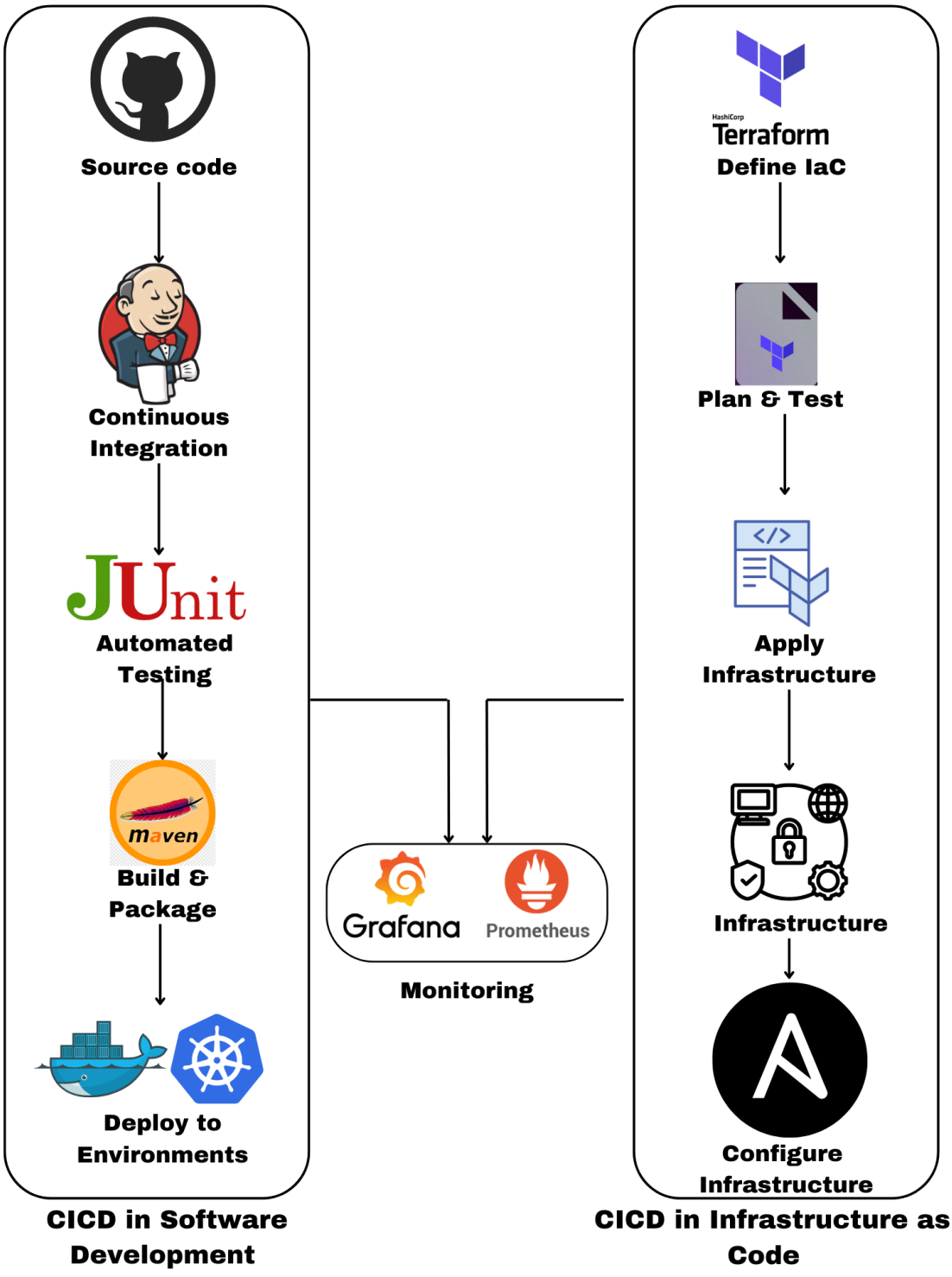
Devops Shack

# CICD in Software Development and CI/CD in IAC



## Introduction

In the software development domain, implementing robust and efficient processes helps meet the continuously evolving demands of the industry. Essentially, the rapid deployment of both infrastructure and software is crucial to maintaining a competitive edge in the market. Adopting methodologies like Infrastructure as Code (IaC) and Continuous Integration/Continuous Deployment (CI/CD) are crucial for swift and consistent software delivery. These approaches transform how software and its infrastructure are built, tested, and deployed.

