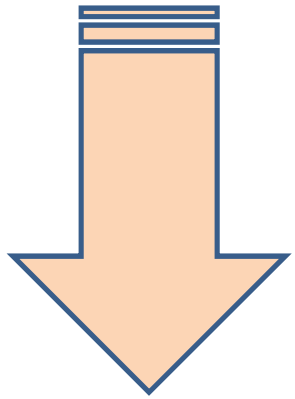
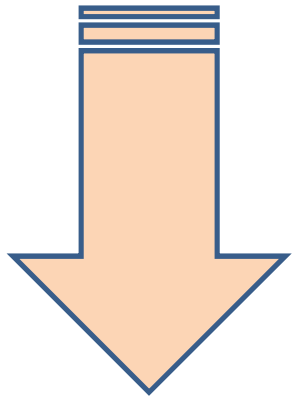
DEVOPS

DevOps



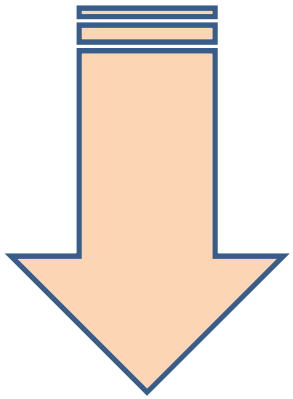
UNIT-2: Continuous Integtration using Jenkins

Introduction – Understanding Continuous Integration, Introduction about Jenkins, Build Cycle, Jenkins Architecture, Installation, Jenkins Management. Adding a Slave node to Jenkins, Building Delivery Pipeline, Pipeline as a Code.



Devops stages: Version Control, **continuous integration, continuous delivery, continuous deployment, continuous monitoring.**

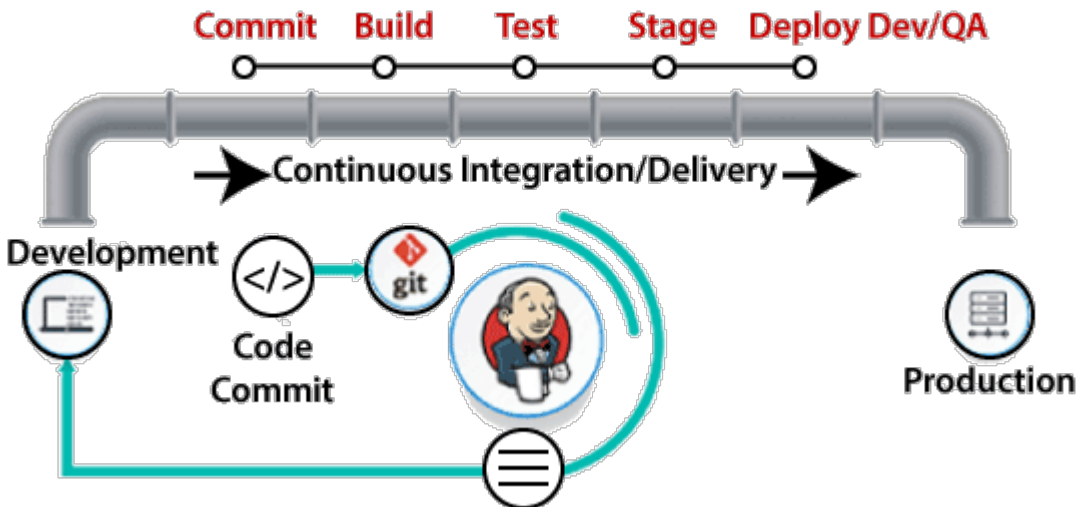
Continuous Integration in DevOps | How it is Performed with Advantages (educba.com)



DevOps – lifecycle

2. Continuous Integration

- This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes **unit testing, integration testing, code review**, and **packaging**.
- The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.
- **Jenkins** is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of **war or jar**. Then this build is forwarded to the test server or the production server.
- Continuous Integration (CI) applies to all types of software projects such as developing websites, Mobile Applications, and Micro services based APIs



DevOps – Continuous Integration

1. **Continuous integration:**

Definition:

- *Continuous integration refers to the build and unit testing stages of the software release process. Every revision that is committed triggers an automated build and test.*

  Continuous Integration in DevOps is the process of automating the build and deploy phase through certain tools and best practices.

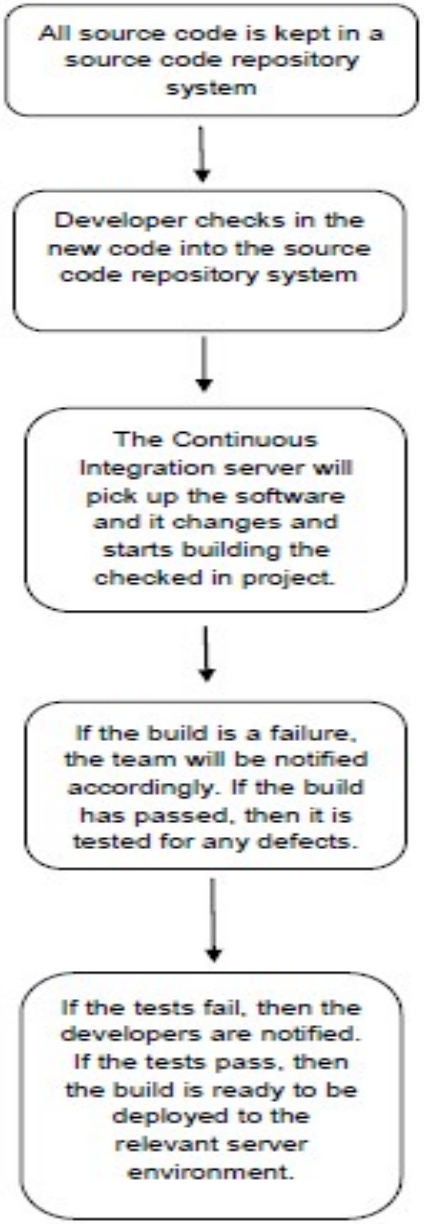
- Continuous integration has become a very integral part of any software development process. The continuous Integration process helps to answer the following questions for the software development team .

Addresses:

- Do all the software components work together as they should?
- Is the code too complex for integration purposes?
- Does the code adhere to the established coding standards?
- How much code is covered by automated tests?
- Were all the tests successful after the latest change?
- Continuous integration is a DevOps software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an **automation component** (e.g. a CI or build service) and a **cultural component** (e.g. learning to integrate frequently). The key goals of continuous integration are to **find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates**.

DevOps – Continuous Integration

continuous integration process working:



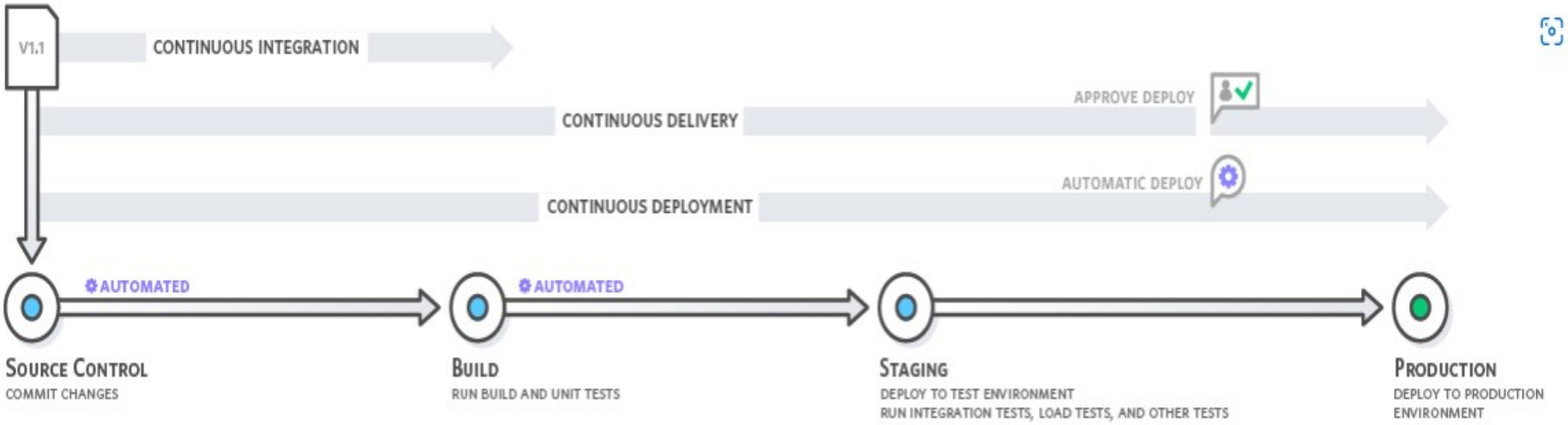
DevOps – Continuous Integration

continuous integration process working: Steps:

