

In this chapter, we will go through the following topics:

- Exploring the benefits of continuous delivery

- Increasing developer productivity
- Simplifying implementation through automation
- Accelerating feedback delivery
- Enhancing testing quality
- Expediting market introduction of new capabilities
- Selecting the tools for a robust continuous deployment process.
- Building a robust and fully automatic CD process
- Measuring considerations for calibrating a CD pipeline
 - Analyzing the lead time in a CD pipeline
 - Evaluating the cycle time in a CD pipeline
 - Assessing the mean time to recovery
 - Examining defect resolution time
 - Understanding test pass rate
- Best practices for adopting continuous delivery
 - Developing Service Level Objectives (SLOs)
 - Automating SLO evaluation with quality gates
 - Automating every repeatable process
 - Keeping everything in version control
 - Providing fast, useful feedback
 - Deploying the same way to every environment
 - Avoiding direct changes in the production environment
 - Deploying to every environment the same way
 - Deploying a copy of the production
 - Including the database
 - Eliminating complexity
 - Establishing observability and continuous monitoring

Exploring the benefits of CD

By using technology, *continuous delivery* expedites the rollout of updated software. It allows developers to share their application updates with others in a central location, such as a development repository or vessel registration, by automating the process:

- Reduce risk
- Faster time-to-market
- Higher quality
- Improved collaboration
- Efficient feedback loop
- Reduced manual intervention
- Scalability
- Continuous improvement
- Supports Agile practices
- Cost savings
- Flexibility
- Regulatory compliance

CD is a software development practice that focuses on automating and streamlining the process of delivering software changes to production or end-users in a frequent and consistent manner. The goal of CD is to enable faster, more reliable, and efficient software releases while minimizing risks and increasing collaboration among development, testing, and operations teams:

- Embrace a mindset of lifelong education.
- Hone your ability to adapt quickly.
- Effective, well-established methods of agile development.
- Embrace the concept of infrastructural automation.
- Increase the rhythm of your speech.

Agile process, system integration, as well as experiment programming are indeed the three mainstays of the larger concept of rapid deployment. With CD, numerous programmers working in different places may be guaranteed their material is constantly added to a centralized source.

One notable advantage of swift deployment is the production-ready status it achieves. To ensure its proper functioning in a real production environment, the software undergoes development, verification, and release into environments that closely emulate the actual setting. During this period, it remains in a staging area.

Constant Delivery's primary objective is to simplify and speed up software deployments so that they may occur on demand with little downtime as well as danger. Negligible, user-unobtrusive deployments are simple to do using methods such as turquoise deploys.

CI and CD workflow:

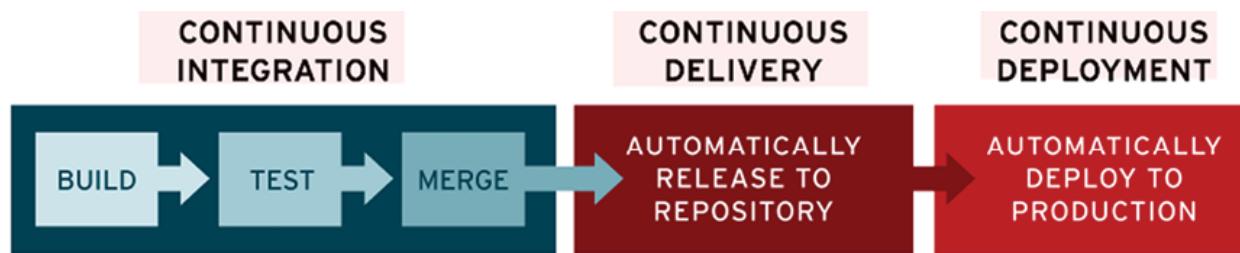


Figure 6.2: CI and CD workflow

Here are some of the benefits of CICD:

- Quicker Mean Time to Resolution (MTTR).
- Smaller backlog, therefore, faster release rate.
- Satisfied customers.
- More open and accountable teams.
- Streamlining code changes and pinpointing issues lead to straightforward maintenance and updates, offering various advantages
- Releasing new versions more often.

The term *continuous deployment pipelines* refers to a set of procedures that are used to reliably roll out updates to current applications. It is a way to put

the CD model into practice, in which all stages of the release process—including automatic building, testing, and peacekeeping missions are managed centrally.

Benefits of CICD:

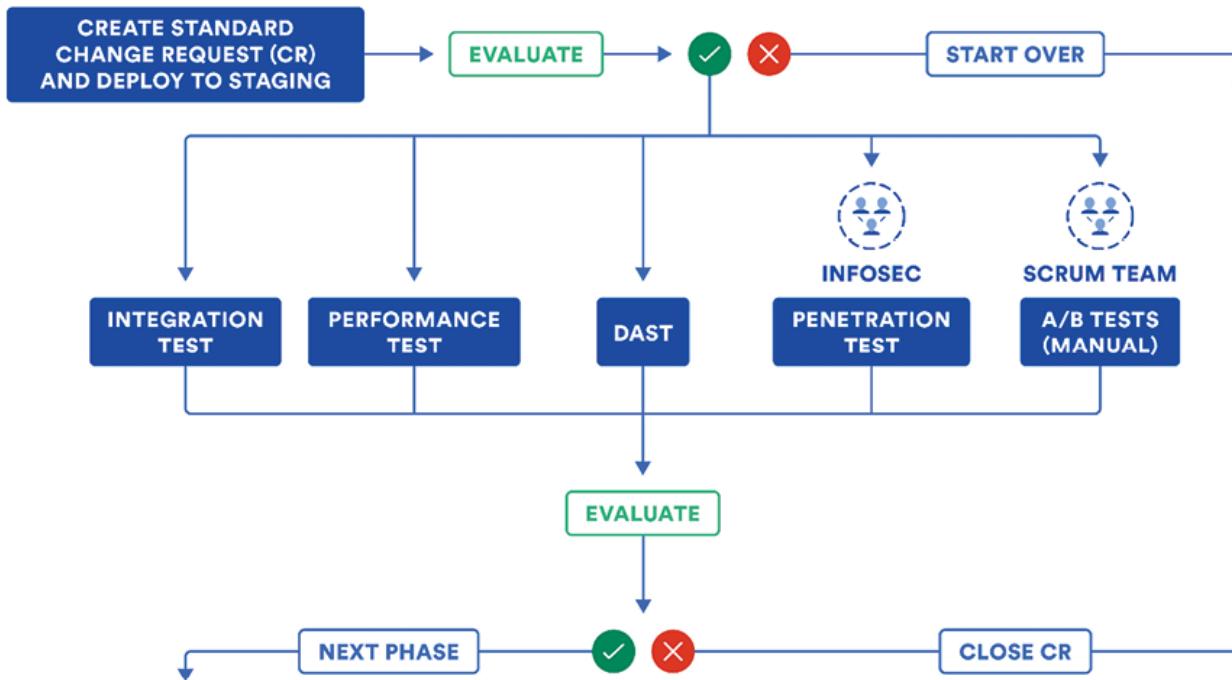


Figure 6.3: Continuous testing workflow

Increasing developer productivity

Enhancing productivity is a perpetual pursuit in the world of software development, where time and efficiency are paramount. In this age of rapidly evolving technology, the quest for increasing developer productivity has become more crucial than ever before.

Make good use of what is already at your disposal. Although it's true that inefficient workers often place the blame for their lack of output on the tools they employ, there's no denying that properly maintained and operated machinery may greatly increase output:

- Select superior equipment
- Make use of programming languages
- Get rid of manual labor

- Use what you already have
- Generate evaluations

DevOps is a collection of practices that encourages an agile way of thinking as well as rapid software development. Words like *continuous deployment*, *ongoing production*, *installation*, *communication*, *mechanization*, and *management* all spring to mind when discussing DevOps.

The term *Programmer Efficiency* is used to describe a company's output in general, not only for a certain period or set of criteria. To evaluate programmer output, a company could set measures or targets to monitor, along with targets to reach or a benchmark for success.

Performance in application development may be thought of as the relation of the program's functional benefits to the time and money spent creating it. Efficiency may be measured in a variety of ways, but most management focuses on two main metrics. Quantitative and qualitative. Quantitative measures of the significance of an action's results, and qualitative measurement metrics assess the maintainability, readability, and reliability of the codebase.

In the following figure, the types of research designs are shown:

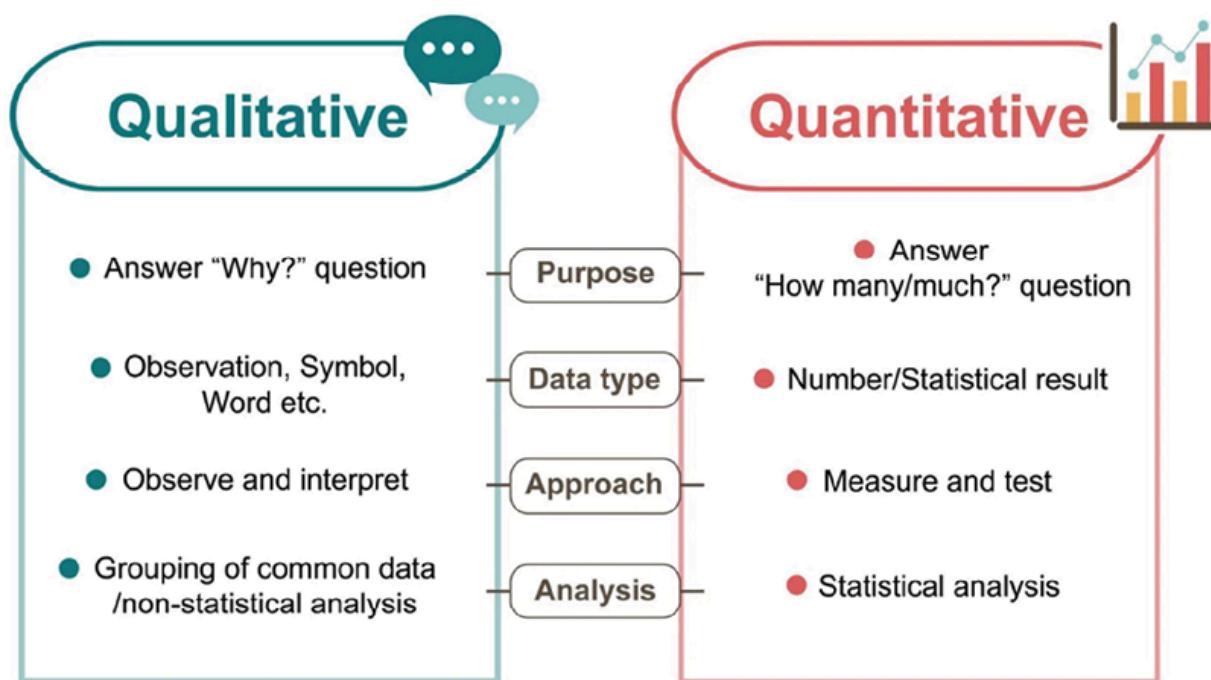


Figure 6.4: Type of research design

Effective time management is the key to the success of this method, which emphasizes the utilization of frequent, brief rests throughout the workday.

Simplifying implementation through automation

By integrating, orchestrating, and automating tools and procedures across production, management, and common service support, CloudBees Continuous Deployment Automate aids organizations in enhancing their technology product delivery for improved innovation and security. Top of Form

Consider a software development team working on a web application. Traditionally, deploying new features or updates involved a manual process that was time-consuming and prone to human error. By implementing automation with tools like CloudBees CD, the team can now seamlessly push code changes to production with just a few clicks. This not only speeds up the delivery process but also reduces the likelihood of deployment errors, resulting in a more reliable and efficient software release pipeline.

When it comes to improving system performance, automation testing may expand the depth and breadth of checks significantly. Unsecured assessment of long procedures that are frequently skipped through in the testing phase is now possible. Moreover, they may be used on a wide variety of machines.

Accelerating feedback delivery

Reducing bottlenecks, fostering more teamwork, cooperation, quickness in fixing bugs and releasing new versions, and constant deployment help organizations enhance the general user experience.

Monitoring plays a vital role in every DevOps implementation, offering valuable insights to the development team. Upon the conclusion of the **Continuous Integration and Continuous Delivery (CI/CD)** process, monitoring serves as the indicator of success or failure for the modified code, thereby closing the feedback loop in DevOps and enabling the start of the next iteration.

Without the proper software delivery technologies, even the best team will be limited in what they can do. Assuring that your team has access to all the

necessary resources, from computers to software to testing programs, is yet another crucial step in successful enterprise software.

However, faster delivery times result in increased sales. Rapid delivery is a key factor in influencing online purchases for 68% of respondents. One day's delay in shipping time may make or break a sale. Whenever service delays occur, 85% of customers go somewhere else.

The feedback loop of software delivery:



Figure 6.5: Feedback loop

Enhancing testing quality

Frequent testing is essential to producing high-quality software. This prevents slacking off and allows for the early identification as well as correction of problems before they become catastrophic. Analysis of past successes in the software development process should be done and the adjustments should be made accordingly.

Function, structure, and procedure are the three pillars of software quality. The testing process and product testing are crucial stages in the software development cycle. Together, they allow for a more streamlined procedure that yields a higher-quality result. This approach not only reduces costs associated with fixes but also enhances transaction capabilities and introduces new features.

Problems can be anticipated as well as mitigated using quality control methods. This type of testing, on the other hand, is done to find bugs as well as make sure they get fixed. **Quality Assurance (QA's)** main goal is

customer satisfaction, so it checks that the company's features and capabilities are in line with the needs of the customer.

In the world of software development, **Test-Driven Development (TDD)** stands as a transformative approach where code is written to pass tests rather than the other way around:

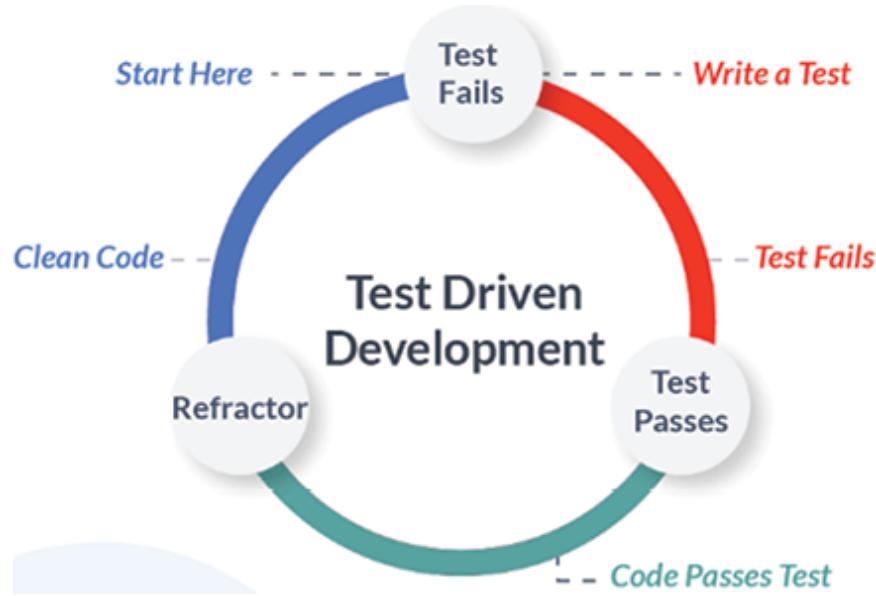


Figure 6.6: Test Driven Development

Expediting market introduction of new capabilities

Continuous Development improves developers' experiences across the board by reducing bottlenecks and speeding up teamwork, fixing bugs faster, as well as delivering software to customers more frequently.

Effective tooling is crucial to the DevOps approach, which enables teams to quickly deploy as well as develop for consumers. These instruments let teams manage complicated systems at scale, put scientists in charge of the rapid pace made possible by DevOps, as well as automating formerly manual processes.

The team utilizes various software shipping frameworks to get the design dictates for release. These concepts are called **software development methodology**, the computer development pipeline, and technology delivery.

The procedure of the application development lifecycle:

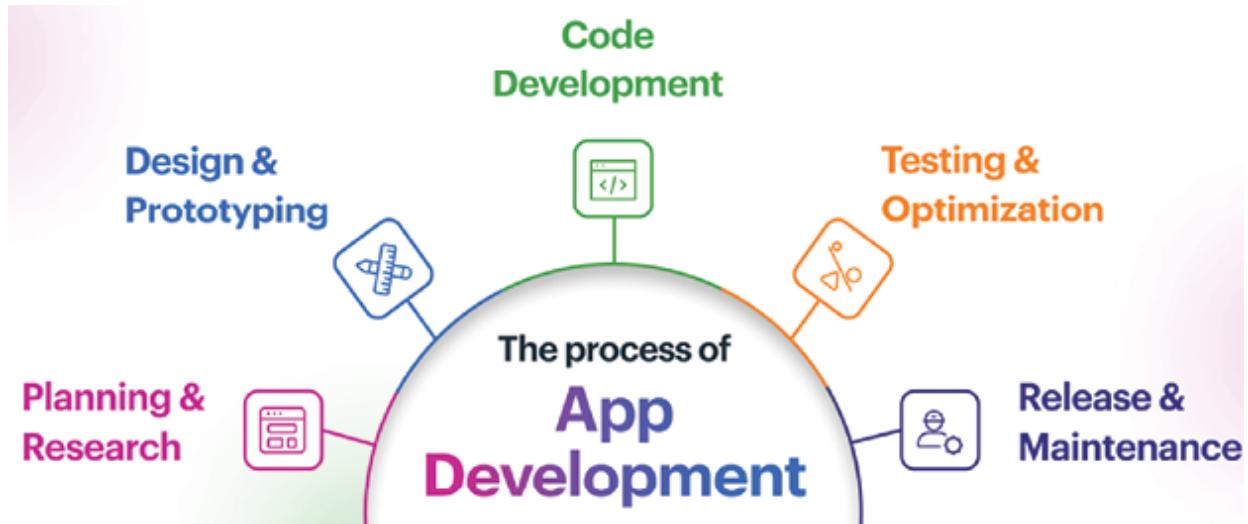


Figure 6.7: Application development

Selecting the tools for a robust CD process

The following are the best tools for CICD:

- AWS Code Deploy
- Octopus Deploy
- Jenkins
- TeamCity
- Deploy Bot
- GitLab
- Bamboo
- Circle
- Code ship
- Google Cloud Deployment Manager

DevOps pipeline and its relevant tools:

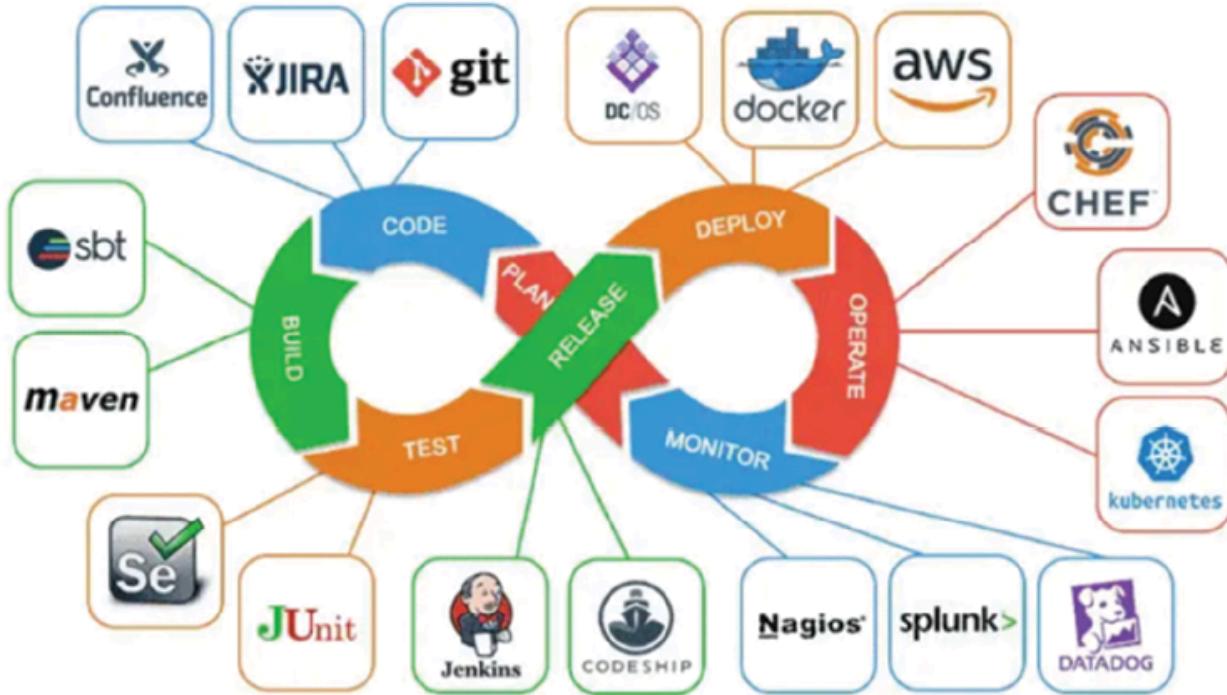


Figure 6.8: DevOps tools

AWS Code Deploy

Code transfers to Amazon EC2 instances or any instance operating on-premises may be automated with the help of **AWS Code Deploy**. With AWS Code Deploy, you can simplify the difficult process of upgrading existing apps, reduce the likelihood of interruption throughout delivery, as well as speed up the delivery of new functionality.

Octopus deploy

Program deployment, as well as distribution, are simplified, regardless of where they reside—on-premises or in the cloud. A release management software that streamlines the process of deploying applications as well as automating DevOps processes is also known as Octopus Deploy.

Jenkins

Jenkins is a Java-based, expansive CI/CD automation DevOps solution for software development lifecycle management. It acts as the backbone that enables the smooth operation of CI/CD processes.

TeamCity

TeamCity is a versatile CI/CD framework that can be used for a wide range of projects as well as development methodologies. DevOps operation may achieve the holy grail of CI/CD/CD successfully using this product.

Deploy Bot

Deploy Bot is a delivery program that automatically pushes software to servers from **Global Information Tracker (GIT)** sources. Sequential gearbox distributions are also possible, with standard or fully bespoke virtual machine allowing programs to be run or generated on servers throughout the installation phase.

GitLab

GitLab simplifies the packaging of apps as well as requirements, the management of vessels, as well as the creation of artifacts for organizations. Integrating GitLab's system software organization and CI/CD processes includes built, secured containers as well as product registration.

Bamboo

Bamboo is a CI platform that may manage the overall software development life cycle by handling the build, testing, and release of a program in one streamlined process.

Circle

CircleCI is a dedicated automated platform exclusively designed for the development, assessment, and release of software projects. This automated workflow operates within job-processing networks. A pipeline represents a set of actions executed by your CI tool whenever changes are made to your software.

Code ship

Internet Codeship aids enterprises in the simultaneous improvement and implementation of apps like Node. JavaScript, Ruby, PHP, as well as Python. The technology is flexible enough to meet the needs of both big and small organizations, letting users personalize web applications as well as manage virtualized infrastructure.

Google Cloud Deployment Manager

Installation Director for Cloud Servers is a solution for automating the provisioning as well as administration of Cloud Services capabilities.

Building a robust and fully automatic CD process

An iterative sequence of operations, considered to be the core integrating as well as simultaneous distribution (CI/CD) pipeline, is what it takes to release updated software. CI/CD pipelines are an approach to automating provisioning at every stage of the **Software Development Life Cycle (SDLC)**.

Businesses may speed up the creation of high-quality software by automating the whole CI/CD process from conception to tester to delivery to management. Each stage of a CI/CD pipeline may be carried out manually, but the railroad's actual value is unlocked when it is automated.

DevOps workflow:



Figure 6.9: DevOps workflow

Measuring considerations for calibrating a CD pipeline

The best way to set up a CD pipeline depends on the nature of your company. It has elements that vary with each group's computer environment, tools, regulatory requirements, as well as development objectives. Because of this, there is also no agreed-upon method for gauging its usefulness.

Following are some of the most important indicators for evaluating and improving the CI/CD pipelines in a DevOps company:

- Successful completion of the examination percentage, that is, the percentage of test scenarios that were successful as a percentage of all test scenarios.
- The total amount of bugs

- Percentage of defects that managed to get through quality control
- Quantity of forks in the code.

A continuous integration/continuous delivery pipeline is a distribution process that makes use of automation technologies as well as enhanced workflow. If done correctly, it will cut down on human mistakes and improve chain reactions across the SDLC, thereby letting groups publish more often but in smaller amounts.

Agility in SDLC:

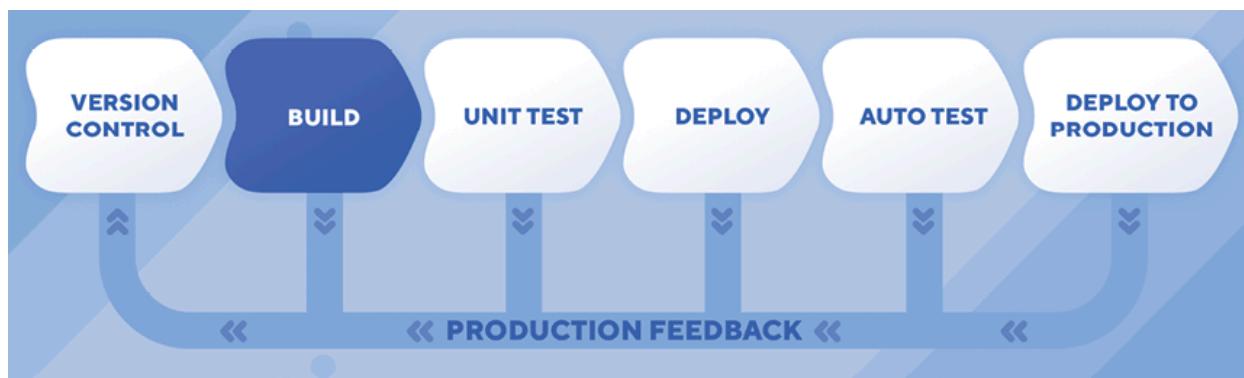


Figure 6.10: Software Development Life Cycle

The use of automated checks to ensure high-quality software is a defining feature of the CI/CD workflow. Software testing is employed at each stage of the software development lifecycle to detect vulnerabilities as well as other problems early on, to transfer material to multiple environments, and to deliver programs to production-ready settings.

Analyzing the lead time in a CD pipeline

The package *wait period* is the time taken by the manufacturers to create your containers. There seem to be numerous more factors that contribute to the total planning horizon, from the time you finally require shipping to the period your clients have it at one's fingertips. The distribution *wait period* seems to be the time it takes from the moment an order is submitted until it is shipped. The waiting period for a shipment advises a business on how long before it must make an obligation to ensure the shipment will come on schedule. It represents one of the most crucial variables to effectively manage inventories as well as distribution channels. Shipment advance is the

amount of time it takes from when such an order is placed to the moment it is delivered to the client. The whole time frame is accounted for, from the moment an order is placed to the moment a product is delivered.

Lead time measurement in software delivery:

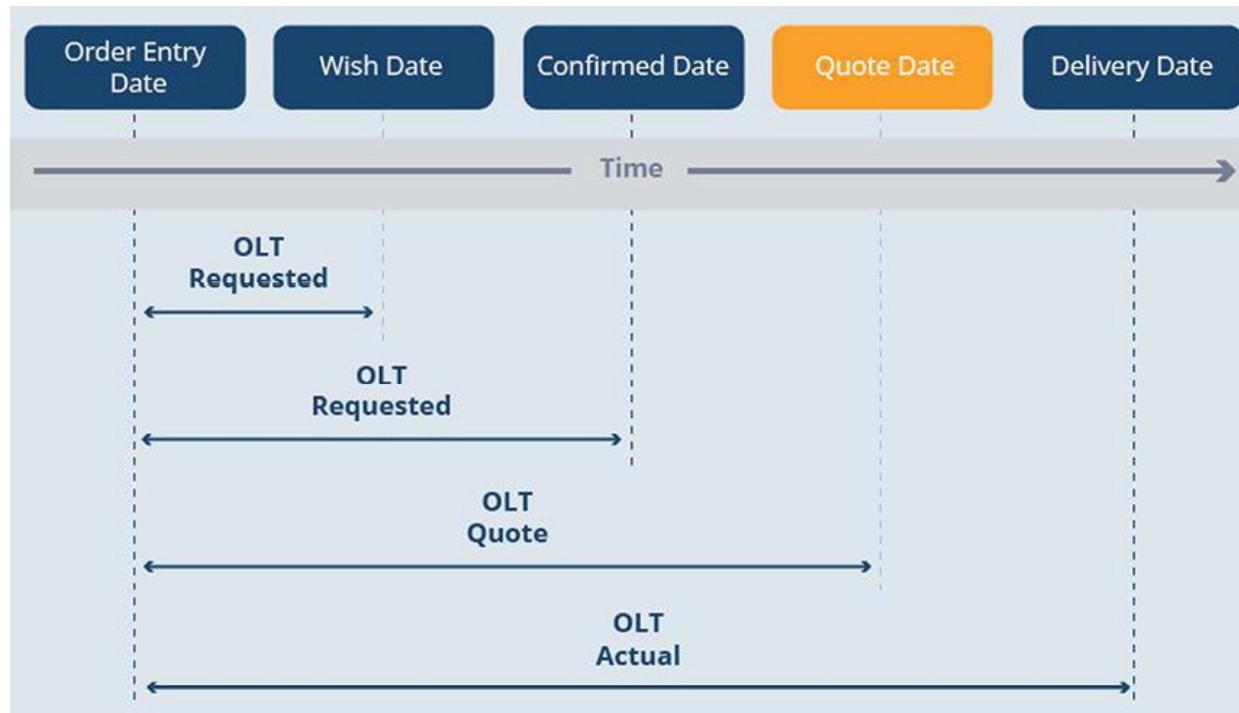


Figure 6.11: Lead time measurement in Software delivery

Evaluating the cycle time in a CD pipeline

Evaluating the cycle time in a CD pipeline refers to the process of assessing and analyzing the amount of time it takes for a software change to move through the various stages of a continuous delivery pipeline. This measurement provides insights into the efficiency and speed of the software delivery process.

Key points about evaluating cycle time in a CD pipeline:

- **Continuous Delivery Pipeline:** A CD pipeline is a series of automated stages and tasks that code changes go through, including building, testing, deployment, and potentially other quality assurance steps. The goal of a CD pipeline is to automate and streamline the delivery of software to production.

- **Cycle time:** Cycle time, in this context, refers to the time it takes for a code change to progress from the moment it is committed (for example, a code push or merge) to the time it is successfully deployed and running in a production environment.
- **Evaluation:** Evaluating cycle time involves measuring and analyzing this duration to gain a clear understanding of the efficiency and effectiveness of the CD pipeline. The evaluation may include tracking cycle time for individual changes, identifying bottlenecks or delays in the pipeline, and monitoring trends over time.
- **Efficiency improvement:** By evaluating cycle time, development teams can identify areas where improvements can be made to streamline the delivery process. This might involve optimizing test suites, enhancing automation, or reducing manual intervention.
- **Continuous improvement:** Continuous Delivery emphasizes the need for ongoing improvement. Regularly evaluating cycle time and making adjustments based on the findings helps organizations achieve faster, more reliable releases and better meet customer needs.
- **Feedback loop:** Cycle time metrics can provide valuable feedback to development teams, allowing them to adjust their processes, tools, and practices to achieve shorter and more predictable delivery times.
- **Quality and speed:** Reducing cycle time while maintaining or improving software quality is a key objective. It ensures that changes can be delivered quickly without sacrificing reliability or stability.
- **Real-time monitoring:** Some organizations use real-time monitoring and visualization tools to track the progress of code changes through the CD pipeline, making it easier to identify bottlenecks and delays as they occur.

In summary, evaluating the cycle time in a CD pipeline is a critical aspect of continuous delivery, helping organizations measure and improve the efficiency, speed, and reliability of their software delivery processes. It plays a significant role in achieving faster time-to-market and more responsive software development practices:



Figure 6.12: Cycle time in software development

Assessing the Mean Time to Recovery

If a system or device fails, the average amount of time it takes to get back up and running is known as the **Mean Time to Recovery (MTTR)**, that is the supposed time for the restoration or scale parameter to restore. All downtime, from the moment an item or system malfunctions to the moment it is completely functional thereafter, is factored in. The normal amount of time it takes to diagnose and fix broken machinery is known as the MTTR, a statistic used in maintained planning and control. MTTR is determined by dividing the sum of all unexpected repairs by the maximum number of replacements. The most common unit of MTTR is hrs. **Time to Repair (TTR)** is the total duration of the interruption, beginning with the first failure and ending when the system is completely functional afterward. The MTTR from errors in each system is determined by calculating the average of these times:

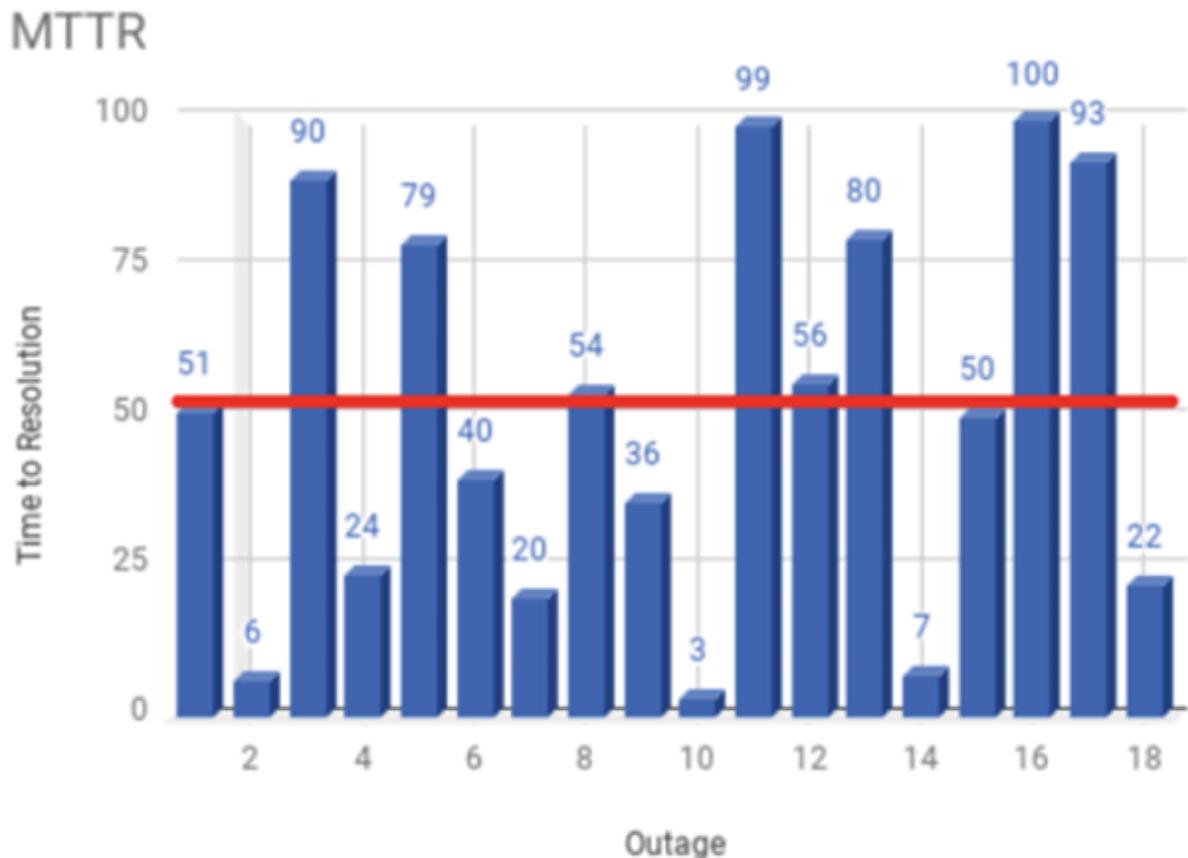


Figure 6.13: Mean Time to Recovery

Examining defect resolution time

One of the most important things for a project to be successful is the timely fixing of bugs. Unresolved critical faults are a common cause of late projects. Loss of money as well as unhappy customers are the direct results of this issue, which also hinders the proper delivery of an item. First, the testing phase will prioritize the faults and allocate them to developers, who will set up a timetable for fixing them. After fixing the problems, the developers send the results to the test manager. Issues may be easily fixed and monitored with this system. **Time to Repair (TTR)** is a metric used to assess how long it typically takes for a company's customer care staff to handle an outstanding complaint or other customer problem. You may determine this by dividing the number of weeks or months spent by the total cases closed, refer to the following figure:



Figure 6.14: Defect management process

Understanding the test pass rate

In software development, *understanding test pass rate* refers to the process of assessing and comprehending the success rate of test cases or tests conducted during the quality assurance and testing phases of a project. The test pass rate is a crucial metric that helps evaluate the quality and stability of a software product.

Here's how it works:

- **Test cases:** During the software development life cycle, various test cases are designed to validate that the software functions correctly and meets its requirements. These test cases can include unit tests, integration tests, regression tests, and more.

- **Execution:** Test cases are executed against the software being developed. Each test case represents a specific scenario or condition that the software must pass to be considered error-free.
- **Test pass:** A test pass occurs when a test case is executed, and the software behaves as expected, meeting the predefined criteria for success. In other words, the test case *passes*.
- **Test fail:** Conversely, a test fail occurs when a test case is executed, and the software does not behave as expected. This indicates a defect or issue in the software.
- **Test Pass Rate:** The test pass rate is the ratio of the number of passed test cases to the total number of executed test cases. It is usually expressed as a percentage. For example, if 90 out of 100 test cases pass, the test pass rate is 90%.

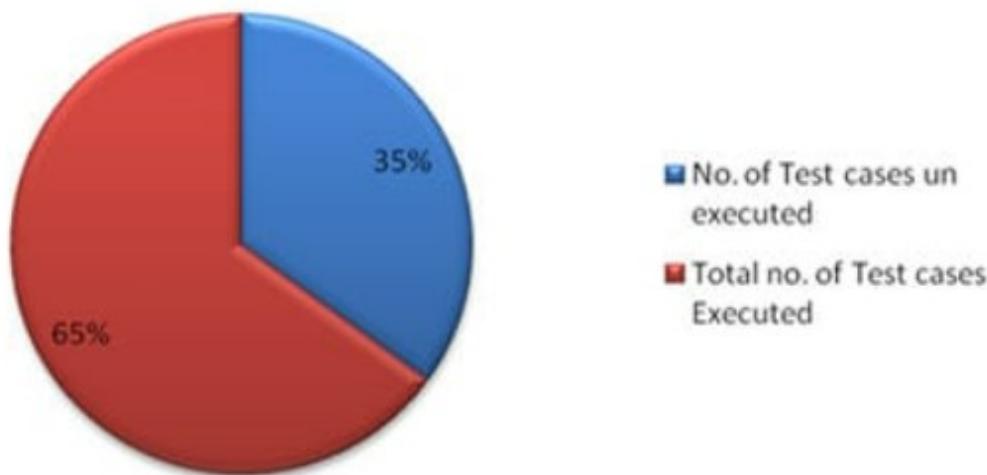
Understanding the test pass rate is essential for several reasons:

- **Quality assessment:** It provides a quantifiable measure of the quality of the software under development. A high test pass rate suggests that the software is meeting its requirements and functioning well.
- **Progress tracking:** It helps project stakeholders, including developers and managers, track the progress of testing efforts and identify areas that may require additional attention or debugging.
- **Defect identification:** Test failures indicate defects or issues in the software, allowing developers to pinpoint and address problems promptly.
- **Release readiness:** The high-test pass rate is a key factor in determining whether the software is ready for release. The low-test pass rate may indicate that more work is needed before release.
- **Risk management:** By monitoring the test pass rate, development teams can assess project risks and allocate resources accordingly to address potential challenges.

Overall, understanding the test pass rate is a fundamental aspect of software quality assurance and plays a significant role in ensuring the reliability and functionality of software products.

In the following figure, test metrics and measurements are shown:

Test execution completion %ge



Test execution status

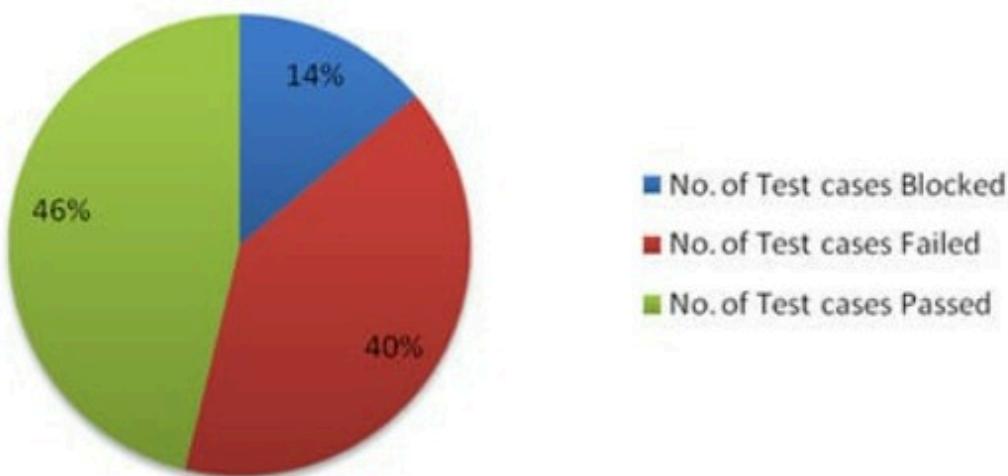


Figure 6.15: Test metrics and measurements

Best practices for adopting continuous delivery

As a component of a software deployment process, **Continuous Delivery (CD)** is a set of methods in which DevOps teams employ robotics to push out new versions of code to many venues in rapid succession. These updates should be delivered often with small improvements toward the code, thereby making the release low-risk, which leads to minimal disruptions for DevOps organizations as well as for end customers. To ensure that modifications can be pushed live at any time with known outcomes, companies that use CD techniques work hard to always maintain technology in a presentable condition.

Continuous deployment is shown in the following figure:

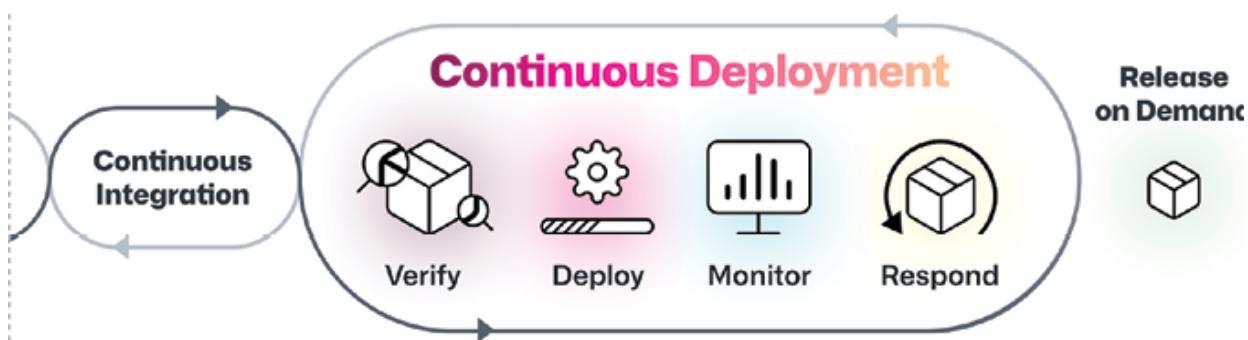


Figure 6.16: Continuous Deployment

Develop Service Level Objectives

An SLA's **SLO (Service Level Objectives)** is a pact between the parties on a defined measure, such as availability or speed of service. If the SLA is the overall contract between you and the client, then the SLOs are the specific commitments you are delivering to that consumer:

- As a first step, identify crucial user personas as well as rank them by their influence on the company.
- The second step is to zero down on the **Key Performance Indicator (KPI)** that will provide you with the clearest picture of the person's journey.
- Third, plan out the time frame for measuring success as well as setting the SLO targets.

Next, build error budgeting, SLO, as well as **Service Level Indicator (SLI)** control panels. Service Level Objectives:

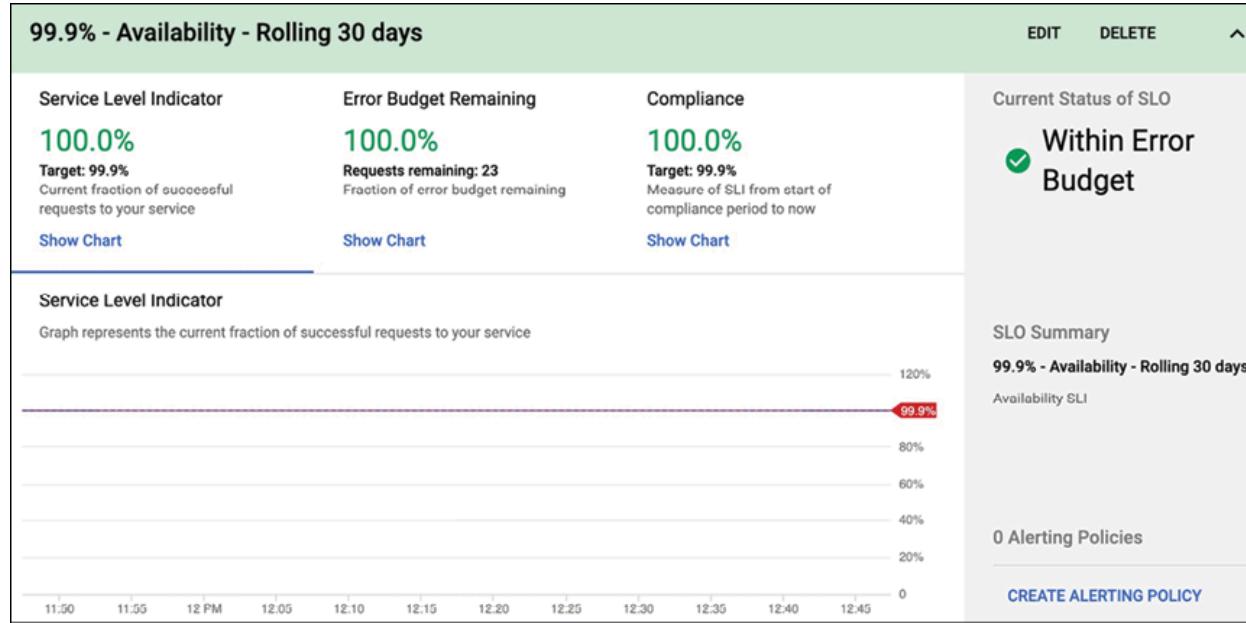


Figure 6.17: Service Level Objectives

Automating SLO evaluation with quality gates

The ideas which help guarantee that an enterprise lives up to its **Service Level Agreements (SLAs)** shine in performance management. We have witnessed that the same ideas begin *moving* into the pipelines for ongoing supply. To be more specific, organizations now utilize the outcomes of SLO assessments as a prerequisite before deploying a release to the operations environment. This is an illustration of four SLIs, the SLOs they're responsible for, or how the sum of those scores is used to determine whether a version meets the blog:

In the following figure, Shift Left: Performance Testing is shown:

Think Big, Act Small



Figure 6.18: Shift-left approach.

