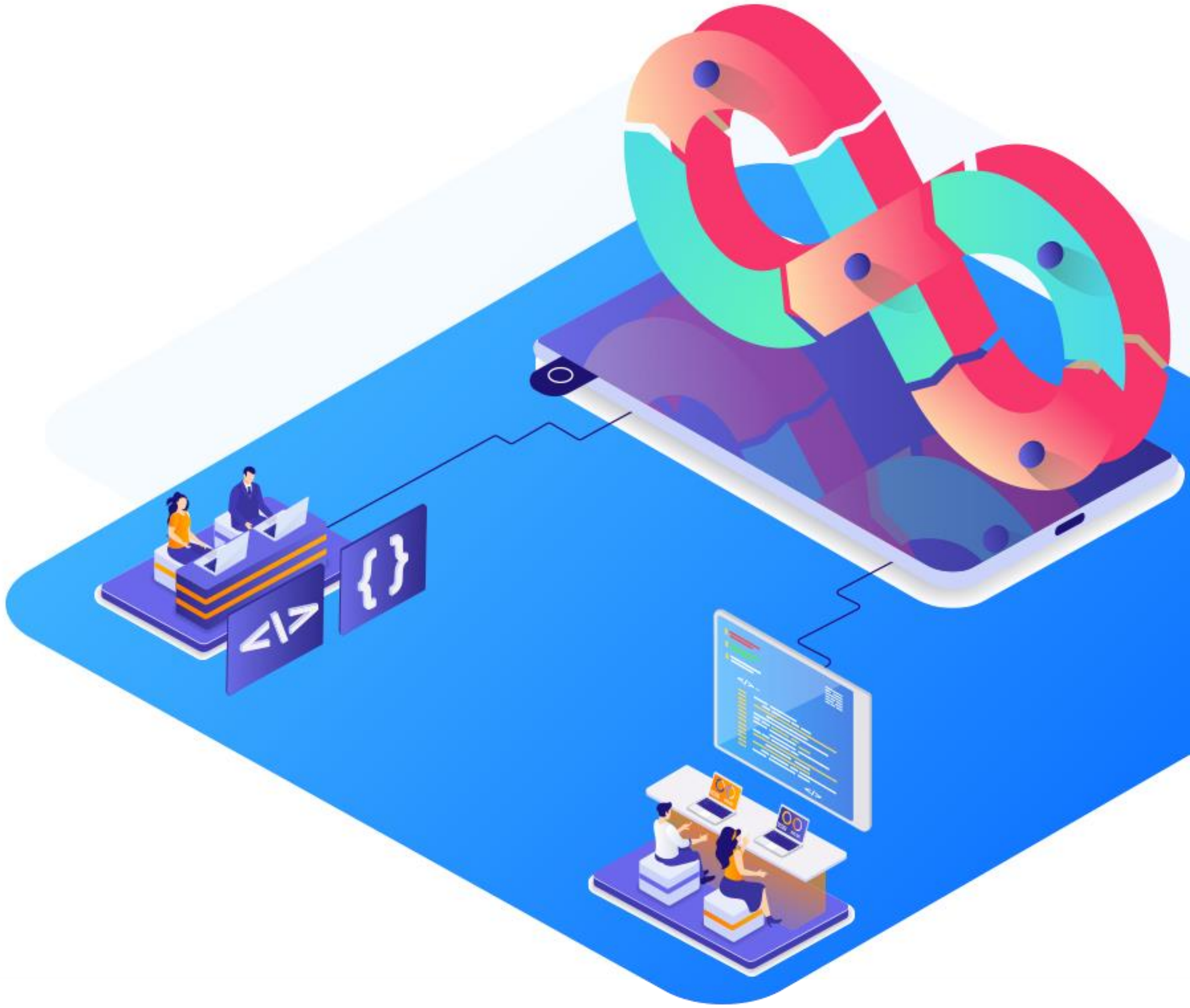
DevOps Foundations: Version Control and CI/CD with Jenkins



Entering the World of DevOps

Learning Objectives

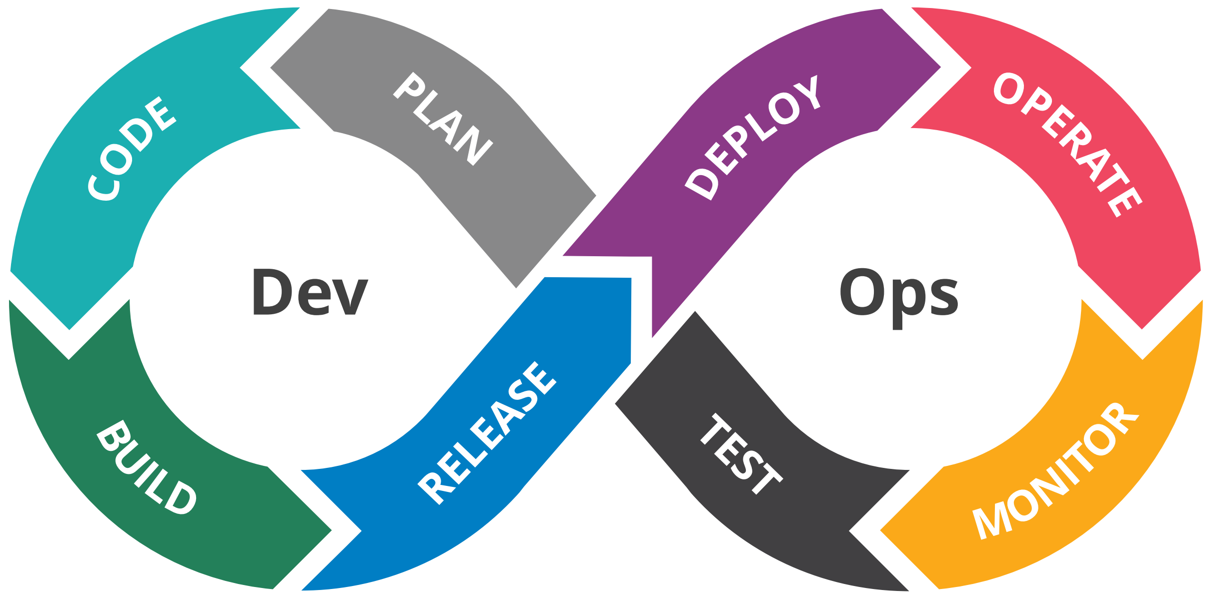
By the end of this lesson, you will be able to:

- Apply DevOps concepts to improve collaboration between development and operations teams, enhancing software quality and deployment frequency
- Comprehend how DevOps and Agile methodologies work together in software development to enhance project flexibility and speed up the product release cycles
- Identify the key aspects of how to transition from traditional approach to DevOps for software development
- Utilize DevSecOps approaches to build a secure CI/CD pipeline

Getting Started with DevOps

What Is DevOps?

It is a software development practice or mindset that promotes collaboration between development and operations.



The aim of DevOps is to optimize the development cycle and ensure continuous delivery of high quality software reducing the time to market.

Benefits of DevOps

Speed	Efficiency	Communication	Reliability
Increases workflow speed resulting in reduction of time-to-market	Streamlines transitions of stages of lifecycle and task management in the pipeline	Enhances team collaboration via a centralized system	Helps achieve consistent project delivery by a robust infrastructure

Benefits of DevOps

Quality

Helps in continuous improvement of project quality through iterative enhancements