1. First, a **developer commits** the code to the version control repository. Meanwhile, the Continuous Integration server on the integration build machine polls source code repository for changes (e.g., every few minutes).
2. Soon after a commit occurs, the **Continuous Integration server detects that changes** have occurred in the version control repository, so the Continuous Integration server retrieves the latest copy of the code from the repository and then **executes a build script**, which integrates the software.
3. The Continuous Integration server **generates feedback** by e-mailing build results to the specified project members.
4. **Unit tests** are then carried out if the build of that project passes. If the tests are successful, the code is ready to be deployed to either the **staging or production server.**
5. The Continuous Integration server **continues to poll for changes** in the version control repository and the whole process repeats.

DevOps – Continuous Integration

- Continuous Integration in DevOps is the process of automating the build and deploy phase through certain tools and best practices.
- There are three major categories of tools associated with CI: Versioning tool, Build tool, and repositories for centralized artifacts. CI pipeline helps the developer to easily commit the code and helps for quality development.
-



DevOps – Continuous Integration

Advantages:

1. Improve Developer Productivity:

- Continuous integration helps your team be more productive by freeing developers from manual tasks and encouraging behaviors that help reduce the number of errors and bugs released to customers.

2. Find and Address Bugs Quicker

with more frequent testing, your team can discover and address bugs earlier before they grow into larger problems later.

. Deliver Updates Faster

- Continuous integration helps your team deliver updates to their customers faster and more frequently.

CI Tools: Jenkins, Bamboo, CircleCI, TeamCity

DevOps – Continuous Integration

Continuous Integration

List Of The Top Continuous Integration Tools

#1) Buddy

#2) Jenkins

#3) Buildbot

#4) ThoughtWorks

#5) UrbanCode

#6) Perforce Helix

#7) Bamboo

#8) TeamCity

#9) CircleCI

#10) Codeship

#11) CruiseControl

#12) Go/GoCD

#13) Travis

#14) Integrity

#15) Strider or Strider CD

DevOps – Tools

Jenkins: https://www.jenkins.io/download/

- Jenkins a DevOps tool for monitoring execution of repeated tasks. It is one of the best software deploy tools which helps to integrate project changes more easily by quickly finding issues.
- It supports **continuous integration and continuous delivery**
- Jenkins is an open source automation tool written in Java programming language **that allows continuous integration..**
- Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

Features:

- It increases the scale of automation
- Jenkins requires little maintenance and has built-in GUI tool for easy updates.
- It offers 400 plugins to support building and testing virtually any project.
- It is Java-based program ready to run with Operating systems like Windows, Mac OS X, and UNIX
- It supports continuous integration and continuous delivery
- It can easily set up and configured via web interface
- It can distribute tasks across multiple machines thereby increasing concurrency.

https://www.jenkins.io/download/

DevOps – Tools

Installing Jenkins:

Windows (jenkins.io)

Jenkins url after installation:

http://localhost:8080/jenkins

Jenkins

Jenkins is commonly used for the following:

- Continuous Integration for application and infrastructure code.
- Continuously deliver pipeline to deploy the application to different environments using Jenkins pipeline as code.
- Infrastructure component deployment and management.
- Run batch operations using Jenkins jobs.
- Run ad-hoc operations like backups, cleanups, remote script execution, event triggers, etc.

Jenkins

Jenkins is commonly used for the following:

- Continuous Integration for application and infrastructure code.
- Continuously deliver pipeline to deploy the application to different environments using Jenkins pipeline as code.
- Infrastructure component deployment and management.
- Run batch operations using Jenkins jobs.
- Run ad-hoc operations like backups, cleanups, remote script execution, event triggers, etc.
- Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.
- With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including **build, document, test, package, stage, deploy , static analysis** and much more.
- Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

Steps by Jenkins:

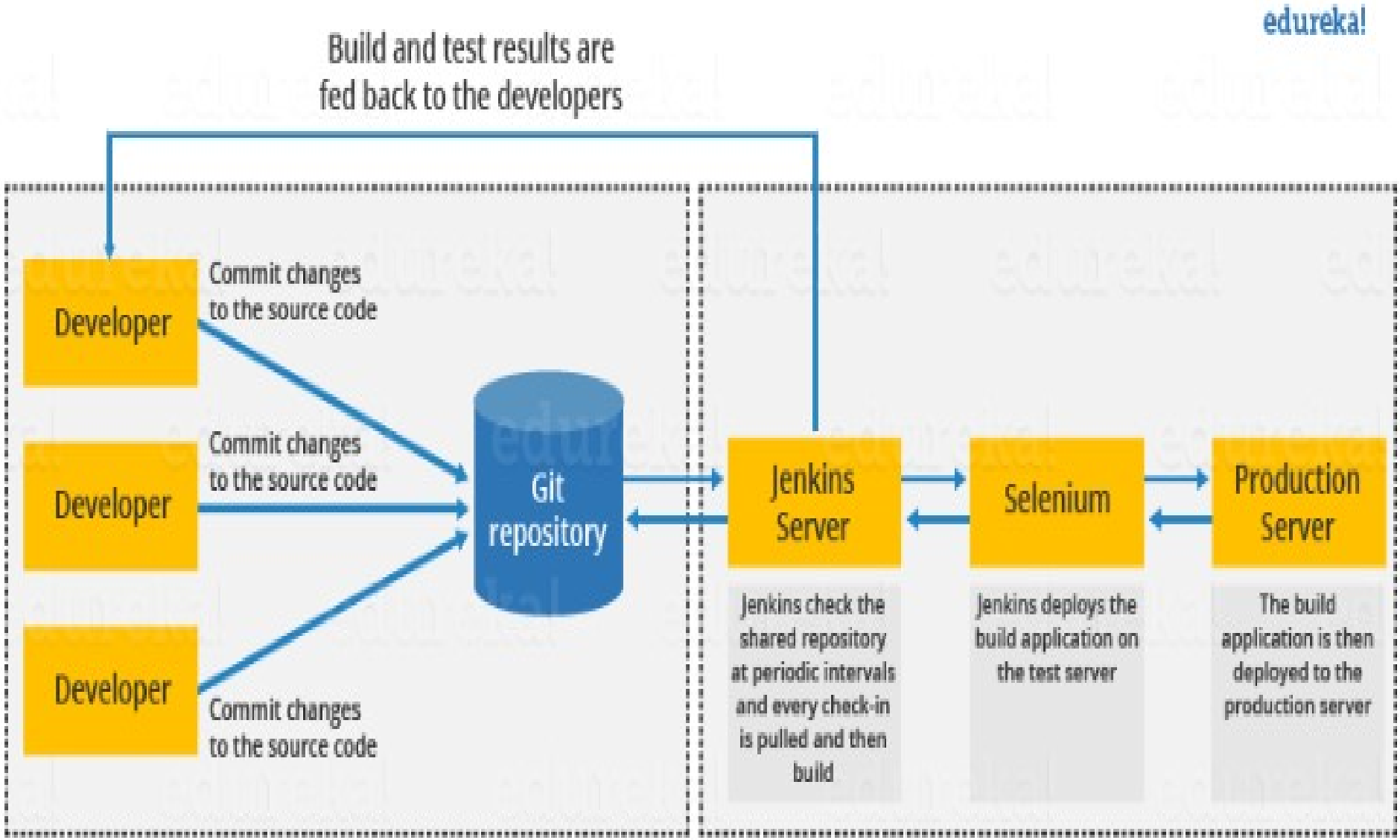
- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script, Archive a build result, Running software tests

How Jenkins works?