Automating every redundant process

Automating every repeatable and redundant process in software release refers to the practice of using automation tools and scripts to eliminate manual and repetitive tasks that are involved in the process of releasing software. In the context of software development and DevOps, this automation typically encompasses various stages of the software release lifecycle:

1. Select appropriate equipment for the distribution channel. Many different automation technologies and frameworks are required to create even a basic delivery pipeline.
2. Robotized test execution. One of the most important aspects of an agile development pipeline is the ability to conduct tests automatically.

3. Constant distribution using a type of system:

- Preparing code for distribution.
- Fabricating containers or images of virtual machines with predetermined settings.
- Streamlining the process of installing and setting up middleware.
- Distribution of files or packages into a live setting.
- Rebooting computer systems, software, or services

Automate everything strategy:

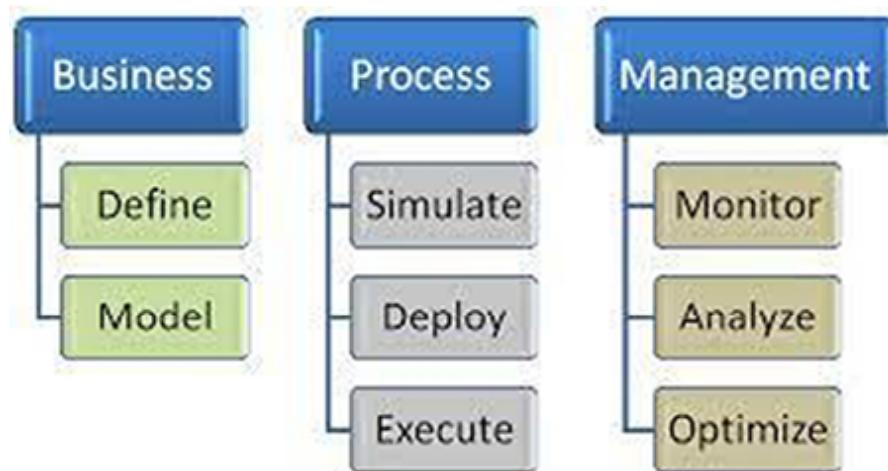


Figure 6.19: Automate everything

The goal of automating these processes is to reduce the manual effort required for each release, minimize the risk of human errors, and accelerate the delivery of new features and updates to end-users. Automation tools and CI/CD pipelines are commonly used to achieve these objectives.

By automating repeatable and redundant tasks, development teams can achieve greater efficiency, consistency, and reliability in their software release processes. This, in turn, leads to faster time-to-market, improved software quality, and better resource utilization.

Keeping everything in version control

Keeping everything in version control refers to the practice of storing and managing all aspects of a software project, including source code, configuration files, documentation, and other assets, within a **version control system (VCS)**. A version control system allows developers to track changes, collaborate with team members, and maintain a history of revisions for all project files. This practice helps ensure that the project remains organized, traceable, and secure, and it provides a means to revert to previous states of the project if needed. Popular version control systems include Git, SVN (Subversion), and Mercurial.

Version control systems are versatile tools that can be used to manage various types of data and assets in addition to source code. Here is a list of the types of data that can be kept in version control:

- **Source code:** This is the most common type of data managed in version control. It includes the code files, scripts, and configuration files that make up a software project.
- **Documentation:** Project documentation, such as design documents, technical specifications, and user manuals, can be stored in version control to track changes and maintain a history.
- **Configuration files:** Configuration files for software applications, server settings, or infrastructure as code (for example, Terraform or Ansible scripts) can be versioned to ensure consistency and traceability.
- **Data files:** Data files, such as databases, data sets, or data transformation scripts, can be managed in version control to track changes and ensure data integrity.
- **Images and graphics:** Graphics assets, images, icons, and other visual elements used in applications or websites can be versioned to maintain design consistency.
- **Web content:** HTML, CSS, and JavaScript files for web development projects can be kept in version control to track changes in web content.
- **Binary files:** While not as efficient as handling text-based files, some version control systems can manage binary files, such as compiled

code, images, audio, and video files.

- **Configurations for Infrastructure: Infrastructure as code (IaC)**: Infrastructure as code (IaC) files, such as those used in tools like Terraform or CloudFormation, can be versioned to manage cloud infrastructure configurations.
- **Scripts**: Automation scripts, batch files, and shell scripts used for deployment or maintenance tasks can be stored in version control.
- **Test data and test scripts**: Test data sets and test scripts used for automated testing can be versioned to ensure consistency in testing environments.
- **Database schema**: Database schema definitions and migrations scripts can be versioned to manage changes to the database structure.
- **Frameworks and libraries**: Third-party libraries and frameworks used in a project can be managed as dependencies in version control to ensure consistent usage across the team.
- **Assets for mobile apps**: Assets such as icons, images, and resources for mobile app development can be versioned.
- **Game assets**: Game development assets, including 3D models, textures, audio files, and level designs, can be managed in version control.
- **Machine learning models**: Machine learning models and their associated code, data preprocessing scripts, and training configurations can be versioned.
- **Configuration files**: Configuration files for development tools, IDE settings, and build scripts can be versioned to ensure consistency among team members.
- **Database backups**: Periodic backups of databases or data can be versioned to keep a history of changes and facilitate disaster recovery.
- **Network configurations**: Network device configurations, firewall rules, and network diagrams can be versioned to manage network infrastructure changes.

- **Configuration management script:** These scripts are used to automate the setup and configuration of servers and environments, ensuring consistency across development, testing, and production environments.

Version control systems provide a structured and organized way to manage these different types of data, helping teams collaborate effectively, track changes, and maintain a reliable history of their projects. The specific capabilities and limitations may vary depending on the chosen version control system:

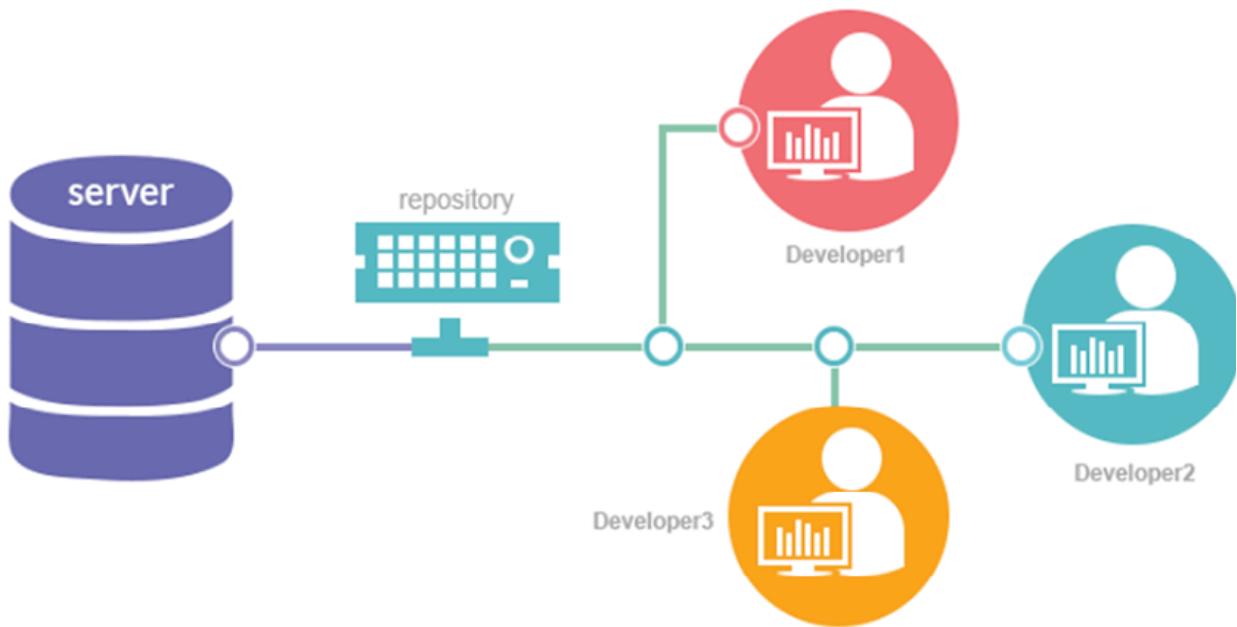


Figure 6.20: Version control

Providing fast, useful feedback

Real people use the product or visit the website, and their opinions regarding their experiences are gathered via customer reviews. They provide the development team with advice as well as suggestions on how to improve the product for future releases. By providing feedback, customers express their thoughts and feelings to businesses in an open forum. Importantly, it gives businesses a measure of consumer happiness to use, which may help them spot issues before they snowball into catastrophes. DevOps is predicated on a core idea- *Information on the flow of code through the pipeline should be collected in real-time, shared with the relevant teams, and used to formalize*

procedures for problem-solving along with continuous improvement. When launching this, it is important to keep the customer's path in mind.

Product Feedback Loop Flywheel:



Figure 6.21: Customer feedback loop

Deploying the same way to every environment

Agile development and deployments retain comparable roles in software development, although the former occurs before the latter. This means that with order fulfillment, the code is always subjected to a final, human certification process before moving from publication to production. In a constant shipment, you are free to experiment with different deployment tactics as well as iterations upon the release of new software packages. Four popular deployment strategies covered here are: **simultaneously** (fly-in spot), **continuous**, permanent, and **turquoise**. When a technology product is available for delivery, the team can rest easy knowing they have a solid rollout strategy in place that details everything from backup procedures as

well as security measures to everyone's individual responsibilities. Workers aim for a seamless installation process in their installation strategies.

Build once deploy many:

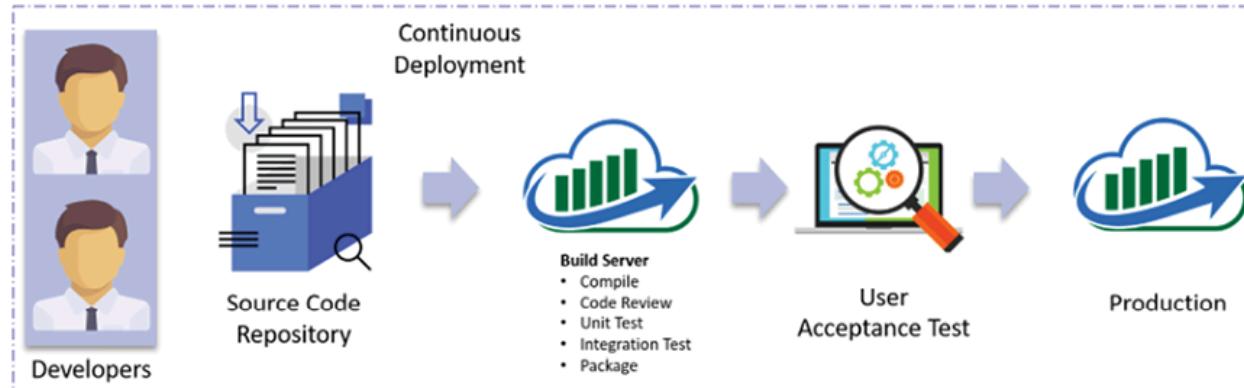


Figure 6.22: Build once deploy many

Avoiding direct changes in the production environment

The distinction is now readily apparent. With continuous delivery, programmers push out updates to clients with the press of a button, while in system implementation, automation is prioritized throughout. With the use of feature flags, we can control who has access to the compiled code and who does not. When new software, goods, or upgrades are ready to be released to the public, they are transferred to what is called a 'production environment', which is the completion of the manufacturing process. This is where the customer will be able to view, touch, and utilize the new products.

Types of environments:

Reasons for having separate environments



Figure 6.23: Types of environments

Deploying to every environment the same way

Continuous integration and deployment have comparable roles in software engineering, although the former occurs before the latter does. This means that with rapid deployment, the program is almost always subjected to a final, human approval procedure before being published to live. In a continuous delivery time, you are free to experiment with different deployment tactics as well as iterations while deploying new software packages. Four popular means of providing are covered here: **all at once** (deploy in place), **continuous**, **permanent**, as well as **blue/green**.

Continuous integration pipeline:

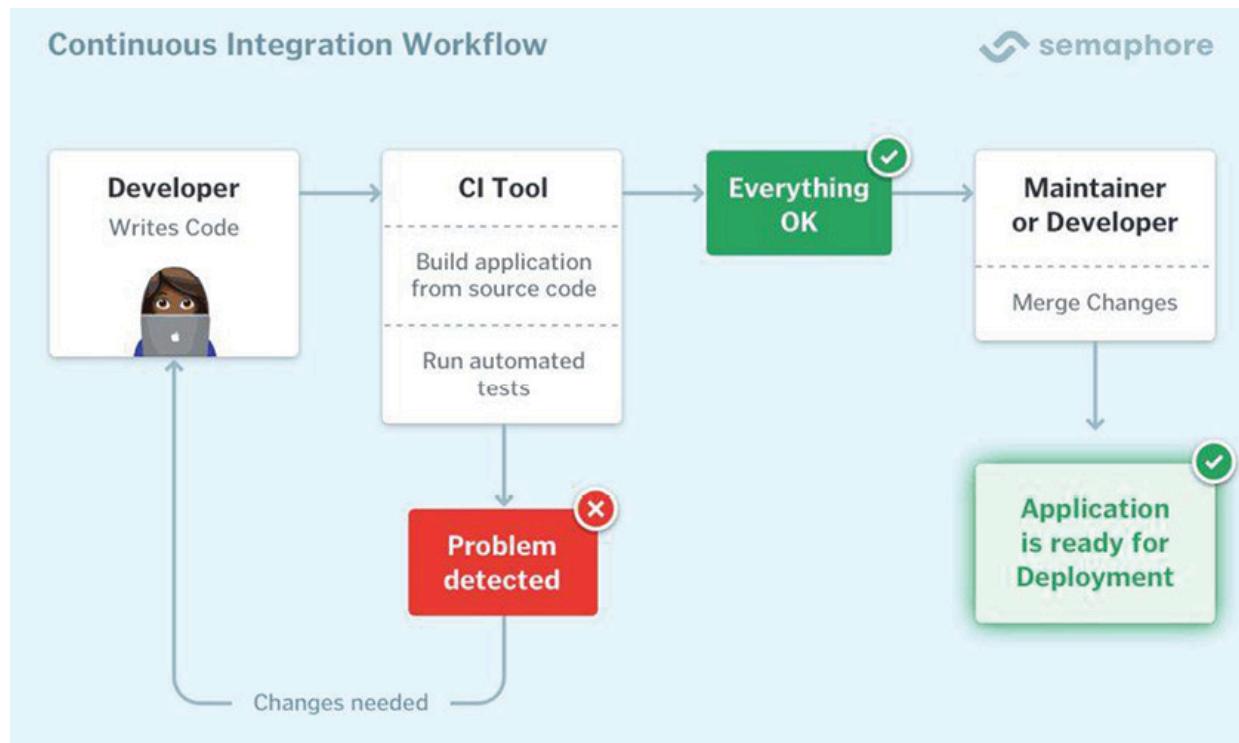


Figure 6.24: Continuous integration pipeline

Deploying in a copy of production

To manage a live application or app, it is necessary for the network or group to have all its nodes and devices licensed. Additionally, the process includes steps such as redeployment, maintenance, deployment, and commenting.

These steps collectively form the process of handling and maintaining a live application or app. There is a time limit associated with each phase, as well as a unique set of mental obstacles that each person in the family would and will need to overcome. Whether it's a testing system, a manufacturing platform, a user's PC, or a smart phone, the software part is the act of getting the program up and running on the desired device. The phrases *software rollout* as well as *application services* may be used indiscriminately.

In the following figure, software deployment stages are shown:

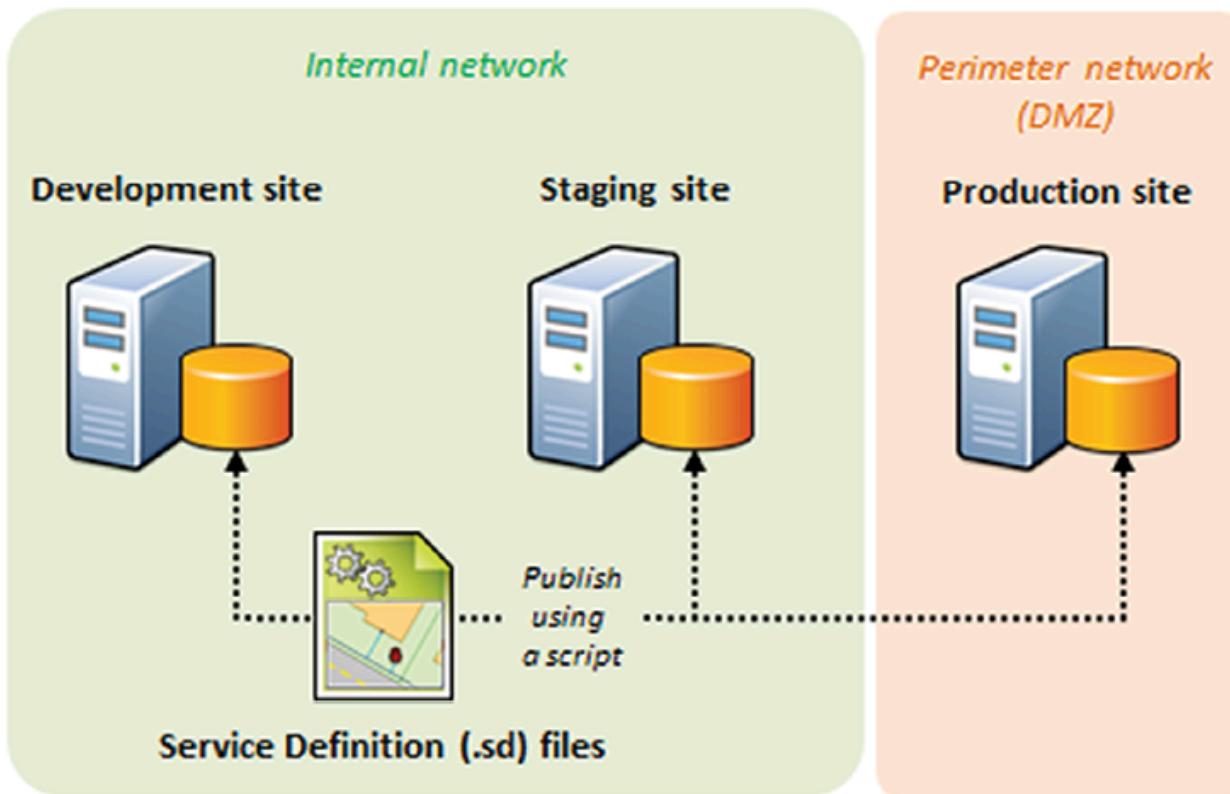


Figure 6.25: Software deployment stages

Including the database

Following is a list of some of the most well recognized and widely used instruments:

- Jenkins.
- TeamCity.
- Liquibase (free)

- Datical (a premium/paid version of Liquibase)
- Redgate (Microsoft Stack)
- Delphix (not simply for database modifications) (not just for database changes)
- DBmaestro (they market and sell as DevOps for databases)

In the following figure, the data pipeline is shown:



Figure 6.26: Data pipeline

Eliminating complexity

Smaller pieces of code (functions, procedures, and methods) are easier to maintain as well as comprehend, but words like `goto`, `halt`, and so on carry and add unnecessary layers of complexity to systems. You may use the term ‘software complexity’ to define a certain collection of features in your program. These qualities are all concerned with the connections your program makes with others. Your code’s complexities are measured in terms of all these qualities. It is similar to getting a score for the functionality of the program.

The complexity of software development and delivery:



Figure 6.27: Software complexity

Establishing observability and continuous monitoring

Surveillance is the practice of keeping *tabs on* and analyzing the health of one's computer systems, using either specialized software or hardware. Metrics and logs may be collected in advance for use in monitoring. The term *reliability* refers to a set of tools or a technological solution that gives teams the capability to continuously analyze their technology. One of the Version control systems for constant monitoring is called **Nagios**. It's a popular free program that is quite popular. Nagios may help in monitoring infrastructure, software, as well as operational procedures in a DevOps environment.

Benefits of continuous monitoring:

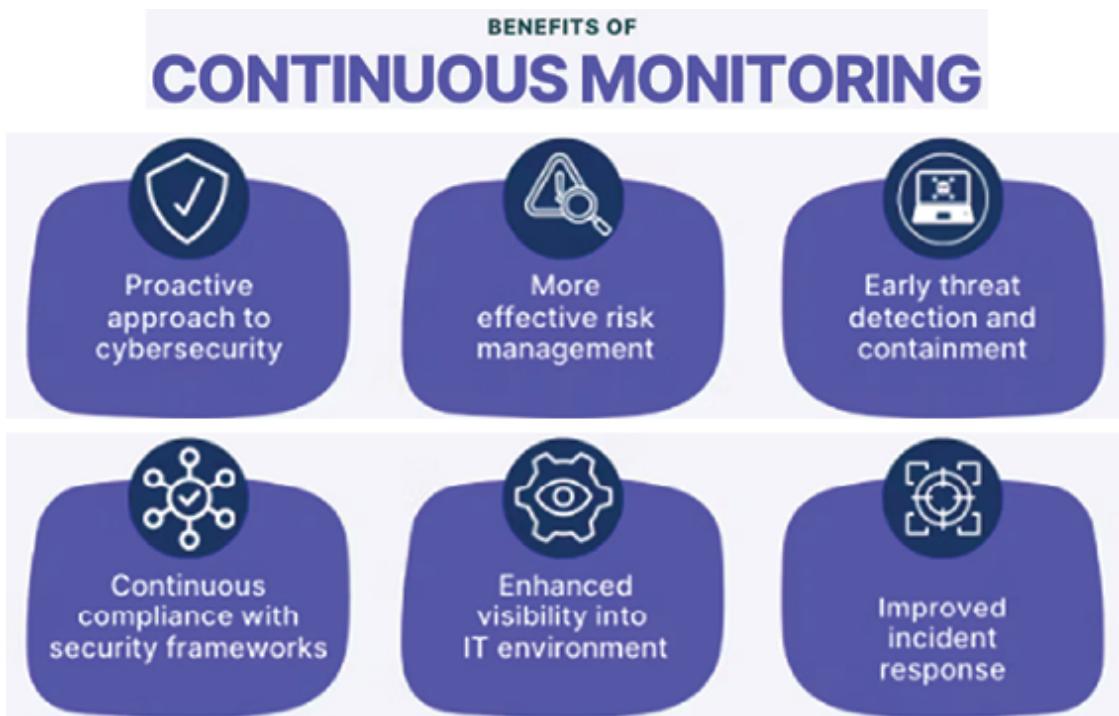


Figure 6.28: Continuous monitoring

Conclusion

In conclusion, CD has revolutionized the way software is developed, tested, and delivered. It represents a fundamental shift from traditional, manual, and error-prone release processes to a streamlined, automated, and efficient approach that prioritizes speed, quality, and reliability. In today's rapidly evolving technology landscape, Continuous Delivery is a cornerstone of modern software development practices. It empowers organizations to innovate, iterate, and deliver value to customers at a pace that was once considered unimaginable, ultimately driving business success and competitiveness in a digital world.

In next chapter we are going to learn Automate testing. It is a software testing technique that involves the use of automated testing tools and scripts to execute test cases and verify the behavior of a software application or system. In automated testing, a testing tool or framework is employed to automate the repetitive and manual steps of running test cases, comparing actual outcomes to expected results, and generating test reports.

CHAPTER 7

Automated Testing

Introduction

Automated testing in software development refers to using software tools and scripts to test the functionality and performance of software applications. Automated testing aims to increase the speed and reliability of the testing process and reduce the manual effort required to test an application. Automated testing is particularly beneficial in regression testing, where it ensures that new code changes do not adversely impact existing functionalities. It also facilitates the identification of defects early in the development process, enabling prompt corrections. Moreover, automated tests can be executed across various environments, configurations, and platforms, ensuring comprehensive test coverage. While automated testing offers numerous advantages, it requires careful planning, maintenance, and continuous integration into the development workflow. Well-designed automated test suites contribute to faster release cycles, improved software quality, and increased confidence in the reliability of the developed software.

Structure

In this chapter, we will learn why automated testing is required in the software development lifecycle. This chapter explains the benefits, best practices, tools, and how to configure automated testing.

The upcoming chapter will delve into the following subjects:

- Adopting more test automation
- Introducing tool integration
- Tracking metrics
- Leverage containerization
- Keeping communication transparent
- Saving time with headless execution
- Exploring multilayer tests
- Integrating performance testing into the delivery cycle
- Building robust continuous testing with tools.
- Continuous testing
- Step-by-step builds robust and fully automatic continuous testing

Objectives

In this chapter, we will learn about how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks, along with minimizing the impact of issues and improving the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Adopting test automation

Adopting test automation is an important aspect of software development, as it allows teams to ensure that their software is of high quality and defect-free. Automated testing allows teams to run tests quickly and easily against their code, which helps to identify any issues or bugs early in the development process. In this section, we will discuss the benefits of adopting more test automation, its implementation and the best practices.

Adopting more test automation can be beneficial for software development in several ways:

- Improves overall software quality.

- Increases speed of the development process.
- Improves collaboration and coordination among teams.
- Create automated test scripts and set up test environments.
- Adhering to best practices such as test-driven development and creating repeatable, reliable, and maintainable tests.
- Aligning test automation strategy with the overall development strategy.
- Using **continuous integration and continuous delivery (CI/CD)** pipeline.
- Automating regression tests to ensure that changes in the software do not introduce new bugs.

One of the main benefits of adopting more test automation is the ability to improve the overall quality of the software. Automated tests enable quick and easy identification and fixing of defects, reducing the number of bugs that make it into production.

Test automation accelerates the development process which allows teams to get feedback on their code more quickly. This can help to reduce the time it takes to develop and release software. Test automation helps improve collaboration and coordination among teams. Automated tests can be run on a regular basis, which allows teams to stay informed on the progress of the project which helps grow communication and collaboration among teams.

To implement test automation, teams need to invest in the right tools and resources. This can include tools like Selenium, Appium, or JUnit, as well as resources like test automation engineers or test automation frameworks. Additionally, investments in training and education need to be made, so teams can learn how to use these tools and resources effectively.

Once teams have invested in the right tools and resources, they can start to implement test automation by creating automated test scripts, setting up test environments, and configuring test runs. Establishing procedures for handling test results, defects, and maintaining test automation in the long term is crucial.

Adhering to best practices for test automation is also important. This includes using a test-driven development approach, which involves writing automated tests before writing code. Additionally, teams should focus on creating repeatable, reliable, and easy to maintain tests.

Teams should also focus on creating a test automation strategy that aligns with their overall development strategy. This can include prioritizing certain areas of the application for automation, and identifying key metrics for measuring the success of their test automation efforts.

Another best practice for test automation is to use a CI/CD pipeline. This allows teams to automatically run tests whenever code is committed to the repository, and it helps to identify issues early in the development process.

Lastly, teams should also focus on maintaining and updating their test automation by regularly reviewing and updating test scripts and adding new tests as needed. Additionally, teams should focus on automating regression tests, which help to ensure that previously developed and validated functionalities within the software remain unaffected by new code changes and updates.

Introducing tool integration

Tool integration is an important aspect of software development, as it allows teams to streamline their workflow and increase efficiency. Tool integration refers to the process of integrating different tools and technologies into a cohesive system, which allows teams to automate and optimize various aspects of the development process. In this section, we will discuss the benefits of tool integration, how to implement tool integration, and best practices for tool integration.

Tool integration helps to streamline the development process. By integrating various tools and technologies into a cohesive system, teams can automate and optimize various aspects of the development process. This includes automating tasks such as testing, deployment, and monitoring, resulting in saving time and money.

There are many tools that can be used for automated testing in software development, some of the most popular include:

- **Selenium:** It is an open-source tool used widely for web application testing. Selenium supports multiple programming languages, including Java, C#, Python, and Ruby.
- **Appium:** Appium is an open-source tool that is used for mobile application testing, and it also supports both Android and iOS platforms.
- **JUnit:** JUnit is a popular open-source tool for unit testing in Java. It provides annotations and assertions for testing, and it is often integrated with other tools like Maven, Gradle, or Eclipse.
- **TestNG:** TestNG is a tool for unit testing like JUnit. It is designed for Java, but it can be used with other programming languages like Groovy, Scala, or Kotlin.
- **Cucumber:** Cucumber is an open-source tool used for **Behavior-Driven Development (BDD)** testing. It supports multiple programming languages, including Java, Ruby, and JavaScript.
- **Jenkins:** Jenkins is an open-source tool used for CI/CD. Jenkins can be used to automate the testing process, by triggering tests whenever new code is pushed to the repository.
- **TestComplete:** TestComplete is a commercial tool that supports automated testing of web, desktop, and mobile applications. It supports multiple programming languages, and it has a record and replay feature for easy test creation.
- **SoapUI:** SoapUI is an open-source tool used for web service testing. SoapUI supports both REST and SOAP web services and allows for functional and security testing.
- **LoadRunner:** LoadRunner is a commercial tool used for load and performance testing. It can simulate thousands of users accessing the system simultaneously; therefore it is often used to test the scalability and capacity of the system.
- **Gatling:** Gatling is an open-source tool used for load and performance testing. It is designed for high-performance systems and can simulate thousands of users accessing the system simultaneously.

These are just a few examples of the many tools that are available for automated testing in software development. The choice of tool will depend on the specific requirements of the project and the programming language used.

Integrating tools for automated testing in a software release process can be a complex task, but it can be accomplished by following these general steps:

1. **Define your testing strategy:** Before integrating tools, it is important to have a clear understanding of your needs, to define a testing strategy outlining the goals and objectives of your efforts.
2. **Choose the right tools:** Select the appropriate testing tools for your project, taking into consideration the programming language, the type of application, and the platform.
3. **Set up the testing environment:** This includes setting up the necessary infrastructure, installing the testing tools, and configuring the necessary test data.
4. **Create test scripts:** Develop test scripts that can be used to automate the testing process. This includes creating unit tests, functional tests, and performance tests.
5. **Integrate the testing tools into your development workflow:** Integrate the testing tools into your development workflow so that the new code, whenever pushed into the repository, can be tested automatically using tools like Jenkins, TravisCI, or CircleCI.
6. **Automate the testing process:** Automate the testing process by using scripts and tools to automatically test the software whenever new code is pushed to the repository.
7. **Monitor test results:** Monitor the test results to identify and fix any issues or bugs that are found.
8. **Implement CI/CD:** Implement a CI/CD pipeline to automatically build, test and deploy your software for a faster and efficient delivery.
9. **Communicate the results:** Communicate the results to the development team, in order to fix any issues, if found.
10. **Review and improve:** Continuously review and improve the testing process, by identifying and addressing issues found, and by using the

results to improve the quality of the software.

By following these steps, you can successfully integrate tools for automated testing into your software release process, which will help to improve the quality of the software and reduce the amount of manual effort required to test the software.

Tracking metrics

Tracking metrics is an important aspect of software development, as it allows developers to measure the performance and quality of their software and make data-driven decisions to improve it. Metrics can be used to track a wide range of information, including the number of bugs found, the amount of time required to fix bugs, and the overall performance of the software.

There are several types of metrics that are commonly used in software development which are as follows:

- **Code coverage:** Code coverage measures the percentage of code that is executed during testing. It is used to measure the quality of the testing process and the effectiveness of the tests.
- **Bug density:** Bug density measures the number of bugs found per line of code. It is used to measure the overall quality of the software by identifying where the software is most prone to bugs.
- **Time to fix:** Time to fix measures the amount of time required to fix a bug, from the time it is reported to the time it is fixed. It serves as a valuable metric to measure the efficiency of the development team and the overall quality of the software.
- **Mean time to recovery (MTTR):** Mean time to recovery (MTTR) measures the amount of time required to recover from a failure. It finds use to measure the reliability and availability of the software.
- **Performance metrics:** Performance metrics measure the performance of the software, such as response time, throughput, and resource usage. These metrics are useful for identifying performance bottlenecks and for measuring the scalability of the software.

- **User engagement metrics:** User engagement metrics measure the engagement of the users with the software, such as the number of active users, the number of sessions, and the retention rate. These metrics are useful for understanding how the users are interacting with the software, and how to improve the user experience.

Tracking metrics is an ongoing process, and it requires a certain level of commitment from the development team. To track metrics effectively, it is crucial to choose the right metrics for your software and to track them consistently over time. This requires a clear understanding of the software development process and a solid plan for tracking the metrics that are important to your software.

To track metrics, you will need to use tools that are designed to collect and analyze data, such as analytics platforms, log analysis tools, or performance monitoring tools. It is also important to have a team of skilled developers and testers who can write and maintain the code that is used to collect and analyze the metrics.

Tracking metrics can help the development team make data-driven decisions, identify and fix issues early in the development process, and improve the overall quality of the software. By tracking metrics consistently and regularly, the development team can have a better understanding of how the software is.

For example, if the development team is tracking the code coverage metric, they can identify areas of the code that are not being tested and improve the test coverage for those areas. This can help to identify bugs that would otherwise go unnoticed and improve the overall quality of the software.

Another example is tracking bug density, this metric can help the development team to identify areas of the code that are most bug-prone and to focus their efforts on fixing those bugs. This can help to reduce the number of bugs in the software, thereby improving the quality of the software.

Tracking the time taken to fix a metric can help the development team measure the efficiency and the overall quality of the software. If the time to fix a bug is high, the team can investigate the reasons and improve their processes to reduce the time.

MTTR is another important metric to track. It assesses the duration needed to recover from a failure, a metric valuable for gauging the software's reliability and availability. If the MTTR is high, the development team may need to investigate the root causes of failures more deeply, refine their incident response processes, and implement improvements to reduce system downtime and enhance overall stability.

In order to implement tracking metrics, teams will need to invest in the right tools and resources. This can include tools such as Google Analytics, Mixpanel, or Prometheus, as well as resources such as data analysts or data visualization tools. Additionally, teams will need to establish a process for collecting, analyzing, and acting upon the metrics.

After acquiring the necessary tools and resources, teams can proceed to implement metric tracking by determining the relevant metrics to monitor and the appropriate data collection methods. For instance, teams might opt to track metrics like error rates, response times, and user engagement utilizing tools such as Google Analytics. Similarly, they can track resource utilization and load times using tools like Prometheus.

It is vital to establish a process for analyzing the metrics and taking action based on the results. For example, teams may establish a weekly or monthly review process to analyze the metrics and identify areas where the software is underperforming. They should also establish a process for addressing issues and implementing improvements based on the metrics.

Best practices for tracking metrics include:

- Identifying the most important metrics to track.
- Establishing a process for collecting and analyzing the metrics.
- Establishing a process for taking action based on the metrics.
- Regularly reviewing the metrics and adjusting as needed.
- Tracking metrics across the entire development lifecycle, from development to production.
- Making sure that the metrics are aligned with the overall business goals and objectives.

- Implementing monitoring and alerting systems to notify teams of anomalies or critical incidents.
- Ensuring that the metrics are accurate, reliable, and consistent.
- Making sure that the data collected is secure and compliant with regulations.
- Using visualization tools to present the metrics in an easy-to-understand format, to help teams make data-driven decisions.

In summary, incorporating metric tracking into software development is crucial. By monitoring metrics like error rates, response times, and user engagement, teams can promptly address issues and prevent them from escalating. This leads to enhanced software quality and reduced maintenance costs. Furthermore, tracking metrics empowers teams to make data-driven decisions and optimize software performance. By implementing effective metric tracking and adhering to best practices, teams can elevate the overall performance, stability, and user satisfaction of their software.

Leverage containerization

Leveraging containerization can greatly benefit the testing process by providing a consistent and isolated environment for running tests. When tests are run in a containerized environment, developers can be sure that the environment is identical every time the tests are run, regardless of the underlying infrastructure which makes the tests repeatable and reliable. There is a possibility of having two separate teams, one dedicated to development and the other focused on Testing, with automated testing providing advantages to both teams.

One way containerization can help testing is by creating a consistent environment for running tests. For example, when using containerization, developers can package their application and its dependencies in a container and run the tests within the same container. This ensures that the test environment is consistent and identical every time the tests are run, regardless of the underlying infrastructure. This eliminates the possibility of test failures caused by differences in the environment, such as different versions of libraries or software.

Another way containerization can help testing is by providing an isolated environment for running tests. For example, while using containerization, developers can run their tests in a separate container, isolated from the production environment. This can help to prevent any test failures from affecting the production environment and it also ensures that the test environment is not impacted by any changes made to the production environment.

The following are the benefits of testing with the use of a containerized platform:

- Consistent testing environment
- Isolated testing environment
- Repeatable and reliable tests
- Speed up test execution
- Cost-effective testing solution
- Improved security during testing
- Easy management of test environments
- Improved scalability during testing
- Better resource utilization during testing
- Improved collaboration and coordination among teams.

A great example of how containerization helps testing is by using test containers. These are containers that are specifically designed to run tests. They are lightweight and disposable, which means they can be created and discarded as needed. This helps to reduce the time required to set up and tear down test environments, which can save time and resources.

For instance, using Docker, developers can create a test container that includes the necessary dependencies for their application, such as a specific version of a programming language or a specific database. Once the test container is created, developers can run their tests within the container, which ensures that the test environment is consistent and identical every time the tests are run. This eliminates the possibility of test failures caused by differences in the environment.

Similarly, Kubernetes can be used to automate the deployment, scaling, and management of containerized applications in a testing environment. It also provides features for monitoring and logging, automatic scaling, and self-healing, which can help to ensure that the testing environment is stable and reliable.

Here are some best practices for setting up a containerization platform for automated testing:

- Use a container orchestration tool such as Kubernetes or Docker Swarm to manage the containers. This allows you to easily scale, deploy and manage containers across multiple environments.
- Use container images that are pre-configured with all the necessary dependencies for your testing environment. This enables you to easily spin up new containers and ensures consistency across all test environments.
- Use version control for container images to keep track of changes and roll back if necessary. This allows you to easily maintain and update your container images.
- Use a centralized container registry to store and distribute your container images. This lets you share container images across teams and environments easily.
- Use environment variables to configure container images. This allows you to easily change the configuration of a container without modifying the container image itself.
- Use a continuous integration tool such as Jenkins or Travis CI to automatically build and test your container images. This allows you to catch issues early on and ensure that container images are always up-to-date.
- Use a container management platform like Kubernetes or OpenShift to automate the deployment of containerized applications. This allows you to easily deploy and manage containers across multiple environments.

- Use a monitoring tool like Prometheus or InfluxDB to collect and analyze metrics from your containers. This allows you to monitor the performance of your containers and identify any issues.
- Use a logging tool like Elasticsearch or Logstash to collect and analyze log data from your containers. This allows you to troubleshoot issues and identify any errors.
- Use a security tool like SELinux or AppArmor to secure your containerized applications. This allows you to protect your containers from unauthorized access and potential attacks.

It is important to keep in mind that containerization is not a one-size-fits-all solution, and the best practices might vary depending on the complexity of the application and the specific requirements of the organization.

In summary, leveraging containerization can greatly benefit the testing process by providing a consistent and isolated environment for running tests. Through this, developers can ensure that the test environment is identical every time the tests are run, regardless of the underlying infrastructure. This ensures the repeatability and reliability of the tests, making it straightforward to reproduce and address any identified issues. Containerization also helps to save time and resources during the testing process by reducing the time required to set up and tear down test environments.

Keeping communication transparent

Keeping communication transparent is an essential aspect of software development which allows teams to work efficiently and effectively towards a common goal. Transparent communication helps to ensure that everyone is on the same page and is kept in the loop regarding what is happening in the project. In this section, we will discuss various strategies and techniques for keeping communication transparent in software development.

Here is how transparent communication is used in DevOps:

- Establish clear and consistent channels of communication.
- Hold regular team meetings.
- Use a project management tool that allows for clear and transparent communication.

- Establish clear and consistent guidelines for communication.
- Encourage open communication among team members.
- Assign clear roles and responsibilities for each team member.
- Encourage feedback and suggestions from team members.
- Communicate progress and updates regularly.
- Address communication breakdowns promptly.
- Regularly review communication processes and make necessary adjustments.

Primarily, it is important to establish clear and consistent channels of communication. This can include regular team meetings, email, instant messaging, or other forms of communication that are appropriate for your team. Establishing clear and consistent channels of communication helps to ensure that everyone knows how to communicate with each other and that everyone is aware of the latest happenings in the project. There should be agile champions or scrum masters taking ownership of communication and keeping track and maintaining checklists. These agile champions, scrum masters, or project managers can use project management tools like Jira and Trello.

Regular team meetings are one of the most effective ways to keep communication transparent. These meetings can be held in person or via video conferencing, and they can be used to discuss progress, share updates, and identify any issues or roadblocks that need to be addressed. For example, a team could schedule a weekly meeting to discuss the progress of the current sprint and to plan for the next sprint. This contributes to ensuring that all individuals are well-informed about the project's progress and facilitates effective communication among team members.

In addition to regular team meetings, it is also important to use a project management tool that allows for clear and transparent communication. For example, a team could use a tool like Trello to track the progress of their project. Trello allows team members to share updates, assign tasks, and track progress in a centralized location. This helps ensure that everyone is aware of what is happening in the project and that everyone can communicate effectively with each other.

Another important aspect of keeping communication transparent is to establish clear and consistent guidelines for communication. For example, a team could establish a guideline that team members should check their email or instant messages at least twice a day and should respond to questions or concerns within 24 hours. This helps to ensure that everyone is aware of what is expected of them and that everyone is able to communicate effectively with each other.

Open communication is also an important aspect of keeping communication transparent. For example, a team could establish a policy that encourages team members to share their ideas, thoughts, and concerns openly. This can help to identify and resolve issues quickly and effectively, and it can also help to improve the overall quality of the software.

Finally, it is essential to establish clear roles and responsibilities for each team member. For example, a team could assign a project manager to coordinate the work of the team and handle communication. This helps to ensure that everyone knows who is responsible for what, and that everyone knows who to go to with questions or concerns.

In conclusion, keeping communication transparent is an essential aspect of software development. By establishing clear and consistent channels of communication, clear roles and responsibilities, clear and consistent guidelines, regular team meetings and open communication, and using project management tools, teams can work efficiently and effectively towards a common goal. This can help to improve the overall quality of the software, and the speed of the development process. Clear and transparent communication allows for better collaboration, problem-solving, and understanding among team members. It is important for the team lead or manager to keep an eye out for any communication breakdown and address them promptly to keep the project on.

Saving time with headless execution

Headless execution is a technique that allows teams to run automated tests without the need for a physical or visual user interface. This can save time and resources, as teams no longer need to manually interact with the software to run tests. In this section, we will discuss the benefits of headless

execution, how to implement headless execution, and best practices for headless execution.

This approach offers several benefits that can lead to time savings and increased efficiency:

- Saves time and resources by running tests without the need for a physical or visual user interface.
- Improves consistency of test results by ensuring tests are run in the same manner across different environments.
- Allows for running tests in parallel, reducing the overall time spent on testing.
- Requires the use of headless browsers such as Headless Chrome, PhantomJS, or HtmlUnit.
- Integrates well with the overall development process, including the CI/CD pipeline.
- Adhering to best practices such as using test-driven development, creating repeatable and reliable tests, and aligning test strategy with overall development strategy.
- Use of test reporting and visualization tools to present results in an easy-to-understand format.
- Use of monitoring and alerting systems to notify teams of any anomalies or critical incidents.
- Training of the team members on the tools and processes used in headless execution.
- Can be used in conjunction with other testing techniques, such as continuous testing, to further improve the efficiency and effectiveness of the testing process.

Headless execution allows you to perform tasks in the background without displaying the graphical user interface. Here is how it can save time in these scenarios:

- **Web scraping:**

- **E-commerce price monitoring:** Suppose you want to monitor the prices of products on various e-commerce websites to find the best deals. You can use headless web scraping to extract price information from multiple websites simultaneously without having to open a web browser. This saves time as you can collect data from different sources in the background, 24/7.
 - **Content aggregation:** Content aggregators and news websites use headless execution to gather news articles, blog posts, or other content from various sources. They can scrape and update their websites with fresh content automatically, without manual intervention.
- **Automated testing:**
 - **Continuous integration (CI):** In software development, headless testing is often used in CI/CD pipelines. Automated tests can be executed in a headless mode, running in the background without a graphical interface. This saves time in the development process by quickly identifying and reporting issues.
 - **Browser compatibility testing:** Testing web applications across different browsers and versions can be time-consuming. Headless testing allows you to run automated tests on various browser configurations simultaneously, significantly reducing the time needed to ensure cross-browser compatibility.
 - **Data extraction and transformation:**
 - **Data extraction and transformation** processes for analytics or data warehousing can be automated using headless execution. This can save time by allowing you to extract, process, and load data in the background without the need for manual intervention.

One of the main benefits of headless execution is that it saves time and resources. By running tests without the need for a physical or visual user interface, teams no longer need to manually interact with the software to run tests. This can save time and resources, as teams can run tests more quickly and efficiently.

Another benefit of headless execution is that it improves the consistency of test results. By running tests without the need for a physical or visual user interface, teams can ensure that test results are consistent across different environments. This practice can enhance the overall software quality by enabling teams to readily pinpoint and address issues.