Consistency

Promotes consistency in workflow and results through iterations

Agility

Responds proactively to changes and enhances planning effectiveness

Adaptability

Ensures efficient real-time response to changes and issues

DevOps Principles

Collaboration

Establish a collaboration between the development and operations team

Automation

Automate processes and increase productivity

Continuous improvement

Experiment, minimize waste, and optimize delivery through iterations

Customer-centric action

Prioritize customer feedback and response through continuous monitoring

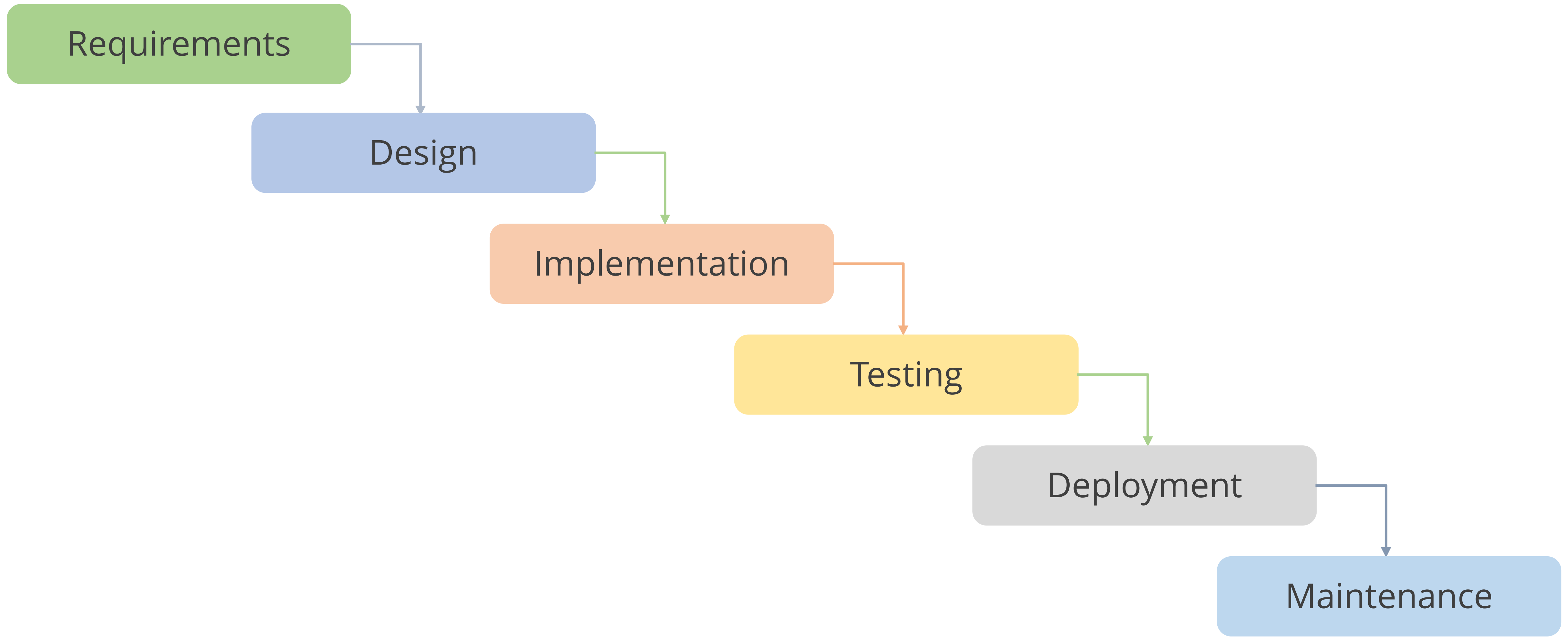
Understanding customer needs

Create products and services centered around user needs

Traditional Approach

To understand why one should choose DevOps, we need to get the understanding of the traditional approach of development.

The traditional software development approach (waterfall method) has a sequence of activities for system designers and developers to plan, create, test, and deploy a software system.



Challenges in the Traditional Approach

Using traditional methods poses major challenges in development due to their drawbacks.

Rigidity in process

Blocks adjustments to project needs or market changes, causing expensive revisions

Long time to market

Impacts fast-paced industries and traditional phased development that delays market entry

Challenges in the Traditional Approach

Using traditional methods poses major challenges in development due to their drawbacks.

Difficulty in handling changes

Encounters development disruptions, leading to delays and higher costs

Limited customer feedback

Delays in traditional development hinder user feedback, compromising product effectiveness

Traditional Approach vs. DevOps

Aspect	Traditional Approach	DevOps
Focus	Improves infrastructure only	Improves infrastructure and application
Speed	Slower recovery and releases	Faster recovery and releases
Development model	Linear model	Continuous integration and deployment
Teamwork	Focuses on specific functions without cross-team collaboration	Emphasizes a collaborative approach where different teams work together

Adopting DevOps Model

Here are the concepts or best practices required to adopt DevOps:

Continuous integration	Merging multiple code changes into a central repository to allow developers to improve software quality
Continuous delivery	Automating building, testing, and deployment of code for feature releases
Monitoring and logging	Monitoring application performance and logs for proactive issue detection and resolution

Adopting DevOps Model

Infrastructure as Code (IaC)	Managing infrastructure using code to enable scalable and consistent deployments
Microservices	Designing applications as small and independent services for flexibility, scalability, and isolated deployment
Communication and collaboration	Fostering teamwork and communication between teams for efficient project delivery and alignment

Assisted Practice

Implementing the DevOps model **Duration: 15 Min.**

Problem statement:

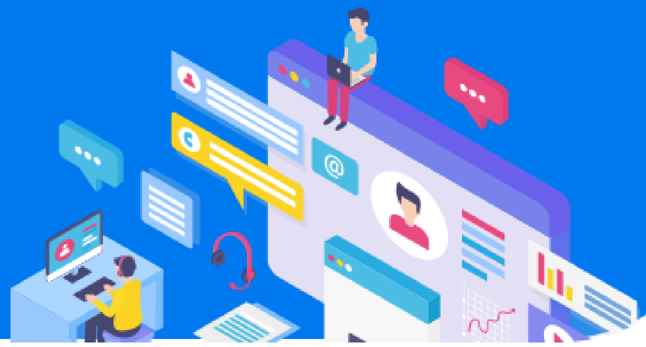
You have been assigned a task to implement DevOps using GitHub to store a Java program and Jenkins to build consistent code packages, enabling continuous integration and continuous deployment.

Outcome:

By completing this demo, you will be able to gain proficiency in implementing a CI/CD pipeline by creating a GitHub repository, adding a Java program, setting up a Jenkins freestyle build job, and automating the build process for continuous integration and deployment.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines



Steps to be followed:

1. Create a GitHub repository
2. Add a Java program to the repository
3. Create a freestyle build job in Jenkins
4. Build the Java program with Jenkins

Quick Check

As an IT manager leading an online music streaming application, you are considering adopting DevOps practices. How would you explain the major difference between DevOps and the traditional approach?

A. DevOps emphasizes separate teams for development and operations.

B. DevOps focuses on manual software deployment.

C. DevOps promotes automation, collaboration, and continuous integration and deployment.

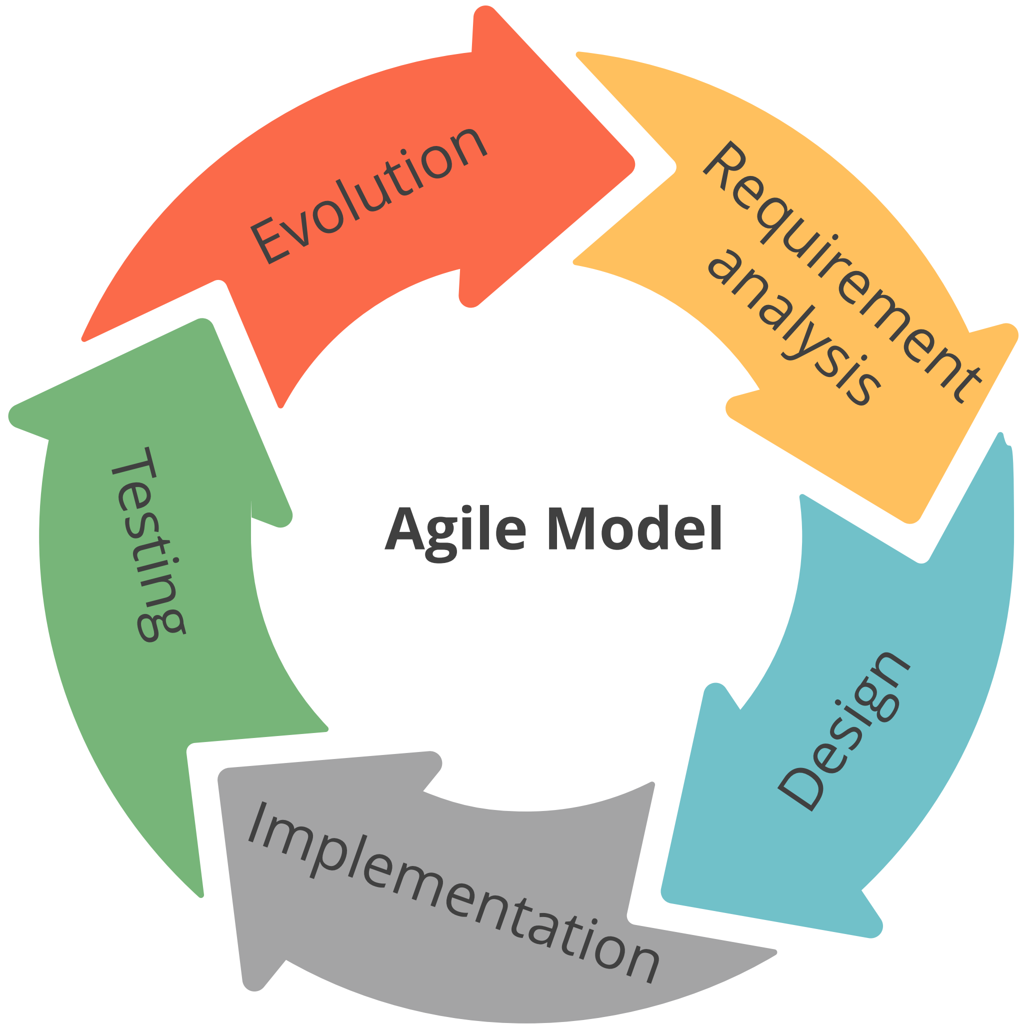
D. DevOps results in slower software delivery compared to the traditional approach.