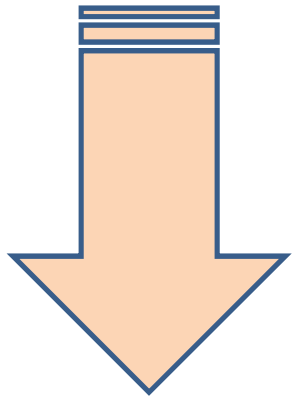
The below diagram is representing the following functions:

- First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
- Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
- If the build fails, then the concerned team will be notified.
- If built is successful, then Jenkins server deploys the built in the test server.
- After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
- It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



Jenkins Advantages & Disadvantages



Jenkins Advantages & Dis Advantages

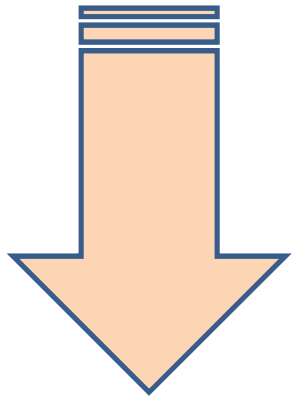
Advantages:

- It is an open source tool.
- It is free of cost.
- It does not require additional installations or components. Means it is easy to install.
- Easily configurable.
- It supports **1000 or more plugins** to ease your work. If a plugin does not exist, you can write the script for it and share with community.
- **It is built in java** and hence it is **portable**.
- It is **platform independent**. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- **Easy support,** since it open source and widely used.
- **Jenkins also supports cloud based architecture** so that we can deploy Jenkins in cloud based platforms.

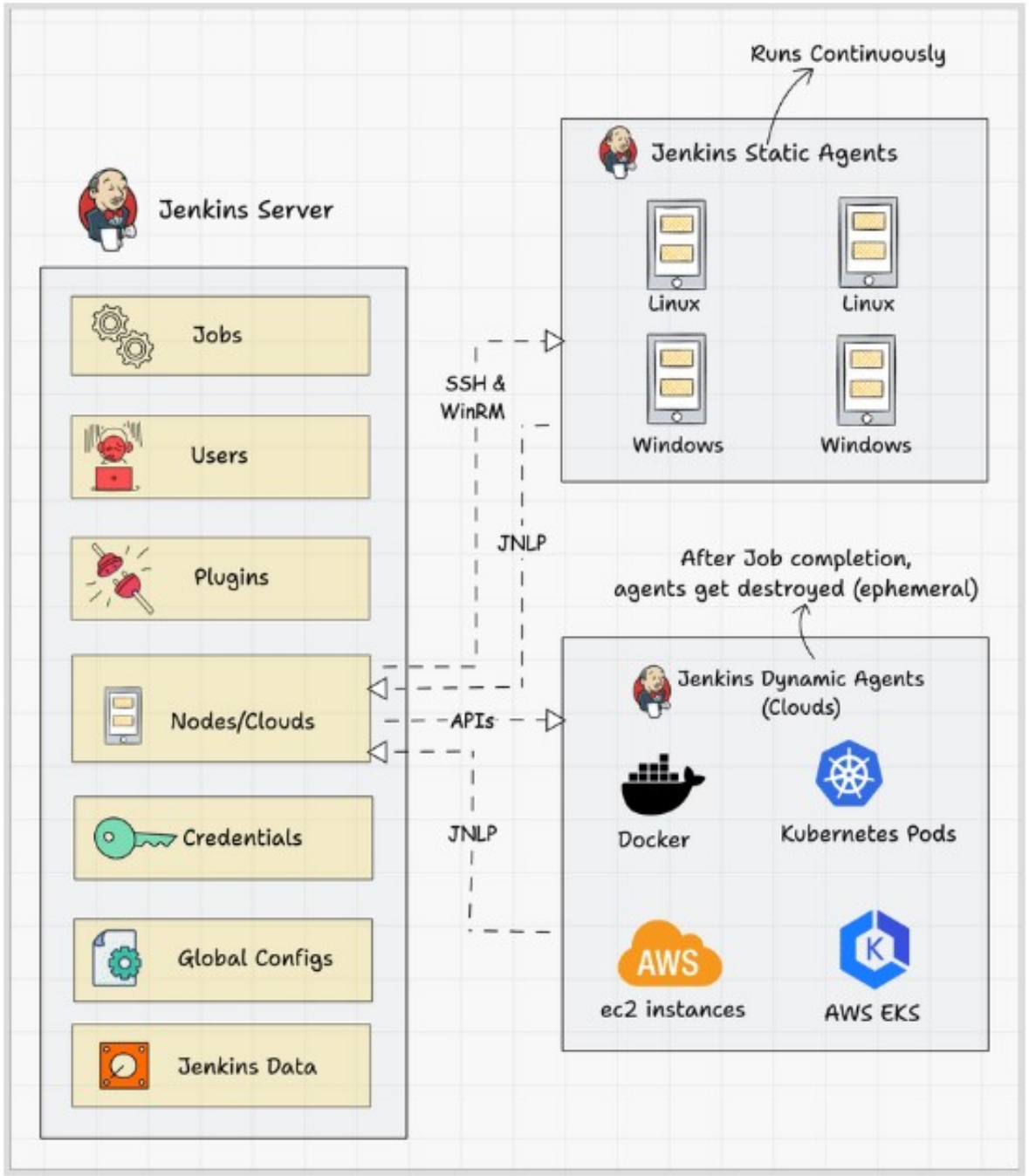
Dis Advantages:

- Its interface is out dated and not user friendly compared to current user interface trends.
- Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
- CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

Jenkins Architecture



Jenkins Architecture



Jenkins Architecture

Following are the key components in Jenkins:

- Jenkins Master Node
- Jenkins Agent Nodes/Clouds
- Jenkins Web Interface

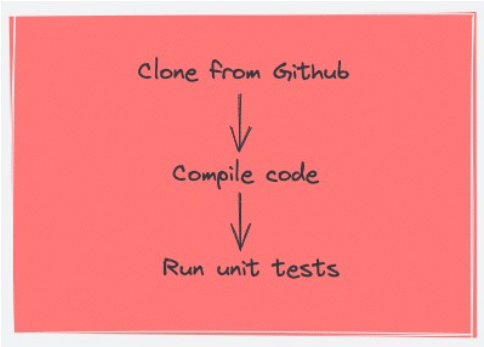
Jenkins Server (Formerly Master):

- Jenkins's server or master node holds all key configurations. Jenkins master server is like a control server that orchestrates all the workflow defined in the pipelines. For example, scheduling a job, monitoring the jobs, etc.

Jenkins Jobs:

- A job is a collection of steps that you can use to build your source code, test your code, run a shell script, run an Ansible role in a remote host or execute a terraform play, etc. We normally call it a Jenkins pipeline.

Jenkins Architecture



If you translate the above steps to a Jenkins pipeline job, it looks like the following.

```
        stage('Code Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/master']],
                    userRemoteConfigs: [[url: 'https://github.com/spring-
projects/spring-petclinic.git']]
                ])
            }
        }

        stage('Code Build') {
            steps {
                 sh 'mvn install -Dmaven.test.skip=true'
            }
        }
```

- There are multiple job types available to support your workflow for continuous integration & continuous delivery.

Jenkins Architecture

Jenkins Plugins:

- Plugins are official and community-developed modules that you can install on your Jenkins server. It helps you with more functionalities that are not natively available in Jenkins.
- For example, if you want to upload a file to s3 bucket from Jenkins, you can install an AWS Jenkins plugin and use the abstracted plugin functionalities to upload the file rather than writing your own logic in AWS CLI. The plugin takes care of error and exception handling.
- You can also download the plugin file and install it. You can also develop your custom plugins.