In order to implement headless execution, teams will need to invest in the right tools and resources. This can include headless browsers such as Headless Chrome, PhantomJS, or HtmlUnit, as well as resources like test automation engineers or test automation frameworks. Additionally, teams will need to invest in training and education to learn how to use these tools and resources effectively.

Once teams have invested in the right tools and resources, they can start to implement headless execution by creating automated test scripts and setting up test environments. Teams should focus on automating test cases that are repeatable, reliable and easy to maintain, and the test cases cover different scenarios and test for different types of defects. It is also important to integrate headless execution into the overall development process. This can include integrating automated tests into the CI/CD pipeline and scheduling automated tests to run at specific intervals.

In addition to implementing headless execution, teams should also adhere to best practices for headless execution. This can include using a test-driven development approach, which involves writing automated tests before writing code.

Additionally, teams should focus on creating tests that are repeatable, reliable, and easy to maintain. They should also focus on creating a headless execution strategy that aligns with their overall development strategy. This can include prioritizing certain areas of the application for testing, as well as identifying key metrics for measuring the success of their headless execution efforts.

For example, Headless Chrome is a headless browser that allows teams to run tests in Chrome without the need for a visual user interface. This can be useful for teams that need to test the functionality of a web application in Chrome. Similarly, PhantomJS is a headless browser that allows teams to run tests in a web kit environment without the need for a visual user

interface. This can be useful for teams that need to test the functionality of a web application in a web kit environment.

It is important to use test reporting and visualization tools to present the results of the tests in an easy-to-understand format. This allows teams to quickly identify issues and track progress over time. For example, teams can use tools like JUnit or TestNG to generate test reports, which provide a detailed breakdown of the test results. These tools also allow teams to generate test reports in different formats, such as HTML or XML.

Another ideal practice is to use monitoring and alerting systems to notify teams of any anomalies or critical incidents. For example, teams can use tools like Nagios or Zabbix to monitor the performance of their headless execution process and receive alerts if any issues are identified.

Finally, it is important to train your team members on the tools and processes used in headless execution. This will ensure that they can effectively implement and maintain the process and troubleshoot any issues that may arise.

In conclusion, headless execution is a method that enables teams to execute automated tests without requiring a tangible or visible user interface. This can save time and resources, as teams no longer need to manually interact with the software.

Exploring multi-layer tests

Multi-layer testing is a technique that allows teams to test different layers of their software in isolation. This can improve the overall quality of the software and reduce the time it takes to test the software. In this section, we will discuss the benefits of multi-layer testing, how to implement multi-layer testing, and best practices for multi-layer testing.

The following are the benefits of multi-layer testing:

- Improves the overall quality of the software by testing different layers in isolation.
- Allows for more thorough testing of specific parts of the software.
- Reduces overall time spent on testing by focusing on individual layers.

- Requires the use of test automation tools such as Selenium, Appium, or JUnit.
- Integrates well with the overall development process, including the CI/CD pipeline.
- Adhering to best practices such as using test-driven development, creating repeatable and reliable tests, and aligning test strategy with overall development strategy.
- Use of test reporting and visualization tools to present results in an easy-to-understand format.
- Use of monitoring and alerting systems to notify teams of any anomalies or critical incidents.
- Training of the team members on the tools and processes used in multi-layer testing.
- Can be used in conjunction with other testing techniques, such as headless execution, to further improve the efficiency and effectiveness of the testing process.

One of the main benefits of multi-layer testing is that it improves the overall quality of the software. By testing different layers of the software in isolation, teams can more easily identify and resolve issues. This could lead to a decrease in the quantity of bugs that reach the production stage, ultimately resulting in time and cost savings.

Another benefit of multi-layer testing is that it allows teams to test specific parts of the software more thoroughly. This can help to improve the overall quality of the software, as teams can more easily identify and resolve issues.

A third benefit of multi-layer testing is that it allows teams to test the software quickly. By testing different layers of the software in isolation, teams can run tests without the need to test the entire software at once. This can help to reduce the overall time it takes to test the software, which can ultimately save time and money.

In order to implement multi-layer testing, teams will need to invest in the right tools and resources. This can include tools like Selenium, Appium, or JUnit, as well as resources like test automation engineers or test automation

frameworks. Additionally, teams will be required to allocate resources for training and education to ensure proficient utilization of these tools and resources.

Once teams have invested in the right tools and resources, they can start to implement multi-layer testing by creating automated test scripts and setting up test environments. Teams should focus on automating test cases that are repeatable, reliable, and easy to maintain. They should also focus on automating test cases that cover different scenarios and test for different types of defects.

Teams should also focus on integrating multi-layer testing into their overall development process. This can include integrating automated tests into the CI/CD pipeline and scheduling automated tests to run at specific intervals.

For example, a team can create automated tests for the **User Interface (UI)** layer, service layer, and database layer in isolation. This way, if a defect is found in the UI layer, the team can quickly identify that it is an issue with the UI layer and not the service or database layer, making the debugging process more efficient.

A team can create automated tests for different functionalities of the software, such as login, registration, and payment gateway in isolation. If a defect is found in the login functionality, the team can quickly identify that it is an issue with the login functionality and not the registration or payment gateway functionality.

In addition to implementing multi-layer testing, teams should also adhere to best practices for multi-layer testing. This can include using a test-driven development approach, which involves writing automated tests before writing code. Additionally, teams should focus on creating tests that are repeatable, reliable, and easy to maintain.

Teams should also focus on creating a multi-layer testing strategy that aligns with their overall development strategy. This can include prioritizing certain areas of the application for testing, as well as identifying key metrics for measuring the success of their multi-layer testing efforts.

Another best practice for multi-layer testing is to use a CI/CD pipeline. This allows teams to automatically run tests whenever code is committed to the

repository, and it helps to identify issues early on in the development process.

In real-world scenarios, multi-layer testing is often used in applications that have a complex architecture such as web applications, mobile applications, and microservices-based applications.

For example, in a web application, teams can use multi-layer testing to test the UI layer, service layer, and database layer in isolation. They can create automated tests for the UI layer to test the functionality and appearance of the web pages, test the service layer to ensure the communication between the UI and the database is working correctly, and test the database layer to ensure data is being stored and retrieved correctly.

In a mobile application, teams can use multi-layer testing to test the UI layer, **Application Programming Interface (API)** layer, and database layer in isolation. They can create automated tests for the UI layer to test the functionality and appearance of the mobile app, test the API layer to ensure the communication between the mobile app and the database is working correctly and test the database layer to ensure data is being stored and retrieved correctly.

In a microservices-based application, teams can use multi-layer testing to test each microservice in isolation. They can create automated tests for each microservice to ensure that each microservice is functioning correctly, and that the communication between microservices is working correctly.

In all these scenarios, teams can use test automation tools such as Selenium, Appium, or JUnit to create automated test scripts for different layers and run them on different test environments. They can also use test reporting and visualization tools to present the results in an easy-to-understand format and use monitoring and alerting systems to notify teams of any anomalies or critical incidents.

It is important to note that multi-layer testing can be a gradual process, and teams may start with automating a few tests and gradually increase the number of automated tests as they become more comfortable with the process.

Finally, teams should also focus on maintaining and updating their multi-layer testing over time. This can include regularly reviewing and updating

test scripts, as well as adding new tests as needed. Additionally, teams should focus on automating regression tests, which help to ensure that changes in the software do not introduce new bugs.

In conclusion, multi-layer testing is a technique that allows teams to test different layers of their software in isolation. This can improve the reliability, stability, and overall quality of the software by identifying and addressing issues specific to each layer before they escalate into more complex problems during integration.

Integrating performance testing into delivery cycle

Performance testing is an important aspect of software development and should be integrated into the delivery cycle to ensure that the system is able to handle the expected load and usage patterns.

One way to integrate performance testing into the delivery cycle is to include it as part of the CI/CD process. This means that performance tests are automatically run whenever new code is committed to the repository. This helps to catch performance issues early on and makes it easier to identify and fix the cause of any problems.

Another way to integrate performance testing into the delivery cycle is to include it as part of the acceptance criteria for new features or changes. This means that the performance of the system is tested before any new feature or change is complete. This helps to ensure that the system remains performant as new features are added and changes are made.

There are a variety of performance testing tools available, such as Apache JMeter, Gatling, and LoadRunner, which can be used to automate performance tests. These tools allow you to simulate different types of loads and usage patterns, such as concurrent users, high traffic, and peak usage.

It is also important to establish clear performance goals and metrics for the system. These goals and metrics should be based on the expected usage patterns and should be used to measure the performance of the system during testing.

Overall, integrating performance testing into the delivery cycle is a crucial step in ensuring that the system can handle the expected load and usage

patterns and that any performance issues are identified and resolved as quickly as possible.

Here are some more details with examples of how to integrate performance testing into the delivery cycle:

- **Define performance test cases:** Define performance test cases that represent the expected usage of the system. For example, for an e-commerce website, the test cases could include the number of concurrent users, the number of items in the shopping cart, and the number of searches per second.
- **Automate performance tests:** Automate the performance test cases using a performance testing tool, such as Apache JMeter or Gatling. This allows you to run the tests quickly and easily and makes it easier to identify and fix any performance issues.
- **Integrate with CI/CD:** Integrate the automated performance tests into the CI/CD pipeline, so that the tests are run automatically whenever new code is committed to the repository. This allows you to catch performance issues early on and makes it easier to identify and fix the cause of any problems.
- **Establish performance metrics:** Establish clear performance metrics, such as response time, throughput, and error rate, that can be used to measure the performance of the system. These metrics should be based on the expected usage patterns and should be used to evaluate the performance of the system during testing.
- **Monitor performance in production:** Monitor the performance of the system in production using tools such as New Relic or AppDynamics. This allows you to track the performance of the system over time and identify any trends or patterns that may indicate a performance issue.
- **Include performance testing in acceptance criteria:** Include performance testing as part of the acceptance criteria for new features or changes. This means that the performance of the system is tested before any new feature or change is complete. This helps to ensure

that the system remains performant as new features are added and changes are made.

For example, if your system is an e-commerce website, you would want to make sure that the system can handle a high number of concurrent users during peak shopping times, such as during the holiday season. You would also want to make sure that the system can handle a high number of searches per second and that the response time for search results is fast.

Here are some benefits of performance testing:

- **Identify bottlenecks and scalability issues early on:** Performance testing helps to identify bottlenecks and scalability issues before the system is deployed to production. This makes it easier to fix the problems and improves the overall performance of the system.
- **Improve user experience:** Performance testing helps to ensure that the system can handle the expected load and usage patterns, which improves the user experience.
- **Increase system stability:** By identifying and fixing bottlenecks and scalability issues, performance testing increases the stability of the system.
- **Improve efficiency:** Performance testing helps to identify and optimize resource-intensive processes, which improves the efficiency of the system.
- **Reduce costs:** By identifying and fixing bottlenecks and scalability issues early on, performance testing can help reduce the costs associated with maintaining and scaling the system.
- **Improve customer satisfaction:** By ensuring that the system is performant, performance testing improves customer satisfaction.
- **Meet Service Level Agreements (SLAs):** Performance testing helps to ensure that the system meets SLAs by measuring and verifying the performance of the system.
- **Facilitates load testing, stress testing, and endurance testing:** Performance testing facilitates load testing, stress testing and

endurance testing, which are essential for identifying the system's breaking point.

- **Improve the quality of the system:** Performance testing helps to identify and fix performance issues, which improves the overall quality of the system.
- **Better decision-making:** Performance testing provides valuable data and insights, which can be used to make informed decisions about the system and its performance.

By integrating performance testing into the delivery cycle and automating the tests, you can quickly identify and fix any performance issues, ensuring that the system can handle the expected load and usage patterns, as well as, provide a better experience for the end-users.

A real-world scenario for performance testing might be a company that has an e-commerce website that they want to test the performance of.

Here is an example of how they might go about performing performance testing:

- **Define performance goals and metrics:** The company defines performance goals and metrics for the e-commerce website, such as the number of concurrent users, the number of items in the shopping cart, and the number of searches per second.
- **Test in a realistic environment:** The company sets up a test environment that is as close as possible to the production environment. This includes using the same hardware, software, and network configurations.
- **Test with realistic data:** The company uses realistic data when testing the e-commerce website, such as representative product data and customer data.
- **Test with realistic loads:** The company tests the e-commerce website with loads that are representative of the expected usage patterns. For example, they simulate a peak shopping period with a high number of concurrent users and a high number of searches per second.

- **Automate performance tests:** The company automates the performance tests using a performance testing tool, such as Apache JMeter or Gatling. This allows them to run the tests quickly and easily and also makes it easier to identify and fix any performance issues.
- **Monitor performance:** The company monitors the performance of the e-commerce website in production using tools such as New Relic or AppDynamics. This allows them to track the performance of the system over time and identify any trends or patterns that may indicate a performance issue.
- **Continuous monitoring:** The company continuously monitors the e-commerce website's performance and uses the data and insights obtained from performance testing to continuously improve the system's performance.
- **Use a combination of testing types:** The company uses a combination of testing types, such as load testing, stress testing, and endurance testing, to get a more complete picture of the system's performance.
- **Test for security:** The company includes security testing as part of the performance testing to ensure that the increased load on the system does not reveal any security vulnerabilities.
- **Test with real users:** The company includes real users in performance testing to get a more accurate picture of how the e-commerce website will perform in production.

By following these steps, the company can ensure that the e-commerce website is thoroughly tested and that any performance issues are identified and resolved as quickly as possible. This will help to improve the performance of the website and provide a better experience for the end-users.

Building robust continuous testing with tools

There are several tools available that can be used to build robust continuous testing environments. Here are a few popular ones:

- **Jenkins:** Jenkins is an open-source automation server that is widely used for CI/CD. It can be used to automate the building, testing, and deployment of software. Jenkins can be used to automate the running of performance tests and also provides a wide range of plugins that can be used to integrate with other tools.
- **Selenium:** Selenium is a widely used open-source automation tool for web browsers. It can be used to automate the testing of web applications, including performance testing. Selenium can be integrated with other testing frameworks, such as JUnit or TestNG, and can also be used in combination with other performance testing tools, such as Apache JMeter.
- **JUnit:** JUnit is a unit testing framework for Java that is widely used in the Java community. It can be integrated with other tools, such as Selenium, to perform continuous testing.
- **TestNG:** TestNG is a testing framework for Java that is similar to JUnit, but with additional features such as support for data-driven testing, test groups, and parameterized tests. It also allows for parallel test execution and can be integrated with other tools to perform continuous testing.
- **Sauce Labs:** Sauce Labs is a cloud-based platform for testing web and mobile applications. It provides a range of tools for automating tests, including Selenium and Appium, and can also be integrated with other tools to perform continuous testing.
- **TestComplete:** TestComplete is a functional and performance testing tool that allows to automate testing of web, desktop, and mobile applications. It provides support for a wide range of technologies and allows for easy integration with other tools.
- **LoadRunner:** LoadRunner is a performance testing tool that simulates a high number of users accessing the system simultaneously. It allows us to measure the system's performance, identify bottlenecks and optimize the system's resources.

These are just a few examples of the many tools that are available for building robust continuous testing environments. The best tool for your

specific use case will depend on the technology stack of your application and the specific requirements of your testing. Setting up a robust continuous testing environment involves several steps, here are some general guidelines:

- **Define your testing strategy:** Define your testing strategy, including the types of tests that will be performed (unit tests, functional tests, performance tests, etc.), the tools that will be used, and the testing environment.
- **Automate your tests:** Automate as many tests as possible, including unit tests, functional tests, and performance tests. This will make it easier to run the tests quickly, and also makes it easier to identify and fix any issues.
- **Integrate with your CI/CD pipeline:** Integrate your automated tests with your CI/CD pipeline, so that the tests are run automatically whenever new code is committed to the repository. This allows you to catch issues early on and makes it easier to identify and fix the cause of any problems.
- **Use a test management tool:** Use a test management tool, such as TestRail, to manage your tests and track the results of your tests.
- **Monitor performance in production:** Monitor the performance of the system in production, using tools such as New Relic or AppDynamics. This allows you to track the performance of the system over time and identify any trends or patterns that may indicate a performance issue.
- **Continuously monitor and improve:** Continuously monitor the system's performance, and use the data and insights obtained from continuous testing to continuously improve the system's performance.
- **Test on different environments:** Test on different environments, such as different operating systems, different browsers, and different devices, to ensure that the system works correctly in different configurations.
- **Include security testing:** Include security testing as part of the continuous testing process, to ensure that the system is secure.

- **Continuously review and update:** Continuously review and update your testing strategy, tools, and environments to ensure that they are up to date and appropriate for your current needs.

By following these guidelines, you can set up a robust continuous testing environment that will help to identify and fix any issues quickly, ensure that the system is thoroughly tested, and provide a better experience for the end-users.

Continuous testing

Continuous testing is an essential part of the software development process, as it allows teams to ensure that their software is of high quality and defect free. Continuous testing refers to the practice of continuously running tests against the code throughout the development process, from the initial stages of development to the final stages of deployment. In this section, we will discuss the benefits of continuous testing, how to implement continuous testing, and best practices for continuous testing.

One of the main benefits of continuous testing is that it helps to improve the overall quality of the software. By continuously running tests against the code, teams can catch and fix defects early in the development process.

Another benefit of continuous testing is that it helps to increase the speed of the development process. By continuously running tests against the code, teams can get feedback on their code more quickly, thereby reducing the time.

A third benefit of continuous testing is that it helps to improve collaboration and coordination among teams. By continuously running tests against the code, teams can stay up to date on the progress of the project, which helps improve communication and collaboration among teams.

In order to implement continuous testing, teams will need to invest in the right tools and resources. This can include tools like Selenium, Appium, or JUnit, as well as resources like test automation engineers or test automation frameworks. This encompasses tools such as Selenium, Appium, or JUnit, alongside assets like test automation engineers or test automation frameworks. Moreover, teams should allocate resources for training and education to ensure proficient utilization of these tools and resources.

Once teams have invested in the right tools and resources, they can start to implement continuous testing by creating automated test scripts and setting up test environments. Teams should prioritize automating test cases that are consistent, dependable, and require minimal maintenance. It's also important to concentrate on automating test cases that encompass diverse scenarios and encompass various types of defects.

Teams should also focus on integrating continuous testing into their overall development process. This can include integrating automated tests into the CI/CD pipeline and scheduling automated tests to run at specific intervals. In addition to implementing continuous testing, teams should also adhere to best practices for continuous testing. This can include using a test-driven development approach, which involves writing automated tests before writing code. Furthermore, it's important for teams to emphasize the development of tests that are consistent, dependable, and straightforward to upkeep.

Teams should also focus on creating a continuous testing strategy that aligns with their overall development strategy. This can include prioritizing certain areas of the application for testing, as well as identifying key metrics for measuring the success of their continuous testing efforts.

Another best practice for continuous testing is to use a CI/CD pipeline. This allows teams to automatically run tests whenever code is committed to the repository, and it helps to identify issues early in the development process.

Finally, teams should also focus on maintaining and updating their continuous testing over time. This can include regularly reviewing and updating test scripts, as well as adding new tests as needed. Additionally, teams should focus on automating regression tests, which help to ensure that changes in the software do not introduce new bugs.

In conclusion, continuous testing is an essential part of the software development process. By continuously running tests against the code, teams can catch and fix defects early in the development process, which can ultimately save time and money. Additionally, continuous testing helps to improve collaboration and coordination among teams, increase the speed of the development process, and improve the overall quality.

Implementing continuous testing involves several steps:

- **Identify the testing requirements:** The first step in implementing continuous testing is to identify the types of tests that need to be run, such as unit tests, integration tests, and acceptance tests.
- **Choose the right tools:** Select the appropriate tools that align with your testing requirements, like Selenium, Appium, JUnit, TestNG, Cucumber and more.
- **Set up test environments:** Set up test environments that mimic the production environment, such as virtual machines, containers, or cloud-based environments.
- **Create automated test scripts:** Create automated test scripts that will be run as part of the continuous testing process. These scripts should be repeatable, reliable, and easy to maintain.
- **Integrate with the development process:** Integrate continuous testing into your overall development process, such as by using a CI/CD pipeline. This allows automated tests to run whenever code is committed to the repository.
- **Monitor and analyze test results:** Monitor and analyze the results of the automated tests. This includes tracking the number of tests run, the number of tests passed, and the number of tests failed.
- **Continuously review and improve:** Continuously review and improve the test automation process by looking at the test coverage, identifying new test scenarios, and updating the test scripts as needed.
- **Implement test reporting and visualization:** Implement test reporting and visualization tools to present the results of the tests in an easy-to-understand format. This allows teams to quickly identify issues and track progress over time.
- **Use monitoring and alerting system:** Implement monitoring and alerting systems to notify teams of any anomalies or critical incidents.
- **Train your team:** Ensure that your team members are trained on the tools and processes used in continuous testing, so that they can effectively implement and maintain the process.

It's important to note that implementing continuous testing is a gradual process. Teams may start by automating a few tests and gradually increase the number of automated tests as they become more comfortable with the process.

How test management tools work

Test management tools and CI/CD tools serve different purposes in the software development and testing process:

Test management tools:

- Test management tools are designed to manage and streamline the testing process.
- They help teams create and organize test cases, test plans, and test scripts.
- Test management tools facilitate test execution, capture test results, and track defects.
- These tools often offer reporting and analytics capabilities to provide insights into testing progress and quality.
- They integrate with test automation tools, enabling the execution of automated test scripts.
- Test management tools are primarily focused on the manual and automated testing aspects of the software development lifecycle.

CI/CD tools:

- CI/CD tools are centered around the automation of the software build, test, and deployment process.
- CI tools automate the building and testing of code changes as developers commit them to a version control system. This ensures that new code doesn't introduce errors or conflicts with existing code.
- **Continuous Deployment/Delivery (CD)** tools automate the process of packaging and deploying code to production environments.
- CI/CD tools enable teams to achieve rapid and reliable software delivery, reducing the time and effort required for manual tasks.

- These tools focus on the automation of code integration, build, and deployment phases in the software development pipeline.

In summary, while both test management and CI/CD tools play essential roles in the software development process, they have different primary functions:

- Test management tools concentrate on managing and organizing test cases and tracking test execution, helping teams ensure software quality through thorough testing.
- CI/CD tools are dedicated to automating the integration, building, and deployment of code, leading to faster and more reliable software development and release processes.

Step-by-step build robust and fully automatic continuous testing

Here is a step-by-step guide on how to build a robust and fully automatic continuous testing environment:

1. **Define your testing strategy:** Define your testing strategy, including the types of tests that will be performed (unit tests, functional tests, performance tests, etc.), the tools that will be used, and the testing environment.
2. **Identify and select the appropriate testing tools:** Identify and select the appropriate testing tools, such as Selenium for functional testing, Apache JMeter for performance testing, and TestNG or JUnit for unit testing.
3. **Create and automate your test cases:** Create and automate your test cases, including unit tests, functional tests, and performance tests. This can be done using the testing tools identified in step 2.
4. **Integrate with your CI/CD pipeline:** Integrate your automated tests with your CI/CD pipeline, such as Jenkins, so that the tests are run automatically whenever new code is committed to the repository.
5. **Monitor performance in production:** Monitor the performance of the system in production, using tools such as New Relic or AppDynamics. This allows you to track the performance of the

system over time and identify any trends or patterns that may indicate a performance issue.

6. **Continuously monitor and improve:** Monitor the system's performance, and use the data and insights obtained from continuous testing to continuously improve the system's performance.
7. **Test with real users:** Include real users in testing, to get a more accurate picture of how the system will perform in production.
8. **Continuously review and update:** Continuously review and update your testing strategy, tools, and environments to ensure that they are up-to-date and appropriate for your current needs.

By following these steps, you can build a robust and fully automatic continuous testing environment that will help to identify and fix any issues quickly while ensuring that the system is thoroughly tested and provides a better experience for the end-users.

We will leverage the widely renowned and prevalent integration tool Jenkins to establish a framework for continuous testing. Here is a step-by-step guide on how to set up a fully automatic continuous testing environment with Jenkins:

1. **Install Jenkins:** Install Jenkins on a suitable machine, either on-premises or in the cloud.
2. **Create a Jenkins job:** Create a Jenkins job that will run your automated tests. This job should be configured to run whenever new code is committed to the repository.
3. **Integrate with your version control system:** Integrate Jenkins with your version control system, such as Git, so that the Jenkins job is triggered whenever new code is committed to the repository.
4. **Configure the Jenkins job to run your tests:** Configure the Jenkins job to run your automated tests, including unit tests, functional tests, and performance tests.
5. **Install and configure the necessary plugins:** Install and configure any necessary Jenkins plugins, such as the Selenium plugin, to run your automated tests.

6. **Use a test management tool:** Use a test management tool, such as TestRail, to manage your tests and track the results of your tests.
7. **Monitor performance in production:** Monitor the performance of the system in production using tools such as New Relic or AppDynamics. This allows you to track the performance of the system over time and identify any trends or patterns that may indicate a performance issue.
8. **Continuously monitor and improve:** Continuously monitor the system's performance and use the data and insights obtained from continuous testing to continuously improve the system's performance.
9. **Conduct testing using actual users:** Incorporate actual users into the testing process to obtain a more precise representation of the system's performance in a real production environment.
10. **Incorporate testing for security:** Integrate security testing within the continuous testing process to verify the system's security.
11. **Continuously review and update:** Continuously review and update your testing strategy, tools, and environments to ensure that they are up-to-date and appropriate for your current needs.

By following these steps, you can set up a fully automatic continuous testing environment with Jenkins that will help to identify and fix any issues quickly, ensure that the system is thoroughly tested, and provide a better experience for the end-users.

Following is a comprehensive walkthrough detailing the configuration process for setting up an automatically triggered job within Jenkins:

1. **Install the necessary plugins:** Install any necessary Jenkins plugins, such as the *Jenkins Pipeline plugin* or the *Git plugin*.
2. **Create a new Jenkins job:** Navigate to the Jenkins homepage and click on **New Item** to create a new Jenkins job.
3. **Configure the job's source code management:** In the **Source Code Management** section, configure the job to pull code from your version control system, such as Git.
4. **Configure the job's build triggers:** In the **Build Triggers** section, configure the job to run automatically by selecting the options that

you want to trigger the build, for example *Build periodically* or *GitHub hook trigger for GITScm polling*.

5. **Configure the job's build environment:** In the **Build Environment** section, configure the job to run on the appropriate environment, such as a specific slave node or cloud-based environment.
6. **Configure the job's build steps:** In the **Build** section, add build steps to run your automated tests. This can include commands to run your unit tests, functional tests, and performance tests.
7. **Save the job:** Click on **Save** to save the job.
8. **Test the trigger:** Once the job is configured and saved, you can test the trigger by committing a change in your version control system and checking whether the job is triggered automatically.
9. **Monitor the job's progress:** Monitor the job's progress by viewing the build logs and watching for any errors or failures.

What happens if any of the jobs within the Jenkins pipeline fail? Here is a step-by-step guide on how to check for failed Jenkins jobs:

1. **Navigate to the Jenkins homepage:** Open your web browser and navigate to the URL of your Jenkins server.
2. **View the build history:** Click on the job that you want to check for failed builds, then click on the **Build History** link on the left-hand side.
3. **Check the build status:** Look at the build status of the job, the failed jobs will be marked in red.
4. **Check the build logs:** Click on the failed build to view the build logs. This will show detailed information about what caused the build to fail, such as error messages or stack traces.
5. **Check the Test Results:** You can also check the test results of the failed build by clicking on the **Test Results** link on the left-hand side. This will show you which test cases have failed.
6. **Check the Console Output:** You can also check the console output of the failed build by clicking on the **Console Output** link on the left-hand side. This will show you the log of the execution and the output of the commands that were run.

7. **Check the Artifacts:** You can also check the artifacts produced by the failed build by clicking on the **Artifacts** link on the left-hand side. This will show you the files produced by the build and can be useful to debug.

The following figure shows how to check the Jenkins job history and job logs from console:

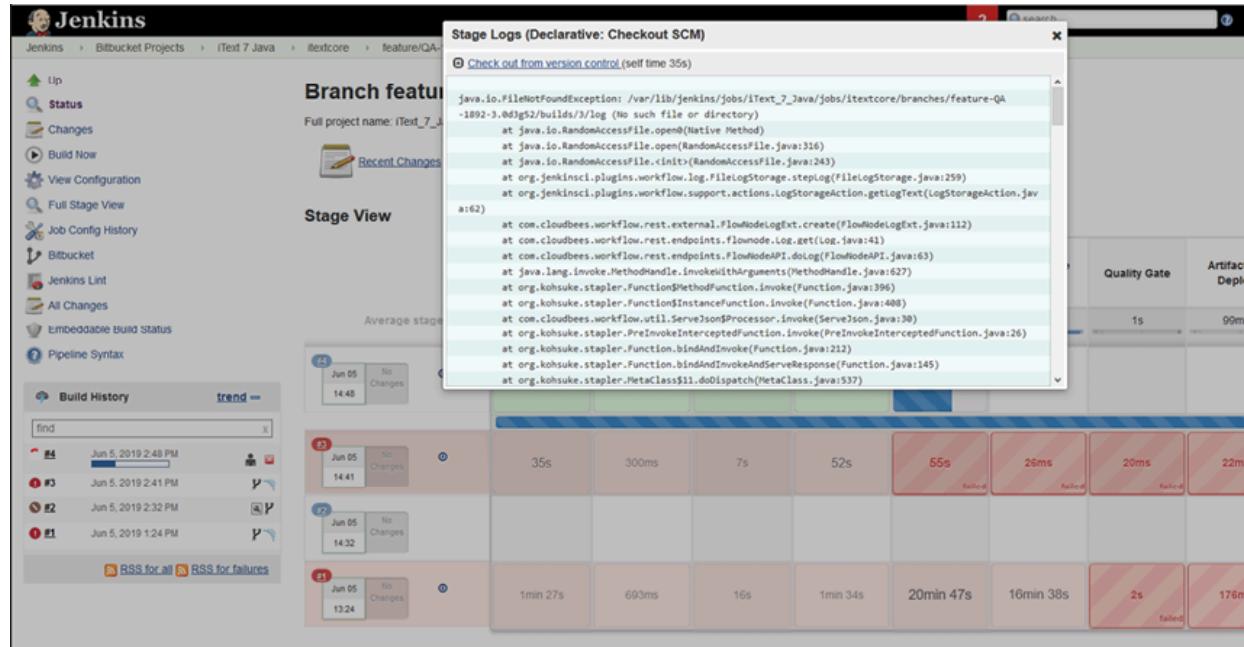


Figure: 7.1: Jenkins jobs history

By following these steps, you can check for failed Jenkins jobs and troubleshoot the problem. This will help you to identify and fix issues quickly, ensure that the system is thoroughly tested.

Here is a step-by-step guide on how to set up notifications on Slack in Jenkins:

1. **Install the Slack plugin:** Install the **Slack Plugin** in Jenkins, by navigating to the **Manage Jenkins | Manage Plugins | Available** and searching for the **Slack Plugin**.
2. **Create a new Jenkins job:** Navigate to the Jenkins homepage and click on **New Item** to create a new Jenkins job.

3. **Configure the Jenkins job:** Add the build steps and triggers to run the job automatically.
4. **Create a new Slack Workspace:** Create a new Slack workspace for your Jenkins notifications or use an existing one if you already have one.
5. **Create a new Incoming Webhook:** In your Slack workspace, create a new incoming webhook by navigating to [Apps and Integrations | Incoming Webhooks](#) | [Create a new incoming webhook](#).
6. **Copy the Webhook URL:** Once the webhook is created, copy the webhook URL and keep it safe.
7. **Configure the Slack plugin:** In Jenkins, navigate to the [Manage Jenkins](#) | [Configure System](#) | [Slack](#) and configure the plugin by providing the webhook URL, the channel, and the Jenkins build job.
8. **Save the configuration:** Click on **Save** to save the configuration.
9. **Test the notification:** Once the job is configured and saved, you can test the notification by running the job, and check whether the notifications are sent to the slack channel.

The following figure shows how to integrate Jenkins with Slack notifications:

The screenshot shows the configuration page for integrating Jenkins with Slack. At the top, the breadcrumb navigation reads "Browse apps > Jenkins CI > New configuration". Below this, the Jenkins CI logo is displayed with the text "An open source continuous integration server." A descriptive paragraph states: "Jenkins CI is a customizable continuous integration server with over 600 plugins, allowing you to configure it to meet your needs." It also notes: "This integration will post build notifications to a channel in Slack." A section titled "Post to Channel" contains instructions: "Start by choosing a channel where Jenkins notifications will be posted." A dropdown menu is shown with the value "# slack-test-channel", and a link "or create a new channel" is visible. At the bottom of the form is a green button labeled "Add Jenkins CI integration".

Figure: 7.2: Integrate Slack notification

By following these steps, you can set up notifications on Slack in Jenkins. This will help you monitor the status of your Jenkins jobs and receive notifications when a build fails or completes.

Here is a step-by-step guide on how to set up email notifications in Jenkins:

1. **Configure the Jenkins email settings:** Navigate to the **Manage Jenkins | Configure System** and scroll down to the **Email Notification** section.
2. **Add the SMTP server information:** Provide the SMTP server information, including the server hostname, port number, username, and password.
3. **Add the recipient email addresses:** Provide the email addresses of the recipients who should receive the notifications. You can also add a recipient list for the different types of notifications such as success, failure, or unstable build notifications.
4. **Test the email configuration:** Test the email configuration by clicking the **Test Configuration** button. This will send a test email to the recipient email addresses provided.
5. **Configure the Jenkins job:** In your Jenkins job, navigate to the **Post-build Actions** section, and add an **Editable Email Notification** action.
6. **Configure the recipient list:** Select the recipient list that you want to use for this job and configure the triggers for when an email should be sent.
7. **Save the job:** Click on **Save** to save the job.
8. **Test the notification:** Once the job is configured and saved, you can test the notification by running the job, and check whether the email notifications are sent to the recipients.

By following these steps, you can set up email notifications in Jenkins. This will help you monitor the status of your Jenkins jobs and receive notifications when a build fails or completes.

Conclusion

In conclusion, automating your testing process is a crucial step in ensuring that your software is thoroughly tested and free of defects. Automating tests allows you to run tests quickly and easily, making it easier to identify and fix any issues that may arise. By integrating automated tests into your CI/CD pipeline, you can catch issues early on, which helps to minimize the impact of any problems that may arise. Additionally, by using a test management tool, you can easily track the results of your tests and monitor the performance of your system over time. Furthermore, by setting up a robust continuous testing environment, you can ensure that your system is thoroughly tested, and that any issues are identified and fixed quickly, which will help to provide a better experience for your end-users.

In the next chapter, we will learn continuous monitoring, which identifies and timely responds to violations of regulatory requirements and potential vulnerabilities in software systems or applications. This practice involves using automated tools, continuous monitoring, and thorough assessments to swiftly pinpoint and address security gaps, ensuring that the software remains compliant with industry standards and well-guarded against potential threats.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.\(bpbonline.com](https://discord(bpbonline.com)



CHAPTER 8

Rapid Detection of Compliance Issues and Security Risks

Introduction

Rapid detection of compliance issues and security risks refers to quickly identifying and assessing potential non-compliance with regulations and security vulnerabilities within an organization's systems, networks, and data. The goal is to detect these issues as soon as possible in order to take appropriate action to resolve them and minimize any negative impact. This can be achieved through various methods, such as real-time monitoring, automated scanning, and incident response processes.

The following figure shows key characteristics of continuous monitoring:

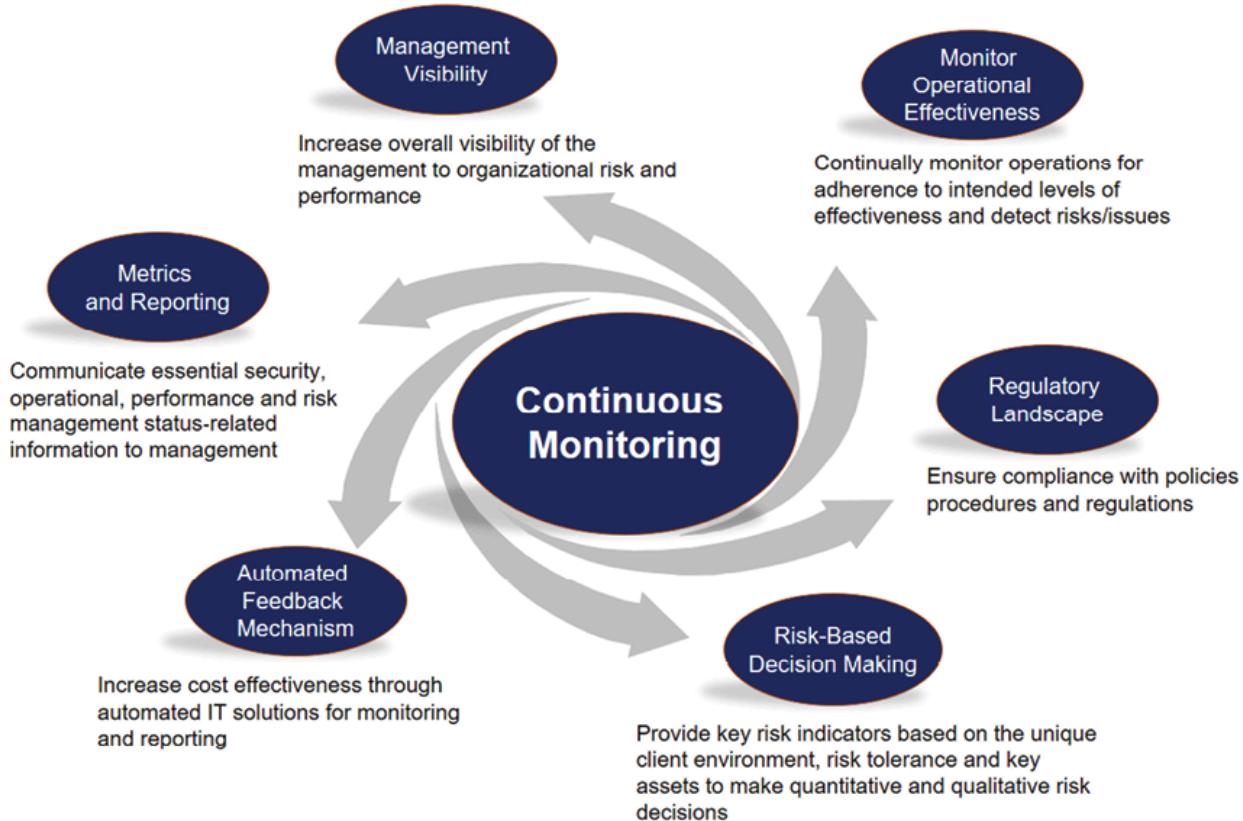


Figure 8.1: Characteristics of continuous monitoring

Structure

In this section, we will learn why continuous monitoring is required in the software development lifecycle. This section explains the benefits, best practices, tools, and how to configure continuous monitoring. In this chapter, we are going to learn the following topics in continuous monitoring:

- Types of continuous monitoring
- Network monitoring
- Infrastructure monitoring
- Configuration monitoring:
- Log monitoring:
- User activity monitoring
- Application Performance Monitoring

- Setup alerts
- Step by step build robust and fully automatic continuous monitoring

Objectives

In this section, we will learn how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks. To minimize the impact of issues and improve the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Types of continuous monitoring

Continuous monitoring can be performed in various ways and can involve the collection and analysis of data from different sources. Some common types of continuous monitoring include:

- **Network monitoring:** Monitoring of network traffic and activities to detect potential security incidents and anomalies.
- **Infrastructure monitoring:** Infrastructure monitoring is to identify potential issues and performance bottlenecks in real-time.
- **Configuration monitoring:** Monitoring of changes in system configurations to ensure they align with established security policies and to detect potential misconfigurations.
- **Vulnerability management:** Regular scans and assessments of systems and applications to identify vulnerabilities that could be exploited by attackers.
- **Log monitoring:** Collection and analysis of log data from systems and applications to detect potential security incidents and to support incident response activities.
- **User activity monitoring:** Monitoring of user activities on systems and applications to detect potential security incidents, such as unauthorized access or suspicious behavior.

- **Application performance monitoring:** Monitoring of application performance to detect potential security incidents and to ensure the performance of critical applications is maintained.

These are some examples of the types of continuous monitoring that organizations may employ as part of their overall security strategy. The specific types of monitoring used will depend on the needs and requirements of the organization.

Continuous monitoring feedback loop:

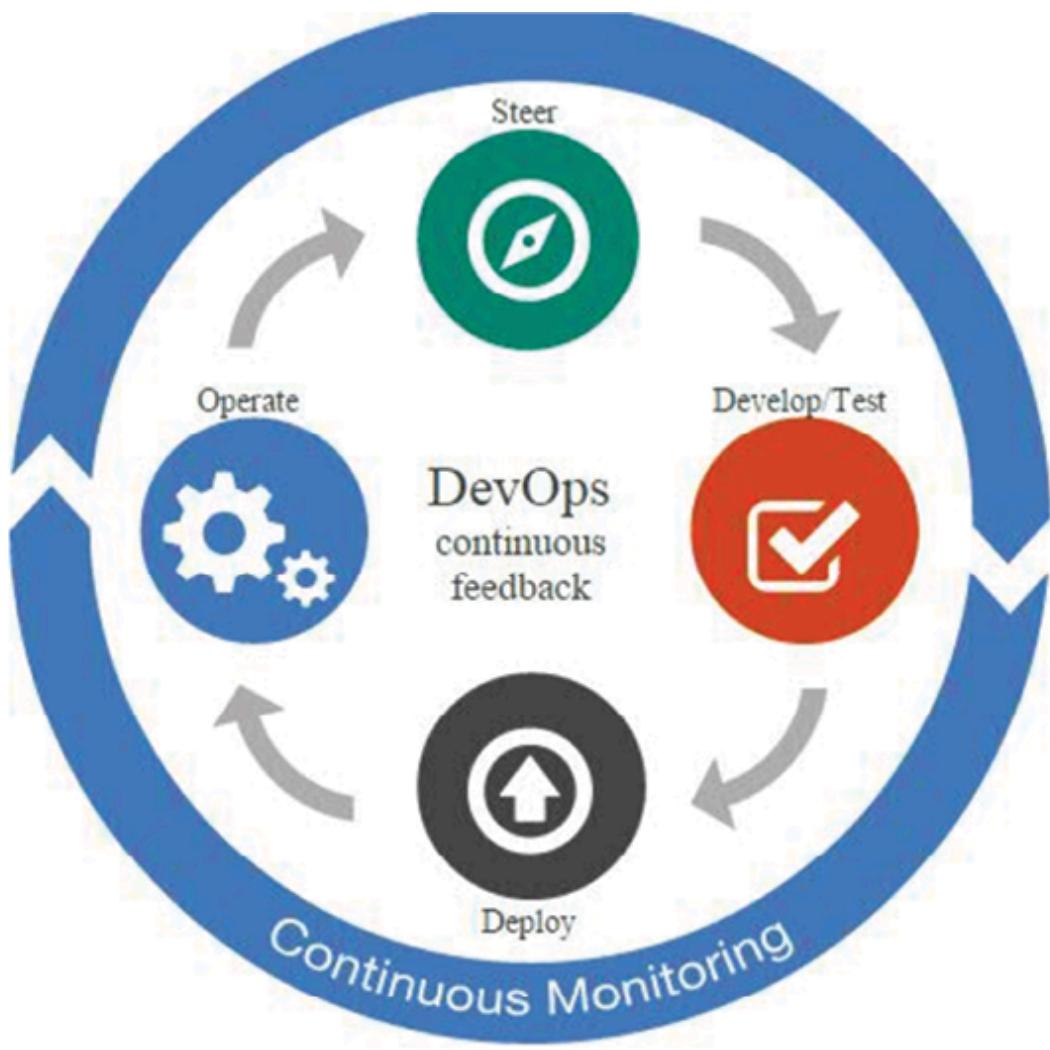


Figure 8.2: Continuous monitoring feedback loop

Network monitoring

Network monitoring is a process of constantly monitoring the activities and performance of a network to ensure that it is operating as expected. The goal is to detect any potential issues or anomalies that could impact the availability and performance of the network and to identify the root cause of any problems that may arise.

Real-time monitoring in network management involves continuous, instantaneous tracking of network performance metrics. It enables immediate alerts to be triggered for critical issues, allowing administrators to respond promptly and minimize potential disruptions. This feature is crucial for maintaining the availability and reliability of network services.

Network monitoring typically involves the use of specialized software and hardware tools to collect data on network performance and utilization and to analyze that data to identify potential issues. This may include monitoring network traffic, device performance, and network infrastructure, among other things. The information gathered through network monitoring can be used to optimize network performance, identify areas for improvement, and ensure that the network is secure and complies with relevant security policies and standards.

Why network monitoring

There are several reasons for using network monitoring, including:

- **Network performance:** Network monitoring helps to ensure that the network is operating at optimal performance levels by detecting potential performance issues and bottlenecks.
- **Availability:** It helps to detect any potential outages or downtime and to quickly identify the root cause, allowing for prompt resolution of issues.
- **Security:** It helps to detect potential security threats and vulnerabilities in the network, such as unauthorized access or malware, and to respond promptly to minimize any negative impact.
- **Compliance:** Network monitoring can be used to ensure that the network complies with the relevant regulations, standards, and security policies.

- **Capacity planning:** Network monitoring can provide valuable insights into network utilization and capacity, allowing organizations to make informed decisions about future network infrastructure investments.
- **Troubleshooting:** Network monitoring can help in identifying the cause of network issues, making it easier to resolve problems quickly.

Overall, network monitoring is a critical component of an organization's IT infrastructure and is essential for ensuring network availability, performance, security, and compliance.