Agile and DevOps

What Is Agile?

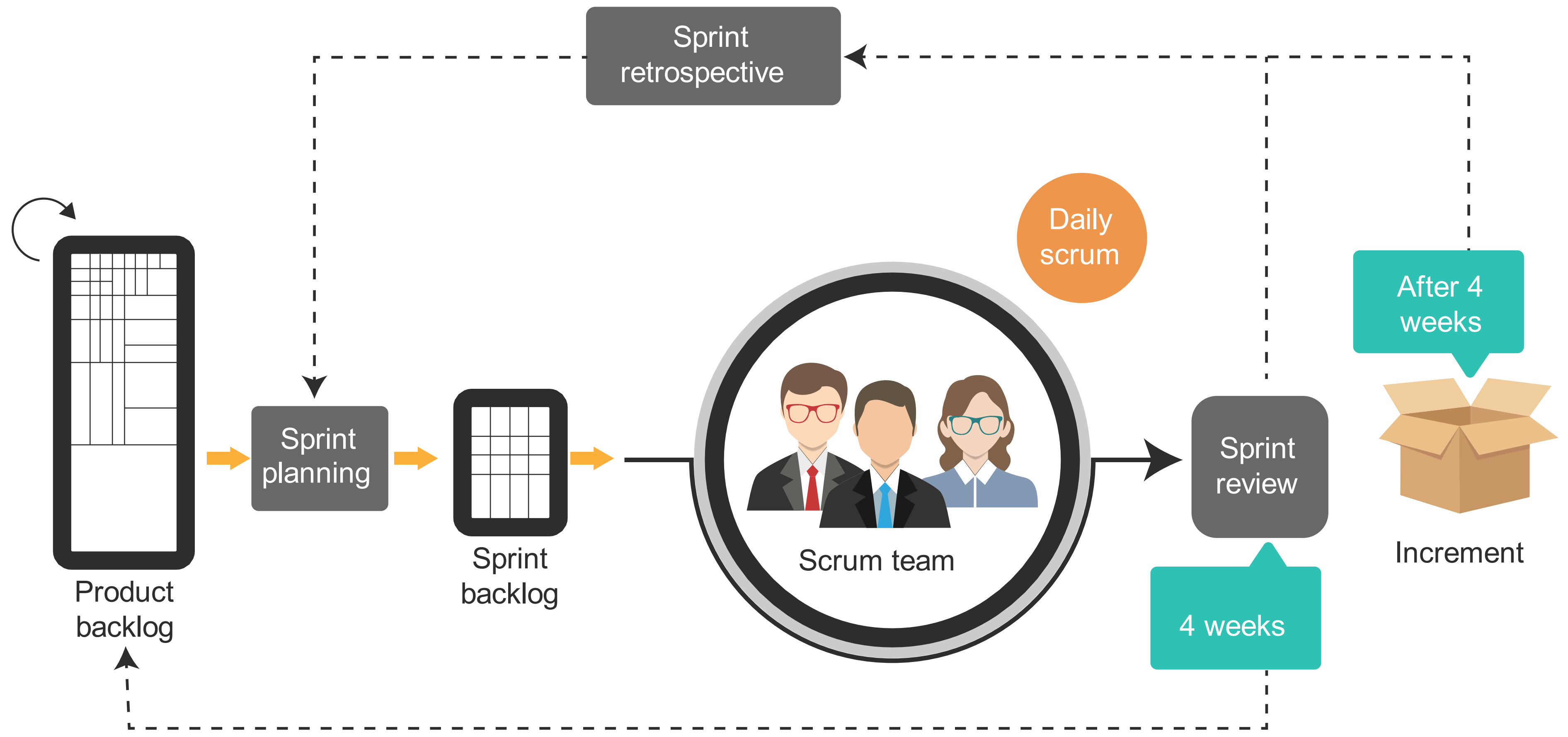
Agile is another popular framework or approach towards optimizing SDLC.

It is an iterative development process that promotes continuous iteration of development and testing throughout the project lifecycle.



Agile Model: Example

Here is a visual representation of the Scrum methodology under Agile for project management.



Advantages of Agile

The advantages of Agile methodologies in software development and project management include:

- Interactions with stakeholders and team members
- Continuous and fast delivery
- Continuous attention to technical excellence and better design
- Adaptation to the regular changing requirements
- Continuous testing and deployment

DevOps vs. Agile

Aspect	Agile	DevOps
Focus	Development process, iterative progress, collaboration, and customer feedback	Entire software delivery lifecycle, collaboration between development and operations
Principles	Customer collaboration, responding to change, delivering working software frequently (Agile Manifesto)	Automation, continuous integration/continuous delivery (CI/CD), monitoring, feedback loops
Practices	Scrum, Kanban, Extreme Programming (XP)	Infrastructure as code (IaC), CI/CD, automated testing, monitoring
Teams	Small, cross-functional teams including developers, testers, and business analysts	Collaboration between development, IT operations, QA, and sometimes security (DevSecOps)
Cycle	Short iterations or sprints (1-4 weeks) resulting in potentially shippable product increments	Continuous flow of code from development to production, enabling frequent and reliable releases
Goals	Deliver high-quality software quickly and respond effectively to changing requirements	Improve the speed and reliability of software delivery, ensuring high-quality software can be released quickly and safely

DevOps with Agile

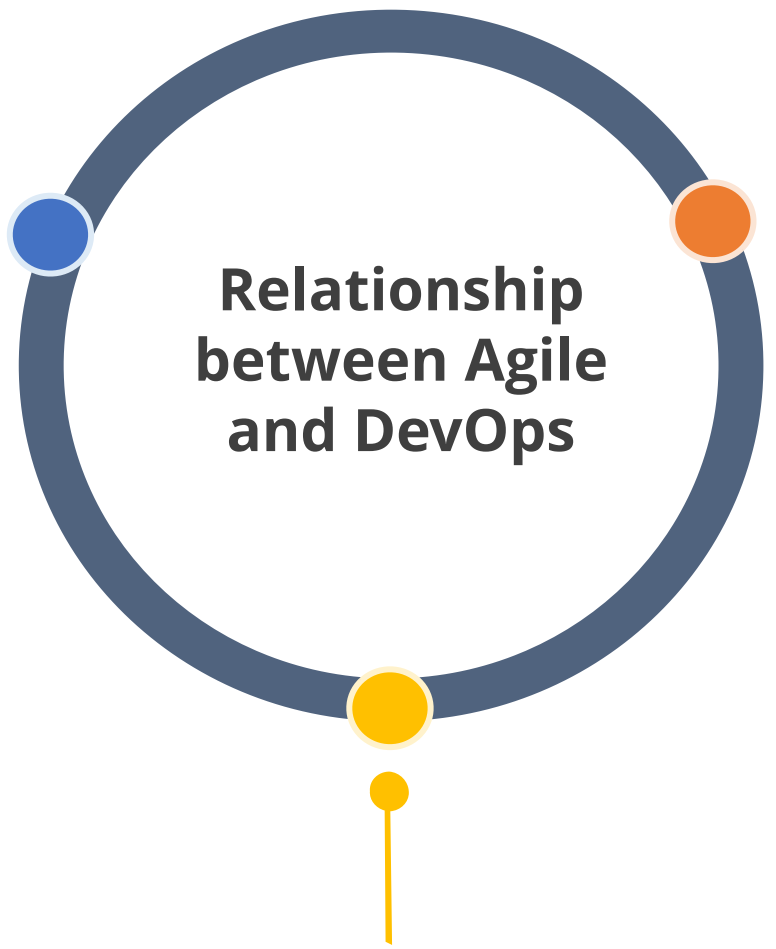
Below are the key points that can be considered when DevOps is combined with Agile:

Satisfy customers through early and continuous delivery of valuable software with each sprint

Deliver working software frequently with a preference for shorter timescale and promote continuous feedback

Promote a culture of collaboration and continuous improvement

DevOps with Agile



Replaces non-human steps using automation tools

Improves the collaboration between the teams

Automates the process and development itself to create a minimal viable product incrementally

Quick Check

As a developer, how do you see the integration of DevOps practices with Agile methodologies benefiting your project?