Jenkins Global Security:

- Jenkins has the following type of primary authentication methods.
- **Jenkins's own user database:-** Set of users maintained by Jenkins's own database. When we say database, its all flat config files (XML files).
- **LDAP Integration**:- Jenkins authentication using corporate LDAP configuration.
- **SAML Single Sign On(SSO)**: Support single signon using providers like Okta, AzureAD, Auth0 etc..
- With Jenkins matric-based security you can further assign roles to users on what permission they will have on Jenkins.
-

Jenkins Architecture

Jenkins Credentials:

- When you set up Jenkins pipelines, there are scenarios where it needs to connect to a cloud account, a server, a database, or an API endpoint using secrets.
- In Jenkins, you can save different types of secrets as a credential.
- Secret text
- Username & password
- SSH keys
- All credentials are encrypted (AES) by Jenkins. The secrets are stored in $JENKINS_HOME/secrets/ directory. It is very important to secure this directory and exclude it from Jenkins backups.

Jenkins Nodes/Clouds:

- You can configure multiple agent nodes (Linux/Windows) or clouds (docker, kubernetes) for executing Jenkins jobs. We will learn more about it in the agent section.

Jenkins Global Settings (Configure System):

- Under Jenkins global configuration, you have all the configurations of installed plugins and native Jenkins global configurations.
- Also, you can configure global environment variables under this section. For example, you can store the tools (Nexus, Sonarqube, etc) URLs as global environment variables and use them in the pipeline. This way it is easier to make URL changes that get reflected in all the Jenkins jobs.

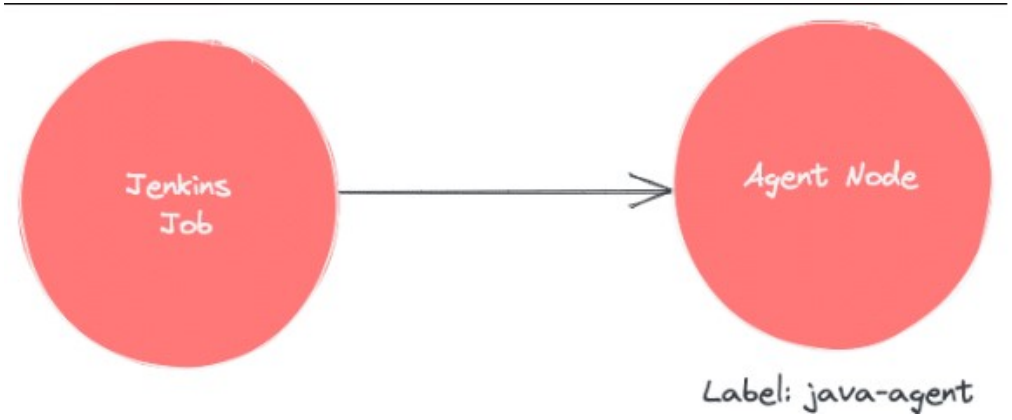
Jenkins Architecture

Jenkins Logs:

- Provides logging information on all Jenkins server actions including job logs, plugin logs, webhook logs, etc.
- Note: All the configurations for the above-mentioned components are present as a config file (XML file) in the Jenkins master nodes data directory.

Jenkins Agent:

- Jenkins agents are the worker nodes that actually execute all the steps mentioned in a Job. When you create a Jenkins job, you have to assign an agent to it. Every agent has a label as a unique identifier.
- When you trigger a Jenkins job from the master, the actual execution happens on the agent node that is configured in the job.



Jenkins Architecture

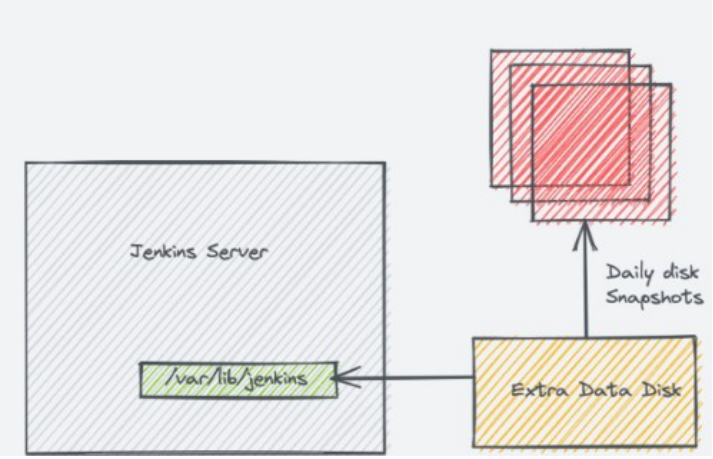
Note: You can run jobs in the Jenkins server without a Jenkins agent. In this case, master nodes acts as the agent. However, the recommended approach is to have a Jenkins master-agent setup for different job requirements so that you don't end up corrupting the Jenkins server for any system-wide configuration changes required for a job.

- You can have any number of Jenkins agents attached to a master with a combination of Windows, Linux servers, and even containers as build agents.
- Also, you can restrict jobs to run on specific agents, depending on the use case. For example, if you have an agent with java 8 configurations, you can assign this agent for jobs that require Java 8 environment.
- There is no single standard for using the agents. You can set up a workflow and strategy based on your project needs.

Jenkins Architecture

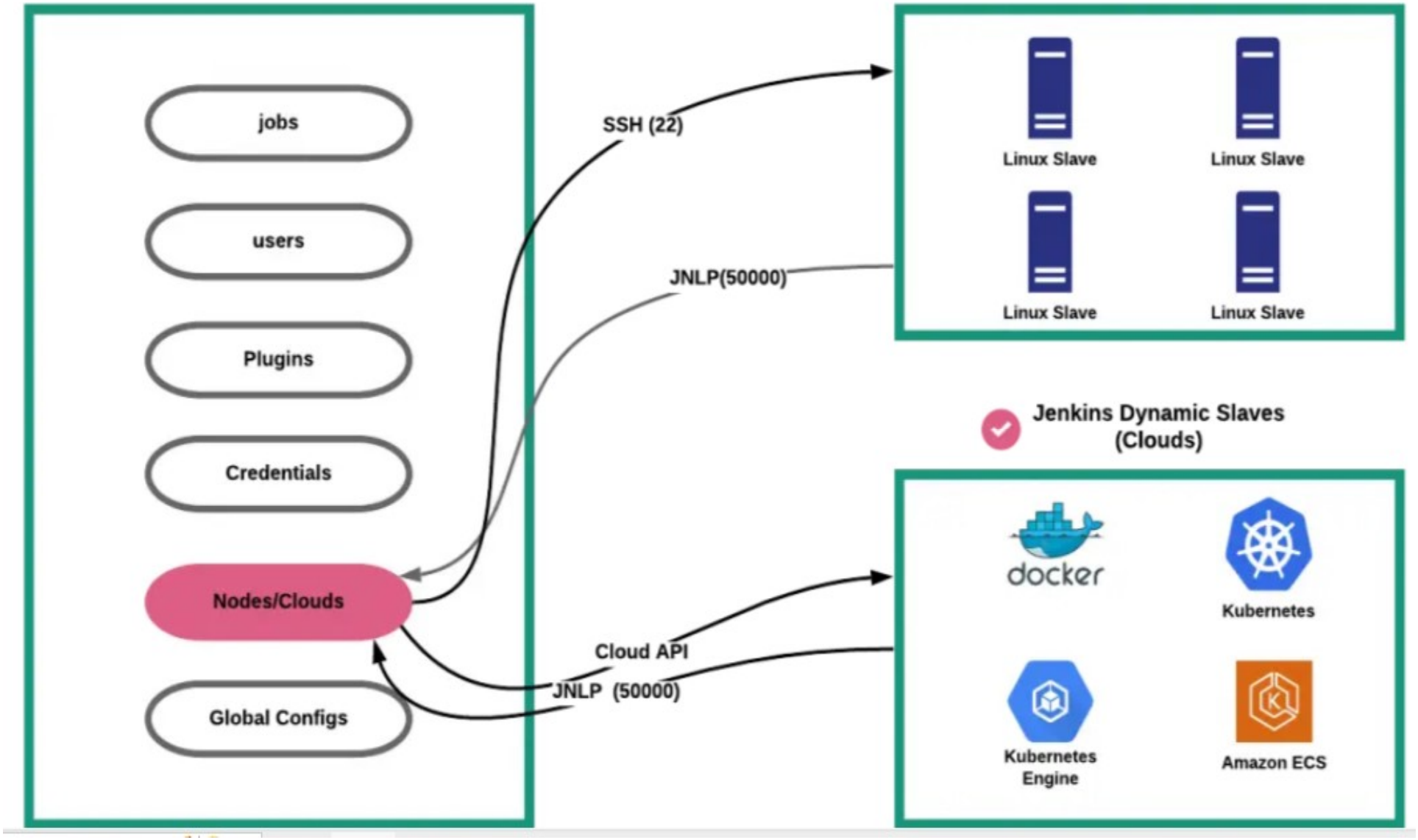
Jenkins Data:

- All the Jenkins data gets stored in the following folder location.
- /var/lib/jenkins/Data includes all jobs config files, plugins configs, secrets, node information, etc. It makes Jenkins migration very easy as compared to other tools.
- If you take a jook at /var/lib/jenkins/ you will find most of the configurations in xml format.
- It is essential to back up the Jenkins data folder every day. For some reason, if your Jenkins server data gets corrupt, you can restore the whole Jenkins with the data backup.
- Ideally, when deploying Jenkins in production, a dedicated **extra volume is attached** to the Jenkins servers that hold all the Jenkins data.



Jenkins Architecture

Jenkins Architecture

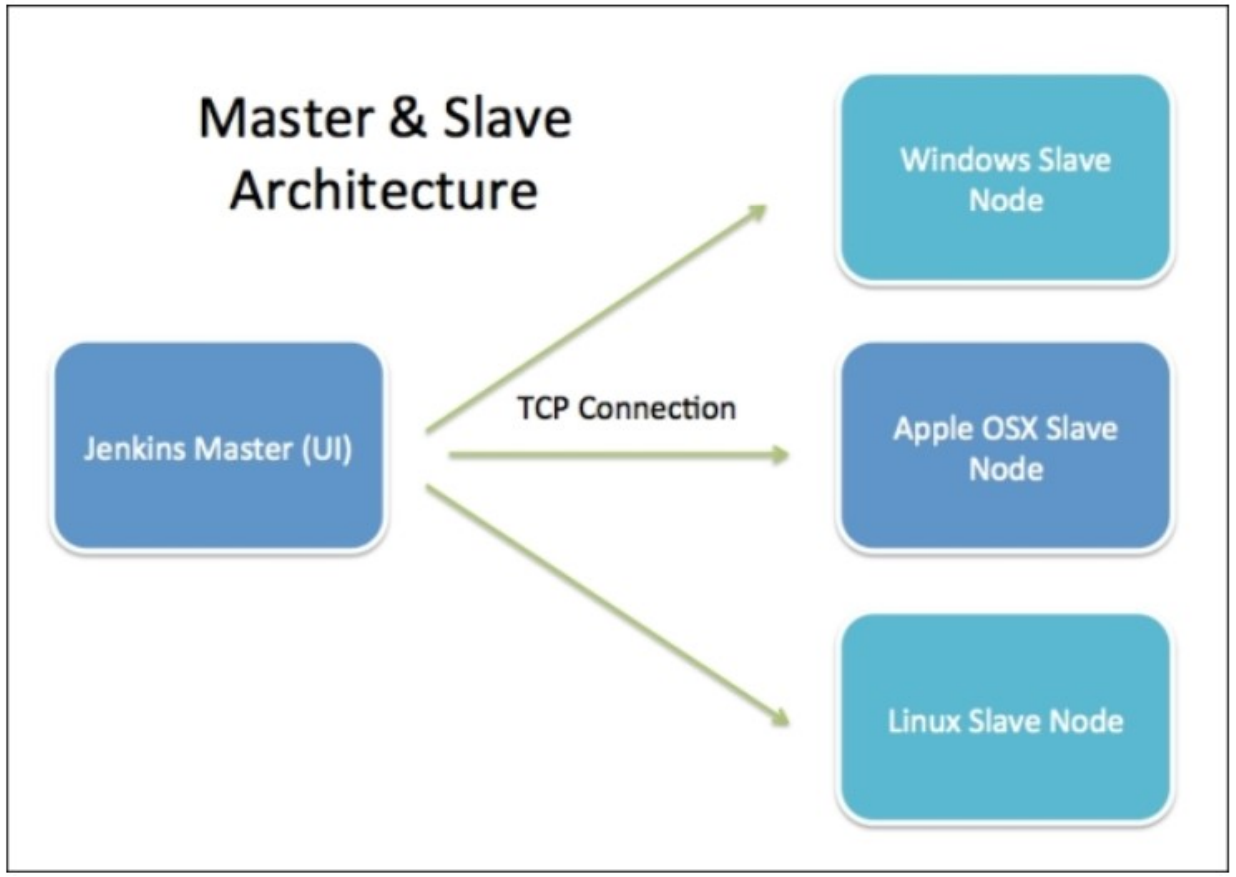


Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

- Jenkins Master/Server
- Jenkins Slave/Node/Build Server



Jenkins Architecture

Jenkins Master:

- The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can **configure the jobs/projects** but the build takes place in Nodes/Slave.
- By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

The server's job or master's job is to handle:

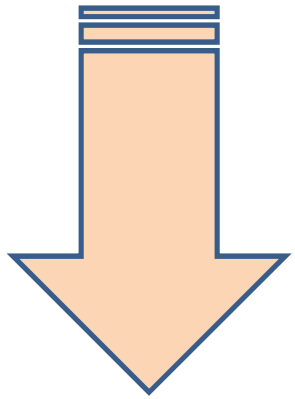
- Scheduling build jobs.
- Dispatching builds to the nodes/slaves for the actual execution.
- Monitor the nodes/slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master/Server instance of Jenkins can also execute build jobs directly.

Jenkins Architecture

Jenkins Slave

- Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.
- As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.

JAVA & TOMCAT SETUP FOR JENKINS



Java & Tomcat Setup for Jenkins

1. Install JDK 11/17/21 version in windows m/c
2. **set JAVA_HOME environment variable to** C:\Program Files\Java\jdk-21
3. Check javac, java versions and **%JAVA_HOME% value**

   javac –version

   java –version

   echo %JAVA_HOME%

4. Install Tomcat 10.1.13 version of tomcat

4.1 download Tomcat 10.1.13 zip from

https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.13/bin/apache-tomcat-10.1.13-windows-x64.zip

4.2 unzip to D:\

4.3 copy jenkins.war file to the location:

D:\apache-tomcat-10.1.13-windows-x64\apache-tomcat-10.1.13\webapps

4.4 change connector port number:

In D:\apache-tomcat-10.1.13-windows-x64\apache-tomcat-10.1.13\conf folder:

In server.xml → <connector port="8080" to <connector port="8181"

4.4 goto D:\apache-tomcat-10.1.13-windows-x64\apache-tomcat-10.1.13\bin

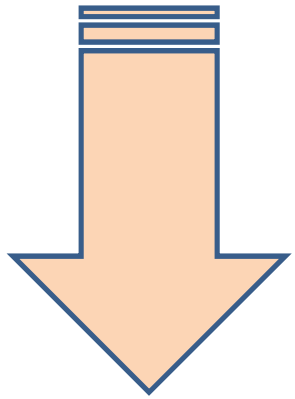
4.5 run startup.bat

D:\apache-tomcat-10.1.13-windows-x64\apache-tomcat-10.1.13\bin>startup.bat

5. Goto browser and give:

http://localhost:8080/jenkins to get Jenkins web UI

Jenkins – GitHub setup



Jenkins – GitHub setup

TO install git plugin:

Jenkins GitHub Setup – javatpoint

From Jenkins dashboard → go to Manage Jenkins → available plgins → search for ‘Git plugin’

And install.