Here is a list of some popular network monitoring tools:

- **Nagios:** An open-source network monitoring tool that provides real-time monitoring and alerting for network devices, systems, and applications.
- **Zabbix:** An open-source network monitoring solution that provides comprehensive monitoring for a variety of network components and applications.
- **SolarWinds Network Performance Monitor (NPM):** A commercial network monitoring tool that provides real-time visibility into network performance and availability.
- **PRTG network monitor:** A commercial network monitoring solution that provides monitoring and reporting for a variety of network devices and applications.
- **ManageEngine OpManager:** A commercial network monitoring tool that provides real-time monitoring and alerting for network devices and applications.
- **WhatsUp Gold:** A commercial network monitoring solution that provides monitoring and reporting for network devices, systems, and applications.
- **Wireshark:** An open-source network protocol analyzer tool that can be used for network troubleshooting and analysis.
- **Spiceworks network monitor:** A free network monitoring tool that provides monitoring and alerting for network devices and applications.

- **Datadog:** A cloud-based network monitoring solution that provides real-time monitoring and reporting for a variety of network components and applications.
- **Cacti:** An open-source network monitoring tool that provides real-time monitoring and reporting for network devices and applications.

These are some popular network monitoring tools available in the market, and the right tool for a particular organization will depend on its specific needs and requirements.

Benefits of network monitoring

There are several benefits to using network monitoring, including:

- **Improved network performance:** Network monitoring helps to ensure that the network is operating at optimal performance levels by detecting potential performance issues and bottlenecks. This allows for prompt resolution of issues, which can help improve network performance.
- **Increased availability:** Network monitoring helps detect any potential outages or downtime and quickly identify the root cause, allowing for prompt resolution of issues. This helps to ensure that the network is always available and minimizes downtime.
- **Enhanced security:** Network monitoring helps detect potential security threats and vulnerabilities in the network, such as unauthorized access or malware, and respond promptly to minimize any negative impact.
- **Compliance:** Network monitoring can be used to ensure that the network complies with relevant regulations, standards, and security policies.
- **Improved capacity planning:** Network monitoring can provide valuable insights into network utilization and capacity, allowing organizations to make informed decisions about future network infrastructure investments.

- **Efficient troubleshooting:** Network monitoring can help in identifying the cause of network issues, making it easier to resolve problems quickly.
- **Better cost management:** Network monitoring can help organizations identify areas for optimization and reduce waste, which can result in cost savings.
- **Capacity management and scalability:** Network monitoring provides insights into current network utilization and helps organizations plan for future growth. This is essential for ensuring that the network infrastructure can handle increased demands and allows for strategic expansion without compromising performance.

Overall, network monitoring is a critical component of an organization's IT infrastructure and is essential for ensuring network availability, performance, security, and compliance. By using network monitoring, organizations can improve their ability to manage and optimize their networks, reduce downtime, improve productivity, and ensure that they follow relevant regulations and standards.

How to use network monitoring

To use network monitoring, the following steps can be followed:

- **Identify network components:** The first step is to identify all the network components that need to be monitored, including servers, routers, switches, firewalls, and other network devices.
- **Choose a network monitoring tool:** The next step is to choose a network monitoring tool that meets the organization's needs and requirements. There are many different network monitoring tools available, both commercial and open source, so it is important to choose one that meets the specific needs of the organization.
- **Install and configure the monitoring tool:** Once a network monitoring tool has been chosen, it should be installed and configured. This typically involves setting up the monitoring agent or software on the network devices to be monitored and configuring the monitoring tool to collect data and generate alerts.

- **Define monitoring and alerting thresholds:** Once the monitoring tool has been installed and configured, monitoring and alerting thresholds should be defined. This involves specifying what conditions should trigger an alert, such as a threshold for network utilization or a change in network status.
- **Monitor network performance:** Once the monitoring tool has been configured, it can be used to monitor network performance in real-time. The monitoring tool will collect data from the network devices and display this information in a dashboard or report, making it easier to identify potential performance issues and respond to them.
- **Respond to alerts:** When an alert is generated, the monitoring tool should be used to quickly identify the root cause of the issue and to respond to it. This might involve troubleshooting the network, making changes to network configurations, or escalating the issue to a higher level of support.
- **Monitor and optimize performance over time:** Network monitoring should be an ongoing process, and the monitoring tool should be used to continually monitor network performance and to make changes to improve performance and availability over time.

By using network monitoring, organizations can improve their ability to manage and optimize their networks, reduce downtime, improve productivity, and ensure that they follow relevant regulations and standards.

Configuring network monitoring

Here are the steps to configure Nagios for network monitoring:

1. **Install Nagios:** The first step is to install Nagios on the server that will act as the monitoring host. Nagios can be installed on a variety of operating systems, including Linux, Unix, and Windows.
2. **Configure the Nagios server:** Once Nagios has been installed, the next step is to configure the Nagios server. This involves modifying the Nagios configuration file to specify the devices and services that will be monitored.

- 3. Install plugins:** Nagios uses plugins to monitor different aspects of the network, such as network performance, service availability, and resource utilization. The plugins should be installed and configured on the Nagios server.
- 4. Define hosts and services:** The next step is to define the hosts and services that will be monitored by Nagios. This involves creating a configuration file for each device or service, specifying the parameters for monitoring, such as hostname, IP address, and monitoring interval.
- 5. Configure alerts:** Nagios can be configured to generate alerts when specific conditions are met, such as when a service goes down or when network utilization exceeds a certain threshold. The alert configuration should be specified in the Nagios configuration file.
- 6. Test the configuration:** Once the configuration has been completed, it should be tested to ensure that Nagios is monitoring the network as expected. This can be done by simulating network conditions, such as a service outage, and verifying that Nagios generates the appropriate alerts.
- 7. Monitor the network:** Once the configuration has been tested and validated, Nagios can be used to monitor the network in real-time. Nagios provides a dashboard and reporting features that make it easy to monitor network performance and availability.

By following these steps, organizations can use Nagios to monitor their network and to ensure that their network is operating at optimal performance levels, with minimal downtime and maximum security.

Infrastructure monitoring

Infrastructure monitoring is the process of continuously monitoring the health, performance, and availability of various components of an IT infrastructure, such as servers, networks, databases, and applications. The goal of infrastructure monitoring is to identify potential issues and performance bottlenecks in real-time, so that the necessary corrective actions can be taken before they cause downtime or affect the user experience.

Infrastructure monitoring includes monitoring various metrics, such as resource utilization (CPU, memory, disk), network performance, availability of services and applications, and the status of hardware components. The monitoring data is collected, analyzed, and visualized in real-time, to provide a comprehensive view of the IT infrastructure and to help IT teams quickly identify and resolve any issues.

Infrastructure monitoring is essential for organizations to ensure the availability and performance of their IT systems and to prevent downtime, which can result in lost revenue and productivity. It also helps organizations to optimize the use of their IT resources, reduce costs, and improve the overall quality of their IT services.

Benefits of infrastructure monitoring

There are several benefits of infrastructure monitoring:

- **Improved availability:** Infrastructure monitoring helps organizations proactively detect and resolve issues that can cause downtime or affect the user experience. By monitoring the health and performance of key components, IT teams can take corrective action before problems occur, ensuring the highest levels of availability for IT services.
- **Increased efficiency:** Infrastructure monitoring provides real-time visibility into the performance and utilization of IT resources, allowing organizations to optimize their use and reduce costs. By monitoring resource utilization, IT teams can identify and resolve performance bottlenecks and improve the efficiency of their IT systems.
- **Better performance:** Infrastructure monitoring helps organizations identify performance issues and resolve them before they affect the user experience. By monitoring key metrics such as network performance, resource utilization, and service availability, IT teams can quickly identify performance issues and take corrective action, improving the overall performance of IT systems.
- **Faster problem resolution:** With infrastructure monitoring, IT teams can quickly identify and resolve problems before they escalate into critical issues. This helps to minimize downtime and reduces the time

it takes to resolve problems, improving the overall efficiency and effectiveness of IT operations.

- **Improved security:** Infrastructure monitoring can also help organizations identify and prevent security threats by monitoring network traffic and identifying suspicious activities. This helps organizations improve the security of their IT systems and to prevent security breaches.
- **Better compliance:** Infrastructure monitoring can help organizations ensure that their IT systems follow industry regulations and standards. By monitoring the configuration and usage of IT systems, IT teams can ensure that their systems are secure, compliant, and in line with best practices.

Overall, infrastructure monitoring is an essential tool for organizations to ensure the availability, performance, and security of their IT systems while reducing downtime and improving the efficiency of IT operations.

Here are some popular infrastructure monitoring tools:

- **Nagios:** An open-source monitoring system that monitors hosts, services, and network devices.
- **Prometheus:** Prometheus is an open-source monitoring and alerting system designed to monitor dynamic and distributed systems.
- **Zabbix:** An open-source monitoring system that provides detailed monitoring of hosts, services, and network devices.
- **SolarWinds:** A commercial network and systems management software that provides real-time monitoring and reporting of network devices, servers, and applications.
- **Datadog:** A cloud-based monitoring platform that provides real-time visibility into the performance and availability of IT systems.
- **New Relic:** A cloud-based performance monitoring and analytics platform that provides real-time insights into application performance, infrastructure performance, and user behavior.
- **AppDynamics:** A performance monitoring platform that provides real-time visibility into the performance of applications, infrastructure,

and business transactions.

- **OpenNMS**: An open-source network management platform that provides real-time monitoring and reporting of network devices, servers, and applications.
- **Splunk**: A real-time data analysis platform that provides centralized logging and real-time monitoring of infrastructure and applications.
- **Ganglia**: An open-source performance monitoring system that provides real-time visibility into the performance of clusters, networks, and grids.

These are just some of the popular infrastructure monitoring tools available. The best tool for an organization will depend on its specific needs and requirements.

Prometheus is designed to scrape metrics from a range of targets, including containers, servers, and applications, and store them in a time-series database. It provides a powerful query language and an alert manager to define and trigger alerts based on metric thresholds.

Configuring infrastructure monitoring

Here are the general steps to configure infrastructure monitoring with Prometheus:

1. **Install Prometheus**: Download and install Prometheus on a server or cluster of servers that will act as the monitoring server.
2. **Configure targets**: Configure Prometheus to scrape metrics from the targets you want to monitor. This can include servers, applications, containers, and other infrastructure components.
3. **Define metrics**: Define the metrics you want to collect from each target, including the frequency of scraping and the format of the data.
4. **Configure alerting**: Set up alerts in Prometheus based on specific metric thresholds. You can use the Prometheus Alert Manager to manage alerts and send notifications.
5. **Start Prometheus**: Start the Prometheus server and verify that it is correctly collecting and storing metrics.

6. **Monitor metrics:** Use Prometheus's query language to monitor metrics and identify trends and performance issues. You can use Prometheus's dashboards and Grafana integration to visualize metrics and make informed decisions.
7. **Maintain Prometheus:** Regularly update and maintain Prometheus to ensure it is operating correctly and collecting accurate metrics.

Note that these steps are general and may vary depending on the specific environment and requirements. It is also recommended to consult Prometheus documentation and online resources for specific guidance and best practices.

Prometheus configuration is YAML.

Following is the standard Prometheus configuration (comments are the lines prefixed with a #):

```
global:  
  scrape_interval:      15s  
  evaluation_interval: 15s  
  
rule_files:  
  # - "first.rules"  
  # - "second.rules"  
  
scrape_configs:  
  - job_name: prometheus  
    static_configs:  
      - targets: ['localhost:9090']
```

In above sample configuration `global`, `rule_files`, and `scrape_configs` are three blocks:

- **global:** It controls the Prometheus server's global configuration.

- **rule_files**: specifies the location of any rules we want the Prometheus server to load.
- **scrape_configs**: It controls what resources Prometheus monitors.

Application Performance Monitoring

Application Performance Monitoring (APM) is the process of measuring, managing, and optimizing the performance of a software application to ensure that it meets the requirements of the user and the business. APM typically involves monitoring the application's resource usage, identifying bottlenecks and problems, and tracking user experience to improve overall application performance.

Benefits of Application Performance Monitoring

The benefits of APM include:

- **Improved user experience**: APM helps identify and resolve performance issues that negatively impact user experience, ensuring that applications are responsive and meet the needs of users.
- **Increased availability**: APM helps detect and resolve issues that could cause an application to become unavailable, reducing downtime and increasing overall availability.
- **Faster problem resolution**: APM enables quick identification and resolution of performance issues, reducing the time it takes to restore normal operation.
- **Increased efficiency**: APM provides insights into application resource usage, enabling optimization and improved efficiency.
- **Better decision-making**: APM provides real-time data and analytics, enabling data-driven decision-making for performance optimization and capacity planning.
- **Improved customer satisfaction**: APM helps to ensure that applications meet the needs of the users and the business, leading to improved customer satisfaction.

- **Enhanced security:** APM can monitor for security breaches and suspicious activity, helping to detect and prevent security threats.
- **Improved collaboration:** APM provides a centralized view of application performance, enabling better collaboration among development, operations, and support teams.
- **Increased ROI:** APM helps to ensure that applications are performing optimally, increasing the **Return on Investment (ROI)** for the business.
- **Better compliance and auditing:** APM can monitor and report on key compliance and auditing requirements, enabling organizations to meet regulatory requirements and ensure data privacy.

When to use Application Performance Monitoring

APM should be used in the following scenarios:

- **During development:** APM can be used during the development process to ensure that applications are performing optimally and meeting performance requirements.
- **Upon deployment:** APM should be implemented immediately upon deployment to monitor application performance and identify any performance issues.
- **When performance issues arise:** APM should be used to identify and resolve performance issues when they arise, reducing downtime and improving overall performance.
- **When adding new features:** APM should be used when adding new features to ensure that the application continues to perform optimally.
- **During capacity planning:** APM provides real-time data and analytics, enabling data-driven decision-making for capacity planning and performance optimization.
- **When monitoring compliance and security:** APM can be used to monitor and report on key compliance and security requirements, ensuring that organizations meet regulatory requirements and maintain data privacy.

- **When scaling applications:** APM should be used when scaling applications to ensure that performance is not impacted as the application grows.
- **During ongoing application management:** APM should be used regularly to monitor application performance and identify and resolve performance issues.
- **Microservices architecture:** If an application is built using microservices, APM can be used to monitor the performance of individual services and the interactions between them.

Here are the top ten best tools for **Application Performance Monitoring (APM)** in detail:

- **New Relic:** New Relic is a cloud-based APM tool that provides real-time monitoring, deep transaction tracing, and custom dashboards to help resolve performance issues.
- **AppDynamics:** AppDynamics is a comprehensive APM solution that provides real-time visibility into application performance, deep transaction tracing, and custom analytics.
- **Azure monitoring:** Azure Application Monitoring is a service provided by Microsoft Azure that allows organizations to monitor the performance of their applications running on the Azure cloud platform.
- **Datadog:** Datadog is a cloud-based APM tool that provides real-time monitoring, customizable dashboards, and deep transaction tracing to help resolve performance issues.
- **Dynatrace:** Dynatrace is a cloud-based APM solution that provides real-time monitoring, deep transaction tracing, and artificial intelligence-powered root cause analysis.
- **Amazon CloudWatch:** Amazon CloudWatch is a cloud-based APM tool that provides real-time monitoring and customizable dashboards for **Amazon Web Services (AWS)** environments.
- **Nagios:** Nagios is a popular open-source APM tool that provides real-time monitoring, customizable dashboards, and deep transaction

tracing to help resolve performance issues.

- **Zabbix:** Zabbix is a comprehensive open-source APM solution that provides real-time monitoring, deep transaction tracing, and customizable dashboards.
- **InfluxData:** InfluxData is a time-series database and APM tool that provides real-time monitoring, deep transaction tracing, and customizable dashboards to help resolve performance issues.
- **ExtraHop:** ExtraHop is a network-based APM tool that provides real-time monitoring, deep transaction tracing, and custom analytics to help resolve performance issues.

Web application monitoring on Azure

Web application monitoring on Azure involves using Microsoft Azure's tools and services to continuously observe, measure, and analyze the performance, availability, and user experience of web applications hosted on the Azure cloud platform. This monitoring process helps ensure that web applications are running smoothly, meeting performance expectations, and quickly identifying and addressing any issues that may arise.

The architecture diagram of Azure monitoring is shown in the next figure:

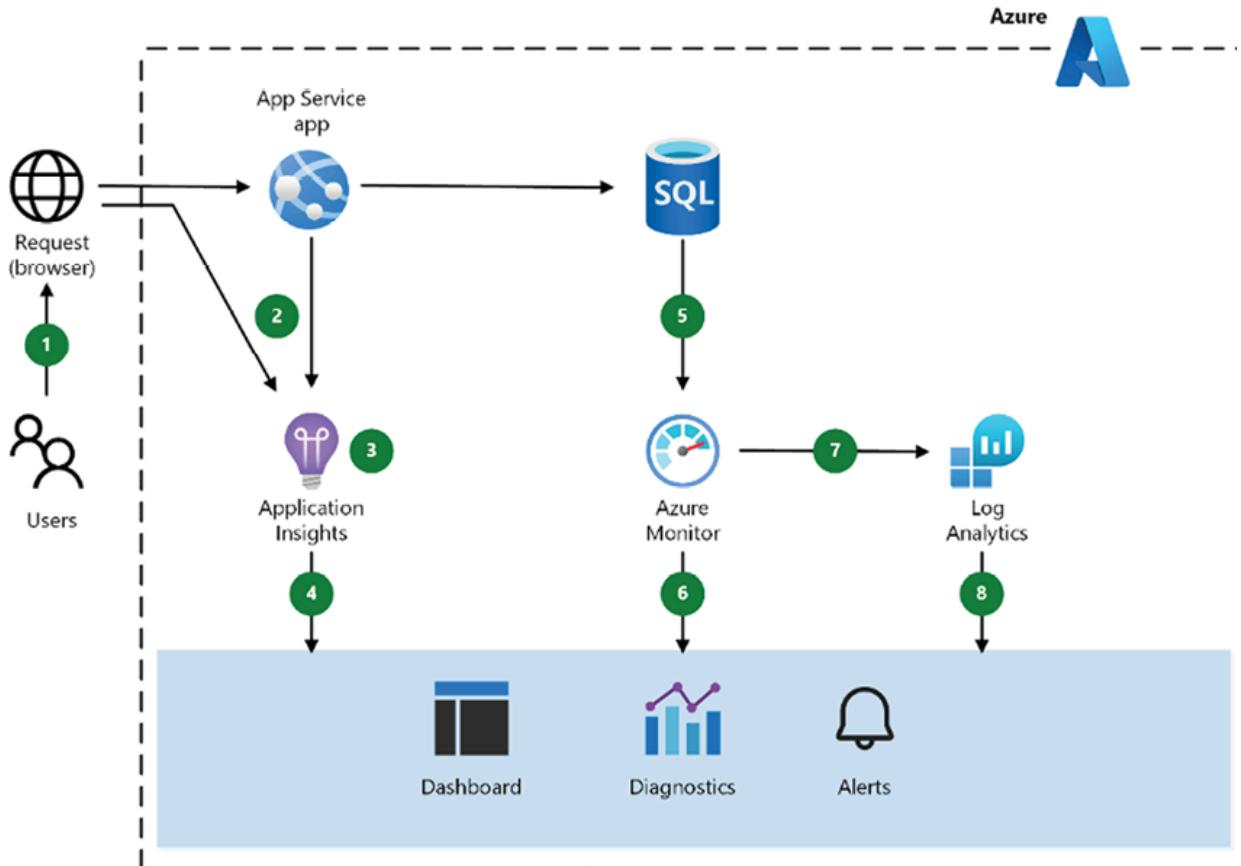


Figure 8.3: Azure Monitoring architect

Here is a general outline of how to configure Azure Application Monitoring:

1. **Set up Azure Monitor:** To use Azure Application Monitoring, you first need to set up Azure Monitor, which provides the foundation for monitoring your applications in Azure.
2. **Enable Insights:** You need to enable Application Insights for each application that you want to monitor. This will provide real-time performance metrics and insights for your application.
3. **Configure Alerts:** You can configure alerts to notify you when specific performance thresholds are exceeded. This allows you to quickly respond to performance issues and resolve them before they impact your users.
4. **Create dashboards:** You can create custom dashboards to visualize your application's performance data and gain insights into performance trends.

5. **Collect logs:** Azure Application Monitoring allows you to collect and analyze logs from your application, making it easier to diagnose and resolve performance issues.
6. **Use Azure monitor logs:** You can use Azure Monitor Logs to search, analyze, and visualize log data from your application, making it easier to diagnose and resolve performance issues.
7. **Integrate with other Azure services:** Azure application monitoring can be integrated with other Azure services, such as Azure functions, Azure container instances, and Azure service fabric, to provide a unified view of application performance across your Azure environment.

Note: This is a general outline of how to configure Azure Application Monitoring. The specific steps and configuration options will vary depending on the specific requirements of your application and the Azure services you are using. It is recommended to refer to the Azure documentation for more detailed instructions.

The following *Figure 8.4* gives a high-level view of Azure Monitor:

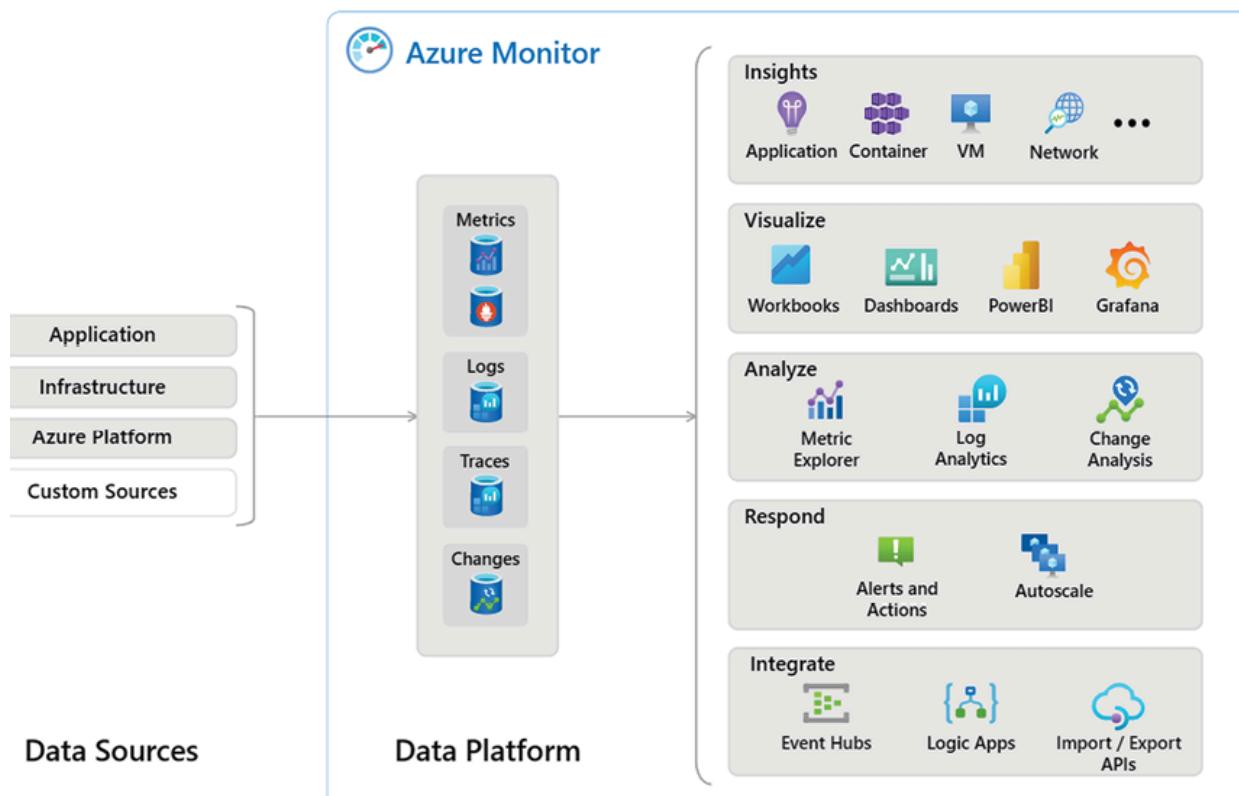


Figure 8.4: High level view of Azure Monitoring

Azure metrics, logs, and distributed traces are the three components of Azure Application Monitoring that provide real-time performance insights into your applications running in Azure.

Followings are the important components of Azure Application Monitoring:

- **Metrics:** Azure Metrics provides real-time performance data for your application, including resource utilization, response time, and error rates. This data is collected and analyzed by Azure Monitor and can be used to identify performance issues and trends.
- **Logs:** Azure Logs provides a way to collect and analyze log data from your application, including information about errors, warnings, and other performance issues. You can use Azure Monitor Logs to search, filter, and visualize log data to diagnose and resolve performance issues.
- **Distributed Traces:** Azure Distributed Traces provides a way to trace the path of a request as it travels through your application and its components. This allows you to see the performance of each component and identify bottlenecks and performance issues.
- **Changes:** Azure Change analysis is a feature of Azure monitor that helps you understand the impact of changes to your Azure resources and applications. It provides a centralized view of changes made to your resources and applications, along with information about when the changes were made and who made them. This makes it easier to identify the root cause of issues and resolve them quickly.

By using Azure Metrics, logs, and distributed traces, you can gain real-time insights into your application's performance, identify performance issues and trends, and quickly resolve performance issues before they impact your users.

Azure Metric Explorer is a feature of Azure Monitor that provides real-time performance data for your Azure resources and applications. It enables you to visualize, analyze, and alert on metrics data collected by Azure Monitor.

Azure metrics explorer dashboard view:

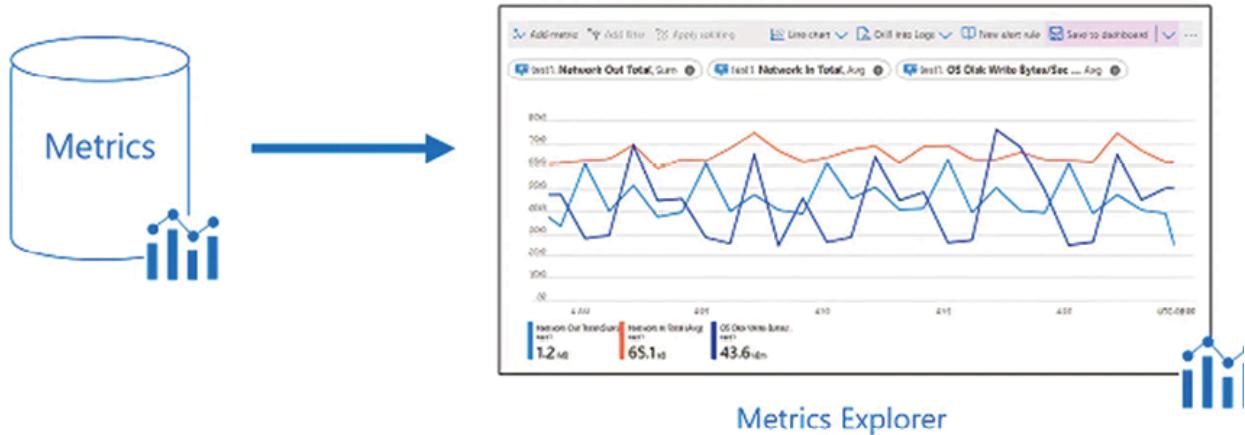


Figure 8.5: Azure metrics explorer dashboard view

Here is how Azure Metric Explorer works:

- **Data collection:** Azure Monitor collects performance data, such as resource utilization, response time, and error rates, from your Azure resources and applications. This data is stored in Azure Metrics and can be accessed through Azure Metric Explorer.
- **Data visualization:** Azure Metric Explorer provides a graphical interface for visualizing your metrics data. You can create charts and graphs to see trends and patterns in your metrics data over time.
- **Data analysis:** Azure Metric Explorer enables you to analyze your metrics data to identify performance issues and trends. You can use Azure Metric Explorer to create alerts and notifications to notify you when specific metric thresholds are exceeded.
- **Data alerting:** Azure Metric Explorer provides the ability to set up alerts based on metrics data. You can create alerts for specific metrics, such as resource utilization or response time, and specify when you want to be notified. For example, you can create an alert to notify you when the CPU utilization of a virtual machine exceeds 90%.
- **Data correlation:** Azure Metric Explorer enables you to correlate metrics data with other data sources, such as logs and traces, to get a complete picture of your application performance.

By using Azure Metric Explorer, you can gain real-time insights into the performance of your Azure resources and applications, identify performance

issues, and resolve them quickly, improving the overall performance and stability of your Azure environment.

Azure Log Explorer is a feature of Azure Monitor that enables you to analyze and search log data from your Azure resources and applications. It provides a centralized view of log data from Azure Monitor, Azure Log Analytics, and other sources, allowing you to identify and resolve issues quickly.

Log Analytics dashboard view:

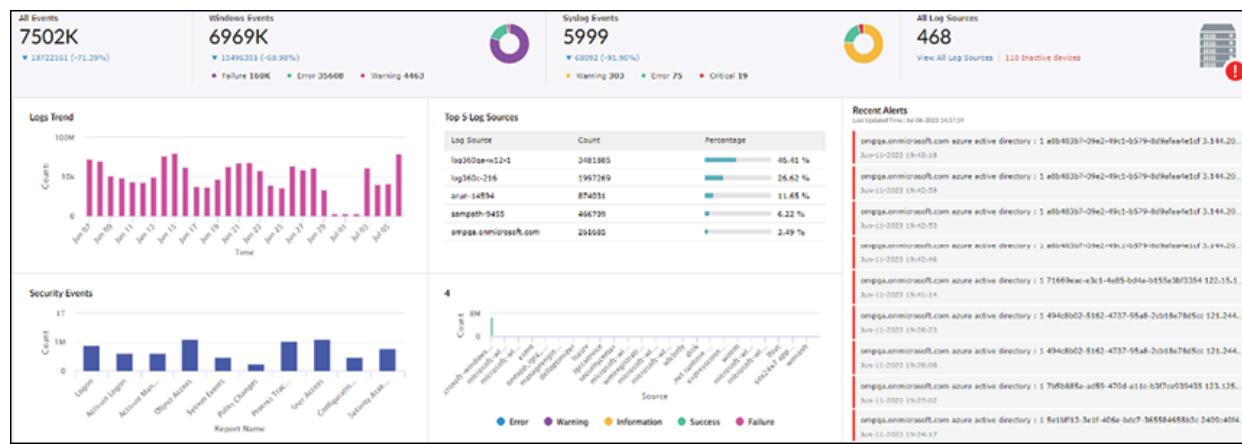


Figure 8.6 Log Analytics dashboard view

Here is how Azure Log Explorer works:

- **Data collection:** Azure Log Explorer collects log data from various sources, such as Azure Monitor, Azure Log Analytics, and other log sources. This log data is stored in Azure Log Analytics.
- **Data visualization:** Azure Log Explorer provides a graphical interface for visualizing log data. You can use Log Explorer to view log data in tables, charts, and graphs, and to search and filter log data to find specific information.
- **Data analysis:** Azure Log Explorer enables you to analyze log data to identify issues and trends. You can use Log Explorer to search and filter log data, create charts and graphs, and create custom log queries to analyze log data.

- **Data alerting:** Azure Log Explorer provides the ability to set up alerts based on log data. You can create alerts for specific log entries, such as error messages or performance issues, and specify when you want to be notified.
- **Data correlation:** Azure Log Explorer enables you to correlate log data with other data sources, such as metrics and traces, to get a complete picture of your application performance.
- **Integration with Azure monitoring:** Azure Log Explorer seamlessly integrates with Azure Monitor Workbooks, empowering you to craft custom dashboards and reports with rich data visualizations. By incorporating log data, you can gain deeper insights and perform sophisticated analysis for a comprehensive view of your Azure resources and applications. This integration enhances your ability to monitor and optimize performance effectively.

By using Azure Log Explorer, you can gain real-time insights into the performance of your Azure resources and applications, identify issues and trends, and resolve them quickly, improving the overall performance and stability of your Azure environment.

Application Insights

Azure Application Insights is a performance monitoring and analytics solution for web applications and services, provided by Microsoft Azure. It provides real-time insights into the performance, availability, and usage of your web applications and services, enabling you to identify and resolve issues quickly.

Application Insights dashboard view:

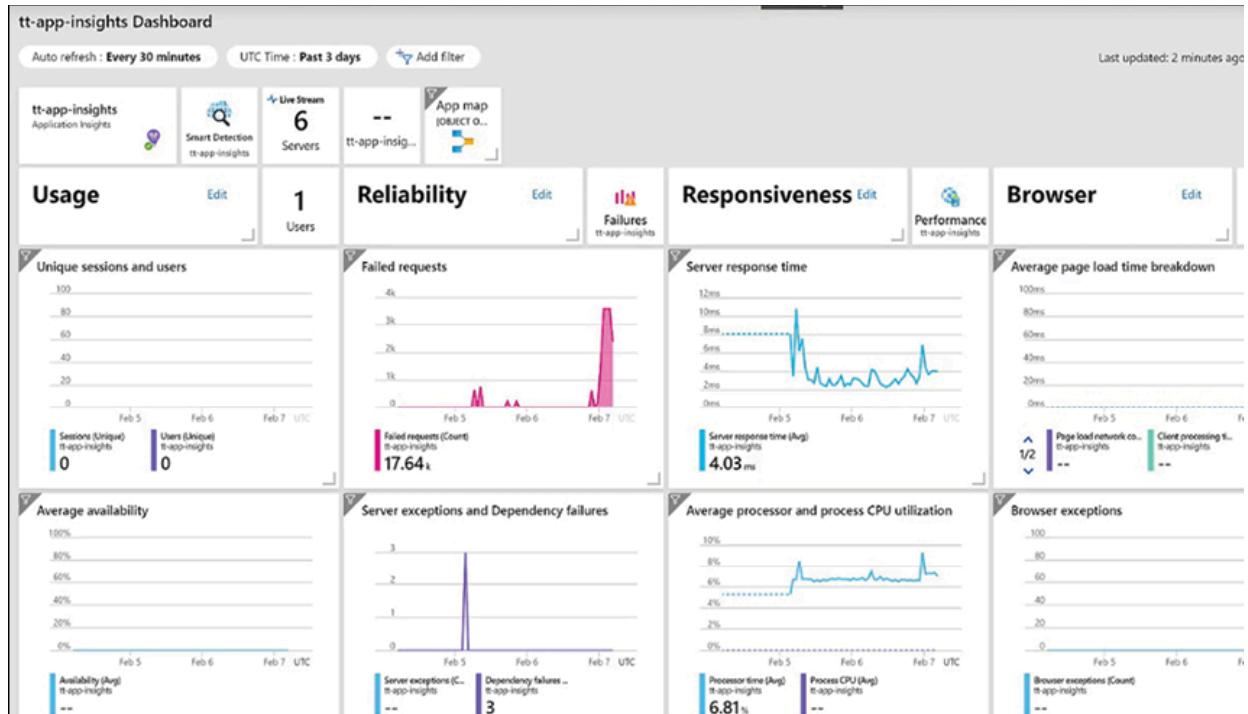


Figure 8.7: Application Insights dashboard

With Application Insights, you can:

- Monitor the performance of your applications in real-time.
- Analyze logs, traces, and performance metrics to identify issues and trends.
- Receive alerts and notifications when issues occur.
- Collect and analyze user and session data to understand usage patterns and user behavior.
- Identify dependencies and track the performance of external services that your applications depend on.
- Monitor the health and performance of your applications on a global scale.

Application Insights integrates with other Azure services and tools, making it easy to monitor and analyze data from across your Azure environment.

With its user-friendly interface, customizable dashboards, and rich set of analytics tools, Application Insights is a powerful solution for improving the performance and reliability of your web applications and services.

Azure Container Insights

Azure Container Insights is a monitoring solution for containers in Azure. It provides real-time performance monitoring, logging, and tracing for containers, enabling you to diagnose and resolve issues quickly.

Container Monitor Insights dashboard view:

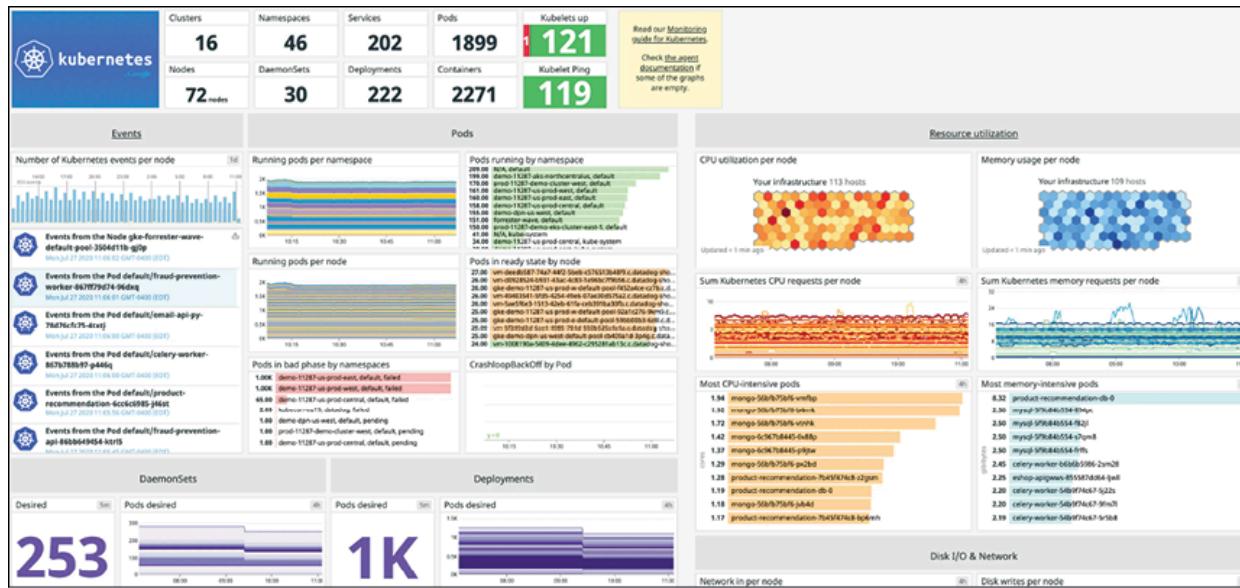


Figure 8.8: Container Monitor Insights dashboard

With Container Insights, you can:

- Monitor the health and performance of your containers in real-time.
- Analyze logs, traces, and performance metrics to identify issues and trends.
- Receive alerts and notifications when issues occur.
- Visualize the structure and relationships of your containers and services, making it easy to understand the dependencies between components.
- Monitor the utilization and performance of the underlying host and infrastructure to identify performance bottlenecks and resolve issues.
- Monitor the performance of your containers across multiple hosts and regions, providing a comprehensive view of your container

infrastructure.

- Integrate with other Azure services and tools, such as Azure Monitor and Azure Log Analytics, to provide a complete picture of the performance and behavior of your containers.

With its powerful set of monitoring and analysis tools, Azure Container Insights helps you maintain the performance, stability, and reliability of your container-based applications and services.

Azure VM Insights

Azure VM Insights is a performance monitoring solution for **Virtual Machines (VMs)** running in Azure. It provides real-time performance monitoring, logging, and tracing for VMs, enabling you to diagnose and resolve issues quickly.

Azure VM insight Dashboard view:

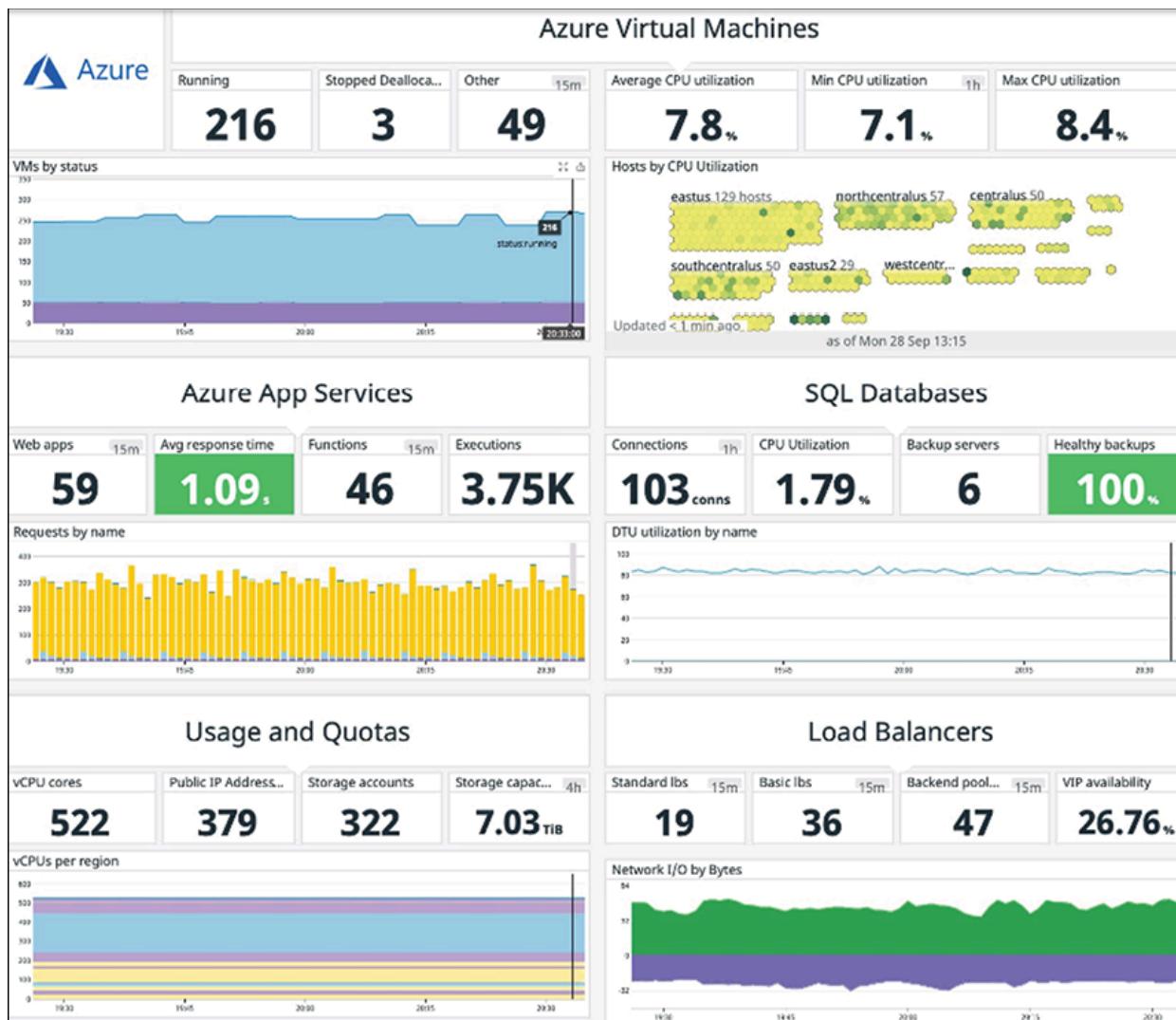


Figure 8.9 Azure VM insight Dashboard

With VM Insights, you can:

- Monitor the health and performance of your VMs in real-time.
- Analyze logs, traces, and performance metrics to identify issues and trends.
- Receive alerts and notifications when issues occur.
- Monitor the utilization and performance of the underlying host and infrastructure to identify performance bottlenecks and resolve issues.
- Monitor the performance of your VMs across multiple hosts and regions, providing a comprehensive view of your virtual machine

infrastructure.

- Integrate with other Azure services and tools, such as Azure Monitor and Azure Log Analytics, to provide a complete picture of the performance and behavior of your VMs.
- Monitor the performance of your VMs running on Windows or Linux operating systems.

With its powerful set of monitoring and analysis tools, Azure VM Insights helps you maintain the performance, stability, and reliability of your virtual machine-based applications and services.

Setup alerts

Azure Monitor Alerts is a feature in Azure Monitor that allows you to set up notifications and take automated actions based on specific conditions. Azure Monitor alerts provide real-time monitoring and alerting for various Azure resources, including virtual machines, web apps, and databases.

The following figure shows how alerts work:

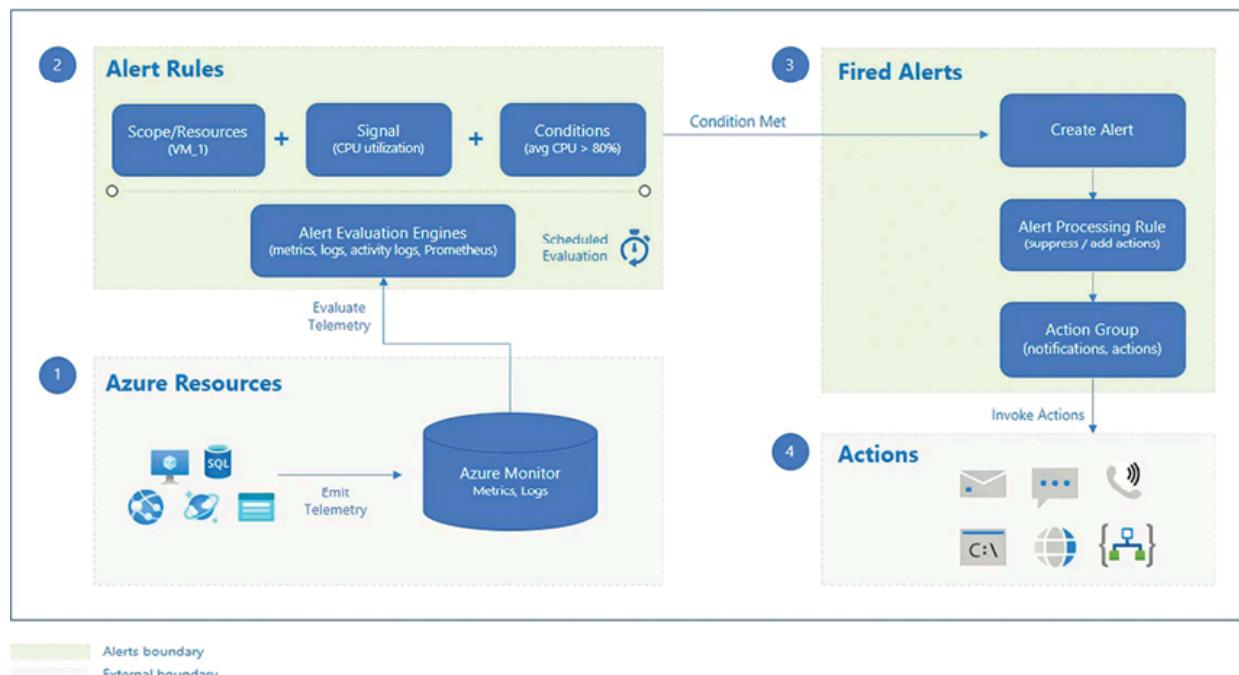


Figure 8.10: Alerts workflow

Azure Monitor Alerts utilize action groups, which contain distinct combinations of recipients and actions that can be utilized across multiple alert rules. Depending on your needs, action groups can initiate actions such as activating webhooks to trigger external actions or integrating with IT service management tools.