A. By emphasizing strict development timelines

B. By promoting collaboration between development and operations teams

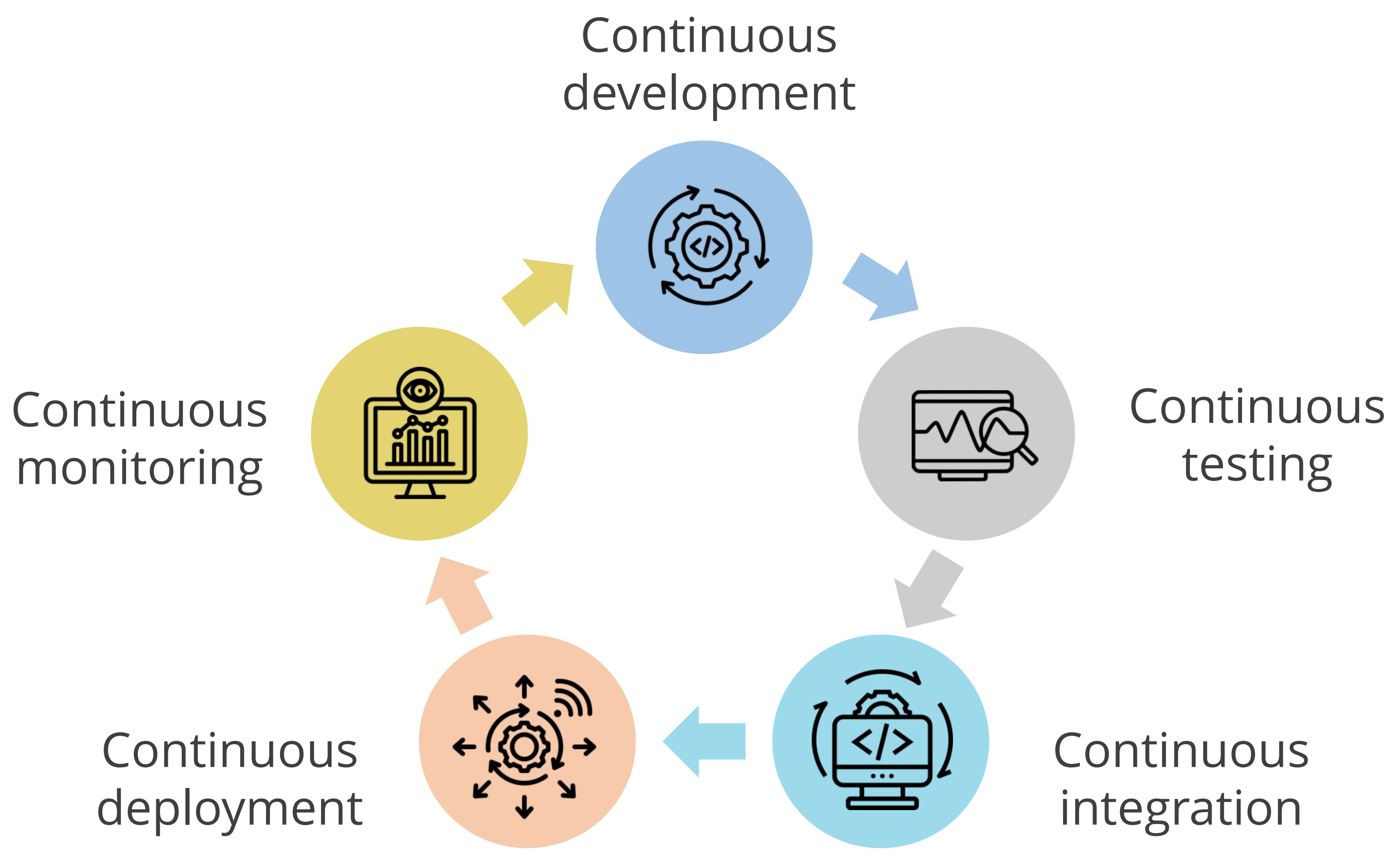
C. By discouraging automation in the development process

D. By following a waterfall approach to software development

DevOps: Architecture, Lifecycle, and Essential Tools

DevOps Architecture

The following architecture of DevOps depicts a flow of the execution for developing a software and managing large distributed applications efficiently:



DevOps Architecture

Continuous development

Involve continuously coding and building new features or updates for software applications

Continuous testing

Automate testing of code changes to ensure quality and identify bugs early in the development process

Continuous integration

Merge code changes into a shared repository and perform automated builds and tests to detect integration issues

DevOps Architecture

Continuous deployment

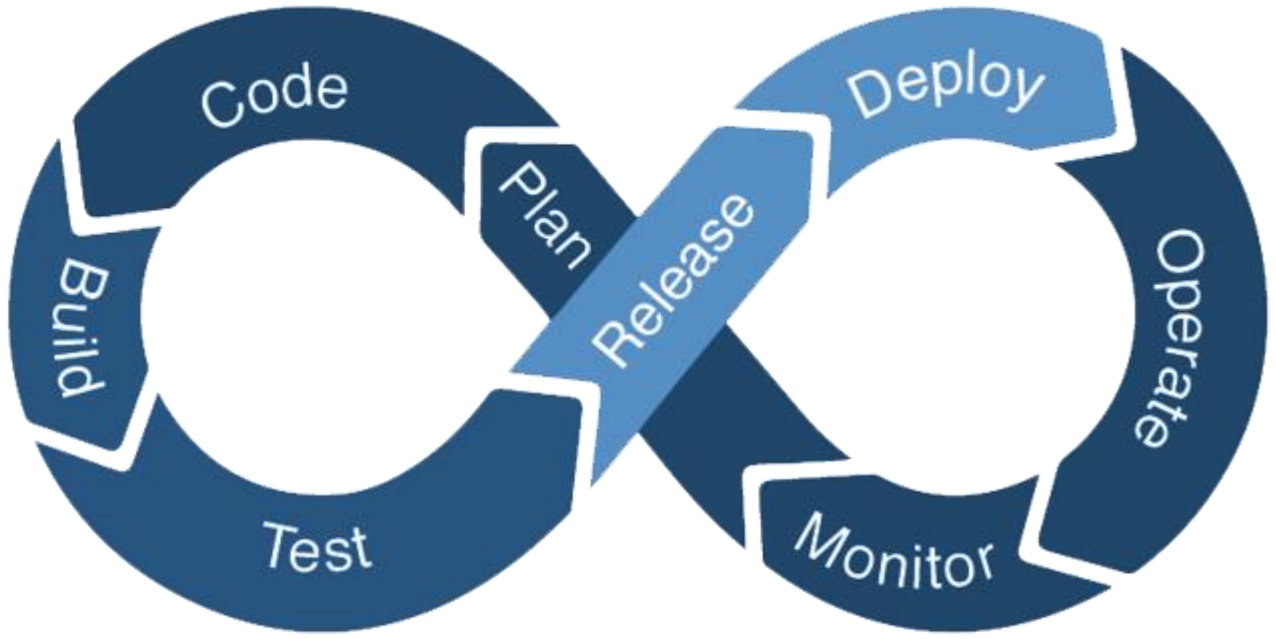
Deploy code changes automatically to testing or production environments after passing through the build and test stages

Continuous monitoring

Monitor application performance and infrastructure in real time to detect issues, gather insights, and ensure system reliability and performance

DevOps Lifecycle

It is a collaborative and iterative process, focusing on delivering software that meets the specific needs of businesses and their users through regular feedback.