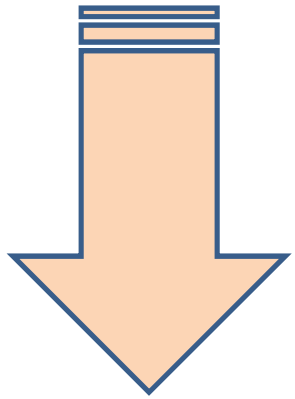
Once installed, restart Tomca tby:

http://localhost:8080/jenkins/restart

Integrating Jenkins with GitHub:

1. **Create a New Job in Jenkins**

Jenkins – Maven setup



Jenkins – Maven Integration

- Maven is build automation tool used basically for Java projects, though it can also be used to build and manage projects written in C#, Scala, Ruby, and other languages.
- Maven addresses two aspects of building software: 1st it describes how software is build and 2nd it describes its dependencies.

Maven Setup:

1. **Download from: https://maven.apache.org/downloa.cgi**

https://dlcdn.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.zip

2. set the **JAVA_HOME** and **MAVEN_HOME** environment variable in your system.

JAVA_HOME= C:\Program Files\Java\jdk-21

MAVEN_HOME= D:\apache-maven-3.9.5-bin\apache-maven-3.9.5

Check from cmd: java –version, javac –version, ,maven -version

3. Go to Maven plugin in Manage Jenkins → Manage Plugin → Available, search for 'maven integration' plugin.

4. Create a new maven job in Jenkins

i. Dashboard → New Item → <enter itemname>,

Jenkins – Creating a Maven Job

Maven Job Creation:

There are multiple ways to build a Maven project with Jenkins:

1. Use a free-style project with a Maven build step
2. Use a Maven-style project
3. Use a Pipeline project with a shell, batch, or powershell build step that calls Maven

Steps:

1. **Install a 'Maven Integration' plug-in in Jenkins.**

   Manage Jenkins → Plugins → 'Available Plug-ins' → search 'Maven Integration' → click 'Install'

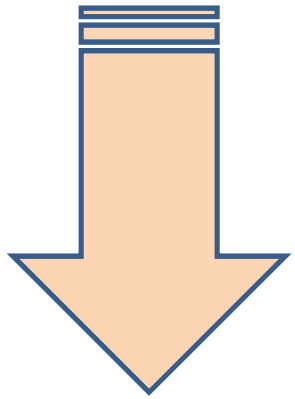
2.

Jenkins – Creating a Maven Job

Features of Maven job type

- ✓ The Jenkins project recommends Pipeline jobs and freestyle jobs for Maven projects. See the earlier section to understand the risks associated with using the Maven job type instead of Pipeline or freestyle jobs.
- ✓ This plugin provides a more advanced integration with additional features like:
- ✓ Automatic configuration of reporting plugins (Junit, Findbugs, ...)
- ✓ Automatic triggering across jobs based on SNAPSHOTs published/consumed
- ✓ Incremental build - only build changed modules
- ✓ Build modules in parallel on multiple executors/nodes
- ✓ Post build deployment of binaries only if the project succeeded and all tests passed

CI/CD pipeline



DevOps – CI/CD Pipeline

- **What is CI/CD?**

(Continuous Integration / Continuous Delivery //Continuous Deployment):

- **Continuous integration** is a software development method where members of the team can integrate their work **at least once a day**. In this method, every integration is checked by an automated build to search the error.
- **Continuous delivery** is a software engineering method in which a team develops software products in a short cycle. It ensures that software can be easily released at any time. (manual)
- **Continuous deployment** (automated) is a software engineering process in which product functionalities are delivered using **automatic deployment**. It helps testers to validate whether the codebase changes are correct, and it is stable or not.

DevOps – CI/CD Pipeline

- https://www.guru99.com/ci-cd-pipeline.html
- https://www.edureka.co/blog/ci-cd-pipeline/

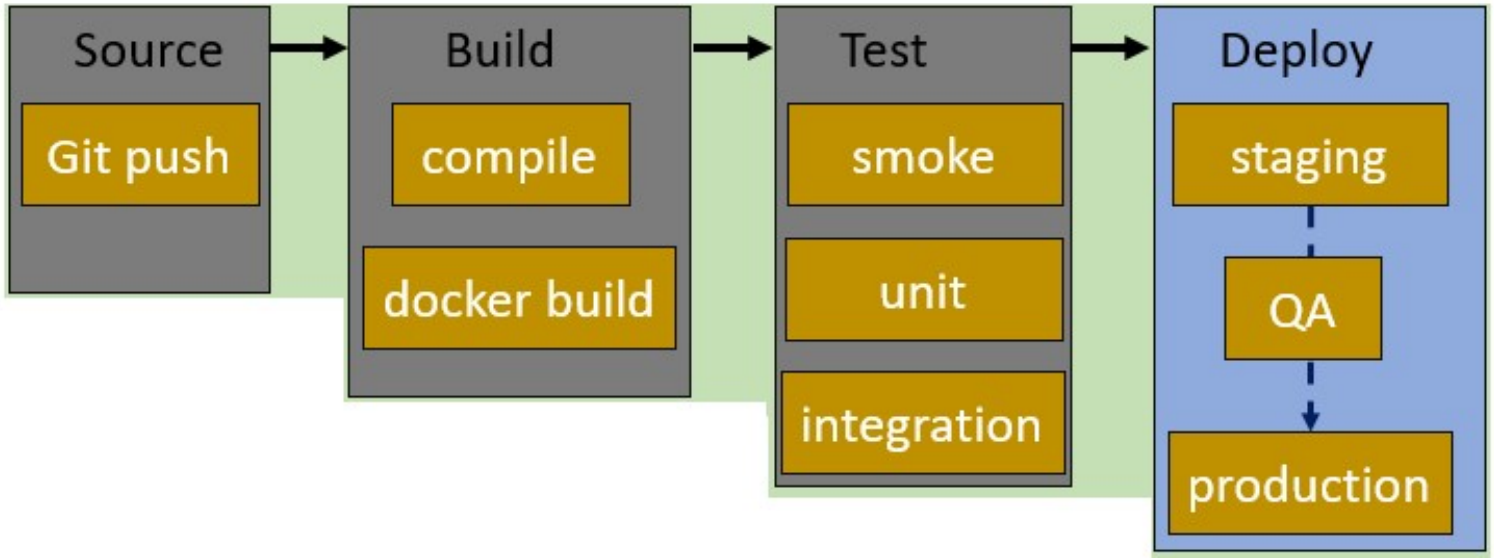
CI/CD Pipeline:

- A CI/CD pipeline automates the process of software integration & delivery.
- It **builds code, runs tests, and helps you to safely deploy** a new version of the software.
- CI/CD pipeline **reduces manual errors, provides feedback to de**velopers, and allows fast product iterations.
- CI/CD pipeline **introduces automation and continuous monitoring** throughout the lifecycle of a software product.
- It involves from **the integration and testing phase to delivery and deployment**. These connected practices are referred as **CI/CD pipeline.**

DevOps – CI/CD Pipeline - stages

- A CI/CD pipeline is a runnable specification of the steps that any developer should perform to deliver a new version of any software. Failure in each and every stage triggers a notification via email, Slack, or other communication platforms. It enables responsible developers to know about the important issues.