Here are the steps to set up Azure Monitor alerts:

1. Log into the Azure portal.
2. Select the resource you want to monitor, such as a virtual machine, web app, or database.
3. Go to the **Alerts** section of the resource.
4. Click on the **New alert rule** to create a new alert.
5. Select the type of alert you want to create. There are several pre-defined alert types, such as high CPU usage, low disk space, or error rates. You can also create custom alerts based on log data.
6. Define the conditions that will trigger the alert. For example, you can set an alert to trigger when CPU usage exceeds a certain threshold.
7. Select the action(s) that will be taken when the alert is triggered. You can choose from several options, such as sending an email or SMS, triggering a webhook, or executing an automation runbook.
8. Define the scope of the alert, such as which resource group or subscription it should apply to.
9. Review the details of the alert and save the alert rule.
10. The alert is now active and will trigger when the specified conditions are met.

By following these steps, you can set up Azure Monitor alerts to monitor the health and performance of your Azure resources and receive notifications or take automated actions when issues occur.

Alerts Dashboard view:

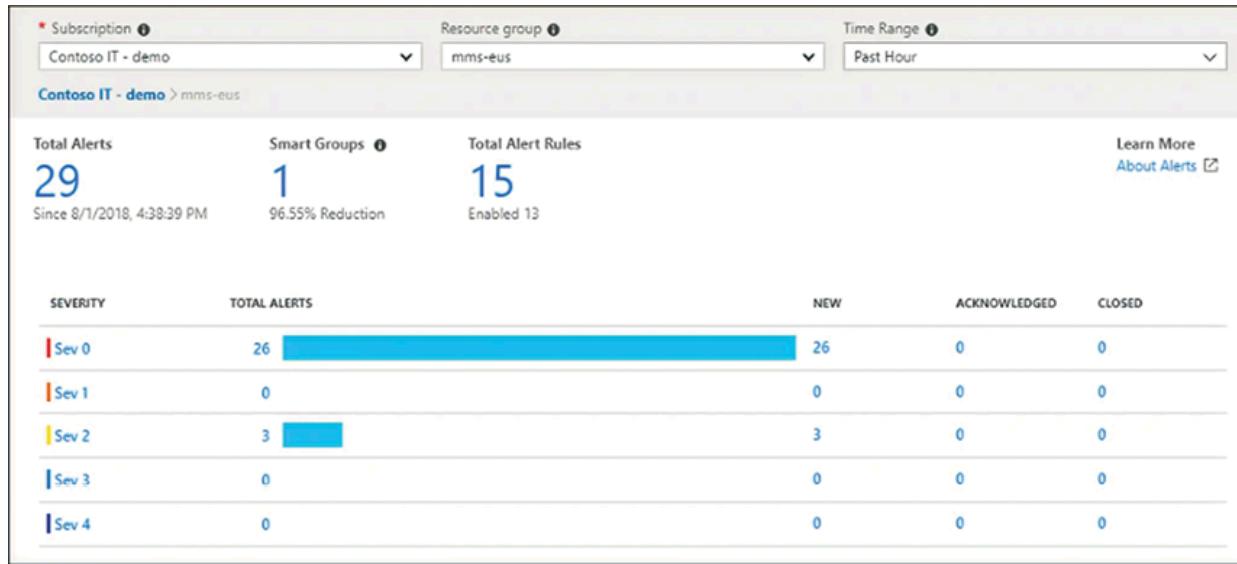


Figure 8.11: Alerts dashboard

Visualize monitoring data

Azure provides several ways to visualize monitoring data and gain insights into the health and performance of your Azure resources. Here are some of the ways to visualize monitoring data in Azure:

- **Dashboards:** Azure Monitor dashboards provide a real-time view of the health and performance of your Azure resources. You can create custom dashboards that display data from multiple sources, including metrics, logs, and alerts.
- **Metric charts:** Azure Monitor provides detailed metric charts that allow you to visualize performance data over time. You can customize the charts to display specific metrics, data ranges, and time ranges.
- **Log analytics:** Azure Monitor Log Analytics is a powerful tool for visualizing and analyzing log data. You can use Log Analytics to create custom queries and visualizations and to gain insights into the health and performance of your Azure resources.
- **Application Insights:** Azure Application Insights is a powerful tool for monitoring web applications. It provides detailed telemetry data, including performance metrics, traces, and exceptions. You can use

Application Insights to visualize performance data and to identify and troubleshoot issues with your web applications.

- **Azure Monitor Workbooks:** Azure Monitor Workbooks are interactive, markdown-based documents that allow you to visualize and analyze monitoring data. You can create custom Workbooks that combine data from multiple sources, including metrics, logs, and alerts.

These are some of the ways to visualize monitoring data in Azure and gain insights into the health and performance of your Azure resources. By using these tools, you can quickly identify issues, understand the root cause, and take appropriate action to resolve problems.

Dashboards

Azure Dashboards provide a flexible and customizable interface that allows you to create the views that are most relevant to your needs. You can add charts, tables, and other visualizations to your dashboards and arrange them in a way that makes sense for your specific use case.

Single pane of glass view for Monitoring and Analytics:

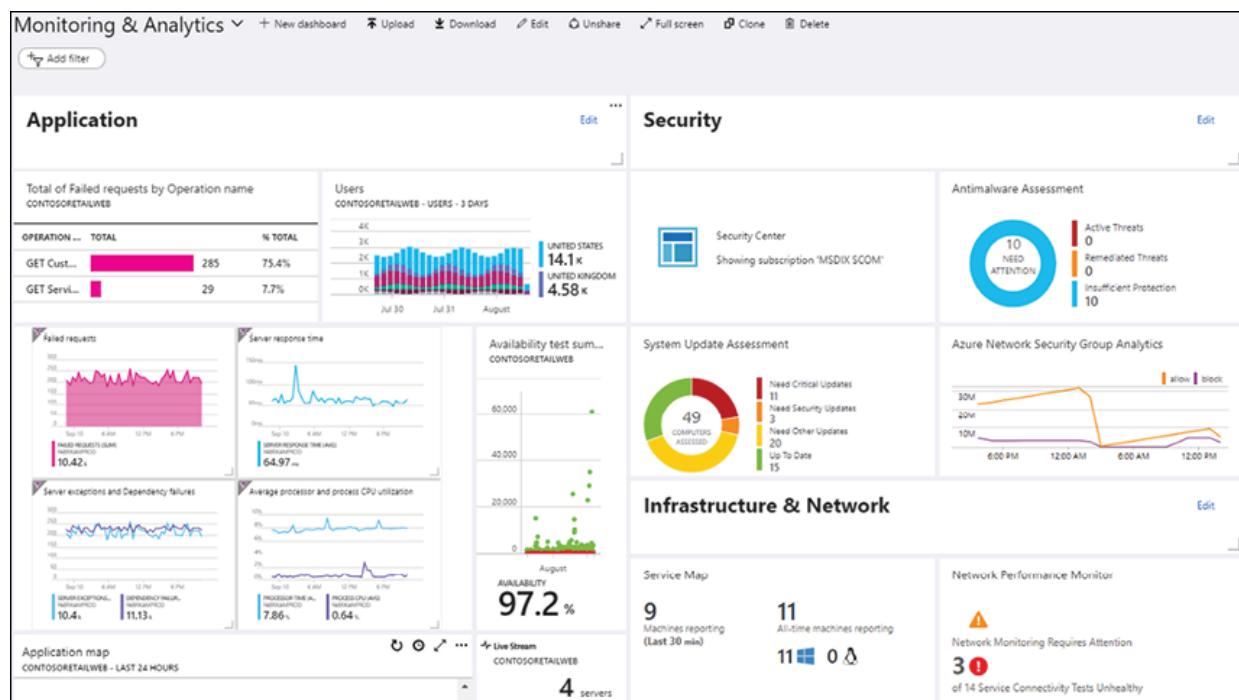


Figure 8.12: Azure Monitoring and Analytics

Azure workbooks

Azure Monitor Workbooks are interactive, markdown-based documents that allow you to visualize and analyze monitoring data in Azure. Workbooks provide a flexible and powerful interface for exploring and visualizing data, making it easier to gain insights into the health and performance of your Azure resources.

End-to-End Application Monitoring: Workbooks enable the creation of customized dashboards that combine metrics, logs, and traces from various Azure services. For example, you can visualize the response times, error rates, and dependency calls of a web application, allowing for thorough performance analysis and troubleshooting.

Azure Workbook Dashboard view:

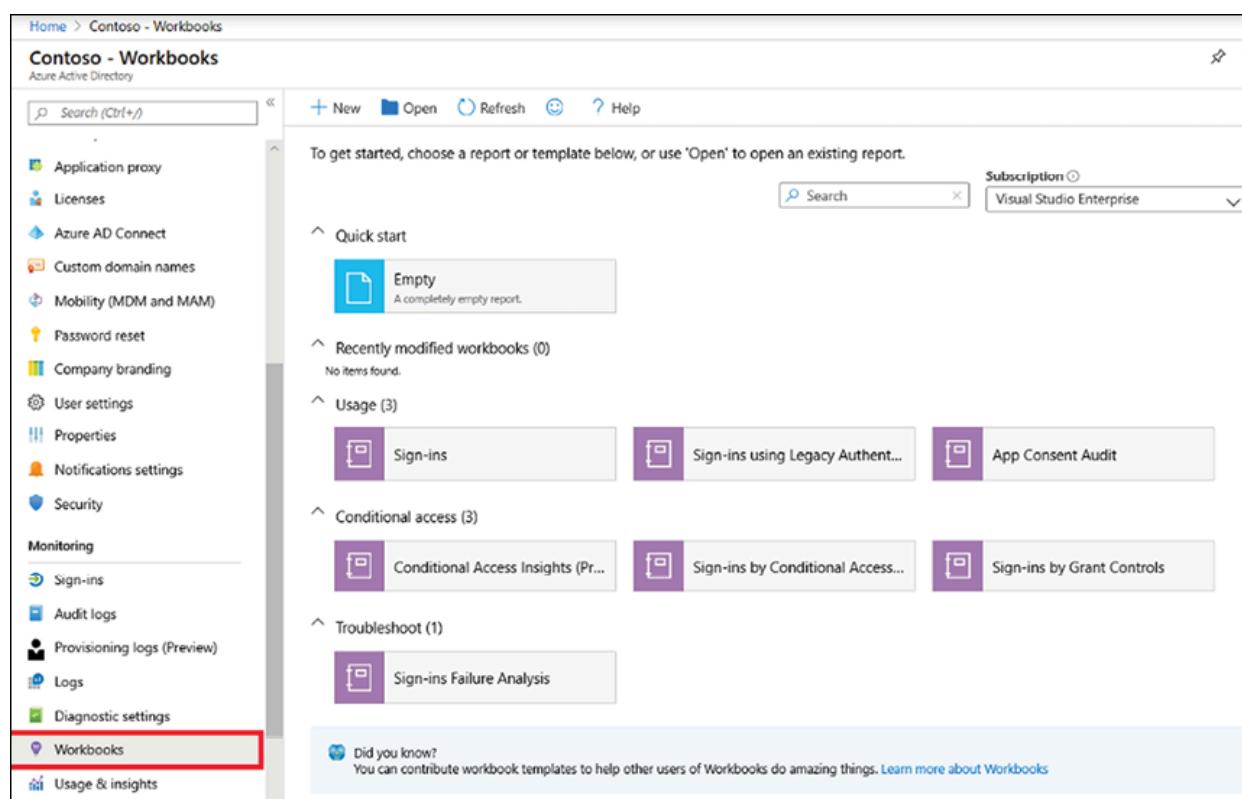


Figure 8.13: Azure workbook dashboard view

Power BI

Power BI is a business intelligence and data visualization tool from Microsoft that enables users to connect, analyze, and visualize data from various sources. With Power BI, you can create interactive reports, dashboards, and visualizations to help you gain insights into your data and make more informed decisions.

With the help of Power BI you can automatically import log data from Azure Monitor. Power BI Dashboard view:

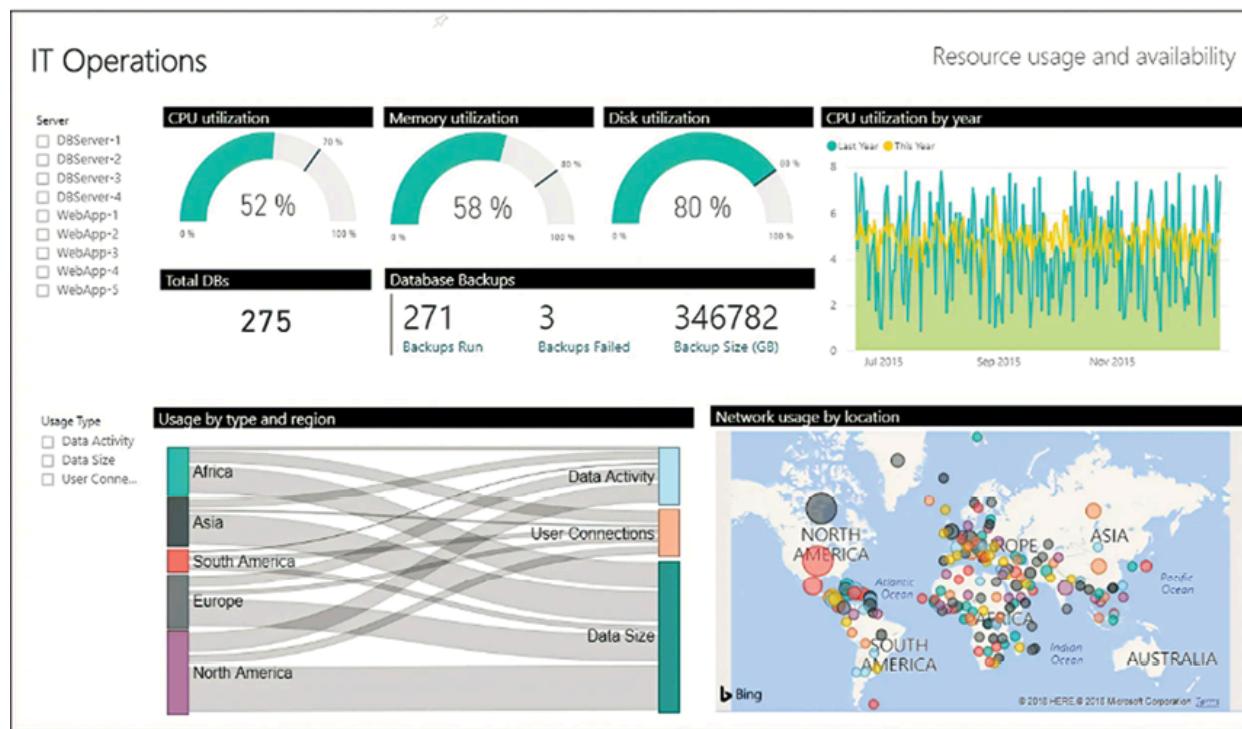


Figure 8.14: Power BI dashboard

Power BI is a business intelligence and data visualization tool from Microsoft that enables users to connect, analyze, and visualize data from various sources. With Power BI, you can create interactive reports, dashboards, and visualizations to help you gain insights into your data and make more informed decisions.

Configuring automatic continuous monitoring

Building a robust and fully automatic continuous monitoring solution requires careful planning, the right tools, and a focus on continuous improvement. Here is a step-by-step guide in full detail with examples:

- 1. Define your monitoring requirements:** Start by defining what you want to monitor and what performance and availability metrics are most important to your business. For example, you might want to monitor response time, error rate, or disk usage.
- 2. Choose your monitoring tools:** Select the right monitoring tools based on your requirements and budget. Consider factors such as scalability, ease of use, and integration with other tools. Some popular options include Azure Monitor, New Relic, and Datadog.
- 3. Set up your monitoring infrastructure:** Configure your monitoring tools, including data collectors, servers, and storage, to ensure that your monitoring solution can scale with your needs. For example, you might set up a centralized monitoring server or configure monitoring agents on each of your application servers.
- 4. Configure your monitoring rules:** Set up monitoring rules and alerts to notify you when performance or availability metrics fall below predefined thresholds. For example, you might configure an alert to notify you when response time exceeds a threshold of two seconds or when disk usage reaches 90% capacity.
- 5. Integrate with your development and operations processes:** Integrate your monitoring solution with your development and operations processes to ensure that monitoring data is available when and where it is needed. For example, you might integrate your monitoring solution with your incident management system or configure alerts to trigger automated remediation scripts.
- 6. Automate your monitoring:** Use automation to automate as much of your monitoring as possible, reducing the need for manual intervention. For example, you might use automation to schedule regular monitoring reports or to trigger automatic failovers in the event of a failure.
- 7. Continuously review and refine your monitoring:** Regularly review your monitoring data and refine your monitoring rules and alerts to ensure that they continue to meet your needs. For example, you might update your thresholds based on actual performance data or refine your alerts to reduce false positives.

8. Invest in continuous improvement: Invest in continuous improvement to ensure that your monitoring solution stays up-to-date and continues to meet your needs over time. For example, you might invest in training for your operations team or explore new monitoring tools and techniques.

In summary, building a robust and fully automatic continuous monitoring solution requires careful planning, the right tools, and a focus on continuous improvement. By following these steps, you can ensure that your monitoring solution provides the data and insights you need to keep your apps running smoothly.

Conclusion

Continuous monitoring is a critical aspect of modern application development and operation. It helps ensure that applications are functioning optimally, providing valuable insights into performance, usage, and user experience. By automating the monitoring process, organizations can save time and resources, respond to issues quickly, and proactively identify potential problems before they become critical. In conclusion, implementing a robust and fully automatic continuous monitoring system is essential for any organization looking to optimize its application performance, user experience, and overall success.

In the next chapter, we are going to learn different types of rollback strategies.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 9

Rollback Strategy

Introduction

A rollback strategy refers to a plan or set of procedures that are implemented to revert a system or software application to a previous state. This is often done in response to an error, malfunction, or unexpected behavior that occurs after a change or update has been made. The term is commonly used in the context of software deployment, where a new version of a program or system is released.

When a rollback is necessary, it means that the changes made in the new version have caused issues or disruptions that were not anticipated. The rollback strategy outlines the steps to undo the changes and restore the system to its previous, stable state.

Having a well-defined rollback strategy is crucial in minimizing downtime, reducing the impact on users, and maintaining the stability and reliability of a system during the software development life cycle.

Structure

This chapter will discuss the best practices for implementing a DevOps rollback strategy. We will explore the key components of a successful rollback plan, including automated rollback processes, version control, testing, communication plans, and post-rollback analysis. By following these best practices, DevOps teams can ensure that they are prepared to respond to any issues, minimize downtime, and maintain the reliability and quality of their applications.

The following are types of rollback policies:

- Manual rollback procedures

- Automated rollback scripts
- Snapshot backups
- Version control
- Hotfix rollback strategy
- Feature toggles rollback strategy
- Immutable infrastructure rollback strategy
- Canary deployment
- A/B testing deployment strategy
- Blue-Green deployment
- Shadow deployment

Objectives

We will learn the importance of having a DevOps rollback strategy to minimize the impact of issues arising during or after deployment. We will delve into the key components of a successful rollback plan, including automated rollback processes, version control, testing, communication plans, and post-rollback analysis.

Additionally, we will provide the best practices and tips for implementing a DevOps rollback strategy that is efficient, reliable, and minimizes downtime.

Whether you are a DevOps team seeking to enhance your rollback processes or preparing for any potential deployment challenges, our recommendations will help you stay prepared and responsive.

Introducing rollback strategies

Rollback strategies are crucial in the **Software Development Lifecycle (SDLC)** as they help to mitigate risks and minimize the impact of failures. Here are some of the different ways of rollback strategies:

- **Manual rollback procedures:** Manual rollback procedures can be established to ensure that developers and operations teams know how to roll back changes in case of problems. This can involve steps such as restoring from backups or manually reversing changes to the code.
- **Automated rollback scripts:** Automated scripts can be set up to automatically revert changes if certain conditions are met, such as if the application fails to start or if an error rate exceeds a certain threshold.

- **Snapshot backups:** Taking periodic snapshots of the system or database allows for easy rollback to a previous state in case of problems.
- **Version control:** A version control system such as Git or **Subversion (SVN)** can be used to keep track of changes made to the code. If a change causes an issue, developers can easily roll back to a previous code version.
- **Hotfix rollback strategy:** A hotfix rollback strategy is a set of procedures put in place to reverse changes made by a hotfix that was applied to a system or application. A hotfix is a patch or update released quickly to address a specific issue or bug in a software system without a full testing cycle.
- **Feature toggles:** Feature toggles are flags that can be turned on or off to enable or disable a specific application feature. If a new feature causes problems, the toggle can be turned off to roll back to the previous state of the application.
- **Immutable infrastructure rollback strategy:** Immutable infrastructure is an approach to infrastructure management where servers, containers, or other infrastructure components are treated as disposable and are replaced with new instances instead of being modified in place. In an immutable infrastructure rollback strategy, the focus is on replacing the problematic component with a new, functioning one instead of rolling back changes made to the existing component.
- **Canary deployment:** In this strategy, a small percentage of users are directed to the new version of the application while the rest of the users continue to use the old version. If the new version works well, more users are redirected to it until it is fully deployed. If there are problems, the rollback is as simple as redirecting all users to the old version.
- **A/B testing deployment strategy:** A/B testing deployment strategy is a process used to test and compare two different versions of a software application, website, or other digital product to determine which version performs better. In this strategy, two different versions of the product are deployed to a small subset of users, and their behavior and feedback are analyzed to determine which version is more effective.
- **Blue-Green deployment:** This strategy involves maintaining two identical production environments, one *blue* and one *green*. The current version of the application runs on the blue environment, while the new version is

deployed to the green environment. If the new version causes problems, the rollback can be as simple as redirecting traffic to the blue environment.

- **Shadow deployment:** Shadow deployment is a technique used to test changes to a production system without impacting users. In this technique, a new version of the system is deployed alongside the existing version, and a small percentage of user traffic is redirected to the new version. The new version runs in parallel with the existing version, and the results are compared to ensure that the new version performs as expected.

Overall, a combination of these rollback strategies can be used to minimize the impact of failures and ensure smooth software development and deployment.

Manual rollback procedures

Manual rollback procedures are steps or actions taken to reverse changes made to a system, application, or database. The purpose of these procedures is to restore the system to a previous state if an error or issue is detected after changes have been made.

Here are the general steps involved in a manual rollback procedure:

1. **Identify the problem:** The first step in any rollback procedure is to identify the issue that needs to be corrected. This can be done through monitoring tools, system logs, or user feedback.
2. **Notify stakeholders:** Once the problem has been identified, it is important to notify all stakeholders affected by the rollback. This includes IT staff, end-users, customers, and other relevant parties.
3. **Determine the rollback plan:** The next step is determining the best course of action to rollback the changes. This may involve rolling back the entire system or only specific components or features.
4. **Execute the rollback:** The rollback can be executed with a plan in place. This involves reversing the changes made during the previous deployment, possibly restoring data from a backup, or undoing configuration changes.
5. **Test the rollback:** After the rollback has been completed, it is important to thoroughly test the system to ensure that it is functioning correctly. This includes checking if the data is intact, the features are working as expected, and that there are no new issues or errors.
6. **Communicate the outcome:** Finally, the outcome of the rollback should be communicated to all stakeholders. This includes explaining the reason for

the rollback, the steps taken to correct the issue, and any impact on the system or users.

7. **Documenting the rollback process:** Documenting the rollback process is a fundamental practice that enhances reliability, reduces risk, and promotes a culture of learning and improvement.

Overall, manual rollback procedures are an important part of any IT system or application deployment process. By having a clear plan in place for rolling back changes, businesses can minimize the impact of errors or issues on their operations and ensure that their systems are functioning as intended.

Automated rollback scripts

Automated rollback scripts are a critical component of DevOps practices, which aim to streamline software development and deployment processes. Rollback scripts allow teams to quickly and automatically revert changes made during a deployment, ensuring that systems can be returned to a stable state in the event of errors or issues.

Here are some key features of automated rollback scripts in DevOps:

- **Version control:** Automated rollback scripts are typically stored in a version control system, which allows developers to track changes made to the script over time. This helps ensure that rollback scripts are always up-to-date and that changes made during deployment can be easily reversed.
- **Triggering mechanisms:** Automated rollback scripts are triggered automatically when an error or issue is detected during a deployment. This can be done through various mechanisms, including monitoring tools, log analysis, or user feedback.
- **Pre-defined steps:** Rollback scripts typically consist of a pre-defined set of steps that need to be executed to reverse changes made during a deployment. These steps may include rolling back database changes, restoring files from backups, or undoing configuration changes.
- **Automated execution:** Once triggered, automated rollback scripts are executed automatically, which helps save time and reduces the risk of human error. This ensures that systems can be returned to a stable state as quickly as possible, minimizing the impact of errors or issues on users.
- **Testing:** Automated rollback scripts are typically tested thoroughly to ensure that they work as expected. This may involve testing the script in a

staging environment or running automated tests to validate that the rollback has been successful.

Automated rollback scripts are an essential component of DevOps practices, which aim to increase efficiency and reduce the risk of errors during software development and deployment. By having automated rollback scripts in place, teams can ensure that systems can be returned to a stable state quickly and effectively in the event of issues or errors.

Snapshot backups

Snapshot backups are a type of backup strategy that captures the current state of a system at a specific point in time, allowing it to be restored to that state in the event of a failure or data loss. The term ‘snapshot’ refers to a point-in-time copy of the data, which can be used to create a backup or to restore the data to a previous state.

Here is a deep dive into snapshot backups:

- **What is a snapshot backup?**

A snapshot backup is a backup strategy that captures the state of a system at a specific point in time. This snapshot can be used to restore the system to the exact state it was in when the snapshot was taken. Snapshot backups are typically used in virtualized environments, such as those that use VMware or Hyper-V, as well as in **Storage Area Network (SAN)** environments.

- **How do snapshot backups work?**

Snapshot backups work by creating a point-in-time copy of the data being backed up. This is typically done by creating a read-only copy of the data, which can be used to create a backup or to restore the system to a previous state. When a snapshot is taken, the system continues to write new data to the original storage location, but the snapshot remains static.

- **Benefits of snapshot backups**

Snapshot backups offer several benefits, including:

- **Faster backups:** Snapshot backups are typically faster than traditional backup methods, as they only capture changes made since the last snapshot was taken.
- **Reduced downtime:** Since snapshot backups can be used to restore a system to a previous state quickly, they can help reduce downtime in the

event of a failure or data loss.

- **Reduced storage requirements:** Snapshot backups typically use less storage space than full backups, as they only capture changes made since the last snapshot was taken.

- **Types of snapshot backups**

There are several types of snapshot backups, including:

- Full Snapshot: This captures the entire state of the system at a specific point in time.
- Differential Snapshot: This captures the changes made since the last full snapshot was taken.
- Incremental Snapshot: This captures only the changes made since the last snapshot, whether it was a full, differential, or incremental snapshot.

- **Limitations of snapshot backups**

While snapshot backups offer several benefits, they also have some limitations. These include:

- **Limited retention:** Snapshots are typically stored for a limited time, after which they are automatically deleted.
- **Limited scope:** Snapshot backups are typically limited to the system being backed up and do not include other systems or databases.
- **Impact on performance:** Taking frequent snapshots can impact system performance, particularly if the system being backed up is under heavy load.
- **Storage requirements:** Snapshots consume storage space, and as you take more snapshots, the storage needs can grow significantly. This may require additional hardware or cloud resources.
- **Limited use for Disaster Recovery:** While snapshots are helpful for quick recovery from system failures, they may not be sufficient for disaster recovery scenarios where the entire system is lost.

Steps to maintain backups of a disk using incremental snapshots

The steps to maintain a backup of a disk using incremental snapshots are as follows:

1. To create a backup of your premium storage disk, begin by taking a snapshot of it named `mypremiumdisk_ss1`. This snapshot will be used to create a backup page blob in a standard storage account named `mybackupstdaccount`. The backup page blob should be named `mybackupstdpageblob`.
2. Next, use the `snapshot Blob` function to create a new snapshot of `mybackupstdpageblob`, which should be named `mybackupstdpageblob_ss1`. Store this new snapshot in the `mybackupstdaccount`.
3. During the backup window, take another snapshot of `mypremiumdisk`, which should be named `mypremiumdisk_ss2`. Store this snapshot in the premium storage account.
4. To get the incremental changes between the two snapshots, use the `GetPageRanges` function on `mypremiumdisk_ss2` with the `prevsnapshot` parameter set to the timestamp of `mypremiumdisk_ss1`. Write these changes to the backup page blob in `mybackupstdaccount`. If there are any deleted ranges in the incremental changes, make sure to clear them from the backup page blob. Use the `PutPage` function to write incremental changes to the backup page blob.
5. After writing the incremental changes, take another snapshot of the backup page blob named `mybackupstdpageblob_ss2`. Delete the previous snapshot `mypremiumdisk_ss1` from the premium storage account.

Repeat steps 4-6 during every backup window to maintain backups of `mypremiumdisk` in a standard storage account.

VM disk snapshot architecture:

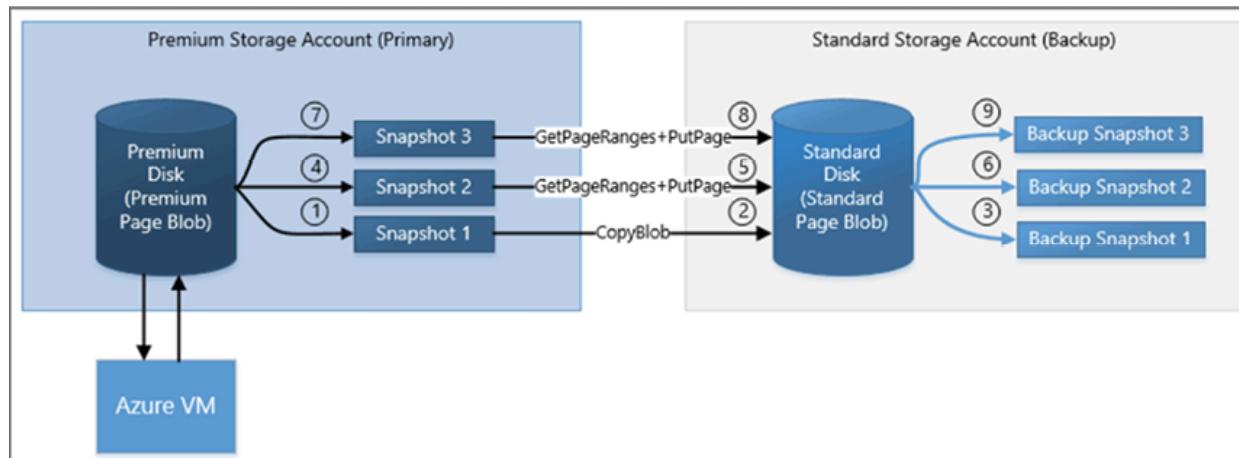


Figure 9.1: Snapshots of vm disk

Steps to restore a disk from snapshots

Here are the steps to restore the premium disk `mypremiumdisk` to an earlier snapshot from the backup storage account `mybackupsstdaccount`:

1. Determine the point in time to which you want to restore `mypremiumdisk`. Suppose you choose the snapshot `mybackupsstdpageblob_ss2`, which is stored in the `mybackupsstdaccount`.
2. In the `mybackupsstdaccount`, promote the snapshot `mybackupsstdpageblob_ss2` to be the new backup base page blob, which should be named `mybackupsstdpageblobrestored`.
3. Take a snapshot of the restored backup page blob and name it `mybackupsstdpageblobrestored_ss1`.
4. Copy the restored page blob `mybackupsstdpageblobrestored` from `mybackupsstdaccount` to `mypremiumaccount` as the new premium disk, which should be named `mypremiumdiskrestored`.
5. Take a snapshot of `mypremiumdiskrestored` and name it `mypremiumdiskrestored_ss1` for future incremental backups.
6. Update the D-Series **Virtual Machine (VM)** to use the restored disk `mypremiumdiskrestored` and detach the old `mypremiumdisk` from the VM.
7. Begin the backup process described in the previous section for the restored disk `mypremiumdiskrestored`, using `mybackupsstdpageblobrestored` as the backup page blob.

By following these steps, you can restore `mypremiumdisk` to an earlier snapshot from the backup storage account `mybackupsstdaccount`.

VM restore workflow:

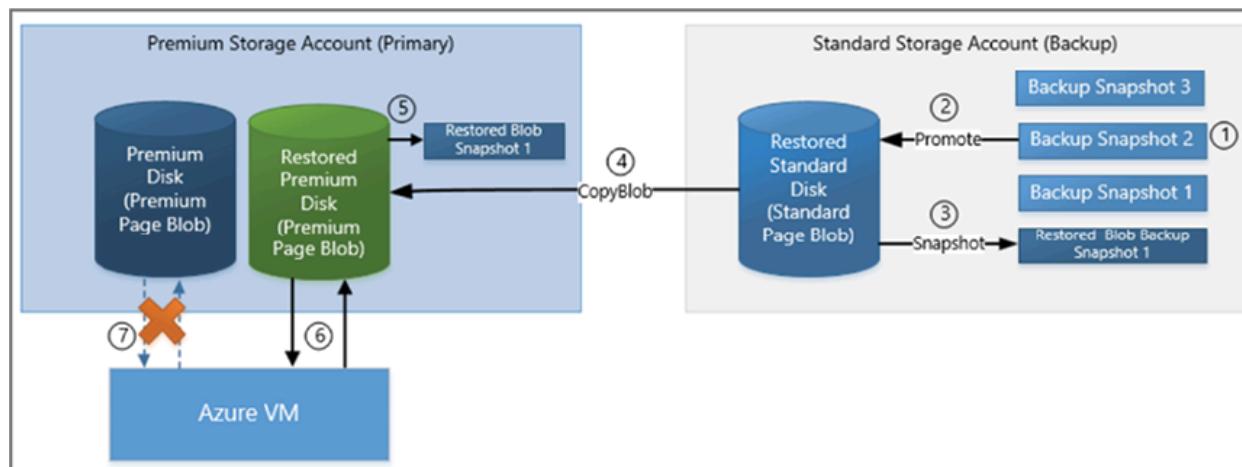


Figure 9.2: Restore VM from snapshots backup

Snapshot backups provide a fast and efficient way to capture the current state of a system at a specific point in time, allowing it to be restored to that state in the event of a failure or data loss. However, they have some limitations and should be used in conjunction with other backup strategies to ensure comprehensive data protection.

Version control

Version control systems can be used as a backup and restore strategy for applications by storing the entire source code and building the artifacts of the application over time. This means that every version of the application is preserved, allowing developers to revert to a previous version if necessary.

Here is a deep dive into how version control can be used for application version backup and restore strategy:

What is version control for application backup and restore?

Version control for application backup and restore involves using a version control system to store the entire source code and build artifacts of the application over time. This ensures that every version of the application is preserved and can be easily accessed if needed.

How does version control work for application backup and restore?

Version control works by storing each version of the application in a repository. The repository contains the entire source code, as well as any build artifacts or binaries generated during the build process. Each version of the application is stored as a commit, which includes metadata such as the author, date, and a message describing the changes made.

Benefits of version control for application backup and restore

Version control for application backup and restore offers several benefits, including:

- Comprehensive data protection: Version control for application backup and restore ensures that the entire history of the application is preserved, providing comprehensive data protection.
- Improved collaboration: Version control makes it easy to collaborate with others on the same application, as changes can be tracked and merged easily.

- **Greater flexibility:** Version control allows users to experiment with different versions of the application, knowing that they can easily revert to a previous version if necessary.

Types of version control for application backup and restore

There are several types of version control for application backup and restore, including:

- **Git:** Git is a popular version control system that is widely used for application development. Git stores each version of the application in a repository, allowing developers to easily track changes over time.
- **Subversion:** Subversion is another version control system that is commonly used for application development. Subversion stores each version of the application in a repository, allowing developers to easily track changes over time.

Limitations of version control for application backup and restore

While version control for application backup and restore offers several benefits, it also has some limitations. These include:

- **Storage requirements:** Version control for application backup and restore can require significant amounts of storage, particularly if large applications are being stored.
- **Performance impact:** Version control for application backup and restore can impact system performance, particularly if large applications are being stored.
- **Cost:** Version control for application backup and restore can be expensive, particularly if advanced version control systems or hosting solutions are used.

Here is a step-by-step guide on how to create a new release version and publish it to production using GitHub:

1. **Create a new branch:** Create a new branch in the repository for the changes you want to make for the new release.
2. **Develop new features/fixes:** Develop new features or fixes for the application in the new branch.
3. **Test the changes:** Test the changes locally to ensure that everything is working as expected.

4. **Automated testing:** Integrate automated testing into your workflow. This may include unit tests, integration tests, and end-to-end tests.
5. **Continuous integration (CI):** Use CI services such as GitHub Actions, Travis CI, or CircleCI to automate the execution of tests whenever changes are pushed to the repository.
6. **Commit changes:** Commit the changes to the new branch and push it to GitHub.
7. **Create a new release:** Go to the repository's **Releases** page on GitHub and click on the **Draft a new release** button.
8. **Fill out release information:** Fill out the release information, such as the tag version, release title, and release description.
9. **Upload release assets:** Upload the build artifact(s) or release assets (for example, binaries, configuration files, and more) to the release.
10. **Publish the release:** Click the **Publish release** button to publish the release.
11. **Deploy the release:** Deploy the release to the production environment. This may involve uploading the package to a server, running a deployment script, or using a deployment tool (for example, Ansible, Chef, Puppet, and more).
12. **Verify deployment:** Verify that the application is running as expected in the production environment.
13. **Update release notes:** Update the release notes with information about the new release and any changes made.
14. **Rollback Plan:** Develop a rollback plan and ensure it is integrated into your CI/CD pipeline. This allows for quick and controlled rollbacks in case of issues in the production environment.
15. **Security considerations:** Implement security checks and scans as part of your automated testing process to ensure that code changes do not introduce security vulnerabilities.
16. **Notify stakeholders:** Notify stakeholders (including customers, users, and more) about the new release and any changes made.

Go to **Releases** of your repository, click **Create new Release**, create release as shown in *Figure 9.3* and click **Publish Release**.

Software release:

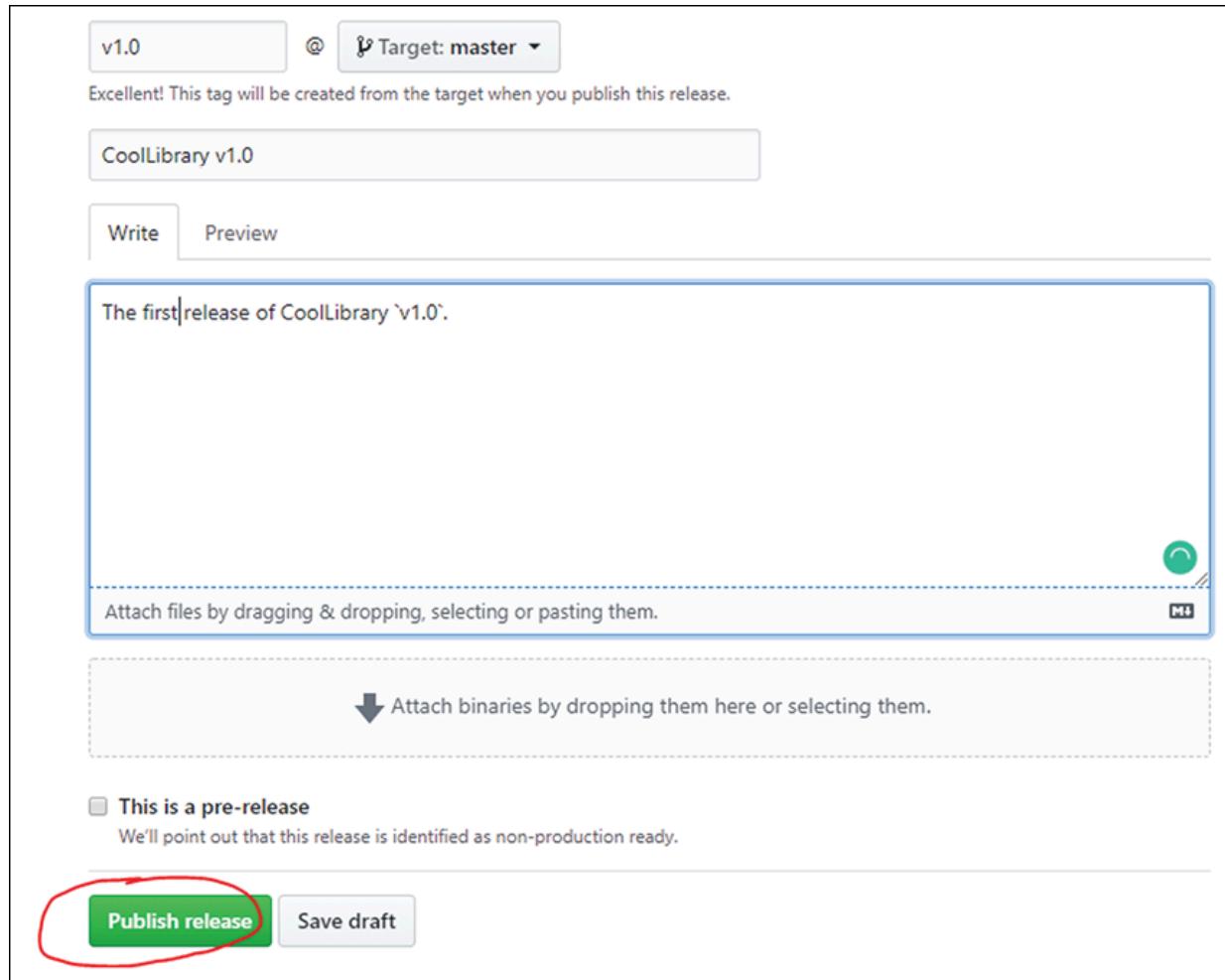


Figure 9.3: Publish release

By following these steps, you can create a new release version and publish it to production using GitHub. It is important to have a well-defined and tested release process in place to minimize downtime and ensure the stability of the production environment. Additionally, it is recommended to use tools and automation to streamline the release process and minimize the potential for errors.

Here is a step-by-step guide on how to revert to a previous release version in GitHub:

1. Go to the repository's **Releases** page on GitHub.
2. Locate the release version you want to revert to.
3. Click on the **Release** to open it.
4. Download the release asset(s) for the version you want to revert to.

5. Deploy the `releasedeploy` the downloaded release asset(s) to the production environment.
6. **Verify deployment:** Verify that the application is running as expected in the production environment.
7. If necessary, update the release notes with information about the reversion.
8. Notify stakeholders (customers, users, and so on) about the reversion, refer to the following figure:

Release Version	Published Date	Commit Hash	File Type
1.7.0	18 days ago	3985d8c	zip, tar.gz
1.6.1	25 days ago	a0d1891	zip, tar.gz
1.6.0	25 days ago	2559f59	zip, tar.gz
1.5.4	on 10 Mar	b0cfe5d	zip, tar.gz

Figure 9.4: List of all the releases

It is important to note that reverting to a previous release version may cause data loss or other unintended consequences, especially if the changes made in the newer release were significant. Therefore, it is recommended to thoroughly test the previous release version before deploying it to production and to have a backup plan in case the reversion is not successful. Additionally, it is recommended to use version control and a well-defined release process to minimize the potential for errors and ensure the stability of the production environment.

In summary, version control can be used as a backup and restore strategy for applications by storing the entire source code and building artifacts over time. While it has some limitations, it offers several benefits and should be used in conjunction with other backup and restore strategies to ensure comprehensive data protection.

Hotfix rollback strategy

A hotfix is a patch or update that is released to address critical issues or bugs in a software application. Hotfixes are often deployed quickly to resolve urgent problems, but they can also introduce new issues if they are not properly tested. In such cases, a hotfix rollback strategy is essential to revert the application to a stable state quickly.

Here are some steps for implementing a hotfix rollback strategy:

1. **Define rollback conditions:** Include application unique requirements before implementing a hotfix. It is important to define the conditions under which the hotfix will be rolled back. For example, if the hotfix causes a 20% increase in error rates or if it negatively impacts user engagement by more than 10%, the hotfix should be rolled back.
2. **Monitor key metrics:** It is important to monitor key metrics such as error rates, user engagement, and performance after the hotfix is deployed. If any of these metrics are negatively impacted, the hotfix should be rolled back immediately.
3. **Rollback plan:** A rollback plan should be in place before the hotfix is deployed. The rollback plan should include steps for disabling the hotfix, deploying a stable version of the application, and communicating with users.
4. **Test the rollback plan:** The rollback plan should be tested thoroughly to ensure that it can be executed quickly and efficiently in case of any issues.
5. **Communicate with users:** Communication is key when it comes to rolling back a hotfix. Users should be informed about the rollback and any impact it may have on their experience with the application.
6. **Evaluate and adjust:** Evaluating and adjusting a hotfix before implementation helps to ensure that the fix is well-prepared, tested, and tailored to the specific needs of the system, which ultimately leads to more effective and efficient problem resolution with minimized risks.
7. **Post-mortem analysis:** After the rollback, a post-mortem analysis should be conducted to identify the root cause of the issue and to prevent similar issues from occurring in the future.