DevOps Lifecycle

Below are the phases of DevOps lifecycle:

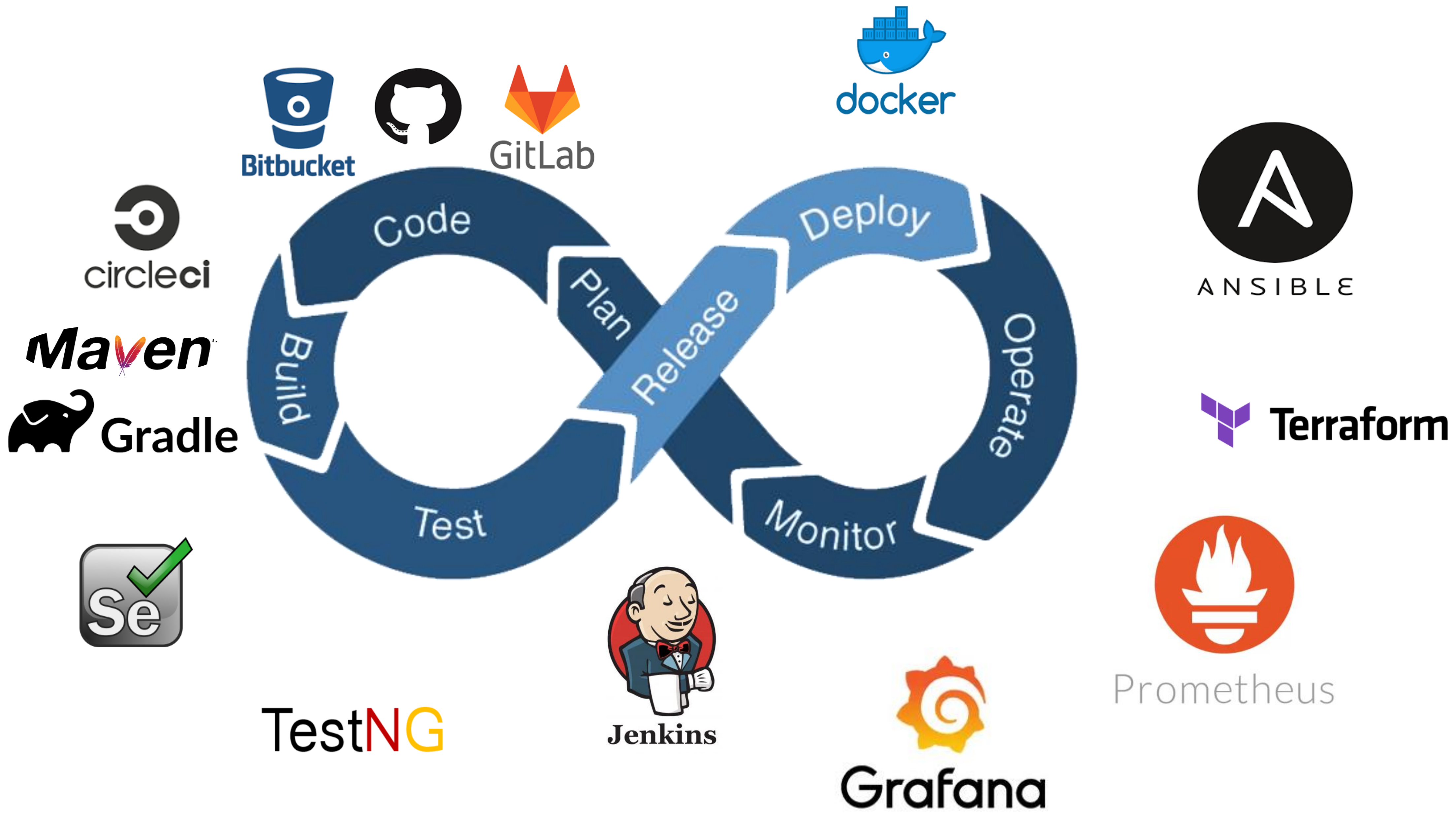
1. Plan	Define project goals, requirements, and timelines, and create a roadmap for development tasks
2. Code	Write and collaborate on code using version control systems for managing changes and revisions
3. Build	Compile, integrate, and automate the build process to create executable software artifacts
4. Test	Conduct automated and manual testing to validate software functionality, performance, and quality

DevOps Lifecycle

5. Deploy	Automate deployment processes to smoothly transition software to target environments
6. Operate	Manage the software in production, handle incidents, and ensure system reliability and availability
7. Monitor	Monitor system performance, availability, and security continuously to ensure optimal functioning and identify areas for improvement

DevOps Tools

Essential tools for implementing DevOps in the software development lifecycle (SDLC) are:



DevOps Tools

SCM tools

For source code management (SCM), version control tools such as Git, GitHub, Subversion, TFS, and Mercurial are used.

Software build tools

For automating the build process of an executable application from source code, software build tools such as Maven, Gradle, Ant, and Grunt are used.

DevOps Tools

Testing tools

In continuous testing phase, the built software is continuously tested for bugs using testing tools such as Selenium, TestNG, and Junit.

Integration tools

CI/CD pipelines are created for procuring updated source code and constructing the build into *.exe* format using tools such as Jenkins.

DevOps Tools

CMT and deployment tools

For the deployment and operations phase, CMT and automation tools such as Jenkins, AWS Code Deploy, Chef, Puppet, Ansible, and Terraform are used.

Monitoring tools

For monitoring system performance and productivity (to reduce or even eliminate downtime), monitoring tools such as Nagios, Grafana and Prometheus are used.

DevOps Tools

Containerization tools

For packaging an application with its required libraries, frameworks, and configuration files to efficiently run it in various computing environments, containerization tools such as Docker and Kubernetes are used.

Companies Using DevOps

Adobe

Walmart

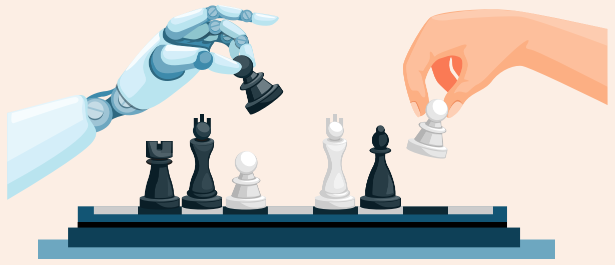
accenture

Google

NETFLIX

Case Study

Infrastructure provisioning Bottleneck (E-commerce startup)

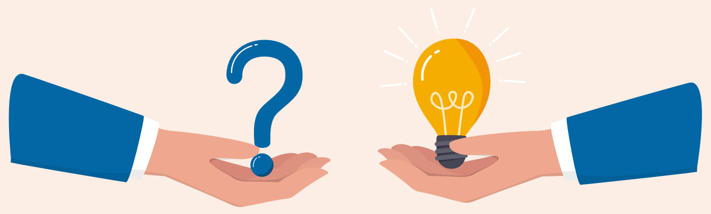


Challenges

- Provisioning new infrastructure environments was a slow manual process, hindering development agility.
- Maintaining consistency across development, testing, and production environments was a challenge.
- Existing infrastructure struggled to handle sudden spikes in traffic during peak shopping seasons.

Case Study

Infrastructure provisioning Bottleneck (E-commerce startup)



Solution

- Implemented Infrastructure as Code (IaC) using tools like Terraform. This allowed developers to define infrastructure configurations in code, enabling automated provisioning through the CI/CD pipeline.
- Defined infrastructure configurations as reusable modules in IaC. These modules could be easily deployed across different environments with slight variations.
- Implemented auto-scaling features in the cloud platform, which automatically scaled infrastructure resources up or down based on real-time traffic demands.

Case Study

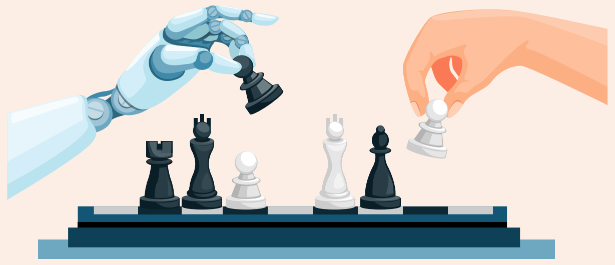
Infrastructure provisioning Bottleneck (E-commerce startup)

Outcome

- Faster deployment of new features and environments
- Increased consistency and repeatability in infrastructure setup
- Simplified environment management for developers and operations teams
- Improved application performance and availability during peak traffic periods
- Reduced infrastructure costs by optimizing resource utilization

Case Study

Netflix infrastructure transformation

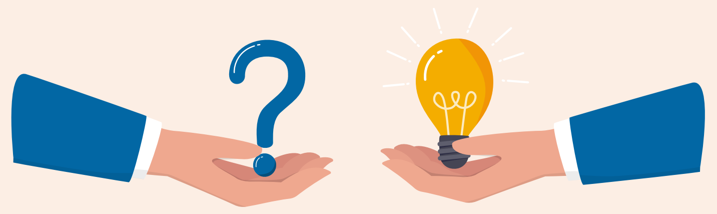


Challenges

- Provisioning new infrastructure environments was a slow manual process, hindering development agility.
- Maintaining consistency across development, testing, and production environments was challenging.
- Existing infrastructure struggled to handle sudden spikes in traffic during peak streaming periods.