Stages of CI/CD pipeline:



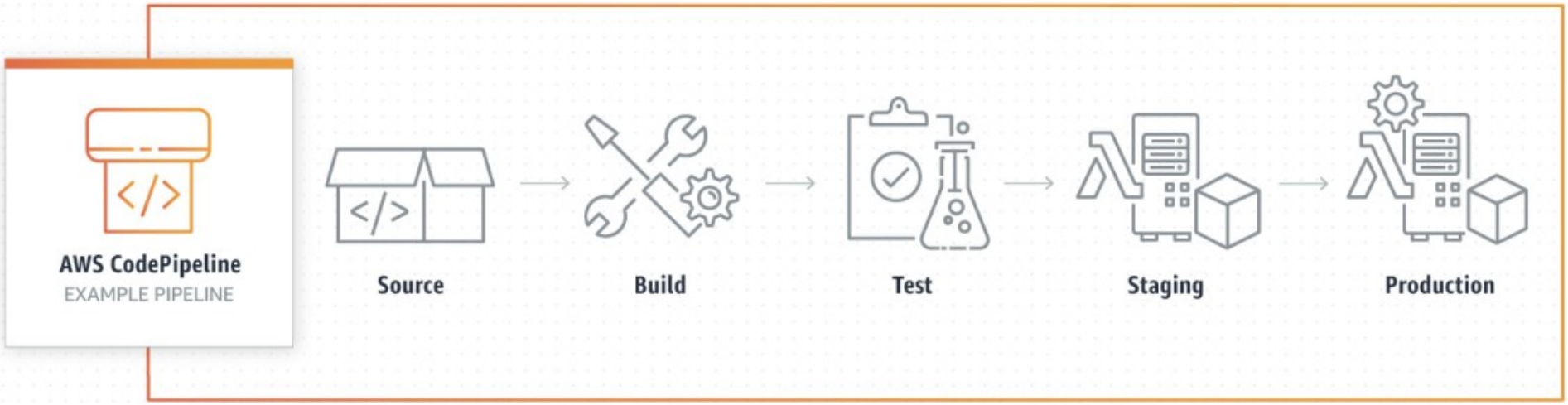
DevOps – CI/CD Pipeline

Process:

1. The developer develops the code and commits the changes to a centralized code repository.
2. When the repot detects a change, it triggers the Jenkins server.
3. Jenkins gets the new code and carries out the automated build and testing. If any issues are detected while building or testing, Jenkins automatically informs the development team via a preconfigured method, like email or Slack.
4. The final package is uploaded to AWS Elastic Beanstalk, an application orchestration service, for production deployment.
5. The elastic beanstalk manages the provisioning of infrastructure, load balancing, and scaling of the required resource type, such as EC2, RDS, or others.

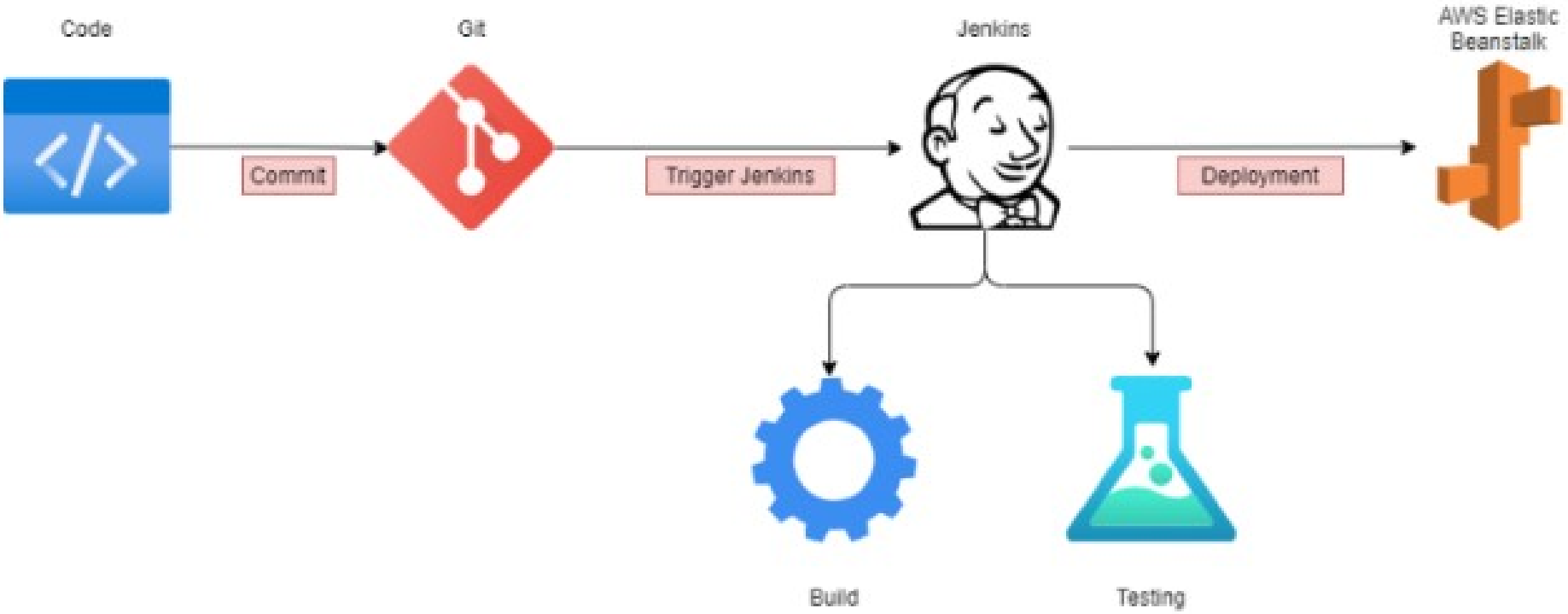
✓ The tools, processes, and complexity of a CI/CD pipeline will depend on the development requirements and business needs of the organization.

DevOps – CI/CD Pipeline – Cloud based CI/CD pipeline - AWS



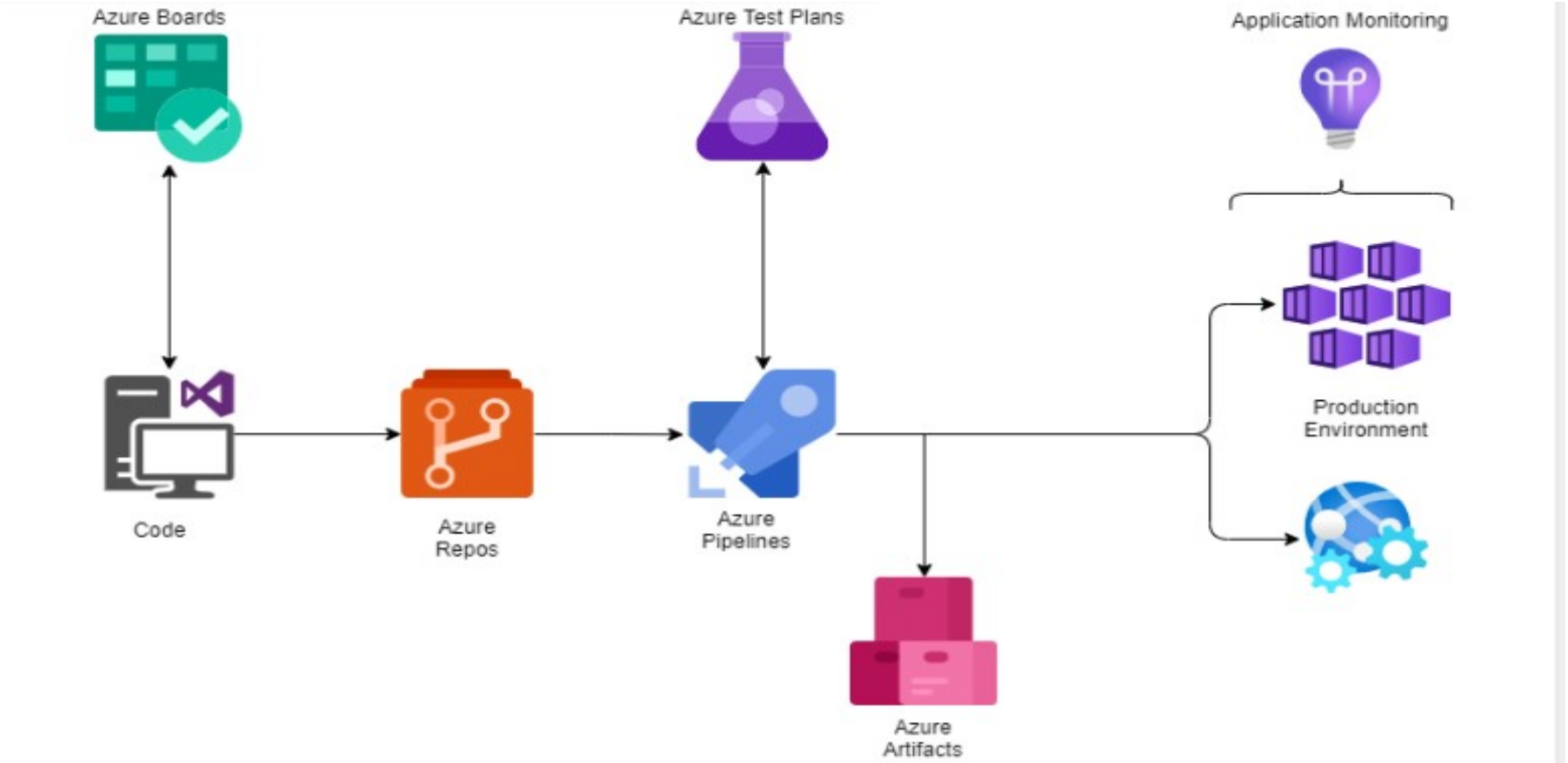
DevOps – CI/CD Pipeline

The following pipeline represents a simple web application development process.