Table of Contents



### CI/CD Meaning: An Overview

CI/CD are software development practices that help improve the efficiency, reliability, and speed of the development and deployment processes. They involve automated pipelines for software deployment and testing meant to improve the product's time to market.

### What Is Continuous Integration (CI)?

Continuous Integration involves integrating code changes regularly from multiple contributors into a shared repository.

Developers submit their code changes to a Version Control System (VCS) like Git, where the changes are merged with the deployed product. Automated

build and testing processes are triggered during the merge process to ensure the new code integrates well with the existing codebase and does not introduce errors.

As such, CI helps detect integration issues early, reduce bugs, and create a consistent and stable codebase.

**What Is Continuous Delivery (CD)?**

Continuous Delivery involves automating software delivery at various stages of the development or production environment.

Once code changes pass the continuous integration phase, automated processes take over to deploy the application to different environments, including testing, staging, and production.

Consequently, CD facilitates faster and more reliable software delivery, reduced manual intervention, and quick bug fixes, as well as feature updates.

**Benefits of CI/CD for Businesses**

CI/CD is a key component of the DevOps methodology, also known as the "shift left" culture that enables collaboration and communication between the development and operations teams to create a more efficient and streamlined software delivery process. Some of the key benefits of implementing CI/CD for companies include:

- **Improved metrics:** CI/CD enhances Software Development metrics by automating the tracking of key performance indicators (KPIs). This includes insights into build success rates, test coverage, and deployment frequency.
- **Better deployment:** CI/CD implementation ensures smoother and more reliable release processes by automating key stages, such as integration, testing, and deployment. This helps businesses reduce the likelihood of deployment failures, improving the overall reliability of software releases.
- **Reduced errors and downtime:** Automation within the CI/CD pipeline significantly decreases the chances of human error during development and deployment. With automated testing, you can detect issues early in the development cycle, which consequently, reduces bugs and vulnerabilities in production.

- **Increased agility:** CI/CD establishes a more agile development environment, enabling businesses to respond quickly to changing market demands.
- **Enhanced security:** Automated security scans and checks integrated into the CI/CD pipeline help identify and address vulnerabilities early in development.
- **Improved time to market:** CI/CD implementation speeds up the production process using automated pipelines for testing and deployment to various environments. This results in faster bug fixes, updates, and a satisfied user base.

**CI/CD and Software/Application Development**

CI/CD helps streamline the development lifecycle and enhances software efficiency. CI/CD pipelines use detailed processes, standards, tools, and automation to rapidly integrate continuous phases—source, build, test, and deploy — with precision.

Moreover, they are tailored to the specific needs of each project, ensuring efficiency and adaptability. Here's how the Software Development Lifecycle (SDLC) benefits from the automated pipelines.

**1. Source**

The first step in a CI/CD pipeline is creating source code, with developers using language-specific tools like IDEs for code-checking features. This phase relies on code repositories and VCS like Git.

**2. Build**

The build process extracts source code, links components, and compiles them into an executable file. Tools generate logs, identify errors, and notify developers. Build tools that are language-dependent and can be integrated into integrated development environments (IDEs), supporting both source creation and building in a CI/CD pipeline.

**3. Test**

After completing static testing in the source code phase, the built code progresses to the dynamic testing phase. This involves functional and unit testing for new features, regression testing to avoid breaking existing

functionalities, and additional integration, user acceptance, and performance tests.

**4. Deploy**

After successfully passing testing, the build becomes a candidate for deployment. Continuous delivery requires approval from human stakeholders prior to deployment, while continuous deployment automatically deploys post-test suite clearance.