Case Study

Netflix infrastructure transformation



Solution

- Adopted cloud-based infrastructure provisioning and automation tools for faster and scalable environment setup
- Implemented Infrastructure as Code (IaC) practices using tools like Terraform to ensure consistency and streamline environment management
- Leveraged cloud scalability to seamlessly scale resources based on demand

Case Study

Netflix infrastructure transformation

Outcome

- Reduced provisioning time from weeks to minutes, enhancing development agility and speed of deployment
- Improved environment consistency, reduced configuration errors, and increased overall efficiency in managing different environments
- Improved scalability and performance during peak traffic periods, ensuring uninterrupted streaming experiences for users and mitigating infrastructure-related issues

Quick Check

Imagine you are tasked with ensuring smooth configuration management across development, testing, and production environments. Which tool would you select to achieve this seamlessly?

A. Docker

B. Chef

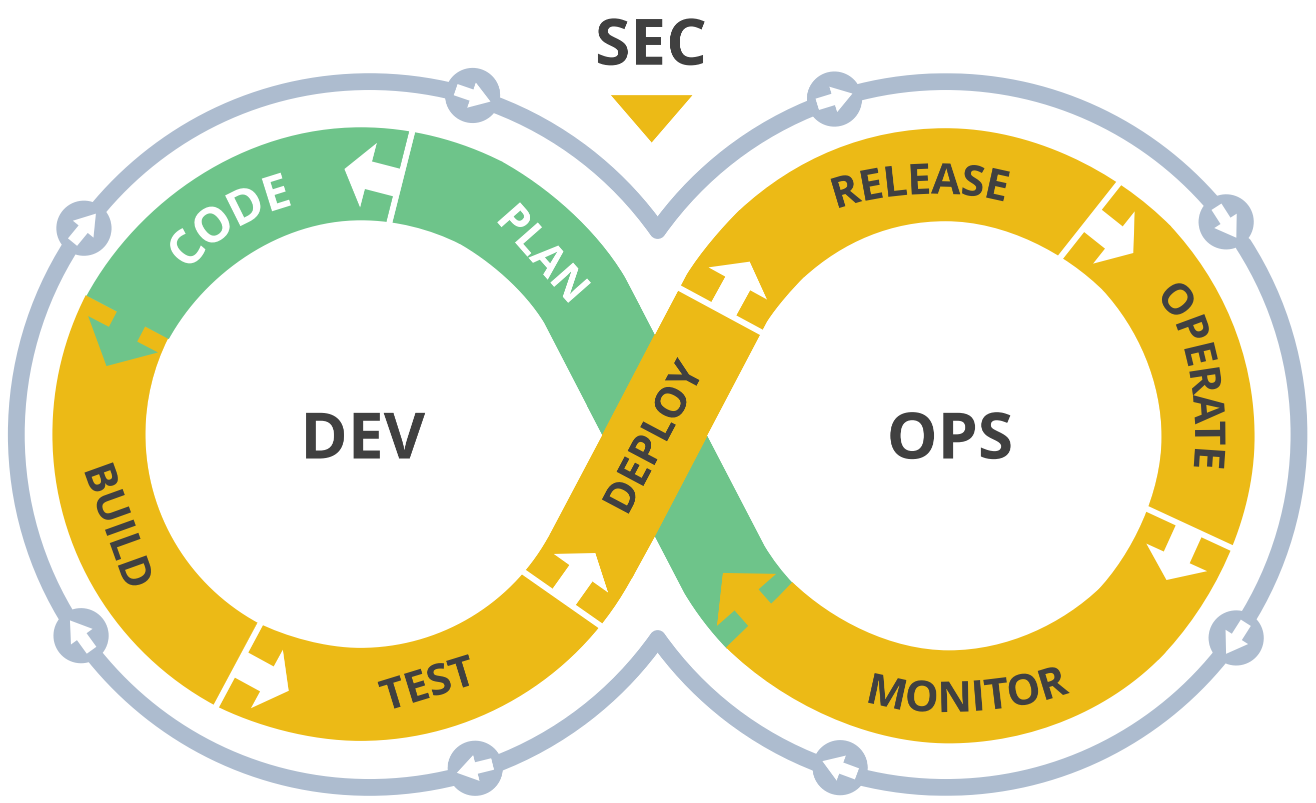
C. Git

D. Ansible

Introduction to DevSecOps

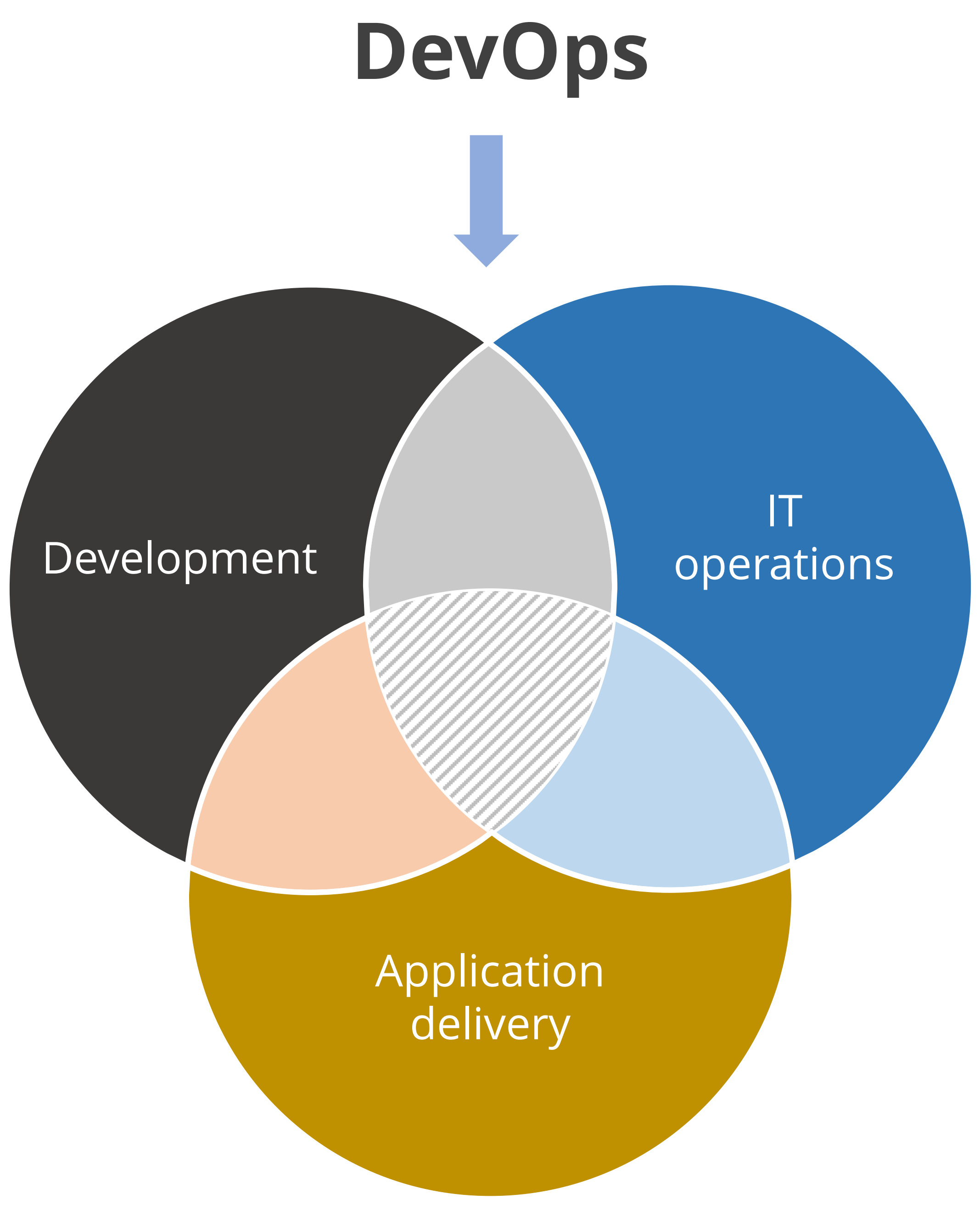
What Is DevSecOps?