Traditional CI/CD pipeline

DevOps – CI/CD Pipeline – Cloud based CI/CD pipeline - Azure



Cloud-based CI/CD pipeline

DevOps – MobaXterm

https://download.mobatek.net/2422024061715901/MobaXterm_Installer_v24.2.zip

DevOps – CI/CD Pipeline – Cloud based CI/CD pipeline - Azure

Process: (for Azure CI/CD Pipeline)

1. A developer changes existing or creates new source code, then commits the changes to Azure Repos.
2. These repo changes trigger the Azure Pipeline.
3. With the combination of Azure Test Plans, Azure Pipelines builds and tests the new code changes. (This is the Continuous Integration process.)
4. Azure Pipelines then triggers the deployment of successfully tested and built artifacts to the required environments with the necessary dependencies and environmental variables. (This is the Continuous Deployment process.)
5. Artifacts are stored in the Azure Artifacts service, which acts as a universal repository.
6. Azure application monitoring services provide the developers with real-time insights into the deployed application, such as health reports and usage information.

DevOps – CI/CD Pipeline - Stages

Source Stage:

- In the source stage, CI/CD pipeline is triggered by a code repository. Any change in the program triggers a notification to the CI/CD tool that runs an equivalent pipeline. Other common triggers include user-initiated workflows, automated schedules, and the results of other pipelines.

Build Stage:

- This is the second stage of the CI/CD Pipeline in which you ***merge the source code and its dependencies***. It is done mainly to build a runnable instance of software that you can potentially ship to the end-user.
- Programs that are written in languages like C++, Java, C, or Go language should be compiled. On the other hand, JavaScript, Python, and Ruby programs can work without the build stage.
- Failure to pass the build stage means there is a fundamental project misconfiguration, so it is better that you address such issue immediately.

Test Stage:

- Test Stage includes the execution of automated tests to validate the correctness of code and the behaviour of the software. This stage prevents easily reproducible bugs from reaching the clients. It is the responsibility of developers to write automated tests.

Deploy Stage:

- This is the last stage where your product goes live. Once the build has successfully passed through all the required test scenarios, it is ready to deploy to live server.

CI/CD Pipeline - Example

- **Source Code Control:** Host code on GitHub as a private repository. This will help you to integrate your application with major services and software.
- **Continuous integration:** Use continuous integration and delivery platform **CircleCI** and commit every code. When the changes notify, this tool will pull the code available in GitHub and process to build and run the test.
- **Deploy code to UAT:** Configure CircleCI to deploy your code to AWS UAT server.
- **Deploy to production:** You have to reuse continuous integration steps for deploying code to UAT.

CI/CD Pipeline - Advantages

- Builds and testing can be easily performed manually.
- It can improve the consistency and quality of code.
- Improves flexibility and has the ability to ship new functionalities.
- CI/CD pipeline can streamline communication.
- It can automate the process of software delivery.
- Helps you to achieve faster customer feedback.
- CI/CD pipeline helps you to increase your product visibility.
- It enables you to remove manual errors.
- Reduces costs and labour.
- CI/CD pipelines can make the software development lifecycle faster.
- It has automated pipeline deployment.
- A CD pipeline gives a rapid feedback loop starting from developer to client.
- Improves communications between organization employees.
- It enables developers to know which changes in the build can turn to the brokerage and to avoid them in the future.
- The automated tests, along with few manual test runs, help to fix any issues that may arise.

CI/CD Pipeline

Why Does the CI/CD Pipeline Matter for IT Leaders?

- CI/CD pipeline can improve reliability.
- It makes IT team more attractive to developers.
- CI/CD pipeline helps **IT leaders**, to pull code from version control and execute software build.
- Helps to move code to target computing environment.
- Enables **project leaders** to easily manage environment variables and configure for the target environment.
- **Project managers** can publish push application components to services like web services, database services, API services, etc.
- Providing log data and alerts on the delivery state.
- It enables programmers to verify code changes before they move forward, reducing the chances of defects ending up in production.

CI/CD Pipeline

Summary:

- A CI/CD pipeline automates the process of software delivery.
- CI/CD pipeline introduces automation and continuous monitoring throughout the lifecycle of a software product.
- Continuous integration is a software development method where members of the team can integrate their work at least once a day.
- Continuous delivery is a software engineering method in which a team develops software products in a short cycle.
- Continuous deployment is a software engineering process in which product functionalities are delivered using automatic deployment.
- There are four stages of a CI/CD pipeline 1) Source Stage, 2) Build Stage, 3) Test Stage, 4) Deploy Stage.
- Important CI/CD tools are Jenkins, Bambo, and Circle CI.
- CI/CD pipeline can improve reliability.
- CI/CD pipeline makes IT team more attractive to developers.
- Cycle time is the time taken to go from the build stage to production.
- Development frequency allows you to analyse bottlenecks you find during automation.
- Change Lead Time measures the start time of the development phase to deployment.
- Change Failure Rate focuses on the number of times development get succeeds vs. the number of times it fails.
- **MTTR (Mean Time to Recovery)** is the amount of time required by your team to recover from failure.
- MTTF (Mean Time to Failure) measures the amount of time between fixes and outages.

CI/CD Tools

Jenkins: Jenkins is an **open-source Continuous Integration server** that helps to achieve the Continuous Integration process (and not only) in an automated fashion.

- Jenkins is free and is entirely written in Java. Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

Features:

- Jenkin will build and test code many times during the day.
- Automated build and test process, saving timing, and reducing defects.
- The code is deployed after every successful build and test.
- The development cycle is fast.

Bamboo: Bamboo is a continuous integration build server that performs – automatic build, test, and releases in a single place. It works seamlessly with JIRA software and Bitbucket.

Features:

- Run parallel batch tests
- Setting up Bamboo is pretty simple
- Per-environment permissions feature allows developers and QA to deploy to their environments
- Built-in Git branching and workflows. It automatically merges the branches.

CI/CD Tools

- CircleCi is a flexible CI tool that runs in any environment like a cross-platform mobile app, Python API server, or Docker cluster. This tool reduces bugs and improves the quality of the application.
- **Features:**
- Allows to select Build Environment
- Supports many languages including C++, JavaScript, NET, PHP, Python, and Ruby
- Support for Docker lets you configure a customized environment.
- Automatically cancel any queued or running builds when a newer build is triggered.

CI/CD Tools

- **AWS CodeBuild:** Is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don't need to provision, manage, and scale your own build servers. CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue.
- **AWS CodePipeline:**

  AWS CodePipeline is a continuous integration and continuous delivery service for fast and reliable application and infrastructure updates. CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define. This enables you to rapidly and reliably deliver features and updates.

AWS CodeDeploy:

AWS CodeDeploy automates code deployments to any instance, including Amazon EC2 instances and on-premises servers. AWS CodeDeploy makes it easier for you to rapidly release new features, helps you avoid downtime during application deployment, and handles the complexity of updating your applications.

CI/CD Tools

- **AWS CodeBuild** is a fully managed continuous integration service that compiles source code, runs tests, and produces ready-to-deploy software packages.