Git workflow:

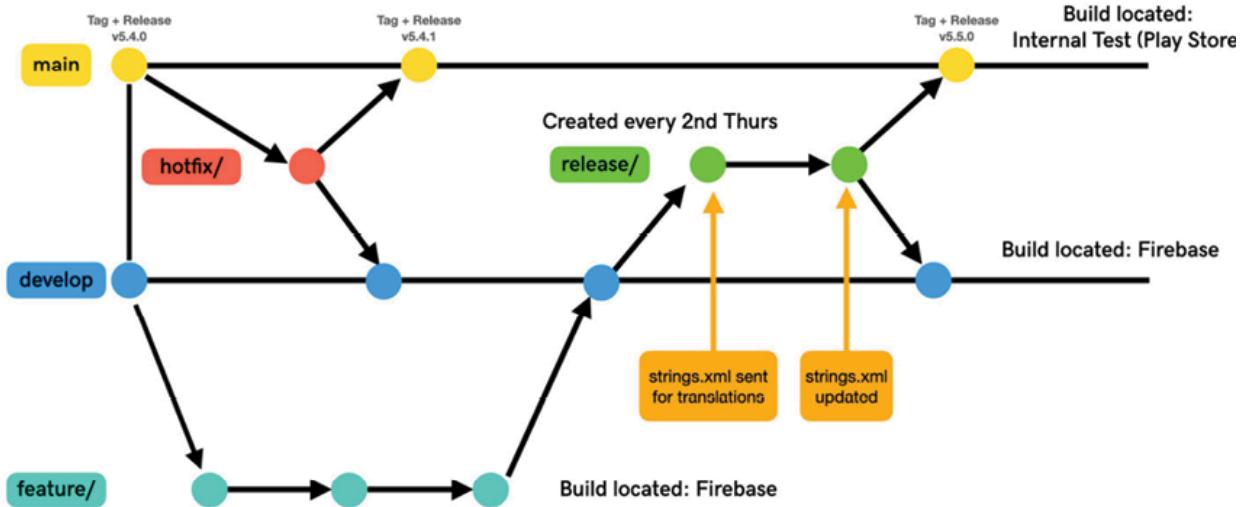


Figure 9.5: Hotfix git flow

In summary, implementing a hotfix rollback strategy involves defining rollback conditions, monitoring key metrics, having a rollback plan in place, testing the rollback plan, communicating with users, and conducting a post-mortem analysis. By following these steps, software development teams can quickly address critical issues while minimizing the impact of any potential issues introduced by the hotfix.

Feature toggles rollback strategy

Feature toggles are a technique used in software development to enable and disable specific features or functionalities of an application without having to deploy a new version. Feature toggles are often used to control the release of new functionality to specific groups of users or to perform A/B (split) testing. A/B testing is used to test multiple versions of a feature or design with different user groups to determine which is more effective, while feature toggles are used to control the release of features to specific user groups or to enable/disable features based on specific conditions.

However, when a feature toggle is turned on, there is always a risk of encountering issues or bugs that were not discovered during testing. In such cases, a rollback strategy is essential to minimize the impact of the issue and quickly revert the application to a stable state.

Here are some steps for implementing a feature toggle rollback strategy:

- 1. Define rollback conditions:** Before implementing a feature toggle, it is important to define the conditions under which it will be rolled back. For

example, if the feature toggle causes a 20% increase in error rates or if it negatively impacts user engagement by more than 10%, the feature toggle should be rolled back.

2. **Monitor key metrics:** It is important to monitor key metrics such as error rates, user engagement, and performance when the feature toggle is enabled. If any of these metrics are negatively impacted, the feature toggle should be rolled back immediately.
3. **Rollback plan:** A rollback plan should be in place before the feature toggle is enabled. The rollback plan should include steps for disabling the feature toggle, deploying a stable version of the application, and communicating with users.
4. **Test the rollback plan:** The rollback plan should be tested thoroughly to ensure that it can be executed quickly and efficiently in case of any issues.
5. **Communicate with users:** Communication is key when it comes to rolling back a feature toggle. Users should be informed about the rollback and its impact on their experience with the application.
6. **Post-mortem analysis:** After the rollback, a post-mortem analysis should be conducted to identify the root cause of the issue and to prevent similar issues from occurring in the future.

Following figure explains the feature branch workflow:

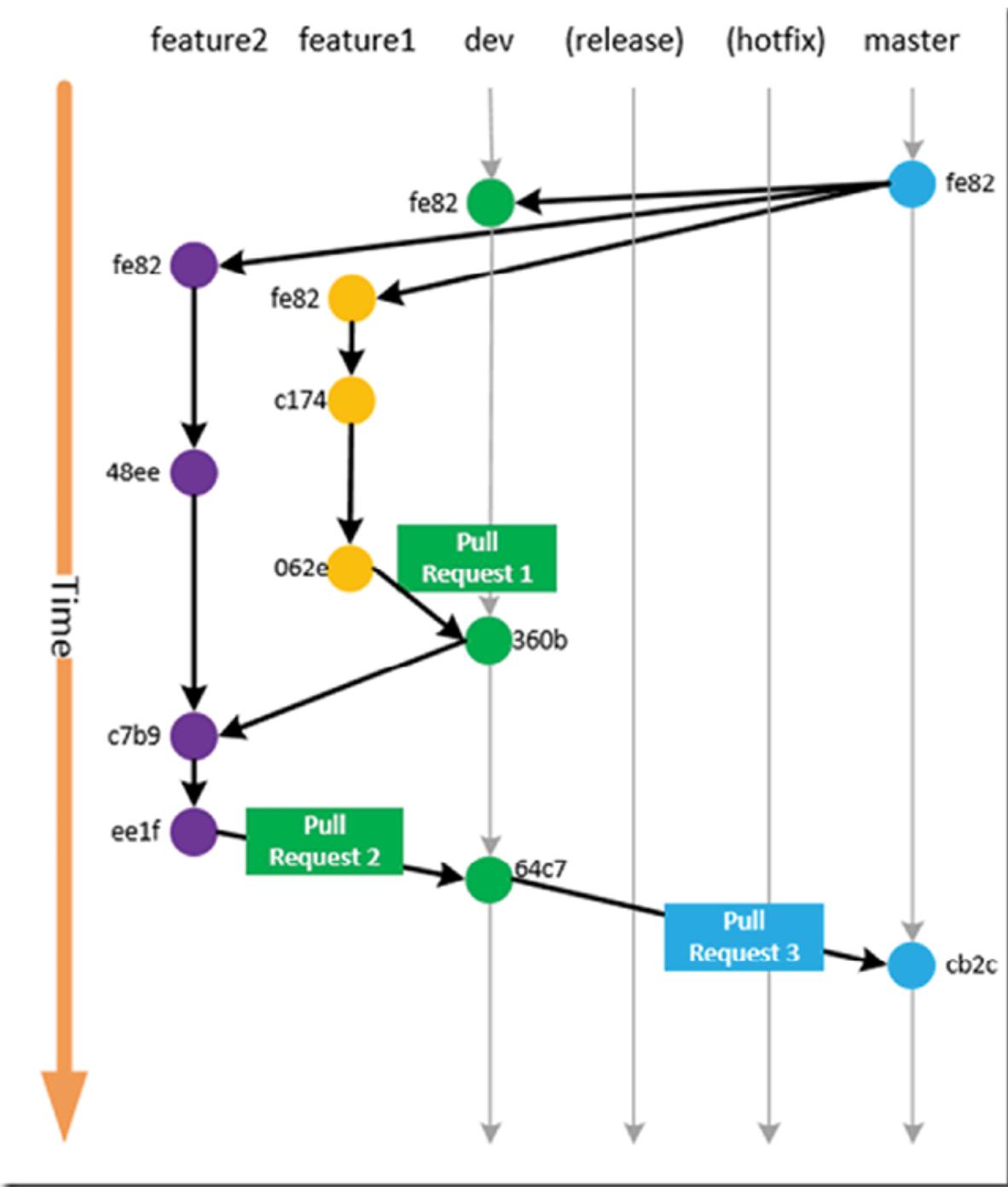


Figure 9.6: Feature branch flow

In summary, implementing a feature toggle rollback strategy involves defining rollback conditions, monitoring key metrics, having a rollback plan in place, testing the rollback plan, communicating with users, and conducting a post-mortem analysis. By following these steps, software development teams can

minimize the impact of issues that arise when feature toggles are enabled and ensure a seamless user experience.

Immutable infrastructure rollback strategy

Immutable infrastructure is a deployment strategy that involves creating and deploying infrastructure components as read-only objects that cannot be changed once deployed. This strategy enables quick and easy rollbacks since the previous version of the infrastructure can simply be redeployed. Here are some in-depth details about immutable infrastructure as a rollback strategy:

- **Definition:** Immutable infrastructure is a deployment strategy that involves creating and deploying infrastructure components as immutable objects that cannot be changed once deployed. Any changes to the infrastructure are made by creating new objects rather than modifying existing objects.
- **Benefits:** Immutable infrastructure offers several benefits as a rollback strategy. Since infrastructure components are deployed as immutable objects, it is easy to roll back changes by redeploying the previous version of the infrastructure. This approach also reduces the risk of configuration drift and ensures consistency across multiple deployments.
- **Implementation:** Immutable infrastructure can be implemented using various tools and frameworks, such as Docker, Kubernetes, or Terraform. These tools enable developers to create and deploy infrastructure components as immutable objects that can be easily rolled back if necessary.
- **Deployment process:** The deployment process for immutable infrastructure involves creating a new object with the updated configuration, testing it thoroughly, and then deploying it to production. The previous version of the infrastructure is left in place until the new version has been thoroughly tested and is ready for deployment. This approach ensures that there is always a stable version of the infrastructure in place that can be rolled back if necessary.
- **Rollback process:** The rollback process for immutable infrastructure is straightforward. To roll back changes, the previous version of the infrastructure is redeployed. Since the previous version is an immutable object, it is guaranteed to be identical to the original version, and there is no risk of configuration drift.

- **Impact analysis:** Immutable infrastructure requires careful impact analysis to ensure that rolling back changes do not affect other parts of the system. Since the infrastructure is deployed as read-only objects, it is important to ensure that rolling back changes does not cause unexpected behavior or errors in other system parts.
- **Monitoring:** Monitoring is an important aspect of immutable infrastructure. Since the infrastructure is deployed as read-only objects, monitoring the objects for any changes or modifications is important. This can be done using various monitoring tools and frameworks, such as Prometheus or Grafana.

Docker image build workflow:

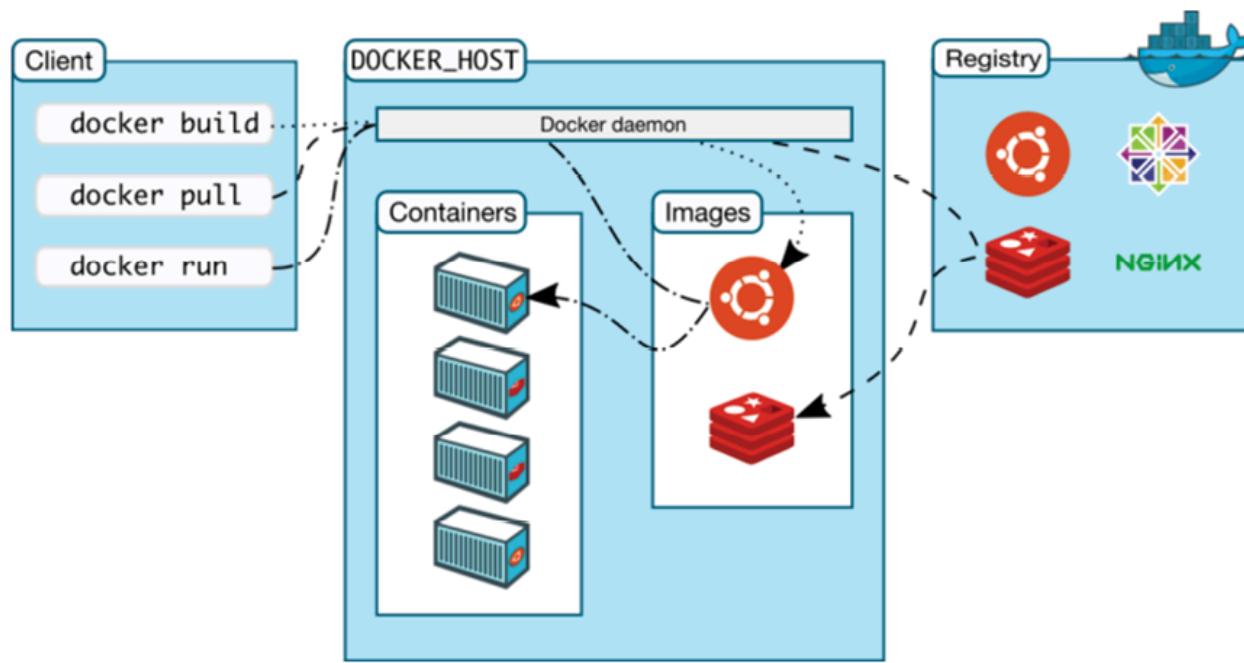


Figure 9.7: Docker build image process

Overall, immutable infrastructure is a powerful deployment strategy that enables quick and easy rollbacks. It ensures consistency across multiple deployments and reduces the risk of configuration drift. However, immutable infrastructure requires careful testing, deployment, and impact analysis to ensure that rolling back changes does not cause unexpected behavior or errors in other parts of the system.

Canary deployment

Canary deployment, also known as canary release or canary testing, is a deployment strategy used in DevOps to roll out new code changes or updates

gradually and minimize the risk of issues or downtime. The canary deployment strategy involves releasing a new version of the application to a small subset of users or servers, known as the ‘canary group’, before rolling it out to the entire production environment.

The canary deployment process typically involves the following steps:

1. **Prepare the release:** Before the release, the new code changes are tested, a build is created, and artifacts are uploaded to the artifact repository.
2. **Deploy to canary:** The new version of the application is deployed to a small subset of users or servers, which represents a small percentage of the overall user base. This canary group can be selected based on various criteria, such as geographic location, user activity, or server configuration.
3. **Monitor and verify:** The performance and stability of the new version of the application are closely monitored by DevOps teams, using various monitoring and analytics tools. If issues or errors are detected, the deployment can be halted, and the issues can be addressed before the release is rolled out to the broader user base.
4. **Rollout:** Once the canary release is verified as stable and reliable, the new version is gradually rolled out to the rest of the production environment. This can be done gradually, with more users or servers added to the release pool over time until the entire user base is updated.

The canary deployment strategy is beneficial in several ways, including:

- **Risk mitigation:** By releasing the new version of the application to a small subset of users or servers, DevOps teams can identify and resolve any issues or errors before they affect the entire user base, minimizing downtime and reducing the risk of issues.
- **Better testing:** Canary deployment provides an opportunity to test the new code changes in a real-world environment, with real user traffic and usage patterns, providing valuable insights and data for further optimization and improvements.
- **Improved user experience:** Canary deployment helps to ensure that the new version of the application is stable, reliable, and performs well before being released to the broader user base, resulting in a better user experience and increased user satisfaction.
- **Early detection** in Canary deployment is essential to minimize the potential negative impact on users and the business. By identifying problems at an

early stage, teams can take corrective actions promptly, whether that involves fixing issues in the new release or rolling back to a stable version.

- **Canary deployment** helps in catching and addressing performance issues proactively, reducing the risk of performance-related outages or slowdowns for the majority of users. It provides a controlled and measured approach to deploying and verifying software releases in production, ultimately leading to a more reliable and high-performance application.

Canary deployment is a strategy that involves rolling out new features or updates gradually to a subset of users or servers before deploying them to the entire system. In Kubernetes, canary deployment can be achieved by using the built-in feature called **Deployment Strategies**.

Here are the steps to set up canary deployment with Kubernetes:

1. Create a new deployment with the desired configuration for the canary deployment. This can be done using the Kubernetes Deployment YAML file.
2. Create a new service for the canary deployment that exposes the pods of the deployment to the internet or to other services in the cluster.
3. Create a new deployment that will act as a stable deployment. This will be the deployment that the canary deployment will gradually replace.
4. Create a new service for the stable deployment that exposes the pods of the deployment to the internet or to other services in the cluster.
5. Set up the canary deployment to gradually replace the stable deployment. This can be done by using the Kubernetes Deployment Strategies. For example, you can use the `RollingUpdate` strategy with a `maxSurge` and `maxUnavailable` value to gradually increase the number of canary pods while decreasing the number of stable pods.
6. Monitor the canary deployment and make sure it is stable before scaling it up to replace the stable deployment completely.
7. Once the canary deployment is stable and has been tested thoroughly, scale it up to replace the stable deployment completely. This can be done by using the Kubernetes Deployment Strategies to gradually decrease the number of stable pods while increasing the number of canary pods.

The parameters in the manifest file can be adjusted to fine-tune rolling updates.
Sample of Kubernetes manifest for Canary deployment:

```

spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  template:
...

```

Figure 9.8: Kubernetes manifest for Canary deployment

In the following figure, canary deployment is shown:

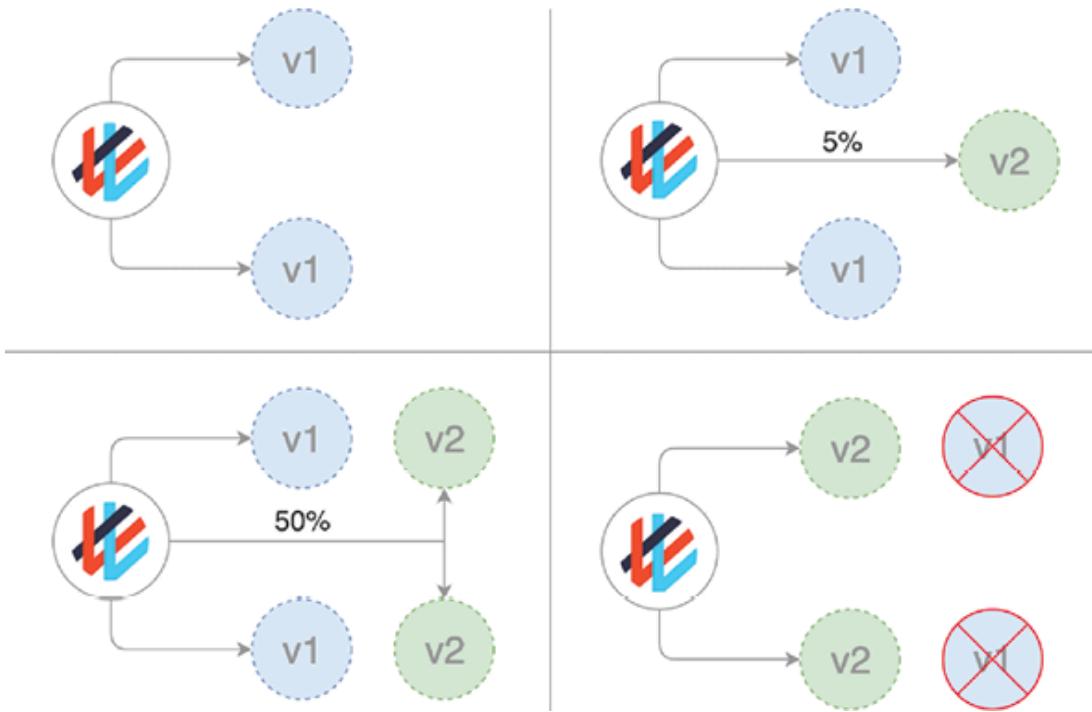


Figure 9.9: Canary deployment

In conclusion, canary deployment is an effective DevOps strategy for minimizing the risk of issues or downtime during code releases or updates. By gradually rolling out changes to a small subset of users or servers and closely monitoring performance and stability, DevOps teams can ensure that the new version of the application is reliable and performs well before releasing it to the broader user base.

A/B testing deployment strategy

A/B testing is a deployment strategy that involves releasing two different versions of an application or feature to a small subset of users and comparing the results to determine which version is better. Here are some in-depth details about A/B testing as a deployment strategy:

- **Definition:** A/B testing is a deployment strategy that involves releasing two different versions of an application or feature to a small subset of users and comparing the results to determine which version is better. This approach allows developers to test new features or changes in a controlled environment before rolling them out to the entire user base.
- **Benefits:** A/B testing offers several benefits as a deployment strategy. It enables developers to test new features or changes in a controlled environment, which helps to reduce the risk of issues or errors. A/B testing also allows developers to measure the impact of changes and make data-driven decisions about which version to release to the entire user base.
- **Implementation:** A/B testing can be implemented using various tools and frameworks, such as Google Optimize, Optimizely, or Adobe Target. These tools enable developers to create and test multiple versions of an application or feature and measure the impact of each version.
- **Deployment process:** The deployment process for A/B testing involves creating two different versions of an application or feature and releasing them to a small subset of users. The results are then measured and analyzed to determine which version is better. The better version is then released to the entire user base.
- **Testing:** A/B testing requires thorough testing to ensure that the results are accurate and reliable. It is important to test the two different versions of an application or feature in different environments, such as development, staging, and production, to ensure that they work in all environments.
- **Impact analysis:** A/B testing requires careful impact analysis to ensure that the results are valid and reliable. It is important to measure the impact of each version on various metrics, such as user engagement, conversion rates, or revenue, to ensure that the results are accurate and meaningful.
- **Continuous monitoring:** Continuous monitoring is an important aspect of A/B testing. It is important to monitor the results of A/B testing continuously and make adjustments as necessary. This can help to improve the accuracy and reliability of the results and ensure that the better version is released to the entire user base.

Here are the steps to set up A/B deployment with Kubernetes:

1. Create two deployments with different configurations for the A/B deployment. This can be done using the Kubernetes Deployment YAML file.
2. Create two services for the A/B deployment that expose the pods of each deployment to the internet or to other services in the cluster.
3. Configure the Service Mesh to split the traffic between the two services. This can be done using the Kubernetes Service Mesh features like Istio or Linkerd. The traffic split can be based on a variety of factors like percentage, HTTP headers, or client IP addresses.
4. Monitor the performance of both versions of the application or service using metrics and logs.
5. Once you have determined which version performs better, you can route all traffic to the better-performing version. This can be done by adjusting the traffic split in the Service Mesh to 100% for the better-performing version.

Different types of A/B (Split) deployment:

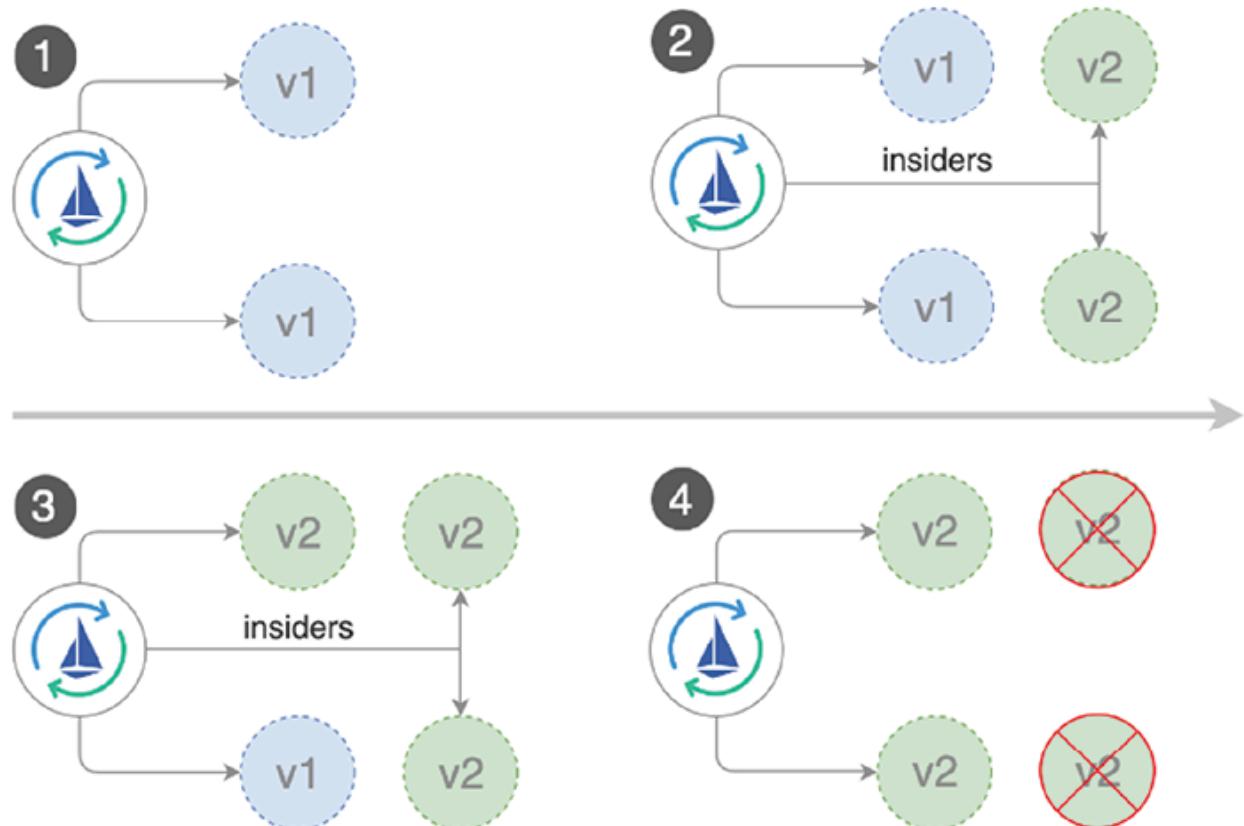


Figure 9.10: A/B Deployment

Overall, A/B testing is a powerful deployment strategy that enables developers to test new features or changes in a controlled environment and make data-driven decisions about which version to release to the entire user base. However, A/B testing requires careful testing, impact analysis, and continuous monitoring to ensure that the results are accurate and reliable.

Blue-green deployment

Blue-green deployment is a popular deployment strategy used in DevOps that involves maintaining two identical environments with only one environment active at a time. The application is initially deployed to the inactive environment, allowing for testing and verification of the new version without affecting the active environment. Once the new version is deemed stable, traffic is redirected from the active (blue) environment to the inactive (green) environment, effectively switching the active environment. If any issues arise, traffic can be quickly redirected back to the previous stable environment.

Here are some in-depth details about blue-green deployment:

- **Maintaining two identical environments:** Blue-green deployment involves maintaining two identical environments, the blue and green environments, with identical infrastructure, configuration, and data. The only difference between the two environments is the application version running on each.
- **Zero downtime deployment:** Blue-green deployment is designed to minimize downtime during the deployment process. The new version of the application is deployed to the inactive environment, allowing for testing and verification without affecting the active environment. Once the new version is deemed stable, traffic is switched from the active to inactive environments, resulting in zero downtime.
- **Quick rollback:** Blue-green deployment makes it easy to roll back to the previous version of the application in case of issues or errors. If any issues are encountered during the deployment process, traffic can be quickly redirected back to the previous stable environment.
- **Reduced risk:** Blue-green deployment helps to reduce the risk associated with deploying new versions of an application. Since the new version is deployed to a separate environment, it can be thoroughly tested and verified before being made active.

- **Increased flexibility:** Blue-green deployment offers increased flexibility compared to traditional deployment strategies. It allows for multiple versions of the application to be deployed simultaneously, and it makes it easy to switch between versions in case of issues or errors.
- **Traffic switching:** Traffic switching is the key component of blue-green deployment. Once the new version of the application is deployed and tested, traffic is switched from the active environment to the inactive environment. This can be done using various traffic-switching mechanisms, such as load balancers or DNS changes.
- **Automation:** Blue-green deployment can be automated using various tools and frameworks, such as Jenkins, Ansible, or Kubernetes. This makes it easy to manage the deployment process and ensures consistency across multiple deployments.

In Kubernetes, Blue-green deployment can be achieved by using the following steps:

1. Create two deployments with identical configurations for the Blue-Green Deployment. This can be done using the Kubernetes Deployment YAML file.
2. Create two services for the Blue-Green Deployment that expose the pods of each deployment to the internet or to other services in the cluster.
3. Use a load balancer to direct all traffic to the service for the *blue* environment.
4. Deploy and test the new version of the application or service in the *green* environment.
5. Once the *green* environment is ready, switch the load balancer to direct all traffic to the service for the *green* environment.
6. Monitor the *green* environment for any issues, and switch back to the *blue* environment if necessary.

Once you are confident that the *green* environment is stable and performing well, you can delete the deployment and service for the *blue* environment.

Different types of Blue-green deployment:

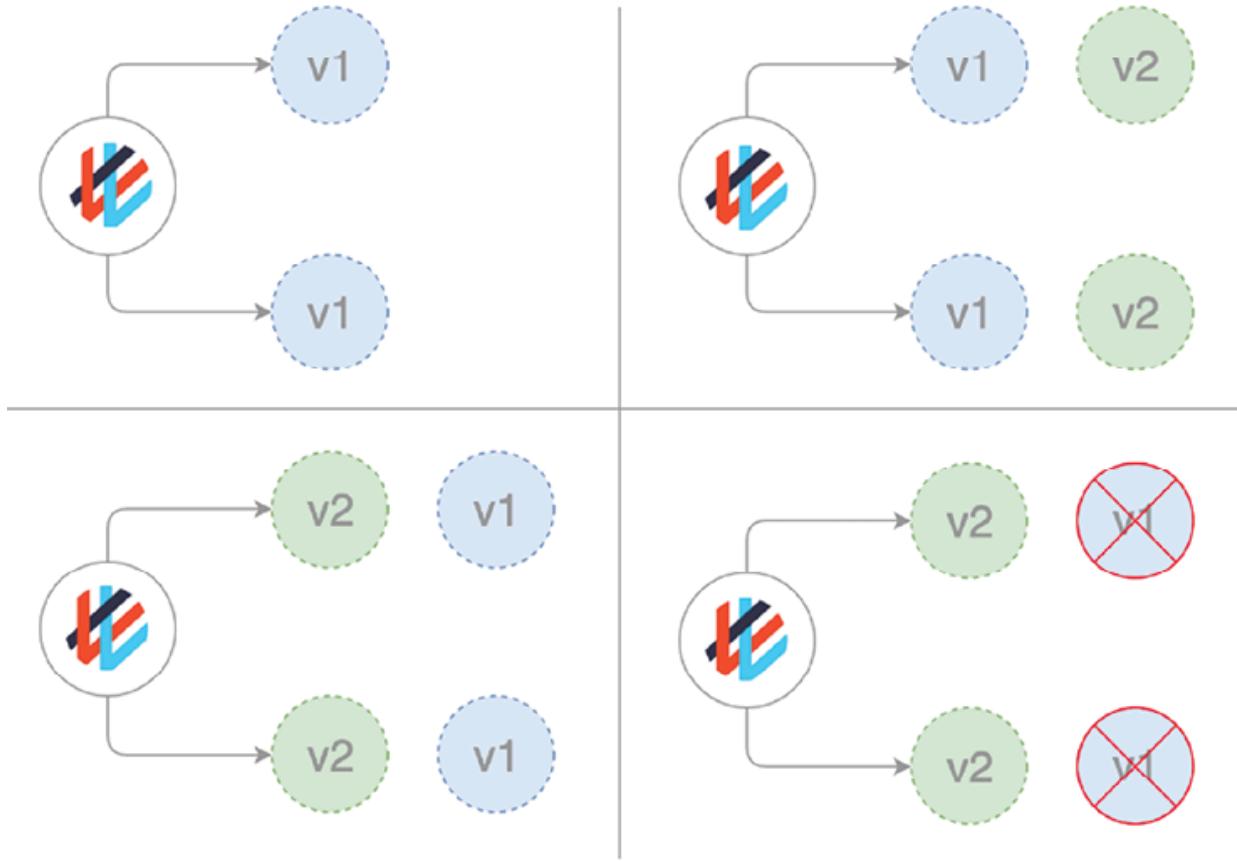


Figure 9.11: Blue-green deployment

Overall, blue-green deployment is a powerful deployment strategy that can help to reduce deployment risk, minimize downtime, and increase flexibility. It is well-suited for larger-scale deployments or major releases where thorough testing and verification are required before making the new version of the application active.

Shadow deployment

Shadow deployment, also known as shadow testing or canary testing, is a technique used in software development to test new features or changes in a live environment without impacting users. It involves deploying a new version of the application to a subset of users, while leaving the existing version running for the rest of the users.

Here are some steps for implementing a shadow deployment:

- 1. Identify the subset of users:** The first step is to identify a subset of users who will receive the new version of the application. This subset should be representative of the entire user base and should be large enough to provide meaningful data but small enough to minimize the impact of any issues.

2. **Deploy the new version:** The new version of the application is deployed to the subset of users while the existing version continues to run for the rest of the users.
3. **Monitor key metrics:** Key metrics such as error rates, performance, and user engagement should be monitored for both the new and existing versions of the application. This will allow developers to compare the two versions and identify any differences in behavior.
4. **Analyze the results:** After a sufficient amount of time has passed, the results of the shadow deployment should be analyzed to determine whether the new version of the application is ready to be deployed to the entire user base. If any issues are detected, they can be addressed before deploying the new version to all users.
5. **Deploy the new version:** Once the new version has been tested and any issues have been addressed, it can be deployed to the entire user base.

In Kubernetes, Shadow deployment can be achieved by using the following steps:

1. Create two deployments with identical configurations for the Shadow deployment. This can be done using the Kubernetes deployment YAML file.
2. Create two services for the Shadow deployment that expose the pods of each deployment to the internet or to other services in the cluster.
3. Use a traffic splitter to split a small portion of traffic, say 1%, to the service for the new version while directing the rest of the traffic to the service for the existing version. This can be done using the Kubernetes Service Mesh features like Istio or Linkerd.
4. Monitor the performance of both versions of the application or service using metrics and logs. Compare the performance and behavior of the new version with the existing version to see if there are any differences.
5. Once you are confident that the new version is performing well and there are no issues, you can increase the traffic split to direct more traffic to the new version.
6. Continue monitoring the performance of both versions and gradually increase the traffic split until all traffic is directed to the new version.
7. Once all traffic is directed to the new version, you can delete the deployment and service for the existing version.

Following is a diagram for Shadow deployment workflow:

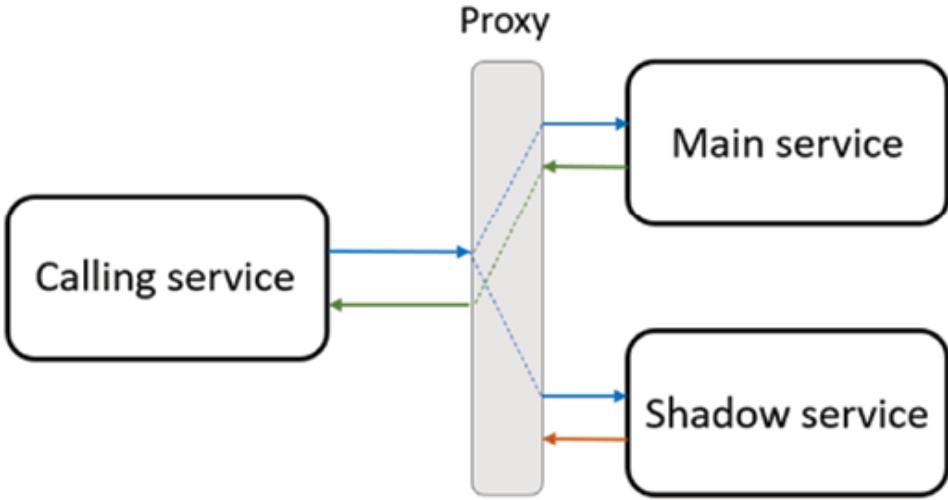


Figure 9.12: Shadow deployment

In summary, implementing a shadow deployment involves identifying a subset of users, deploying the new version of the application to that subset, monitoring key metrics, analyzing the results, and deploying the new version to the entire user base. By following these steps, software development teams can test new features or changes in a live environment without impacting users and ensure a smooth transition to the new version of the application.

Conclusion

In conclusion, rollback strategies play a critical role in DevOps deployment by providing a safety net that can be used to quickly and effectively recover from unexpected errors or issues that arise during the deployment process. With the increasing complexity of modern software systems, having a robust and reliable rollback strategy is essential to ensuring the stability and reliability of the entire system.

We have covered several types of rollback strategies in this topic, and each has its own strengths and weaknesses. The feature toggle rollback strategy, for example, is an effective approach that allows developers to disable problematic features or changes quickly and easily, while the immutable infrastructure rollback strategy provides a more comprehensive approach to rollback by ensuring that the previous version of infrastructure can be redeployed quickly and easily.

The blue-green deployment and A/B testing rollback strategies offer a controlled environment for testing changes and can help mitigate the risk of issues or errors. However, these approaches also require thorough testing and impact analysis to ensure that the results are accurate and reliable.

Ultimately, the choice of a rollback strategy depends on the specific needs and goals of the project. However, regardless of the chosen strategy, it is important to ensure that the deployment process is reliable, stable, and can be reverted to a previous state if necessary. By prioritizing the implementation of a robust rollback strategy, developers can mitigate risk, reduce downtime, and improve the overall stability and reliability of the system.

In the next chapter, we are going to cover Automated Infrastructure (Infrastructure as a code). We will learn how to enable organizations to deploy, configure, and manage their infrastructure in a scalable and efficient manner with the help of Automated Infrastructure (IaC).

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

[https://discord.bpbonline.com](https://discord(bpbonline.com)



CHAPTER 10

Automated Infrastructure

Introduction

Automated infrastructure refers to the practice of using automation tools and processes to manage, provision, configure, and maintain the underlying infrastructure of an IT environment. This includes physical servers, virtual machines, cloud resources, networking components, storage, and more. The primary goal of automated infrastructure is to streamline operations, improve efficiency, reduce manual intervention, and ensure consistency in infrastructure management.

Structure

In this chapter, we will learn why automated testing is required in the software development lifecycle. This section explains the benefits, best practices, tools, and how to configure automation testing:

- Infrastructure as Code
- Pipeline as Code
- Platform as Code
- Policy as Code
- GitOps methodology
- Best tools for IaC

Objectives

In this chapter, we will learn how to improve the overall quality and efficiency of the software development process by automating repetitive and time-consuming testing tasks. We will also delve into how to minimize the impact of issues and improve the traceability of tests by integrating automated tests into the CI/CD pipeline and monitoring the system's performance over time.

Infrastructure as Code

Infrastructure as Code (IaC) is a software engineering practice that manages and provides computer infrastructure through machine-readable definition files rather than manual configuration and management processes. It is a critical component of DevOps and is used to automate infrastructure resource deployment, configuration, and management.

Traditionally, setting up and managing infrastructure resources such as servers, networks, storage, and other components involved a lot of manual work. System administrators would manually configure servers, install software, and handle various dependencies. This approach was time-consuming, error-prone, and needed more scalability.

With IaC, the entire infrastructure stack is treated as code. Infrastructure resources are defined, provisioned, and managed using code-based definition files. These files can be version-controlled, shared, and automated, enabling teams to collaborate effectively and manage infrastructure predictably and consistently.

IaC involves using declarative or imperative programming languages or configuration management tools to define the desired state of the infrastructure. Declarative languages, such as YAML or JSON, describe the desired end-state of the infrastructure. Imperative languages, such as scripting or **Domain-Specific Languages (DSLs)**, provide step-by-step instructions to reach the desired state.

Here are some key concepts and benefits of IaC:

- **Version Control:** Infrastructure code can be stored and versioned using popular version control systems like Git. This enables tracking

changes, collaboration, and rollbacks, similar to application code.

- **Reproducibility:** IaC allows the recreation of infrastructure environments in a consistent and reliable manner. By defining infrastructure as code, teams can replicate the exact configuration across different domains, reducing inconsistencies and ensuring reliable deployments.
- **Automation:** Infrastructure provisioning and management tasks can be automated using IaC tools. This eliminates manual intervention, reduces human error, and increases operational efficiency.
- **Scalability:** Infrastructure can be dynamically provisioned and scaled up or down based on demand. With IaC, you can define templates or scripts that automatically adjust resource allocation based on workload requirements.
- **Collaboration:** Multiple team members can share and collaborate on infrastructure code. This fosters collaboration and knowledge sharing, allowing teams to work together to improve and optimize infrastructure configurations.
- **Auditing and compliance:** IaC provides an audit trail for infrastructure changes, making tracking and understanding modifications easier. It also helps enforce compliance by ensuring infrastructure adheres to specific security and regulatory standards.
- **Disaster recovery:** Infrastructure code can quickly rebuild environments in case of failures or disasters. By defining infrastructure as code, recovery processes can be automated and executed rapidly, minimizing downtime.
- **Resource efficiency:** IaC contributes significantly to resource efficiency by enabling precise provisioning, dynamic scaling, standardization, automation, and continuous optimization of infrastructure resources. This not only reduces operational overhead but also results in cost savings and a more sustainable, resource-efficient IT environment.

Popular IaC tools include Terraform, AWS CloudFormation, Azure Resource Manager, and Ansible. These tools provide a high level of abstraction and

allow you to define infrastructure resources and dependencies in a platform-agnostic way. Overall, IaC brings numerous benefits to managing infrastructure resources. It improves agility, reduces human error, enables scalability, and promotes collaboration, making it a fundamental practice in modern infrastructure management and deployment workflows.

IaC is a powerful tool that can help organizations improve reliability, reduce costs, improve security, and increase the agility of their infrastructure.

Here are some examples of how IaC can be used:

- To provision a new virtual machine
- To configure a firewall
- To deploy a new application
- To update a software package
- To patch a security vulnerability

IaC can be used to manage any type of infrastructure, including:

- Compute resources (for example, virtual machines, bare metal servers)
- Storage resources (for example, disks, file shares)
- Networking resources (for example, switches, routers, firewalls)
- Application resources (for example, web servers, databases)
- Security resources (for example, firewalls, intrusion detection systems)

There are many different IaC tools available. Some of the most popular IaC tools include:

- HashiCorp Terraform
- AWS CloudFormation
- Azure Resource Manager
- Google Cloud Deployment Manager

How to choose the best IaC tool

While it is subjective to determine the *best* tool for IaC, Terraform is widely regarded as a popular and influential choice among many practitioners. Here are several reasons why Terraform is often considered advantageous:

- **Multi-cloud and Multi-platform support:** Terraform supports a broad range of cloud providers, including AWS, Azure, and GCP. It also extends its support to on-premises infrastructure and various third-party services.
- **Declarative syntax:** Terraform uses a declarative approach to infrastructure provisioning, allowing you to specify the desired state of your infrastructure without explicitly defining the steps to reach that state.
- **Resource reusability:** Terraform promotes reusability through its module system. Modules allow you to encapsulate and abstract infrastructure configurations, making sharing and reusing them easier across projects.
- **Dependency management:** Terraform automatically manages dependencies between resources, ensuring the correct resource creation or modification order.
- **State management:** Terraform maintains a state file that keeps track of the current state of your infrastructure.
- **Execution plan:** Terraform provides a detailed execution plan before applying changes to the infrastructure. The execution plan helps you understand the modifications that Terraform will make, including resource creation, editing, or destruction.
- **Vibrant community and ecosystem:** Terraform has a thriving and rich ecosystem of providers, modules, and plugins. The community actively contributes to the development of Terraform, shares best practices, and creates reusable modules.
- **Integration with other tools:** Terraform integrates well with tools commonly used in DevOps and infrastructure management. It can be combined with configuration management tools like Ansible or Puppet to manage application-level configurations. Additionally, Terraform

integrates with popular CI/CD systems, allowing you to incorporate infrastructure changes into your deployment pipelines seamlessly.

The following are advantages of Terraform:

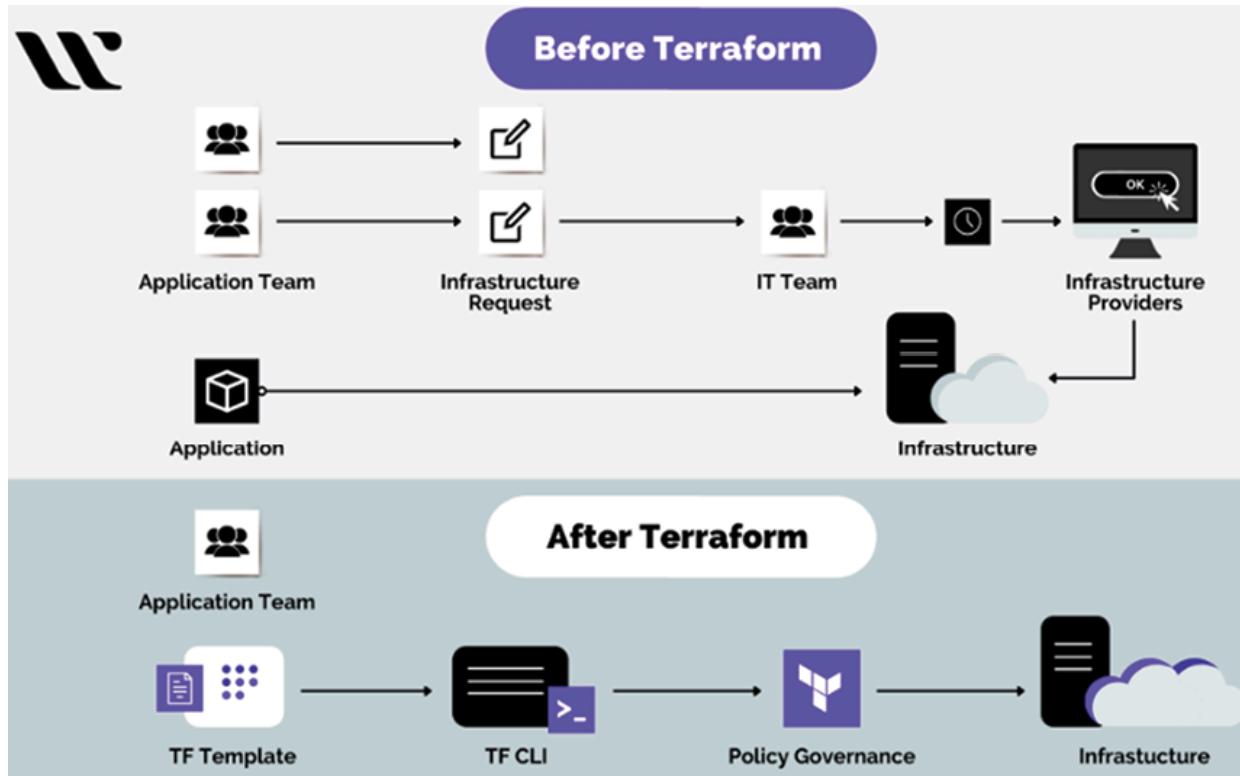


Figure 10.1: Advantages of Terraform

Introducing Terraform

Terraform is an open-source infrastructure provisioning and management tool developed by HashiCorp. It enables you to define, create, and manage infrastructure resources declaratively using configuration files. Terraform supports various cloud providers, as well as on-premises infrastructure.