**CI/CD and Infrastructure as Code**

IaC involves managing, configuring, and provisioning infrastructure using code. Organizations present their infrastructure in code, enabling version control and automated deployment through tools like Terraform and Ansible.

In IaC, CI/CD helps optimize the management and provisioning of infrastructure through automation. Additionally, CI/CD in IaC involves a streamlined process from defining the infrastructure as code to implementing a robust CI/CD pipeline.

Let's explore the steps to enable IaC and CI/CD;

**Step 1: Define Infrastructure as Code**

Define the desired infrastructure configuration using code, enabling automated provisioning and management.

You can use tools like Terraform, Pulumi, and Crossplane.

**Step 2: Create a CI/CD Pipeline**

The second step is to build a CI/CD pipeline that will automate the integration, testing, and deployment of IaC changes.

You can use tools like GitLab CI/CD, Jenkins, or Azure DevOps.

**Step 3: Integrate IaC and CI/CD**

Integrate IaC practices with CI/CD processes to ensure a unified and automated approach to infrastructure management.

This can be achieved by automatically deploying infrastructure changes within the CI/CD pipeline. This guarantees infrastructure alignment with the latest code modifications, maintaining consistency.

**Benefits of CI/CD in IaC**

Integrating CI/CD in IaC has several benefits, including cost reduction, enhanced agility, and fast time to market. Let's explore some of them below.

- **Cost reduction:** Efficient automation minimizes manual efforts, leading to cost savings in infrastructure management.
- **Better quality:** Automated testing ensures consistent and error-free infrastructure configurations, enhancing overall quality.
- **Enhanced agility:** Quick and automated deployment of infrastructure changes enables increased adaptability and responsiveness.
- **Synchronization between code and infrastructure**: Software and IaC changes can be part of the same CI/CD pipeline. Both changes can be deployed using the same pipeline, ensuring the product and its environment are never out of sync.
- **Fast-time to market:** Accelerated development and deployment cycles result in a faster time-to-market for applications and services.

**Comparing CI/CD in Software/Application Development vs CI/CD in IaC**

CI/CD in software development and Infrastructure as Code (IaC) share fundamental principles, enabling efficiency and collaboration.

Let's explore the common aspects below:

- **Automation**
    - **IaC:** CI/CD automates infrastructure provisioning and management through code.
    - **Software development:** CI/CD automates the building, testing, and deployment of software.
- **Version Control**
    - **IaC:** CI/CD utilizes version control systems (e.g., Git) to manage changes in infrastructure code.
    - **Software development:** Employs version control for source code management

- **Collaboration**
    - **IaC:** CI/CD encourages collaboration by allowing multiple team members to contribute to infrastructure code.
    - **Software development:** Promotes collaborative coding and joint development efforts.

The table below compares distinctive features of CI/CD in software/application development vs IaC.

| **Aspects** | **Software/Application Development** | **Infrastructure as Code (IaC)** |
|---|---|---|
| Focus | Software/application building, testing, and deployment | Infrastructure provisioning and management |
| Code Structure | Application source code | Defining infrastructure and configurations as code |
| Deployment Process | Involves automating the build, test, and deployment for systematic release of applications | Involves automating the provisioning and configuring of infrastructure through code |
| Lifecycle Management | Software Development Lifecycle (SDLC) | Focuses on maintaining and evolving infrastructure |
| Final Output | Deployed and operational software applications | Configured and provisioned infrastructure |
| Tools | Jenkins, Travis CI, etc. | Terraform, Pulumi, Crossplane, etc. |

## Conclusion

Implementing CI/CD benefits both traditional software development and Infrastructure as Code (IaC). While CI/CD practices streamline software development lifecycles, enhancing efficiency and reliability, their application in IaC optimizes infrastructure management through automation and consistency.

Whether applied to software or infrastructure, the principles of CI/CD, such as automation, version control, and collaboration, contribute to the overall success and competitiveness of modern development practices.