What is CICD?

CI/CD, standing for Continuous Integration/Continuous Delivery (or Deployment), is a software development practice that automates the process of building, testing, and deploying code.

It's a core part of DevOps and aims to deliver software more frequently and reliably. Essentially, CI/CD pipelines streamline the software delivery process, enabling teams to respond to changes and user feedback more quickly.

* **Continuous Integration (CI):**

Developers frequently merge code changes into a shared repository, and automated tests are run to verify the reliability of the code. This helps catch issues early and reduces the risk of integration problems.

* **Continuous Delivery/Deployment (CD):**

Once code changes are integrated and tested, they are automatically prepared for release (Continuous Delivery) or deployed to production (Continuous Deployment). Continuous Deployment happens without manual intervention, while Continuous Delivery might require manual approval before release to production.

**Benefits of CI/CD:**

* **Faster time-to-market:** Automated processes enable quicker delivery of new features and updates.
* **Improved software quality:** Automated testing helps identify issues early, leading to better quality code.
* **Reduced risk:** CI/CD helps reduce the risk of integration problems and major failures.

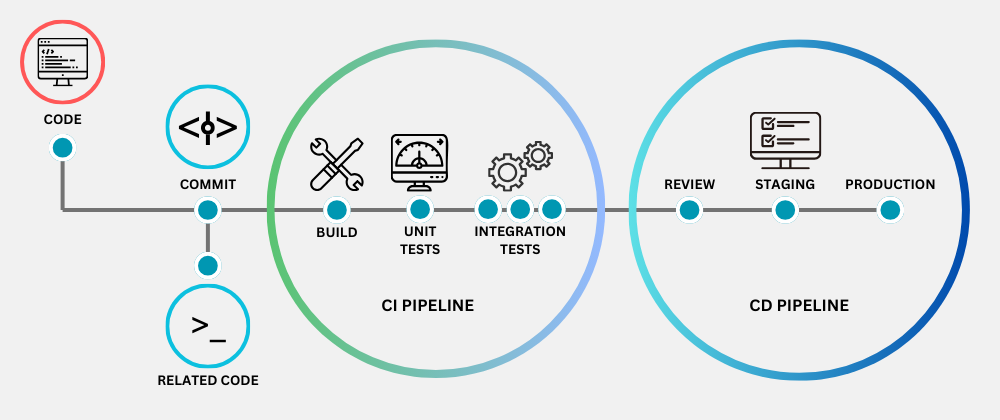
**Increased collaboration:** The CI/CD pipeline fosters collaboration between development and operations teams.

**Tools used for CI/CD.**

Several tools can be used to implement CI/CD pipelines, including:

* **Version control systems:** Git
* **Build and test automation tools:** Jenkins
* **Deployment tools:** Docker

**Cloud-based CI/CD platforms:** AWS Code Pipeline.



**What is Jenkins?**

Jenkins is an open-source automation server used for automating tasks related to building, testing, and deploying software. It's a popular tool for implementing continuous integration and continuous delivery (CI/CD) workflows, helping development teams automate parts of the software development lifecycle. Jenkins is known for its flexibility and extensive plugin ecosystem, allowing it to be adapted to various development workflows.

**Key components and concepts**

Jobs

Pipelines

Agents

Plugins

**Jenkins Jobs**

* **Freestyle project** - Lets you run custom commands and scripts.
* **Pipeline** - For stacking tasks together into an automated workflow.
* **Multibranch Pipeline** - When you have code in multiple branches, and want to build from each.

Java Installation (21 or 17 Versions only Compatiable to Jenkins)

https://www.openlogic.com/openjdk-downloads

Jenkins Installation

<https://www.jenkins.io/download/thank-you-downloading-windows-installer/>

While uninstallation you have to do…

1.uninstall

2.Remove Jenkins folder from Program Files

3.Remove Jenkins folder from Program Data.

For initial Admin password

1.C:\programData\Jenkins\Jenkins\Secrets\Initialadminpassword

Here you can find you password(Open with Notepad)

Paste there.

Select suggested plugins

**Types of Pipelines**

1.Declarative pipeline

Scripted pipeline

**Scripted Pipeline**

Scripted Pipeline is the original pipeline syntax for Jenkins, and it is based on Groovy scripting language. In Scripted Pipeline, the entire workflow is defined in a single file called a Jenkinsfile. The Jenkinsfile is written in Groovy and is executed by the Jenkins Pipeline plugin. Scripted Pipeline provides a lot of flexibility and control over the workflow, but it can be more complex and verbose than Declarative Pipeline.

Here is an example of a simple Scripted Pipeline:

node {  
 stage('Build') {  
 // Build the application  
  
 }  
 stage('Test') {  
 // Run the tests  
   
 }  
 stage('Deploy') {  
 // Deploy the application  
  
 }  
}

**Declarative Pipeline**

Declarative Pipeline is a more recent addition to Jenkins and provides a more structured and simpler syntax for defining pipelines. Declarative Pipeline is based on the Groovy programming language, but it uses a Groovy-based DSL (Domain-Specific Language) for pipeline configuration. The main benefit of Declarative Pipeline is its readability and ease of use, as it is designed to be more intuitive and less verbose than Scripted Pipeline.

Here is an example of a simple Declarative Pipeline:

pipeline {  
 agent any  
 stages {  
 stage('Build') {  
 steps {  
 // Build the application  
  
 }  
 }  
 stage('Test') {  
 steps {  
 // Run the tests  
 }  
 }  
 stage('Deploy') {  
 steps {  
 // Deploy the application  
 }  
 }  
 }  
}