Terraform flow chart:

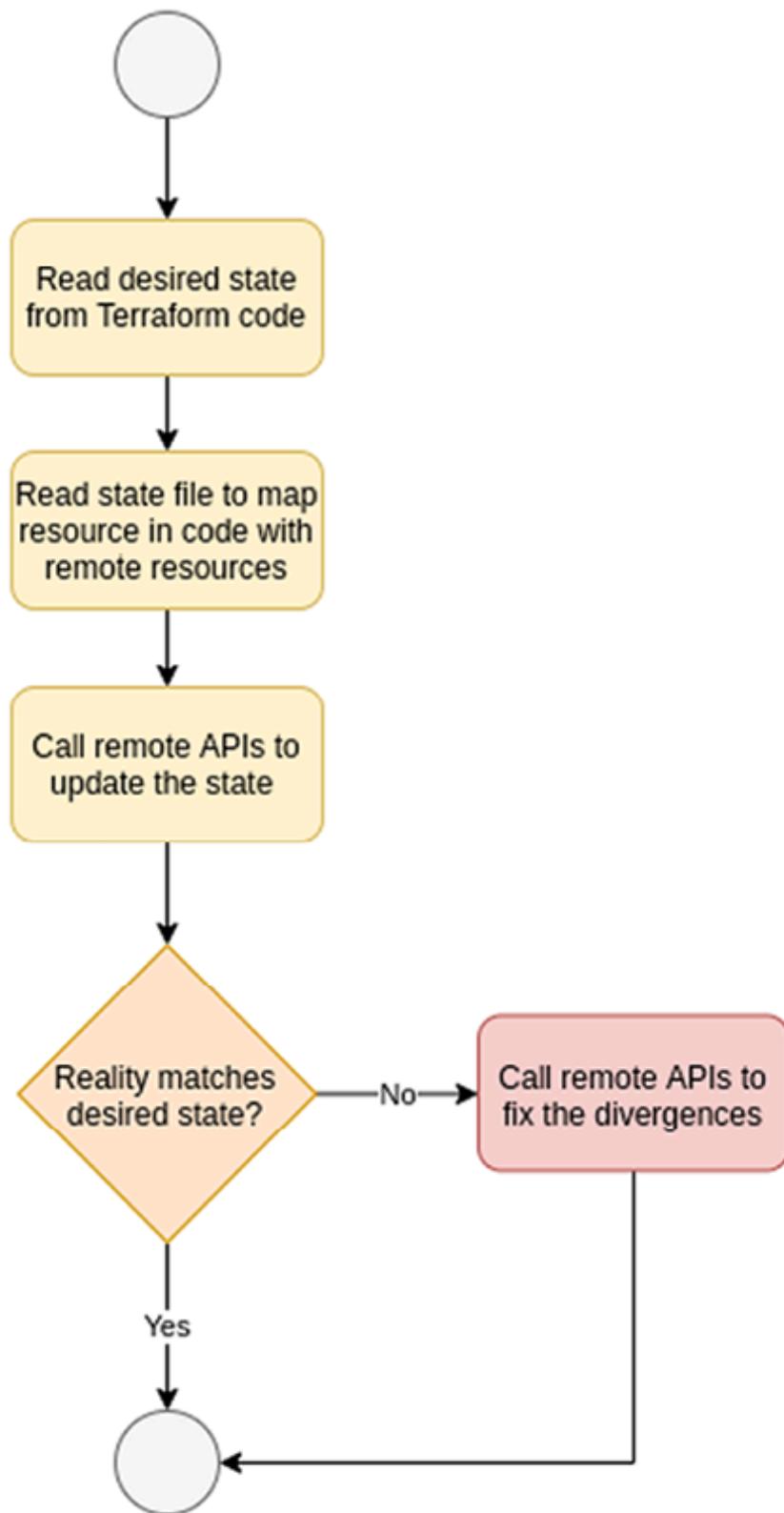


Figure 10.2: Terraform process.

The Terraform workflow is explained in the following points:

- Analyzing the HCL code, Terraform determines the intended state of the infrastructure.
- By examining the state file, Terraform identifies the resource identifiers it is responsible for managing.
- Verifying the disparities between the current state and the desired state, Terraform conducts a comparison. (*Figure 10.2*)
- Terraform takes the necessary actions to align the actual state with the desired state, ensuring consistency.

Terraform workflow:

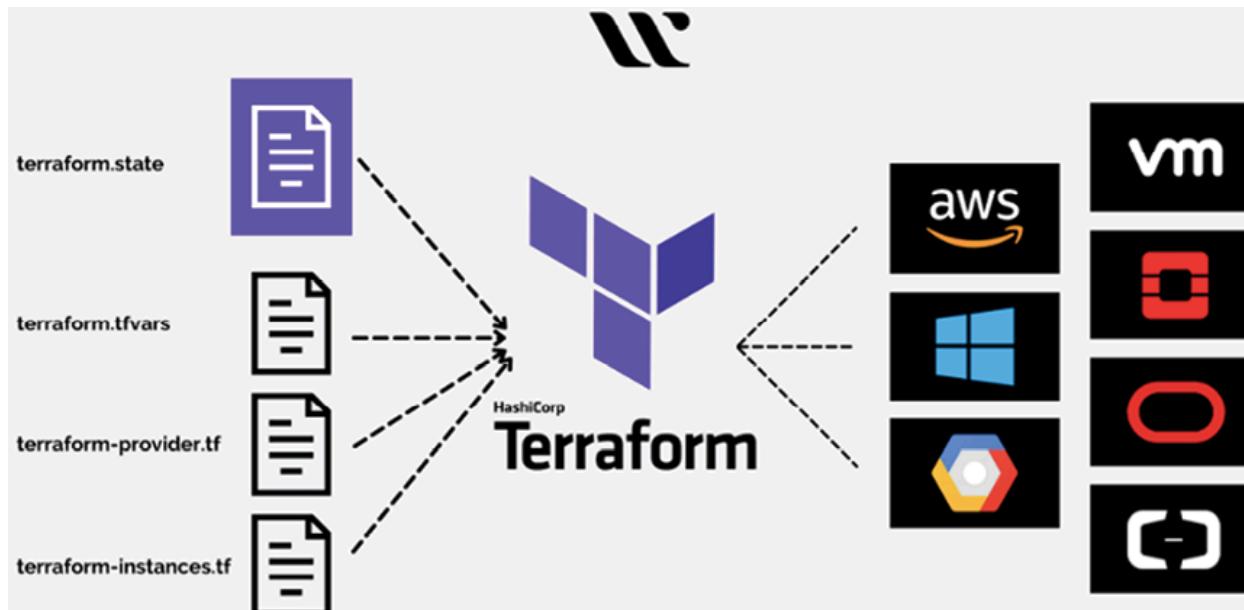


Figure 10.3: How terraform works.

Here is an example of using Terraform to create compute resources in AWS, specifically an EC2 instance:

```
# Declare the provider (AWS)
provider "aws" {

    region = "us-west-2"  # Replace with your desired
    region
```

```

}

# Define the EC2 instance resource

resource "aws_instance" "example" {

    ami              = "ami-0c94855ba95c71c99"      #
Replace with desired AMI ID

    instance_type   = "t2.micro"                      #
Replace with desired instance type

    key_name        = "my-key-pair"                  # Replace
with your SSH key pair name

    tags = {
        Name = "My EC2 Instance"
    }
}

```

In this example, we start by specifying the AWS provider block, indicating the region where we want to create the resources.

Next, we define an `aws_instance` resource called `example`. We set properties such as the **Amazon Machine Image (AMI)** ID, the instance type, and the SSH key pair name. You will need to replace these values with the ones appropriate for your setup. Additionally, we include a `tags` block to assign a name tag to the EC2 instance for better identification.

To use this Terraform configuration, follow these steps:

1. Install Terraform and ensure it is set up correctly on your system.
2. Create a new directory and place a file named `main.tf` in it.
3. Copy the above Terraform code into `main.tf`.
4. Open a terminal or command prompt, navigate to the directory containing `main.tf`, and run the following commands:

- `terraform init` (to initialize the Terraform working directory)
 - `terraform plan` (to preview the changes that Terraform will make)
 - `terraform apply` (to create the AWS resources)
5. When prompted, review the changes and type `yes` to confirm and proceed with the resource creation.

Terraform will then communicate with AWS, create the EC2 instance based on the provided configuration, and output the status and details of the created resources.

Please note that this is a basic example, and you can customize the configuration further based on your specific requirements by including additional resources, configuring networking, defining security groups, and more.

Pipeline as a Code

Pipeline as a Code refers to the practice of defining and managing software delivery pipelines using code. It represents the entire pipeline infrastructure, including the build, test, and deployment processes, as code artifacts that can be version-controlled, reviewed, and executed.

Traditionally, setting up and managing software delivery pipelines involved manual configuration and scripting, which could be error-prone and difficult to maintain. Pipeline as a Code introduces a more structured and scalable approach by treating pipelines as software, using DSLs or configuration files to define the pipeline steps, dependencies, and conditions.

Popular tools for implementing Pipeline as a Code include Jenkins with `Jenkinsfile`, GitLab CI/CD with `.gitlab-ci.yml`, and AWS CodePipeline with AWS CloudFormation or **AWS CDK (Cloud Development Kit)**.

Pipeline as a Code offers several benefits for software development and deployment processes. Some of the key advantages include:

- **Reproducibility and consistency:** By defining the pipeline as code, you can ensure consistent and reproducible execution of your software delivery process across different environments. Every step,

configuration, and dependency are explicitly defined in the code, reducing the risk of errors or inconsistencies due to manual setup or configuration.

- **Version control and collaboration:** Treating the Pipeline as Code allows you to leverage version control systems like Git to manage changes, track history, and collaborate effectively. Developers can review, comment, and propose changes to the pipeline just like they do with application code, promoting collaboration and ensuring pipeline changes go through proper review processes.
- **Scalability and flexibility:** With Pipeline as a Code, you can easily scale and adapt your pipelines to meet the evolving needs of your project. Code-driven pipelines can be modularized, reused, and shared across teams and projects, saving time and effort. They can also be easily modified or extended to incorporate new steps, stages, or integrations, enabling flexibility and agility in your software delivery process.
- **Automation and efficiency:** Automating the pipeline using code eliminates the need for manual intervention and repetitive tasks. This leads to increased efficiency and reduced human error. Automated pipelines can trigger builds, tests, and deployments based on defined conditions, enabling faster and more reliable software delivery.
- **IaC integration:** Pipeline as a Code can seamlessly integrate with IaC practices. You can define and provision the necessary infrastructure resources alongside your application deployment, ensuring consistent and automated infrastructure setup. This integration streamlines the overall deployment process and enhances the reproducibility of the application and its supporting infrastructure.
- **Auditability and traceability:** By representing your pipeline as code, you have improved visibility into the entire software delivery process. You can easily track, and audit changes made to the pipeline over time, enabling better traceability and compliance. This can be crucial in regulated industries or for maintaining a reliable software delivery process.

Following diagram is for software build and deploy lifecycle:



Figure 10.4: CICD pipeline

Here is an example of a `Jenkinsfile` that demonstrates a simple build and deploy pipeline for a web application:

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                // Checkout source code from  
                version control  
                git 'https://github.com/example/my-  
                web-app.git'  
                // Build the application  
                sh 'npm install' // or any build  
                command specific to your application  
            }  
        }  
    }  
}
```

```

stage('Test') {
    steps {
        // Run tests
        sh 'npm test' // or any test
        command specific to your application
    }
}

stage('Deploy') {
    steps {
        // Deploy the application
        sh 'npm run deploy' // or any
        deployment command specific to your application
    }
}
}

```

In this example, the `Jenkinsfile` defines a pipeline with three stages: Build, Test, and Deploy. Here is an overview of what each stage does:

- **Build:** This stage checks out the source code from a Git repository (replace <https://github.com/example/my-web-app.git> with your repository URL) and then builds the web application using a specific build command (`npm install` in this case, but you can customize it based on your project's requirements).
- **Test:** This stage runs tests for the web application. It executes a test command (`npm test` in this example) that you can modify according to your project's testing framework and requirements.

- **Deploy:** This stage is responsible for deploying the built application. It executes a deployment command (`npm run deploy` in this case) that you can customize to match your specific deployment process. It might involve uploading the application to a server, pushing it to a cloud platform, or any other deployment strategy you use.

You can extend this `Jenkinsfile` to include additional stages, such as static code analysis, security scans, or notifications, based on your project's needs. Remember to adjust the commands and configuration according to your application's build and deployment requirements.

Note: Make sure to have Jenkins properly configured with the necessary plugins and agents to execute the pipeline stages successfully.

Platform as Code

Platform as Code (PaC) refers to the practice of defining and managing the infrastructure and platform components of an application or system using code. It involves treating the entire platform stack, including servers, networks, databases, load balancers, and other infrastructure resources, as code artifacts that can be version-controlled, automated, and managed programmatically.

With PaC, infrastructure, and platform configurations are defined in code, typically using a DSL or configuration files. These code artifacts capture the desired state of the platform stack and can be deployed, updated, and managed using automation tools and processes.

Some key aspects of PaC include:

- **Infrastructure automation:** PaC emphasizes automating the provisioning, configuration, and management of infrastructure resources. By defining infrastructure components as code, you can leverage automation tools to create, modify, and destroy resources in a consistent and repeatable manner.
- **Version control and collaboration:** Just like application code, platform code can be stored in version control systems, enabling versioning, collaboration, and traceability. Multiple team members can

collaborate on infrastructure changes, review code, and propose modifications, improving collaboration and transparency.

- **Consistency and reproducibility:** By representing the platform stack as code, you ensure consistency and reproducibility in your infrastructure deployments. The code artifacts capture the desired state of the platform, including server configurations, network settings, security policies, and more. This allows you to recreate the same infrastructure stack reliably across different environments and avoid configuration drift.
- **Scalability and agility:** PaC enables scalability and agility by abstracting the infrastructure details into code. It allows you to define and manage complex infrastructure setups programmatically, making it easier to scale resources up or down, provision new environments, and accommodate evolving application requirements.
- **Continuous Delivery and Integration:** Platform as Code aligns well with the principles of continuous delivery and integration. By defining the infrastructure as code, you can integrate it seamlessly with your CI/CD pipelines, automating the provisioning of environments, running tests, and deploying applications consistently and repeatedly.

Here is an example of Terraform code that demonstrates a simple platform infrastructure setup using IaC principles:

File name: `main.tf`

```
# main.tf

# Provider configuration (for example, AWS)
provider "aws" {
  region = "us-west-2"
}
```

```
# VPC

resource "aws_vpc" "example_vpc" {
    cidr_block = "10.0.0.0/16"
    tags = {
        Name = "example-vpc"
    }
}

# Subnet

resource "aws_subnet" "example_subnet" {
    vpc_id      = aws_vpc.example_vpc.id
    cidr_block = "10.0.1.0/24"
    availability_zone = "us-west-2a"
    tags = {
        Name = "example-subnet"
    }
}

# Security Group

resource "aws_security_group" "example_security_group" {
    name          = "example-sg"
    description   = "Example security group"
    vpc_id        = aws_vpc.example_vpc.id
```

```
ingress {
    from_port      = 80
    to_port        = 80
    protocol       = "tcp"
    cidr_blocks   = ["0.0.0.0/0"]
}

egress {
    from_port      = 0
    to_port        = 0
    protocol       = "-1"
    cidr_blocks   = ["0.0.0.0/0"]
}

}

# EC2 Instance

resource "aws_instance" "example_instance" {
    ami              = "ami-0c94855ba95c71c99"  #
    Replace with your desired AMI ID
    instance_type   = "t2.micro"
    subnet_id       = aws_subnet.example_subnet.id
    vpc_security_group_ids =
    [aws_security_group.example_security_group.id]

    tags = {
```

```
Name = "example-instance"  
}  
}
```

In this Terraform code, we define a basic platform infrastructure setup on AWS. Here is an overview of what each resource block does:

- **Provider configuration:** This block defines the provider configuration for AWS, specifying the desired region (us-west-2 in this example).
- **VPC:** The `aws_vpc` resource block creates a **Virtual Private Cloud (VPC)** with the specified CIDR block (10.0.0.0/16) and assigns it a name tag.
- **Subnet:** The `aws_subnet` resource block creates a subnet within the VPC, with the specified CIDR block (10.0.1.0/24) and availability zone (`us-west-2a`). It is associated with the VPC created earlier and given a name tag.
- **Security group:** The `aws_security_group` resource block creates a security group named `example-sg` within the VPC. It allows inbound traffic on port 80 (HTTP) from any source IP address and allows all outbound traffic. Adjust the rules as per your requirements.
- **EC2 instance:** The `aws_instance` resource block creates an EC2 instance using the specified AMI ID, instance type, subnet ID, and security group ID. This example uses a `t2.micro` instance type and is associated with the previously created subnet and security group. It is given a name tag as well.

To use this Terraform code, make sure you have the AWS CLI configured with appropriate access credentials. Run `terraform init` to initialize the Terraform project, followed by `terraform plan` to review the proposed changes. Finally, execute `terraform apply` to create the infrastructure resources.

Note: Replace the AMI

Popular tools that support PaC include Terraform, AWS CloudFormation, **Azure Resource Manager (ARM)** templates, and Google Cloud Deployment Manager. These tools provide infrastructure orchestration capabilities and allow you to define, manage, and automate infrastructure deployments using code.

In summary, PaC enables organizations to treat infrastructure and platform components as code, bringing the benefits of version control, automation, consistency, and scalability to managing infrastructure resources. It aligns infrastructure provisioning and management with modern software development practices and helps streamline applications' overall delivery and operation.

Configuration as Code

Configuration as Code (CaC) refers to the practice of defining and managing system configurations using code. It represents configuration settings, parameters, and policies as code artifacts that can be version-controlled, automated, and managed programmatically.

Traditionally, system configurations were managed through manual processes, configuration files, or **Graphical User Interfaces (GUIs)**. However, CaC introduces a more structured and reproducible approach by treating configurations as code that can be written, reviewed, versioned, and deployed using automation tools and techniques.

Popular tools that support CaC include Ansible, Chef, Puppet, SaltStack, and PowerShell **Desired State Configuration (DSC)**. These tools provide the ability to define configurations in code and automate the process of applying those configurations to systems.

Here is an example that showcases the configuration of an AWS S3 bucket:

```
# main.tf

# Provider configuration (for example, AWS)
provider "aws" {
    region = "us-west-2"
```

```
}

# S3 Bucket

resource "aws_s3_bucket" "example_bucket" {

    bucket = "example-bucket"
    acl     = "private"

    versioning {
        enabled = true
    }

    lifecycle {
        prevent_destroy = false
    }

    logging {
        target_bucket = "example-logs-bucket"
        target_prefix = "s3-logs/"
    }

    tags = {
        Name          = "example-bucket"
        Environment = "production"
    }
}
```

```
 }  
}
```

In this Terraform code, we define the configuration for an AWS S3 bucket using the `aws_s3_bucket` resource block. The following are the key configuration aspects:

- **Provider configuration:** Similar to the previous example, the provider configuration specifies the AWS region.
- **S3 bucket:** The `aws_s3_bucket` resource block defines an S3 bucket with the desired name (`example-bucket` in this case) and **Access Control List (ACL)** set to `private`:
 - **versioning:** The `versioning` block enables versioning for the bucket by setting `enabled` to true.
 - **lifecycle:** The `lifecycle` block allows the destruction of the bucket when Terraform is destroyed (by default). If you want to prevent bucket destruction, set `prevent_destroy` to true.
 - **logging:** The `logging` block configures S3 server access logging. It specifies the target bucket (`example-logs-bucket`) where access logs will be stored and the prefix (`s3-logs/`) for the log file names.
 - **tags:** The `tags` block defines key-value pairs as tags for the S3 bucket. In this example, we set tags for the bucket's name and environment.

To use this Terraform code, ensure you have the appropriate AWS CLI credentials configured. Initialize the Terraform project using `terraform init`, preview the planned changes with the `terraform plan`, and apply the configuration with `terraform apply`.

Remember to customize the resource configuration to suit your requirements, including the bucket name, ACL settings, versioning, lifecycle rules, logging details, and tags.

Once applied, Terraform will create the S3 bucket with the specified configuration, allowing you to manage the bucket's state and properties as code.

Overall, CaC enables organizations to manage system configurations in a more structured, automated, and scalable manner. By treating configurations as code, teams can achieve consistency, reproducibility, and auditability while reducing manual effort and improving overall system management.

Policy as Code

Policy as Code (PaC) is an approach that involves defining, managing, and enforcing policies using code. It treats policies as code artifacts that can be version-controlled, tested, reviewed, and deployed using automation tools and processes. PaC aims to bring the benefits of automation, consistency, and scalability to policy management and enforcement.

Policies are sets of rules, regulations, or guidelines that define desired behavior, best practices, security measures, compliance requirements, or governance standards within an organization or system. Traditionally, policies were often documented and enforced through manual processes or external tools. However, PaC introduces a more structured and automated approach to policy management.

Key aspects of PaC include:

- **Policy definition:** Policies are defined using code. This can be done using a DSL, configuration files, or programming languages. Policy code specifies the desired rules, conditions, constraints, or actions that need to be enforced.
- **Automation:** PaC leverages automation tools and processes to enforce policies consistently and reliably. Code-driven policies can be integrated into deployment pipelines, continuous integration systems, or infrastructure orchestration tools to automate policy checks and enforcement.
- **Version control and collaboration:** Policy code can be stored in version control systems, allowing for versioning, collaboration, and change tracking. Multiple team members can collaborate on policy development, review and propose modifications and track policy changes over time.

- **Testing and validation:** Policy code can be tested and validated like any other code artifact. Automated tests can be performed to ensure policy compliance and correctness. This helps catch potential policy violations or misconfigurations early in the development process.
- **Auditability and compliance:** PaC provides traceability and auditability. Policies are transparent and verifiable since they are represented as code. Policy enforcement and compliance can be easily tracked, logged, and audited.
- **Consistency and scalability:** PaC enables consistent policy enforcement across different environments and systems. Code-driven policies can be applied consistently, avoiding variations and configuration drift. The automated nature of PaC allows policies to be scaled and applied across large deployments or complex systems.
- **Integration with DevOps:** Policy as Code aligns well with DevOps practices. It can be seamlessly integrated into the software development lifecycle, allowing policies to be treated as code artifacts that go through the same development, testing, and deployment processes.

PaC can be applied to various domains, including security policies, compliance regulations, cloud governance, infrastructure provisioning rules, access control, and more. Tools such as **Open Policy Agent (OPA)**, AWS Config, HashiCorp Sentinel, and Chef InSpec provide capabilities for implementing PaC.

Here is an example code snippet that demonstrates how to define and enforce an Azure policy using Terraform:

```
# main.tf

# Provider configuration (for example, Azure)
provider "azurerm" {
  features {}
}


```

```
# Resource Group

resource "azurerm_resource_group"
"example_resource_group" {

    name      = "example-resource-group"
    location = "West US"

}

# Policy Definition

resource "azurerm_policy_definition"
"example_policy_definition" {

    name          = "example-policy-definition"
    display_name = "Example Policy Definition"
    description  = "Enforces example policy"

    policy_rule = <<POLICY_RULE
{



    "if": {

        "field": "location",
        "in": ["West Europe", "North Europe"]
    },
    "then": {
        "effect": "deny"
    }
}
```

```

POLICY_RULE
}

# Policy Assignment

resource "azurerm_policy_assignment"
"example_policy_assignment" {

    name                  = "example-policy-
assignment"