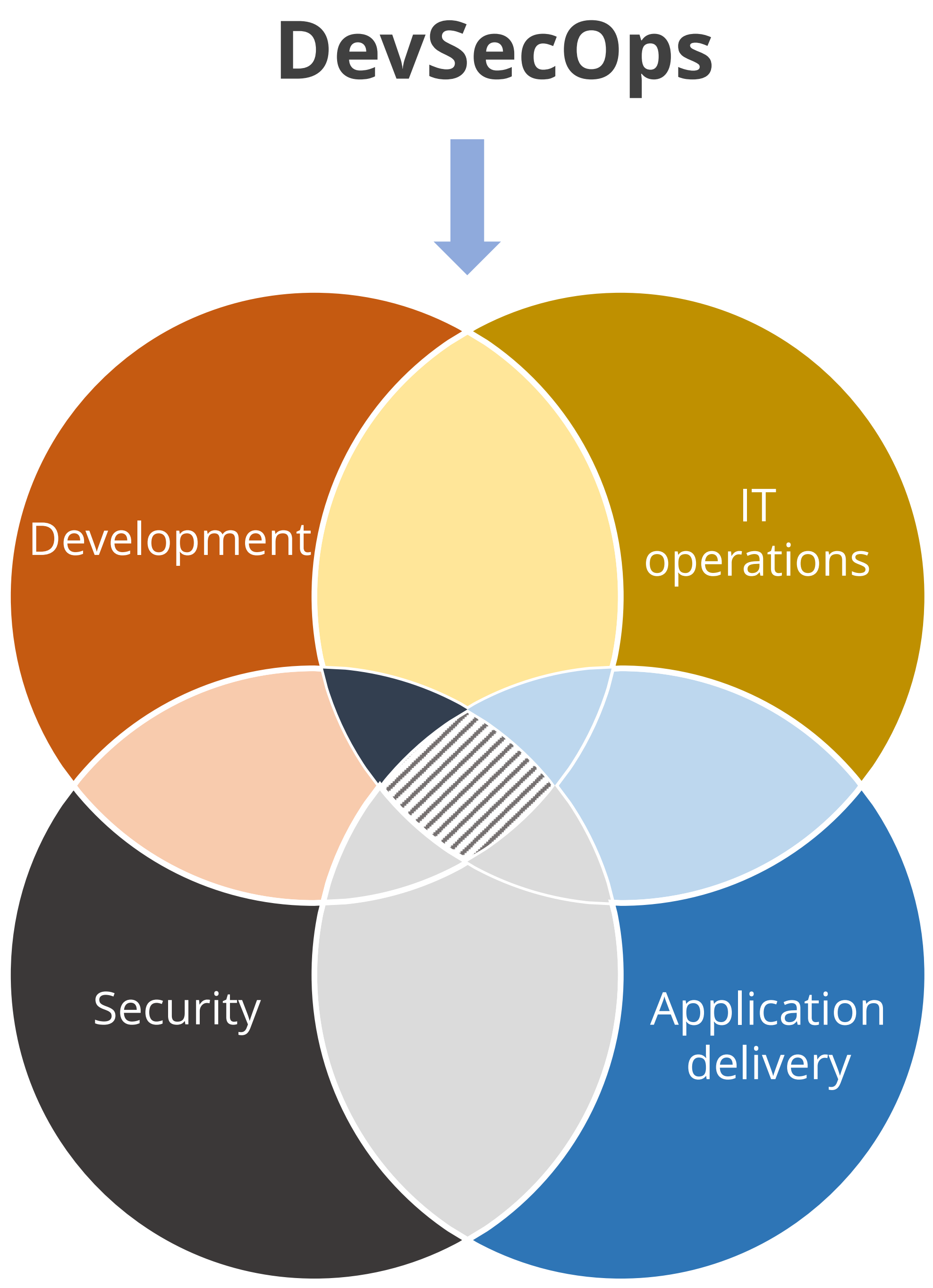
It is an extension of DevOps practice that is used to automate, monitor, and apply security at all phases of the software and DevOps lifecycle.



It is an approach that combines application development, security, operations and Infrastructure as Code (IaC) in an automated continuous integration and continuous delivery (CI/CD) pipeline.

DevOps vs. DevSecOps





DevOps Vs. DevSecOps

Aspect	DevOps	DevSecOps
Focus	Integrates development and operations	Extends DevOps with security practices
Key components	CI/CD pipelines and automation tools	CI/CD pipelines and security tools
Goals	Faster delivery and collaboration	Secure and efficient software delivery
Approach	Emphasizes automation and collaboration	Integrates security throughout SDLC
Security	Limited focus on security aspects	Security is integrated from the beginning

Six Pillars of DevSecOps

Collective responsibility

Implement shared responsibility to foster cohesive cloud security guided by the cloud security team

Collaboration and integration

Promote collaboration to build a security-focused culture, enhancing teamwork and knowledge sharing in DevSecOps

Logical implementation

Utilize a versatile security model to ensure safe application development and data integrity

Six Pillars of DevSecOps

Bridging compliance and development

Address the compliance-development gap to identify necessary controls and implement required software measures effectively

Automation

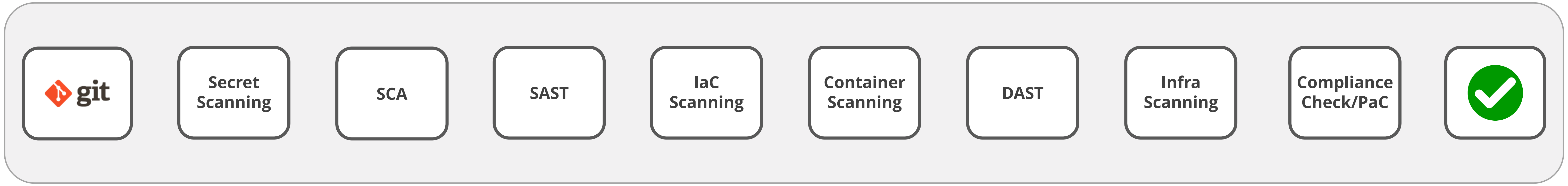
Automate security in DevSecOps to boost efficiency, reduce workloads, ensure quality checks, and cut labor costs

Measure, manage, and execute

Manage security tasks skillfully during both development and post-delivery phases

OWASP DevSecOps Guidelines

It provides a roadmap for incorporating security measures throughout the DevOps process, promoting a culture of security and collaboration.



Roadmap for implementing a basic pipeline

OWASP DevSecOps Guidelines

SAST (Static Application Security Test)

Identify vulnerabilities in source code through static analysis without executing the program

SCA (Software Composition Analysis)

Manage and assess open-source components for security risks and compliance issues

IAST (Interactive Application Security Testing)

Test real-time applications during runtime to detect vulnerabilities

DAST (Dynamic Application Security Test)

Assess web applications in an active state to uncover vulnerabilities and security weaknesses

OWASP DevSecOps Guidelines

IaC scanning

Examine Infrastructure as Code (IaC) files like Terraform or Helm Charts to detect configuration errors and security gaps

Infrastructure scanning

Evaluate the security status of infrastructure components like servers, networks, and databases for potential vulnerabilities