    scope                =
azurerm_resource_group.example_resource_group.id

    policy_definition_id =
azurerm_policy_definition.example_policy_definition
.id

}

```

In this example, we define an Azure policy using Terraform. Here is an overview of the key elements:

- **Provider configuration:** The provider configuration block sets up the Azure provider.
- **Resource group:** The `azurerm_resource_group` resource block creates an Azure resource group with the specified name and location.
- **Policy definition:** The `azurerm_policy_definition` resource block defines an Azure policy. It specifies the name, display name, and description of the policy. The `policy_rule` attribute contains the policy rule expressed as JSON. In this example, the policy denies resources from being created in the *West Europe* and *North Europe* Azure regions.
- **Policy assignment:** The `azurerm_policy_assignment` resource block assigns the policy to the previously created resource group. It specifies the name, scope (resource group ID), and the ID of the policy definition.

To use this Terraform code, ensure you have the appropriate Azure CLI credentials configured. Initialize the Terraform project using `terraform init`, preview the planned changes with the `terraform plan`, and apply the configuration with `terraform apply`.

Remember to customize the resource names, locations, policy rules, and assignments to match your requirements.

Once applied, Terraform will create the policy definition and assign it to the resource group. Azure Policy will enforce the policy rule, denying resource creation in the specified regions.

Make sure you have the necessary permissions and prerequisites in Azure, such as an active subscription and appropriate role assignments, to create and assign policies using Terraform.

In summary, PaC enables organizations to define, manage, and enforce policies using code artifacts. By treating policies as code, teams can automate policy enforcement, achieve consistency, scalability, and auditability, and integrate policy management into the development and deployment processes.

GitOps methodology

GitOps is a methodology for implementing and managing continuous delivery and infrastructure as code practices using Git as the single source of truth for declarative infrastructure and application configurations. It extends the principles of DevOps by leveraging version control, automation, and Git workflows to streamline the deployment and management of software systems.

In GitOps, the desired state of the entire system, including infrastructure, applications, and configurations, is declaratively defined in Git repositories. The Git repository is a central source of truth, containing version-controlled code and configuration files. The desired state represents the target state of the system, and any changes to the system are made by modifying the desired state in the Git repository.

Key aspects of the GitOps methodology include:

- **Declarative infrastructure:** Infrastructure is defined and managed declaratively using IaC tools such as Terraform or CloudFormation. Infrastructure changes are made by updating the infrastructure code stored in the Git repository.
- **Application configuration:** Application configurations, such as environment-specific settings or feature toggles, are also defined declaratively in the Git repository. Changes to application configurations are made by modifying the corresponding configuration files.
- **Continuous Integration and Delivery:** GitOps encourages using CI/CD practices to automate the deployment and release processes. CI/CD pipelines are triggered based on changes in the Git repository, ensuring that deployments are automated, consistent, and reproducible.
- **Version control:** GitOps leverages the powerful version control capabilities of Git to track and manage changes to infrastructure and application configurations. Git provides features such as branching, merging, and pull requests, enabling collaboration, code review, and change management.
- **Pull-based deployments:** In GitOps, the desired state stored in the Git repository is continuously reconciled with the actual state of the system. This reconciliation is typically done through an operator or an automated synchronization process. The reconciliation process pulls changes from the Git repository and applies them to the target system, ensuring that the actual state matches the desired state.
- **Observability and monitoring:** GitOps emphasizes observability and monitoring of the system. Monitoring tools collect metrics, logs, and alerts, providing visibility into the system's health and performance. GitOps promotes the practice of storing and versioning monitoring configurations along with the rest of the system's configurations.

By adopting GitOps, organizations can achieve several benefits:

- **Consistency and reproducibility:** GitOps ensures that infrastructure and application configurations are consistent across different

environments and deployments. The declarative nature of GitOps allows for reproducible deployments, reducing the risk of configuration drift and human errors.

- **Traceability and auditability:** GitOps provides a clear audit trail of all changes made to the system. Every change, including infrastructure modifications and configuration updates, is recorded in the Git commit history, enabling traceability, accountability, and compliance.
- **Collaboration and review:** GitOps promotes collaboration and code review practices. Git workflows such as pull requests enable team members to review and discuss changes before they are merged into the main branch, ensuring quality and reducing the likelihood of introducing errors.
- **Increased efficiency:** With GitOps, deployments and updates can be automated, reducing the manual effort required for infrastructure and application management. This improves efficiency, accelerates development cycles, and allows teams to focus on delivering value rather than managing infrastructure.

Popular tools and technologies that support the GitOps methodology include Kubernetes, Helm, Argo CD, Flux CD, and Jenkins X. These tools provide capabilities for declarative deployments, Git-based workflows, and automated synchronization between Git repositories and target systems.

GitOps WorkFlow

The GitOps workflow follows steps that enable continuous delivery and management of infrastructure and applications using Git as the single source of truth. Here is a typical GitOps workflow:

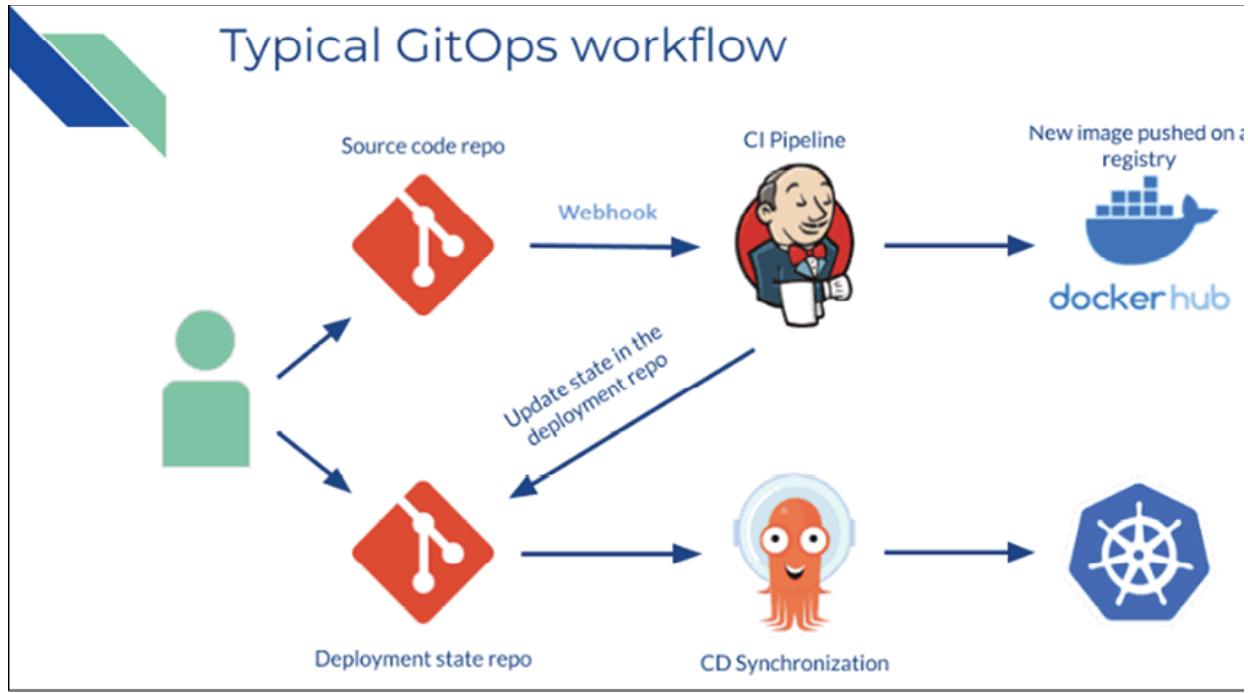


Figure 10.5: GitOps workflow

In the following section, the GitOps workflow is explained:

- **Define infrastructure and application configurations:** Use IaC tools like Terraform or CloudFormation to define your infrastructure configurations. Write application configurations, such as environment-specific settings, in configuration files. Store all these configurations in a Git repository.
- **Pull requests and review:** When changes need to be made to the infrastructure or application configurations, create a branch in the Git repository. Make the necessary modifications in the branch and open a pull request. Reviewers then assess the changes, provide feedback, and ensure the proposed changes align with best practices and policies.
- **Continuous integration:** Set up a CI/CD pipeline triggered by changes to the Git repository. The CI/CD pipeline should build, test, and package the infrastructure and application code. It ensures that the changes are validated and ready for deployment.
- **Infrastructure deployment:** Once the CI/CD pipeline completes the build and testing stages, it deploys the updated infrastructure configurations to the target environment. This deployment is typically

done using the IaC tool, which provisions and configures the infrastructure resources according to the desired state defined in the Git repository.

- **Application deployment:** Following the infrastructure deployment, the CI/CD pipeline deploys the application code and configuration changes to the target environment. This may involve building and packaging the application artifacts, deploying them to the appropriate servers or containers, and configuring any necessary environment variables or dependencies.
- **Automated synchronization:** As part of the GitOps approach, an automated synchronization mechanism constantly monitors the Git repository. It detects changes in the desired state of the system and applies them to the target environment. This synchronization can be facilitated by GitOps-specific tools like Argo CD or Flux CD, which pull changes from the Git repository and reconcile the actual state with the desired state.
- **Observability and monitoring:** Configure monitoring and observability tools to collect metrics, logs, and alerts from the system. These tools provide visibility into the health, performance, and compliance of the infrastructure and applications. Monitoring configurations are version-controlled in the Git repository, ensuring consistency and reproducibility.
- **Rollbacks and roll forwards:** In case of issues or rollbacks, GitOps allows you to revert changes by rolling back to a previous version of the configuration in the Git repository. Similarly, roll forwards can be performed by applying newer versions of the configuration to recover from failures or update the system.

Throughout the GitOps workflow, the Git repository serves as the single source of truth for the desired state of the system. Changes are made by modifying the configurations stored in the Git repository, and automation tools ensure that the actual state aligns with the desired state.

Use cases of GitOps

GitOps can be applied to various use cases across different domains and industries. Here are some common use cases of GitOps:

- **Infrastructure provisioning and configuration:** GitOps is widely used for managing infrastructure provisioning and configuration. It allows teams to define IaC using tools like Terraform or CloudFormation and store the infrastructure configurations in a Git repository. Changes to the infrastructure are made through pull requests and automated pipelines, ensuring consistent and reproducible infrastructure deployments.
- **Application deployment and release management:** GitOps simplifies the deployment and release management of applications. By storing application configurations, deployment manifests, and release information in a Git repository, and teams can easily manage application versions, rollbacks, and environment-specific configurations. GitOps tools can automatically deploy and manage applications based on changes in the Git repository.
- **Multi-environment and multi-cloud management:** GitOps enables the management of multiple environments and cloud platforms. With Git as the single source of truth, teams can define and version infrastructure and application configurations for different environments (for example, development, staging, and production) and target multiple cloud providers. GitOps facilitates consistent and synchronized deployments across environments and cloud platforms.
- **Microservices and kubernetes operations:** GitOps is well-suited for managing microservices and Kubernetes-based architectures. By storing Kubernetes manifests, helm charts, and configuration files in Git, teams can track changes, conduct code reviews, and automate deployments. GitOps tools like Argo CD or Flux CD can monitor the Git repository for changes and automatically reconcile the desired state with the actual state of the cluster.
- **Compliance and auditing:** GitOps provides a transparent and auditable approach to infrastructure and application management. By tracking all changes in the Git repository, teams can easily demonstrate compliance with regulatory requirements and industry

standards. GitOps allows for easy rollbacks, change tracking, and traceability, supporting auditing and compliance processes.

- **Disaster recovery and resilience:** GitOps facilitates disaster recovery and resilience by capturing and versioning infrastructure and application configurations. In the event of a failure or disaster, teams can quickly recover by applying the desired state from a previously known good state stored in Git. GitOps helps maintain consistency and reliability across different environments and enables rapid restoration of services.
- **Collaborative development and DevOps practices:** GitOps encourages collaboration and enables DevOps practices. By using Git as the collaboration platform, teams can work together on infrastructure and application configurations, conduct code reviews, and ensure best practices are followed. GitOps aligns with the principles of DevOps by promoting automation, continuous integration, continuous delivery, and version control.

These are just a few examples of the many use cases for GitOps. The flexibility and benefits of GitOps make it applicable to a wide range of scenarios, providing improved efficiency, consistency, traceability, and automation in managing infrastructure and applications.

Overall, GitOps provides a framework for implementing continuous delivery, infrastructure as code, and version control practices in a unified manner.

Best tools for IaC

Here are some widespread IaC tools:

- **Terraform:** Terraform, developed by HashiCorp, is one of the most widely used IaC tools. It supports multiple cloud providers and infrastructure platforms, allowing you to define and provision resources using a declarative configuration language.
- **AWS CloudFormation:** AWS CloudFormation is a native IaC tool provided by **Amazon Web Services (AWS)**. It enables you to define and manage AWS infrastructure resources using JSON or YAML

templates. It integrates well with other AWS services and provides a high level of automation and scalability.

- **Azure Resource Manager (ARM) Templates:** Azure Resource Manager is the IaC tool for Microsoft Azure. ARM Templates, written in JSON, allow you to define and provision Azure resources and manage dependencies between them.
- **Google Cloud Deployment Manager:** Google Cloud Deployment Manager is the IaC tool provided by **Google Cloud Platform (GCP)**. It allows you to define and manage GCP resources using YAML or Python configuration files.
- **Ansible:** Ansible is an open-source configuration management and provisioning tool that can also be used for Infrastructure as Code. It uses a simple and human-readable YAML-based language and can manage both on-premises and cloud resources.
- **Puppet:** Puppet is a popular configuration management tool that supports Infrastructure as Code practices. It uses a declarative language called Puppet DSL and allows you to define and manage infrastructure configurations across multiple platforms.
- **Chef:** Chef is another configuration management and automation tool that can be used for Infrastructure as Code. It uses a **domain-specific language (DSL)** called Chef Infra and provides robust infrastructure automation capabilities.
- **SaltStack:** SaltStack, also known as Salt, is a configuration management and orchestration tool that can be used for Infrastructure as Code. It uses YAML or Jinja-based configuration files and provides features like remote execution and configuration drift management.
- **Pulumi:** Pulumi is a modern Infrastructure as Code tool that allows you to define and manage infrastructure using familiar programming languages such as JavaScript, Python, Go, and TypeScript. It supports multiple cloud platforms and provides a high level of flexibility and extensibility.

These are just a few examples of popular IaC tools available in the market. Each device has its features, strengths, and target platforms, so choosing the

right tool depends on your specific requirements, cloud provider preferences, and the team's familiarity with the tooling ecosystem.

Conclusion

In conclusion, automating your testing process is crucial in ensuring that your software is thoroughly tested and defects-free. Automating tests allows you to run tests quickly and easily, making it easier to identify and fix any issues that may arise. By integrating automated tests into your CI/CD pipeline, you can catch issues early on, which helps to minimize the impact of any problems that may arise. Additionally, by using a test management tool, you can easily track the results of your tests and monitor the performance of your system over time. Furthermore, by setting up a robust continuous testing environment, you can ensure that your system is thoroughly tested and that any issues are identified and fixed quickly, which will help to provide a better experience for your end-users.

In the next chapter, we will cover DevSecOps. DevSecOps is an approach to software development that integrates security practices into the DevOps (Development Operations) process. It emphasizes the importance of security early in the software development lifecycle rather than treating it as an afterthought.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



CHAPTER 11

Focus on Security: DevSecOps

Introduction

Focus on security is an essential aspect of DevSecOps, which is a methodology that combines the principles of development, security, and operations to create a more secure and efficient software development lifecycle. DevSecOps emphasizes the importance of integrating security considerations throughout the entire development process, from planning and design to testing and deployment.

In DevSecOps, security is not treated as an afterthought or a separate process, but as an integral component of the entire software development lifecycle. The focus on security is achieved by adopting a security-first approach that includes secure coding practices, continuous security testing, automated security controls, continuous compliance monitoring, and security training and awareness.

Introduction of DevSecOps:

DevOps + Security: DevSecOps

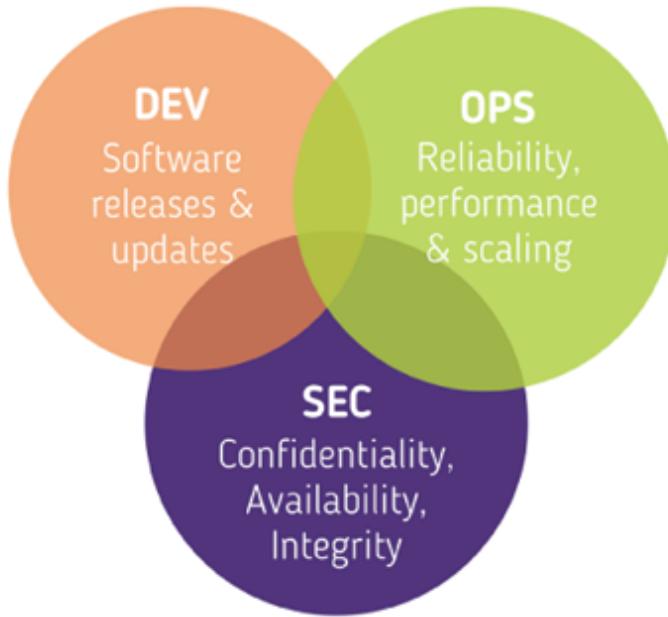


Figure 11.1: DevSecOps introduction

Structure

In this chapter, we will learn about DevSecOps. It is an approach to software development that integrates security practices into the **Development Operations (DevOps)** process. It emphasizes the importance of security early in the software development lifecycle rather than treating it as an afterthought.

We will learn the following topics:

- Principles of DevSecOPS
- Implement security by design
- Integrate compliance and governance
- Implement security into the deployment pipeline
- Maintain visibility and control
- Collaboration between DevOps and security
- Define and implement IAM, RBAC and 2FA

- Implement least privilege model
- Segregate the DevOps network
- Use password manager

Objectives

By the end of this chapter, we will learn the concepts of DevSecOps and its importance in today's fast-paced software development environment.

We will go through a comprehensive guide to the principles, practices, and tools used in DevSecOps, including best practices for integrating security into every stage of the software development lifecycle. You will understand the risks associated with traditional software development approaches and how DevSecOps can mitigate those risks. We will cover case studies and examples of successful DevSecOps implementations to illustrate the benefits of this approach.

Principles of DevSecOps

DevSecOps is an approach to software development that emphasizes security considerations throughout the entire software development lifecycle, from planning and design to testing and deployment. DevSecOps involves collaboration between development, security, and operations teams to ensure that security requirements are met at every stage of the process (see *Figure 11.2*):

Best practices of DevSecOps:



Figure 11.2: DevSecOps best practices

Some key principles and practices of DevSecOps include:

- **Secure coding practices:** Developers must write secure code and follow secure coding practices, including using secure libraries and frameworks, regularly reviewing and testing code for vulnerabilities, and incorporating security requirements into the design phase.
- **Continuous security testing:** Security testing is integrated into every stage of the development pipeline, with automated security scans, code analysis, and penetration testing to identify and remediate vulnerabilities as early as possible.
- **Automated security controls:** Security controls are automated and integrated into the pipeline, including access controls, authentication, and encryption, to ensure that security is enforced consistently and reliably.

- **Continuous compliance monitoring:** Compliance requirements are integrated into the development pipeline, with continuous monitoring and reporting of compliance status to ensure that software applications and infrastructure meet regulatory and security requirements.
- **Recovery and planning:** Recovery and planning are the essential components of a robust and secure software development and deployment pipeline. They help ensure that the development process is resilient to security threats and can recover from potential security incidents. Here's how recovery and planning fit into DevSecOps.
- **Security training and awareness:** All members of the development team receive regular security training and awareness to stay up to date on the latest threats, best practices and to ensure that security is a shared responsibility across the organization.

Some examples of DevSecOps in action include:

- Automating security scans and tests during the development process to identify vulnerabilities before they reach production.
- Integrating security into the **Continuous Integration and Continuous Delivery (CI/CD)** pipeline to ensure that security controls are enforced consistently across the development lifecycle.
- Using **Infrastructure As Code (IAC)** and configuration management tools to automate the deployment and configuration of secure infrastructure and applications.
- Implementing secure coding standards and practices, such as OWASP Top 10 and SANS Secure Coding, to ensure that code is written with security in mind.
- Incorporating security requirements into the design phase of software development, such as threat modeling and security architecture reviews, to proactively identify and mitigate security risks.

DevSecOps is an approach to software development that emphasizes the integration of security considerations throughout the entire software development lifecycle, from planning and design to testing and deployment. By adopting DevSecOps principles and practices, organizations can deliver

more secure and reliable software applications while also improving their efficiency and agility.

Automate security processes

Automating security processes in DevOps involves the use of tools and technologies to automate security testing, assessments, and other security-related tasks throughout the software development lifecycle. Here is how it works:

- **Automating security testing:** Security testing is a critical part of the DevOps process. Automated security testing tools can help identify security vulnerabilities in the code, such as **Cross-Site Scripting (XSS)** and SQL injection. By automating security testing, teams can test the code more frequently and thoroughly, which reduces the risk of vulnerabilities.
- **Continuous security monitoring:** In addition to testing the code, DevOps teams can also use automated tools to continuously monitor the security of their applications and infrastructure. This can include tools that scan for vulnerabilities in production environments, as well as tools that monitor system logs and alert teams to potential security incidents.
- **Integration with the DevOps pipeline:** Automating security processes also involves integrating security tools and processes into the DevOps pipeline. This means that security is not an afterthought but rather an integral part of the development process. For example, security testing can be incorporated into the CI/CD pipeline so that code is tested for security vulnerabilities before it is deployed.
- **DevOps automation tools:** There are a variety of DevOps automation tools that can be used to automate security processes, including tools for vulnerability scanning, configuration management, and identity and access management. These tools can help ensure that security is integrated into the entire DevOps pipeline, from development to deployment.
- **DevSecOps culture:** Finally, automating security processes is not just about using tools and technologies but also about creating a

DevSecOps culture. This means promoting collaboration between development, security, and operations teams, and ensuring that security is a top priority throughout the software development lifecycle. Automating security processes is a key part of this culture, but it is just one piece of the puzzle.

Here are some popular tools that can be used to automate security processes in DevOps:

- **Jenkins**: Jenkins is a popular open-source automation server that can be used to automate security testing and other DevOps processes.
- **GitLab**: GitLab is an integrated DevOps platform that includes features for continuous integration and deployment, as well as security testing and monitoring.
- **SonarQube**: SonarQube is an open-source platform for continuous code quality inspection, including security testing.
- **OWASP ZAP**: OWASP ZAP is an open-source web application security testing tool that can be used to automate security testing in DevOps pipelines.
- **Qualys**: Qualys is a cloud-based security and compliance platform that can be used to automate vulnerability scanning and other security-related tasks.
- **Ansible**: Ansible is an open-source automation tool that can be used to automate security configuration management tasks, such as setting up firewalls and applying security patches.
- **Chef**: Chef is another automation tool that can be used to automate security configuration management tasks.
- **HashiCorp Vault**: HashiCorp Vault is a tool for managing secrets, such as passwords and API keys, and can be used to automate access management in DevOps pipelines.
- **Kubernetes**: Kubernetes is a container orchestration platform that includes features for automating security tasks, such as container scanning and network policy enforcement.

- **Aqua security:** Aqua Security is a cloud-native security platform that can be used to automate security testing and monitoring in Kubernetes and other cloud environments.

Collaborate across teams

Collaborating across teams is a key DevSecOps best practice that involves bringing together teams from different parts of the organization to work towards a common goal of delivering secure and reliable software.

Here are some examples of how teams can collaborate effectively in a DevSecOps environment:

- Development and security teams can work together to define security requirements and standards for applications. This ensures that security is built into the application from the beginning.
- Operations teams can collaborate with development teams to ensure that security and compliance requirements are met in production environments.
- Security teams can provide training and guidance to development and operations teams on best practices for security and compliance.
- Teams can collaborate on incident response and remediation, with clear roles and responsibilities defined in advance.
- Teams can share data and insights to identify and mitigate security risks. For example, security teams can provide threat intelligence to development teams to help them design and implement more secure applications.
- Teams can collaborate on security testing, with development teams responsible for writing, running tests, and security teams responsible for providing guidance and reviewing the results.
- Cross-functional teams can be formed to work on specific projects or initiatives, with members from development, security, and operations teams working together to achieve a common goal.

By collaborating across teams, organizations can break down silos and ensure that everyone is working towards the same goal of delivering secure

and reliable software. This leads to faster delivery times, fewer security incidents, and better overall outcomes for the organization.

Implementing security by design

Implement Security by Design is a key DevSecOps best practice that involves building security into the development process from the very beginning. This means that security is considered at every stage of the development lifecycle, from planning and design to testing and deployment.

Here are some examples of how organizations can implement security by design:

- **Threat modeling:** Teams can use threat modeling techniques to identify potential security threats and vulnerabilities at the design stage. This involves thinking through how attackers might try to exploit the system and designing security controls to mitigate those risks.
- **Secure coding:** Development teams can use secure coding practices to ensure that applications are designed with security in mind. This includes avoiding common security flaws such as SQL injection and cross-site scripting and following best practices such as input validation and output encoding.
- **Security testing:** Security testing can be integrated into the development process to identify and remediate security issues early on. This includes both static analysis, which involves analyzing code before it is executed, and dynamic analysis, which involves analyzing code as it is being executed.
- **Security automation:** Automated security tools and processes can be integrated into the development process to ensure that security is considered at every stage. For example, automated vulnerability scanning can be used to identify and remediate vulnerabilities in applications and infrastructure. This encompasses the utilization of automated security tools such as static code analyzers for code review, vulnerability scanners to identify potential weaknesses, and automated

compliance checks to ensure adherence to security standards and regulations

- **Security training:** All members of the development team, including developers, operations staff, and QA testers, should receive training on best practices for security and compliance. This helps to ensure that everyone has a basic understanding of security principles and can contribute to building secure applications.

By implementing security by design, organizations can ensure that security is a fundamental part of the development process, rather than an afterthought. This leads to more secure applications, fewer security incidents, and better overall outcomes for the organization.

Using secure coding practices

Using secure coding practices is a critical aspect of DevSecOps, as it helps ensure that software applications are developed with security in mind from the very beginning. Here are some examples of secure coding practices that are commonly used in DevSecOps:

- **Input validation:** Developers should validate all input data from users, third-party systems, and other sources to ensure that it meets expected formats and does not contain malicious code. Input validation helps prevent common attacks like SQL injection and cross-site scripting.
- **Authentication and authorization:** Developers should use strong authentication and authorization controls to ensure that users have appropriate access to resources and data. This includes implementing two-factor authentication, password policies, and role-based access controls.
- **Encryption:** Developers should use strong encryption algorithms and protocols to protect sensitive data at rest and in transit. This includes using HTTPS/TLS for web applications, encrypting data stored in databases, and using secure communication protocols for APIs.
- **Secure coding frameworks and libraries:** Developers should use secure coding frameworks and libraries that have been vetted and

tested for security vulnerabilities. This includes using frameworks like Spring Security for Java applications and libraries like OWASP ESAPI for common security functions like input validation and encryption.

- **Regular code reviews:** Developers should conduct regular code reviews to identify and remediate security vulnerabilities. This includes reviewing code for common security issues like buffer overflows, race conditions, and insecure third-party library dependencies.
- **Security testing:** Developers should conduct regular security testing, including penetration testing, vulnerability scanning, and static and dynamic code analysis, to identify and remediate security issues as early as possible.
- **Secure coding standards and guidelines:** Developers should follow established secure coding standards and guidelines, such as the OWASP Top 10 and SANS Secure Coding, to ensure that code is written with security in mind. Secure Coding Standards and Guidelines encompass not only coding practices, but also extend to include architectural and design considerations, ensuring a holistic approach to application security.

By incorporating these secure coding practices into their development processes, DevSecOps teams can help reduce the risk of security vulnerabilities in software applications. Additionally, these practices can help increase the speed and efficiency of the development process by identifying and fixing security issues early in the development lifecycle.

Integrating compliance and governance

Integrating compliance and governance into the DevSecOps process is essential to ensure that software applications and infrastructure meet regulatory and security requirements. Here are some examples of how to integrate compliance and governance into the DevSecOps process:

- **Establish compliance requirements:** The first step in integrating compliance and governance into DevSecOps is to identify and

establish the relevant compliance requirements for the organization, such as HIPAA, PCI-DSS, GDPR, and so on.

- **Incorporate compliance into the development pipeline:** Compliance requirements should be integrated into every stage of the development pipeline, from planning and design to testing and deployment. This includes regular compliance testing and reporting, as well as automated compliance checks and audits.
- **Continuous monitoring and reporting:** Compliance status should be continuously monitored and reported throughout the development lifecycle, with regular reports and dashboards that provide real-time visibility into compliance status and risk.
- **Implement security controls:** Security controls, such as access controls, authentication, and encryption, should be automated and integrated into the development pipeline to ensure that security is enforced consistently and reliably.
- **Compliance automation:** Compliance automation involves automating the compliance process to reduce the risk of human error and increase the speed and efficiency of compliance testing and reporting. This can be done through the use of tools like Chef Compliance, InSpec, and AWS Config Rules.
- **Compliance as code:** Compliance as code involves using **Infrastructure as Code (IAC)** and configuration management tools to automate the deployment and configuration of secure infrastructure and applications. This can help ensure that compliance requirements are consistently and reliably enforced across the development lifecycle.
- **Compliance training and awareness:** All members of the development team should receive regular compliance training and awareness to stay up to date on the latest compliance requirements and best practices.

By integrating compliance and governance into the DevSecOps process, organizations can ensure that software applications and infrastructure meet

regulatory and security requirements, reduce the risk of compliance violations and penalties, and improve overall security and reliability.

Implementing DevSecOps training and education

Implementing DevSecOps Training and Education is a key DevSecOps best practice that involves ensuring that all members of the development team, including developers, operations staff, and QA testers, receive training on best practices for security and compliance.

Here are some ways organizations can implement DevSecOps training and education:

- **Training programs:** Organizations can develop training programs that cover key security principles and practices, as well as specific tools and techniques for implementing security in the development process.
- **Role-based training:** Different members of the development team may have different security responsibilities, so it is important to provide role-based training that is tailored to the specific needs of each team member.
- **On-demand training:** Online training modules and resources can be made available to team members so they can access training whenever they need it.
- **Workshops and exercises:** Hands-on workshops and exercises can be used to reinforce key security concepts and provide team members with practical experience in implementing security in the development process.
- **Cross-functional training:** Organizations can bring together members of different teams, such as developers and security experts, for training sessions that promote collaboration and help to break down silos.
- **Certification programs:** Certification programs can be used to validate the security knowledge and skills of team members, as well as to provide ongoing professional development opportunities.

By implementing DevSecOps training and education, organizations can ensure that all members of the development team have a basic understanding of security principles and best practices. This helps to build a culture of security within the organization and ensures that security is considered at every stage of the development process.

Integrating security into the deployment pipeline

Integrating security into the deployment pipeline is an essential part of DevSecOps. Here are some examples of how to integrate security into the deployment pipeline:

- **Static code analysis:** Developers can use static code analysis tools to identify and fix security vulnerabilities in the code before deployment. These tools analyze the code for known security issues and provide actionable feedback to the developer.
- **Dynamic Application Security Testing (DAST):** DAST tools simulate attacks against the deployed application to identify potential security vulnerabilities. These tests are typically performed as part of the deployment pipeline and can provide early detection of security issues.
- **Container security:** Containerized applications require special attention to security, and DevSecOps teams should use container security tools to scan images for known vulnerabilities, enforce security policies, and monitor container behavior for potential security issues.
- **Infrastructure as code (IAC) security:** IAC tools like Terraform and CloudFormation can be used to create secure infrastructure by automatically configuring security settings and enforcing security policies during deployment.
- **Automated vulnerability scanning:** DevSecOps teams can use automated vulnerability scanning tools to scan deployed applications for known vulnerabilities and misconfigurations.
- **Continuous security testing:** Security testing should be integrated into the deployment pipeline, with regular automated security tests and

code reviews to identify and remediate security issues as early as possible.

- **Monitoring and incident response:** DevSecOps teams should have a plan in place for monitoring deployed applications and responding to security incidents. This includes implementing real-time monitoring and alerting, as well as regular incident response training and testing.
- **Secret management:** By integrating secret management into your CI/CD pipeline, you reduce the risk of security breaches due to mishandled secrets and ensure that your applications and infrastructure are protected. Secrets, such as API keys, passwords, and cryptographic keys, need to be protected and managed effectively.

By integrating security into the deployment pipeline, DevSecOps teams can identify and remediate security issues early in the development process, reduce the risk of security breaches, and improve overall application security and reliability.

Maintaining visibility and control

Maintaining visibility and control is a DevSecOps best practice that involves maintaining real-time visibility and control over the software development process and the underlying infrastructure.

Here are some examples of how organizations can maintain visibility and control:

- **Use automation:** Automation tools can be used to track changes to code, configurations, and infrastructure in real-time. This helps to ensure that all changes are authorized and that unauthorized changes can be quickly identified and remediated.
- **Implement continuous monitoring:** Continuous monitoring tools can be used to detect and respond to security threats in real-time. This includes monitoring network traffic, system logs, and other data sources to detect abnormal behavior and potential security incidents.
- **Use centralized logging:** Centralized logging tools can be used to collect and analyze logs from across the infrastructure. This helps to

identify potential security issues and provides valuable information for incident response and forensic analysis.

- **Implement access controls:** Access controls can be used to limit access to sensitive systems and data. This includes implementing role-based access controls, two-factor authentication, and other measures to ensure that only authorized users have access to sensitive resources.
- **Conduct regular security assessments:** Regular security assessments, such as penetration testing and vulnerability scanning, can be used to identify potential security vulnerabilities and prioritize remediation efforts.
- **Patch management:** Patch management is a critical aspect of DevSecOps, as it plays a significant role in maintaining the security, stability, and reliability of software and infrastructure. Patch management helps address known vulnerabilities in software components, libraries, and operating systems. By regularly applying patches, you reduce the attack surface and mitigate the risk of security breaches.

Maintaining visibility and control, organizations can ensure that all changes to the development process and underlying infrastructure are authorized and monitored, and that the potential security issues are quickly identified and addressed. This helps to reduce the risk of security incidents and ensures that the organization is better able to respond to incidents if they do occur.

Collaboration between DevOps and security

DevOps and security collaboration is a crucial aspect of DevSecOps. It involves bringing together the development, operations, and security teams to work collaboratively to ensure that security is integrated into every aspect of the development lifecycle. Here are some examples of how to foster collaboration between DevOps and security teams:

- **Shared responsibility:** DevOps and security teams should have a shared responsibility for security. This means that security is everyone's responsibility, not just the responsibility of the security team. Developers, operations personnel, and security professionals

should work together to identify and address security issues throughout the development lifecycle.

- **Cross-functional training:** DevOps and security teams should receive cross-functional training to ensure that everyone understands the importance of security and has a basic understanding of security best practices.
- **Regular communication:** DevOps and security teams should communicate regularly to ensure that everyone is on the same page and that security issues are identified and addressed in a timely manner. This can include regular meetings, status updates, and incident response planning.
- **Security automation:** DevOps and security teams should work together to automate security testing and remediation wherever possible. This can include automating vulnerability scanning, configuration management, and incident response.
- **Risk management:** DevOps and security teams should work together to identify and prioritize security risks based on the potential impact on the business. This can include conducting risk assessments and implementing risk mitigation strategies.
- **Agile security:** DevOps and security teams should adopt an agile approach to security, with continuous testing and improvement throughout the development lifecycle. This means that security is not an afterthought, but an integral part of the development process.
- **DevSecOps culture:** DevOps and security teams should work together to create a DevSecOps culture that emphasizes collaboration, communication, and continuous improvement. This means creating a culture where security is integrated into every aspect of the development process, and where everyone takes ownership of security.

By fostering collaboration between DevOps and security teams, organizations can ensure that security is integrated into every aspect of the development process, reduce the risk of security breaches, and improve overall application security and reliability

Threat modelling

Threat modelling is a technique used in DevSecOps to identify potential security threats and vulnerabilities early in the software development process. The goal of threat modeling is to identify potential threats and risks to the system and to prioritize security controls and countermeasures to mitigate those risks.

Here are some steps that can be taken to implement threat modelling in DevSecOps:

- 1. Identify the system components:** The first step is to identify the components of the system being developed. This includes identifying the software, hardware, and data components of the system.
- 2. Identify potential threats:** The next step is to identify potential threats to the system. This includes considering potential attacks and vulnerabilities that could be exploited by attackers, such as injection attacks, privilege escalation, and denial of service attacks.
- 3. Evaluate potential impact:** Once potential threats have been identified, it is important to evaluate the potential impact of each threat. This includes considering the likelihood of the threat occurring, the severity of the impact if the threat is realized, and the potential for cascading effects.
- 4. Prioritize security controls:** Based on the potential impact of each threat, prioritize the implementation of security controls and countermeasures to mitigate the risks. This includes identifying which controls are most effective in mitigating each threat and considering the costs and resource requirements of implementing each control.
- 5. Continuously review and update:** Threat modeling is an iterative process and should be reviewed and updated regularly throughout the software development lifecycle. This helps to ensure that new threats are identified and that security controls remain effective over time.

By implementing threat modeling in the DevSecOps process, organizations can identify potential security threats and vulnerabilities early in the development process, prioritize security controls and countermeasures to mitigate those risks, and ultimately improve the security posture of the software being developed.

Compliance as code

compliance as code is a concept in DevSecOps that involves automating compliance processes and integrating them into the deployment pipeline. The idea is to define compliance requirements as code, and then use automation to verify compliance at each stage of the development process. This approach allows organizations to ensure that compliance requirements are met consistently and efficiently, while also enabling faster, more reliable deployments.

Here are some steps for implementing compliance as code in DevSecOps:

- 1. Define compliance requirements:** The first step is to define compliance requirements based on applicable regulations, industry standards, and internal policies. These requirements should be clearly documented and communicated to all stakeholders.
- 2. Define compliance checks as code:** Next, compliance checks should be defined as code. This can include creating scripts or templates that automate compliance checks and integrate them into the deployment pipeline.
- 3. Automate compliance testing:** Once compliance checks are defined as code, they can be automated to ensure that compliance requirements are met at every stage of the development process. This can include using automated testing tools to verify compliance, as well as implementing real-time monitoring and alerting to identify potential compliance issues.
- 4. Implement version control:** To ensure that compliance requirements are met consistently over time, it is important to implement version control for compliance-related code. This will enable organizations to track changes to compliance requirements and ensure that they are applied consistently across all deployments.
- 5. Conduct regular audits:** Even with Compliance as Code in place, it is important to conduct regular audits to ensure that compliance requirements are being met. Audits can help identify potential compliance issues, as well as areas for improvement in the Compliance as Code process.

6. Change management: Change management plays a crucial role in the DevSecOps (Development, Security, and Operations) process by ensuring that changes are implemented effectively and securely. Change management helps identify potential risks associated with code changes, infrastructure updates, or configuration modifications. It ensures that changes are thoroughly reviewed and tested to minimize the risk of vulnerabilities and security issues.

By implementing compliance as code in DevSecOps, organizations can ensure that compliance requirements are met consistently and efficiently, while also enabling faster, more reliable deployments. This approach can help reduce the risk of compliance-related issues, improve overall security and reliability, and enable organizations to demonstrate compliance to regulators and other stakeholders.

Container security

Container security refers to the set of practices and technologies used to secure the containers used in a DevSecOps environment. Containers are a popular tool for DevSecOps as they provide a way to package and deploy applications quickly and efficiently, but they can also introduce security risks if not properly secured.

Here are some steps that can be taken to implement container security in DevSecOps:

- 1. Use secure base images:** Start by using secure base images for your containers. This means using base images that have been scanned and certified for security vulnerabilities and best practices.
- 2. Limit container privileges:** Containers should be run with the least privilege required to operate. This means limiting the access and permissions given to containers to only what is necessary.
- 3. Scan for vulnerabilities:** Use container scanning tools (**Clair, Trivy and Aqua Security Trivy**) to identify vulnerabilities and security risks in the containers being used in your environment. These tools can identify known vulnerabilities in the container images and provide recommendations for remediation.

4. **Implement access controls:** Implement access controls to restrict access to the containers and the data within them. This includes implementing role-based access controls, implementing network segmentation, and monitoring access to the containers.
5. **Monitor container activity:** Monitor container activity in real-time to identify any suspicious activity that could indicate a security breach. This includes monitoring container logs, network traffic, and system activity.
6. **Continuously update and patch:** As new vulnerabilities are identified, make sure to continuously update and patch your containers to mitigate any risks.

By implementing container security in the DevSecOps process, organizations can ensure that the containers used in their environment are secure and meet the necessary security standards. This helps to reduce the risk of security breaches and improve the overall security posture of the software being developed.

DevSecOps metrics

DevSecOps metrics refer to the use of data and analytics to measure and improve the security and compliance of DevSecOps processes. The idea is to establish a set of metrics that can be tracked over time to identify areas for improvement and measure the effectiveness of security and compliance measures.

Here are some steps for implementing DevSecOps metrics:

1. **Key Performance Indicators (KPIs):** The first step is to define KPIs that will be used to measure the security and compliance of DevSecOps processes. KPIs should be aligned with organizational goals and objectives, and should be relevant, measurable, and actionable.
2. **Collect and analyze data:** Once KPIs are defined, data should be collected and analyzed to track progress over time. This can include data from security tools, compliance reports, and other sources.
3. **Establish benchmarks:** To measure the effectiveness of security and compliance measures, it is important to establish benchmarks for each

KPI. Benchmarks should be based on industry standards, best practices, and organizational goals.

4. **Implement continuous monitoring:** To ensure that DevSecOps Metrics are up-to-date and accurate, it is important to implement continuous monitoring. This can include using automated tools to collect and analyze data in real-time, as well as implementing real-time alerts and notifications to identify potential issues.
5. **Use data to drive improvement:** Once data is collected and analyzed, it should be used to drive improvement. This can include identifying areas for improvement, implementing process changes, and measuring the impact of those changes over time.
6. **Share metrics with stakeholders:** Finally, DevSecOps Metrics should be shared with stakeholders, including developers, operations personnel, security professionals, and senior management. This will help ensure that everyone is aligned on organizational goals and objectives and will help drive accountability and ownership of security and compliance measures.

By implementing DevSecOps Metrics, organizations can gain greater visibility into the security and compliance of DevSecOps processes, identify areas for improvement, and measure the impact of process changes over time. This approach can help reduce the risk of security and compliance-related issues, improve overall application security and reliability, and enable organizations to demonstrate compliance to regulators and other stakeholders.

Securing the public endpoints

Securing public endpoints in the cloud is crucial to protect against unauthorized access and data breaches. Here are some best practices and solutions to consider:

- **Implement strong authentication mechanisms:** One of the most important ways to secure public endpoints is to implement strong authentication mechanisms such as **multi-factor authentication (MFA)** and OAuth. MFA adds an extra layer of security by requiring additional verification steps, while OAuth provides secure authorization for third-party applications.

- **Use HTTPS and SSL/TLS certificates:** Use HTTPS to encrypt traffic and SSL/TLS certificates to ensure secure communication between the user and the endpoint. This helps to prevent unauthorized access and data interception.
- **Implement access controls:** Implement access controls to restrict access to public endpoints based on user roles and permissions. This includes implementing firewalls, IP whitelisting, and network segmentation.
- **Regularly update and patch:** Ensure that public endpoints are regularly updated and patched to address any security vulnerabilities that may be discovered. This includes monitoring for and applying software updates, security patches, and configuration changes.
- **Use a Web Application Firewall (WAF):** A WAF provides an additional layer of security to protect against common web-based attacks, such as SQL injection and cross-site scripting (XSS).
- **Monitor and log activity:** Monitor public endpoint activity in real-time and log all events to help identify and respond to any security incidents. This includes monitoring access logs, network traffic, and system activity.
- **Conduct regular security audits:** Conduct regular security audits to identify potential vulnerabilities and risks. This includes reviewing access controls, monitoring logs, and conducting penetration testing.

By implementing these best practices and solutions, organizations can help secure their public endpoints in the cloud and protect against unauthorized access and data breaches. It is important to regularly review and update security measures to stay ahead of emerging threats and vulnerabilities.

Define policy and governance

Defining policies and governance in cloud DevOps is crucial for ensuring that the organization follows best practices and adheres to regulatory compliance requirements.

Here are some steps for defining best practices for policy and governance in cloud DevOps:

1. **Identify regulatory and compliance requirements:** The first step is to identify the regulatory and compliance requirements that are relevant to the organization's industry and location. This includes regulations such as GDPR, HIPAA, and PCI DSS.
2. **Define policy guidelines:** Based on the identified regulatory and compliance requirements, define policy guidelines that outline the best practices that should be followed to ensure adherence to these requirements. This can include guidelines for access control, data protection, and disaster recovery.
3. **Establish role-based access control:** Implement role-based access control to ensure that only authorized personnel have access to sensitive data and applications. This can be done using tools such as AWS Identity and Access Management (IAM) and Azure Active Directory (AD).
4. **Automate policy enforcement:** Use automation tools to enforce policy guidelines and ensure that best practices are being followed consistently. For example, use automated configuration management tools to ensure that servers and applications are configured in compliance with policy guidelines.
5. **Establish compliance reporting:** Implement compliance reporting to provide visibility into the organization's compliance status. This can include regular audits and reporting tools that provide real-time visibility into compliance status.
6. **Implement continuous monitoring:** Implement continuous monitoring to identify and remediate potential compliance issues in real-time. This can include using tools such as AWS CloudTrail and Azure Security Center to identify security and compliance issues.

By following these best practices for defining policy and governance in cloud DevOps, organizations can ensure that they are adhering to regulatory and compliance requirements while also improving their overall security posture. Additionally, implementing automation and continuous monitoring can help reduce the time and effort required to maintain compliance, allowing DevOps teams to focus on delivering value to the organization.

User right network tools to filter traffic

The right network tools to filter traffic in public cloud environments depend on the specific needs and requirements of the organization. However, here are some commonly used network tools that can help filter traffic in public cloud environments:

- **Virtual Private Network (VPN):** VPNs are used to create secure connections between remote users and the cloud infrastructure. VPNs can help filter traffic by providing secure access to cloud resources and enabling traffic filtering based on IP address, port, protocol, and other criteria.
- **Network Security Groups (NSGs):** NSGs are used to filter traffic at the network layer in Azure cloud environments. NSGs allow you to create and apply security rules to filter traffic based on source and destination IP addresses, ports, protocols, and other criteria.
- **Security Groups (SGs):** SGs are used to filter traffic at the network layer in AWS cloud environments. SGs allow you to create and apply security rules to filter traffic based on source and destination IP addresses, ports, protocols, and other criteria.
- **Firewall:** Firewalls are used to filter traffic based on predetermined security rules. In cloud environments, firewalls can be implemented as virtual appliances or cloud-based services. Firewalls can help filter traffic based on IP address, port, protocol, and other criteria.
- **Intrusion Detection and Prevention Systems (IDS/IPS):** IDS/IPS are used to detect and prevent network-based attacks. IDS/IPS can help filter traffic by monitoring network traffic and alerting administrators of suspicious activity.
- **Web Application Firewalls (WAFs):** WAFs are used to filter traffic to web applications. WAFs can help filter traffic based on HTTP request methods, headers, cookies, and other criteria.
- **Distributed Denial of Service (DDoS) Protection:** DDoS protection services are used to mitigate DDoS attacks by filtering traffic and blocking malicious requests.
- **Next-generation Firewall (NGFW):** Securing your on-premises network with NGFW is an effective way to protect your organization

from a wide range of network threats. There are so many famous **Network Virtual Appliances (NVA's)** available like **Palo Alto Networks, Check Point, Cisco, Juniper, ZScaller** etc.

These are some of the commonly used network tools to filter traffic in public cloud environments. It is important to evaluate the specific needs and requirements of the organization and choose the right tools accordingly.

Define and implement IAM, RBAC and 2FA

Defining and implementing **Identity and Access Management (IAM)**, **Role-Based Access Control (RBAC)**, and **Two-factor Authentication (2FA)** is essential in ensuring the security of cloud environments.

Here are some steps to follow:

1. **Define IAM and RBAC:** IAM defines the roles and responsibilities of users in the cloud environment, while RBAC defines the access privileges based on the user's role. Define the roles required for different users and assign them the necessary access privileges.
2. **Implement IAM and RBAC:** Implement IAM and RBAC policies to ensure that users only have access to the resources required to perform their job functions. Use the principle of least privilege and limit the access of each user to the minimum necessary resources.
3. **Implement 2FA: Two-factor Authentication (2FA)**: adds an extra layer of security by requiring users to provide two forms of authentication to access their accounts. Implement 2FA for all users, including administrators, to ensure secure access to cloud resources.
4. **Use Identity Providers (IdPs):** Use IdPs to manage user identities and authentication. Identity Providers such as Google, Microsoft, and Okta provide secure authentication and **Single Sign-On (SSO)** capabilities.
5. **Audit and Monitor IAM and RBAC:** Regularly audit and monitor IAM and RBAC policies to ensure that they are being enforced correctly. Use logging and monitoring tools to detect and respond to suspicious activity.

6. Implement Continuous Identity and Access Management:

Implement continuous Identity and **Access Management (IAM)** to ensure that access privileges are continuously reviewed and updated. This includes regular reviews of IAM policies, RBAC assignments, and user access privileges.

By defining and implementing IAM, RBAC, and 2FA, organizations can ensure that only authorized users have access to cloud resources, reducing the risk of data breaches and other security incidents.

Implementing least privilege model

Implementing the least privilege model is a DevOps best practice that aims to reduce the attack surface of an application or system by ensuring that users and processes have only the minimum level of access required to perform their tasks. This helps to prevent unauthorized access and reduce the potential impact of a security breach.

Here are some steps to implement the least privilege model:

- 1. Define access requirements:** Identify the minimum level of access that is required for each user and process to perform their tasks. This includes defining the roles and responsibilities of each user and process, as well as the data and resources they need to access.
- 2. Assign privileges accordingly:** Based on the access requirements defined in step 1, assign privileges to users and processes accordingly. This includes granting access to specific data and resources, as well as defining the level of access (for example, read-only, write, or admin access).
- 3. Monitor access:** Regularly monitor access to ensure that users and processes are only accessing the resources they need to perform their tasks. This can include using tools such as AWS CloudTrail and Azure Active Directory to monitor user activity and identify potential security threats.
- 4. Review and update privileges:** Regularly review and update privileges to ensure that users and processes have only the minimum level of access required to perform their tasks. This includes removing

access for users who no longer need it, as well as updating privileges in response to changes in roles and responsibilities.

5. **Implement role-based access control:** Implement role-based access control to ensure that users and processes have only the privileges required for their specific roles and responsibilities. This can include using tools such as AWS IAM and Azure RBAC to define roles and assign privileges accordingly.

By implementing the least privilege model as a DevOps best practice, organizations can reduce their attack surface and improve their overall security posture. This not only helps to prevent unauthorized access and data breaches, but also improves the reliability and performance of applications and systems by reducing the potential impact of security incidents.

Segregating DevOps network

Segregating the DevOps network is a best practice in DevSecOps that involves isolating the DevOps network from other networks to reduce the risk of unauthorized access, data breaches, and other security incidents. Here are some steps to implement this practice:

1. **Identify the DevOps network:** Identify the network used by the DevOps team to perform their tasks. This may include tools such as source control, build servers, and deployment environments.
2. **Segregate the DevOps network:** Segregate the DevOps network from other networks, including the corporate network and the production environment. This can be done by implementing network segmentation and firewall rules to restrict access to the DevOps network.
3. **Limit access to the DevOps network:** Limit access to the DevOps network to only those individuals who require access to perform their job functions. This includes implementing strong authentication mechanisms and access controls such as RBAC.
4. **Monitor the DevOps network:** Monitor the DevOps network for suspicious activity, such as unauthorized access attempts or data exfiltration. Implement logging and monitoring tools to detect and respond to security incidents.

5. **Test network segmentation:** Regularly test network segmentation to ensure that it is effective and working as intended. This can be done by performing penetration testing and vulnerability assessments.

By segregating the DevOps network, organizations can reduce the risk of data breaches and other security incidents by limiting access to only those who require it. It also ensures that any security incidents that occur within the DevOps network are contained and do not spread to other networks.

Using password manager

Using a password manager or vault is a DevOps best practice that involves securely storing and managing passwords and other sensitive information used by applications and systems.

Here are some steps to implement a password manager or vault as a best practice:

1. **Evaluate password managers or vaults:** Research and evaluate different password managers or vaults to find one that meets the specific needs and requirements of the organization. Consider factors such as security, ease of use, compatibility with existing tools, and cost.
2. **Configure the password manager or vault:** Once a password manager or vault has been selected, configure it according to the organization's needs and requirements. This includes setting up user accounts, defining access controls, and configuring integrations with other tools and systems.
3. **Train users:** Provide training and guidance to users on how to use the password manager or vault effectively and securely. This includes educating them on password best practices, such as creating strong passwords, not reusing passwords, and not sharing passwords with others.
4. **Integrate with CI/CD pipelines:** Integrate the password manager or vault with the organization's CI/CD pipelines to ensure that passwords and other sensitive information are securely managed throughout the entire software development lifecycle. This includes

automating the process of retrieving and using passwords during build and deployment processes.

5. Monitor usage: Regularly monitor usage of the password manager or vault to ensure that it is being used effectively and securely. This includes monitoring access logs and activity reports, as well as conducting periodic audits and reviews.

By using a password manager or vault as a DevOps best practice, organizations can improve the security and reliability of their applications and systems. This helps to prevent unauthorized access and data breaches, while also reducing the risk of human error and improving overall operational efficiency.

Conclusion

In conclusion, DevSecOps is a critical practice for organizations to adopt in order to ensure the security and reliability of their software applications and systems. By integrating security into the DevOps pipeline and making it a core part of the development process, organizations can reduce the risk of security breaches and vulnerabilities, while also improving their ability to respond to threats and incidents.

Throughout this chapter, we have explored some of the best practices for DevSecOps, including using secure coding practices, integrating compliance and governance, and collaborating between DevOps and security teams. We also discussed the importance of implementing a least privilege model, using a password manager or vault, and monitoring and measuring DevSecOps metrics.

While these best practices are not exhaustive, they provide a solid foundation for organizations to build upon as they implement DevSecOps practices. By following these guidelines, organizations can establish a culture of security and make security an integral part of their development process, ultimately leading to more secure, reliable, and resilient software applications and systems.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

<https://discord.bpbonline.com>



Index

A

- A/B testing deployment strategy [229, 230](#)
- Agile [6-8, 43, 44](#)
 - Agile methodologies
 - about [44](#)
 - Dynamic Systems Development Method (DSDM) [45](#)
 - Extreme Programming (XP) [45](#)
 - Feature-Driven Development (FDD) [45](#)
 - Kanban [45](#)
 - Lean software development [45](#)
 - Scrum [45](#)
 - Amazon Machine Image (AMI) [244](#)
 - Amazon Web Services (AWS) [12, 262](#)
 - Apache Subversion (SVN) [28](#)
 - AppDynamics [71](#)
 - Appium [150](#)
 - Application Performance Monitoring (APM)
 - about [190](#)
 - Azure Application Insights [197, 198](#)
 - Azure Application Monitoring [192, 193](#)
 - Azure Container Insights [199](#)
 - Azure VM Insights [200, 201](#)
 - benefits [190](#)
 - need for [191](#)
 - tools [191, 192](#)
 - Application Programming Interface (API) [163](#)
 - automated rollback scripts
 - about [213](#)
 - features [213](#)
 - automated testing [159](#)
 - automate security processes
 - about [269](#)
 - tools [269, 270](#)
 - automatic continuous monitoring solution
 - configuration [206, 207](#)
 - AWS Code Deploy [126](#)
 - Azure
 - web application monitoring [192](#)
 - Azure Application Insights [197, 198](#)
 - Azure Application Monitoring

- components 195
- configuration 193, 194
- Azure Container Insights 199
- Azure DevOps
 - about 15
 - features 15
- Azure Log Explorer
 - features 196
 - working 197
- Azure Metric Explorer
 - about 196
 - features 195
 - working 196
- Azure Monitor alerts
 - about 201
 - dashboards 204
 - data monitoring, visualizing 203, 204
- Power BI 205, 206
- setting up 202
- workbooks 204, 205
- Azure Resource Manager (ARM) 251
- Azure VM Insights 200, 201

B

- Bamboo
 - about 13, 14, 108, 127
 - advantages 108
 - features 13, 14
- Behavior-Driven Development (BDD) 150
- Bitbucket 28
- Bitrise
 - about 16
 - features 16
- blue-green deployment 231, 232
- Buddy
 - about 109
 - advantages 109
- buildkit 62
- build once, deploy many approach
 - about 58, 59
 - challenges 59, 60
 - cons 73
 - library dependencies 60, 61
 - limitation, testing 61
 - pros 72, 73
 - wrapping up 61

C

cache optimization 62
Cacti 184
canary deployment 226-229
CD pipeline consideration
 cycle time, evaluating 130, 131
 defect resolution time, examining 132
 lead time, analyzing 129, 130
 Mean Time to Recovery (MTTR), assessing 131, 132
 measuring 128, 129
 test pass rate 133-135
Check Point 283
CI/CD pipeline tools
 about 10
 Azure DevOps 15
 Bamboo 13, 14
 Bitrise 16
 CircleCI 12
 CodeShip 16
 Drone CI 17
 GitLab CI 13
 GoCD 15
 Jenkins 11
 Semaphore 14
 TeamCity 11
 Travis CI 12
 working 10, 11
CI/CD tools 171
Circle 127
CircleCI 12
 about 109
 advantages 109
Cisco 283
CI tools 107
Cloud Development Kit (CDK) 52, 245
CodeShip
 about 16
 features 16
compliance and governance
 implementing 273
compliance as code 278, 279
Configuration as Code (CaC) 251-253
Consistent Occlusion Concept (CCP) 32
containerization
 best practices 155, 156
 using 154, 155
container security 279, 280
continuous delivery

versus continuous deployment 70, 71
continuous delivery, benefits
 developer productivity, increasing 121, 122
 exploring 119-121
 feedback delivery, accelerating 123
 fully automatic CD process, building 128
 implementation, simplifying through automation 123
 market capability 125
 robust CD process, building 128
 robust CD process tools, selecting 125
 test quality, enhancing 124
continuous delivery, best practices
 about 135
 complexity, eliminating 145
 copy production, deploying 143, 144
 database 144, 145
 direct change, avoiding 142
 environment, deploying 141-143
 feedback 140, 141
 observability and continuous monitoring 146
 redundant process, automating 137, 138
 Service Level Objectives (SLO) 136
 SLO evaluation, automating with quality gates 137
 version control 138-140
continuous deployment
 about 70
 tools 71, 72
 versus continuous delivery 70, 71
continuous integration and delivery (CI/CD)
 about 10
 benefits 115
 feasibility 115
 principles 112
 workflow example 114
continuous integration (CI) 13, 27, 76
continuous monitoring
 types 181
continuous testing
 about 168, 169
 implementing 170
 test management tools, working 171
conventional agreement 85
conventional procurement process
 about 85-87
 benefits 86
 build binaries, setting up 87
 drawbacks 86, 87
 usage 86

Cross-Site Scripting (XSS) 269
Cucumber 150
customer relationship management (CRM) 79

D

Datadog 184
data extraction and transformation 159
delivery cycle
 performance testing, integration 163-165
Deploy Bot 127
deployment pipeline security
 integrating 274, 275
design-bid-build 85
Desired State Configuration (DSC) 251
development environment
 managing 89
Development Operations (DevOps)
 about 2, 6-8, 266, 276, 277
 advantages 4, 5
 best practices 8
 classification 2, 3
 disadvantages 5
 need for 46
 overview 45-47
 techniques 6
 usage 3, 4
 version control tools 27
DevOps, best practices
 application monitoring 10
 automated testing 9
 change management 9
 continuous delivery 10
 continuous deployment 9
 continuous integration 9
 continuous product 10
 dashboard automation 10
 environment building 9
 integrated configuration 9
 stakeholder method 9
DevOps challenges
 about 17
 CD practice 18
 change management 19
 continuous learning 20
 deployment and releases 19
 dev and ops toolset 20
 dev, versus ops 18
 key metrics 20

legacy architecture, to microservices 18
legacy infrastructure, to microservices 18
test automation strategy, implementing 18
tools and technologies 19

DevOps implementation
about 21
automation 22
collaboration, encouraging 21
compatible tools, selecting 22
continuous deliver 23
continuous integration 23
customer satisfaction 21
monitoring 23, 24
organizational silos 21
performance reviews, defining 22
practice, evaluating 21
real-time visibility, ensuring 23
scale limitation 22

DevOps network
segregating 285

DevOps principle
about 49
virtualized technology platforms, using 50

DevSecOps metrics 280, 281

DevSecOps principles
about 267
automate security processes 269
best practices 268
examples 268
teams collaboration 270

DevSecOps training and education
implementing 274

distributed system
about 32
application, decomposing into services 33, 34
business capability, establishing 34

Docker Compose 63

Docker container
versus Docker image 63

Docker file method 66

Docker image
anatomy 64
building 62
commands 66, 67
creating 65

Docker file method 66
interactive method 66
repositories 65

use cases 63
versus Docker container 63
Docker image, types
 buildkit 62
 cache optimization 62
 Docker Compose 63
 multi-stage build 62
 single-stage build 62
 third-party tools 63
Domain-Specific Languages (DSLs) 238, 262
Drone CI
 about 17
 features 17

E

environment cleaning 88, 89
environment overlays
 about 67
 application deployment, affecting 69, 70
 benefits 68
 working 68
Extreme Programming (XP) 9, 44

F

feature environment
 about 38
 accelerated shipping 52
 benefits 39, 40
 Cloud Development Kit (CDK) 52
 code review process 40
 feature branch strategy 40, 41
 information process automation 42
 information process orchestration 42
 installation process, accelerating 42
 modification, evaluating 54, 55
 operation coordination 43
 production organization 43
 robotics 51
 routing 51
 three-step fundamental concept 51, 52
feature environment, deployment automation
 about 53
 characteristic elements 53
 feature climate, creating 53
 unique atmosphere 53
feature toggle rollback strategy
 about 223, 224

implementing 223
fully automatic CD process
 building 128
fully automatic CI process
 building 111-114
fully automatic continuous testing environment
 building 172-177

G

Gatling 151
Git
 defining 83-85
 using 34, 35
 working 35
GitHub 27
GitLab
 about 27, 110, 127
 key points 110
GitLab CI 13
GitOps methodology
 about 257
 benefits 258
 key aspects 257
 use cases 260, 261
 workflow 258-260
Global Information Tracker (GIT) 127
GoCD 15
Google Cloud Deployment Manager 127
Google Cloud Platform (GCP) 262
Graphical User Interfaces (GUIs) 251

H

headless execution
 about 158-160
 benefits 158
Hg 28, 29
hotfix rollback strategy
 about 221
 implementing 222

I

Identity and Access Management (IAM)
 about 283
 defining 283
immutable infrastructure rollback strategy 225, 226
infrastructure

change management 50
Infrastructure as a Service (IaaS) 47
Infrastructure as Code (IaC)
 about 238
 benefits 238, 239
 examples 239
 tools 240, 241, 261, 262
 types 239
infrastructure monitoring
 about 186
 benefits 187
 configuration 188, 189
 tools 187, 188
Internet Codeship 127
Internet Service Providers (ISPs) 50

J

Jenkins
 about 11, 107, 126, 150
 advantages 108
Juniper 283
JUnit 150

K

Key Performance Indicator (KPI) 136

L

least privilege model
 implementing 284, 285
LoadRunner 150

M

ManageEngine OpManager 183
manual rollback procedure 212
Mean Time to Recovery (MTTR) 131, 152
Mean Time to Resolution (MTTR) 120
metrics
 tracking 152-154
 types 152
multi-layer tests
 benefits 161
 exploring 161-163
multi-stage build 62

N

Nagios 146, 183
network monitoring
about 182
benefits 184
configuration 185, 186
need for 183
tools 183
usage 185
Network Performance Monitor (NPM) 183
Network Virtual Appliances (NVAs) 283

O

Object-oriented Design (OOD) 32
Octopus deploy 126
Open Policy Agent (OPA) 254

P

Palo Alto Networks 283
password manager
using 286
Perforce 28
performance testing
benefits 165
integration, into delivery cycle 163-165
performance 166
pinning 61
pipeline
measuring 92-94
monitoring 92, 93
Pipeline as a Code
about 244
benefits 245-247
Platform as a Service (PaaS) 47
Platform as Code (PaC)
about 247, 250, 251
key aspects 248
policy and governance
defining 282
Policy as Code (PaC)
about 253, 256
key aspect 254
Power BI 205, 206
production deployment 99-102
production environment
managing 91
PRTG network monitor 183
public endpoints

securing 281

Q

quality analyst (QA) 70

R

Rational Unified Process (RUP) 77

right network tools

 using, to filter traffic 282, 283

robust CD process

 building 128

robust CD process, tools

 AWS Code Deploy 126

 Bamboo 127

 Circle 127

 Deploy Bot 127

 GitLab 127

 Google Cloud Deployment Manage 127

 Internet Codeship 127

 Jenkins 126

 Octopus deploy 126

 selecting 125, 126

 TeamCity 127

robust CI process

 building 111-114

 tools, building 106-110

robust continuous testing environment

 setting up 167, 168

robust continuous testing tools

 building 167

 Jenkins 167

 JUnit 167

 LoadRunner 167

 Sauce Labs 167

 Selenium 167

 TestComplete 167

 TestNG 167

robust testing environment

 building 172-177

Role-Based Access Control (RBAC)

 about 283

 implementing 284

rollback strategies

 about 210, 211

A/B testing deployment strategy 229

automated rollback scripts 213

blue-green deployment 231

canary deployment 226
feature toggle rollback strategy 223, 224
hotfix rollback strategy 221
immutable infrastructure rollback strategy 225, 226
manual rollback procedure 212
shadow deployment 233
snapshot backups 213, 217
version control 217

S

secure coding practices
 using 272
security by design
 examples 271
 implementing 271
 secure coding practices, using 272
security collaboration 276, 277
Selenium 150
Semaphore 14
 features 14
Service Level Agreements (SLAs) 137
Service Level Indicator (SLI) 136
Service Level Objectives (SLO) 136
shadow deployment 233, 234
shadow testing 233
shortened installation phase
 about 47
 creating 47
 features 48
Single-responsibility Principle (SRP) 32
Single Sign-On (SSO) 284
single-stage build 62
snapshot backups
 about 213
 benefits 214
 disk backup, maintaining 215
 disk restore 216
 limitations 214, 215
 types 214
 working 214
SoapUI 150
Software as a Service (SaaS)
 about 47, 79
 examples 79, 80
software build
 automating 94-97
software development
 challenges 48, 49

Software Development Life Cycle (SDLC) 75, 128
software development methodology 125
source code management (SCM) 26
source control 26
Spicework 183
Splunk 71
staging environment
managing 90, 91
Storage Area Network (SAN) 214
streamline testing
about 97, 98
analysis 98
design 99
planning 98
Subversion (SVN) 211

T

TeamCity 127
about 11, 108
advantages 108
features 12
team effort
about 102
benefits 103
examples 104-106
teams collaboration
examples 270, 271
Terraform
about 241
configuring 244
example 243
workflow 242
test automation
adopting 148, 149
TestComplete 150
Test-Driven Development (TDD) 124
test environment
managing 90
test management tools
working 171
TestNG 150
test pass rate
about 133
working 133, 134
threat modelling 277, 278
Time to Repair (TTR) 132
tool integration 150, 151
traditional software development 76-82

transparent communication 156-158

Travis CI

about 12, 109

advantages 109

Two-factor Authentication (2FA)

about 283

implementing 284

U

user interface (UI) 99

V

version control

about 26, 217

benefits 26, 27

benefits, for application backup and restore 218

bug suppressing 30, 31

concurrent growth 31

final result 31

for application backup and restore 217

limitations, for application backup and restore 218

need for 30

release version, creating 218-221

types, for application backup and restore 218

working, for application backup and restore 217

version control system (VCS) 26, 27, 83, 138

version control tools

Apache Subversion (SVN) 28

Bitbucket 28

GitHub 27

GitLab 27

Hg 28, 29

in DevOps 27

Perforce 28

working 29, 30

Virtual Machines (VMs) 200

visibility and control

maintaining 275, 276

W

web application monitoring

on Azure 192

web scraping 159

Wireshark 183

Z

[Zabbix 183](#)

[ZScaller 283](#)