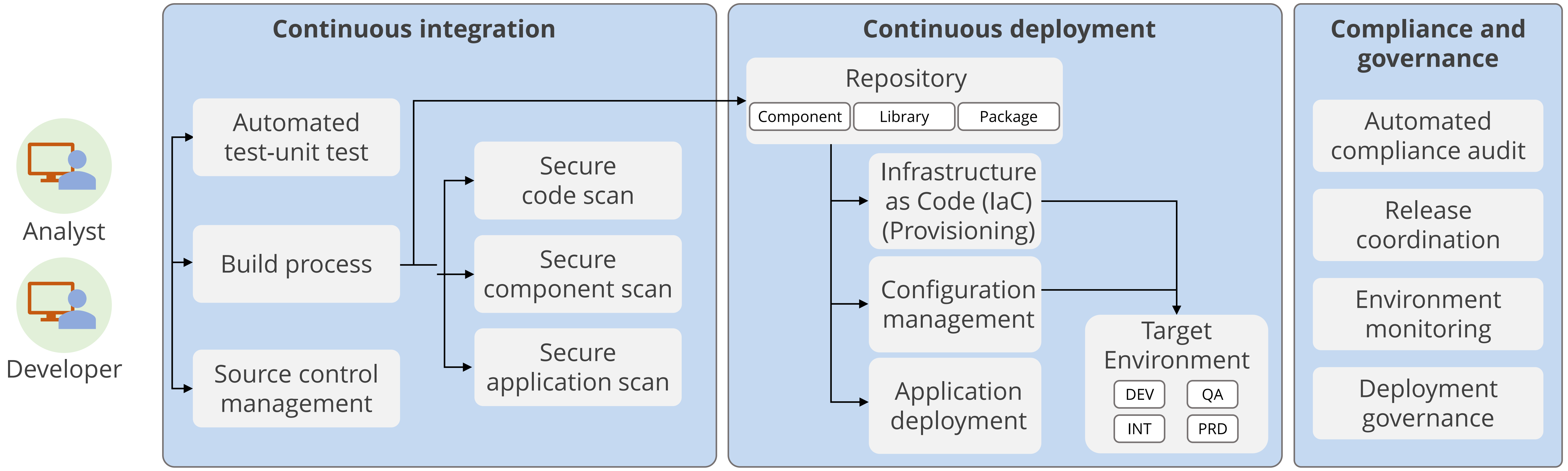
Compliance check

Ensure systems and applications adhere to regulatory and industry-specific security standards and guidelines

DevSecOps Architecture

It is the design and implementation of integrated security practices within the software development and deployment pipeline.

Below is the architecture of DevSecOps based on security aspects:



DevSecOps Architecture

Security aspects in all the three phases of DevSecOps architecture:

Continuous integration (CI)

Automates the process of integrating code changes from multiple developers, fostering early detection of security vulnerabilities through integration testing

Continuous deployment (CD)

Streamlines the deployment process, enabling frequent release of secure code by incorporating security checks throughout the deployment pipeline

Continuous compliance (CC)

Ensures ongoing adherence to security regulations, by automating security scans and enforcing compliance policies throughout the development lifecycle

DevSecOps Architecture

Security aspects in all the three phases of DevSecOps architecture:

Continuous integration (CI)

- Secure code reviews
- Secure dependency management
- Secret scanning
- Access control

Continuous deployment (CD)

- Immutable infrastructure
- Least privilege
- Security testing

Continuous compliance (CC)

- Policy as code
- Configuration management
- Vulnerability scanning
- Compliance reporting

Best Practices in DevSecOps

Shift left

Implement security practices early in development process to enhance security posture from the outset

Foster collaboration

Promote shared responsibility and collaboration for effective outcomes across teams

Adopt automation

Automate CI/CD with integrated security for rapid and consistent software delivery

Best Practices in DevSecOps

Prioritize risk management

Emphasize the importance of managing risk by implementing essential security controls to mitigate potential threats effectively

Enforce access controls

Apply role-based access controls for comprehensive security during development

DevSecOps Tools

DevSecOps tools assist in early identification of security vulnerabilities during development.

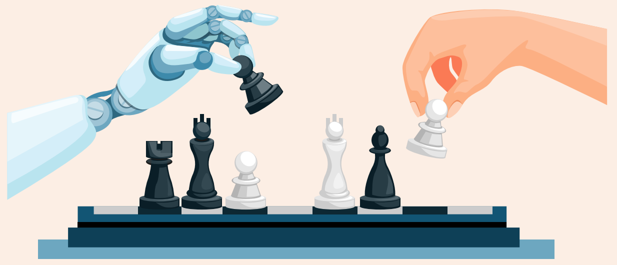
Snyk is a developer-focused security platform that scans code and dependencies for vulnerabilities, providing actionable insights and recommendations for secure coding practices and container security.

OWASP ZAP (Zed Attack Proxy) is a user-friendly, open-source DAST tool for web applications, offering manual and automated scanning to detect vulnerabilities like SQL injection and XSS.

SonarQube is an open-source tool for continuous code quality and security inspection, seamlessly integrated into CI/CD pipelines, supporting multiple languages with static code analysis and vulnerability detection.

Case Study

Secure software delivery: DevSecOps implementation at PayPal



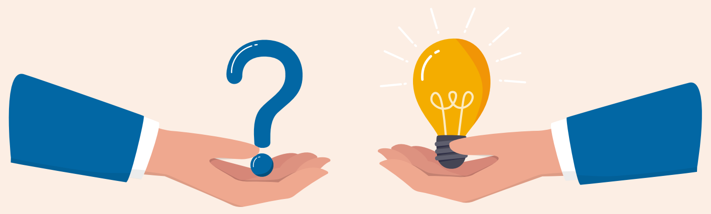
Challenges

- PayPal faced challenges in ensuring secure software delivery due to increasing cyber threats and compliance requirements.
- Traditional methods were inadequate to handle the fast-evolving vulnerabilities in their software.
- The challenge is implementing an integrated and automated security approach across the development cycle to address coordination issues in app development with multiple team members.

PayPal

Case Study

Secure software delivery: DevSecOps implementation at PayPal



Solution

- PayPal adopted DevSecOps, integrating security into the software delivery pipeline with automation tools for testing and compliance.
- They enforced secure coding guidelines and promoted better team collaboration to share security responsibilities.
- This streamlined security practices and enhanced software resilience by making security an integral part of the development process.

Case Study

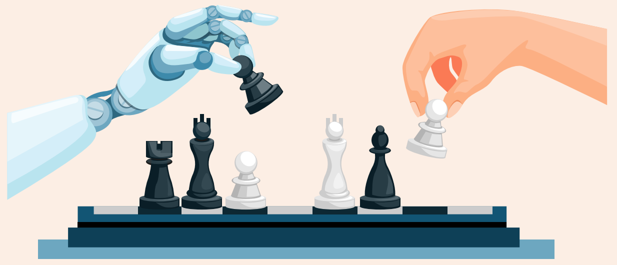
Secure software delivery: DevSecOps implementation at PayPal

Outcome

- Enhanced security posture through early vulnerability detection
- Accelerated software delivery with automated testing
- Reduced security risks (due to adherence to regulatory compliance)
- Decreased security incidents and vulnerabilities

Case Study

Securing digital operations: DevSecOps implementation at Allianz



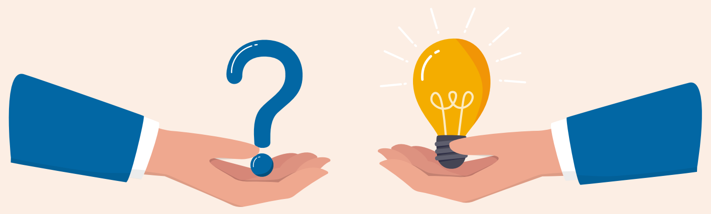
Challenges

- Allianz faced cybersecurity challenges endangering customer data and financial systems.
- Traditional security methods were insufficient against evolving threats.
- Lack of teamwork between development and operations teams caused delays, urging a more integrated approach.

Allianz

Case Study

Securing digital operations: DevSecOps implementation at Allianz



Solution

- Allianz adopted DevSecOps to integrate security into development and operations, fostering teamwork and automating security checks.
- They implemented automated security checks throughout the software development stages.
- Employees were trained in secure coding practices as part of their DevSecOps initiatives.

Case Study

Securing digital operations: DevSecOps implementation at Allianz

Outcome

- Enhanced security posture through early vulnerability detection with DevSecOps
- Automated feature deployment to ensure security and compliance standards
- Improved collaboration for better operational efficiency
- Continuous improvement through feedback loops and metrics

Quick Check

As a DevOps lead in the team, how would you define the **shifting left** approach in DevSecOps and its significance in ensuring early security considerations within the development processes?

A. Delaying security checks until deployment

B. Including security practices early in the development process

C. Focusing only on operational security

D. Ignoring security concerns

Key Takeaways

- The aim of DevOps is to shrink the development cycle and ensure continuous delivery of high-quality software.
- Agile is an iterative development process that promotes continuous iteration of development.
- The lifecycle of DevOps streamlines software development in seven phases, starting with planning and progressing to speed up the process.
- The main objective of DevSecOps is to automate, monitor, and apply security at all phases of the software lifecycle.
- DevSecOps tools assist in early identification of security vulnerabilities during development.

Thank You