

**Why Jenkins:**

When working on a project with different teams, developers often face issues with different teams using different CI tools, version management, and other tools.

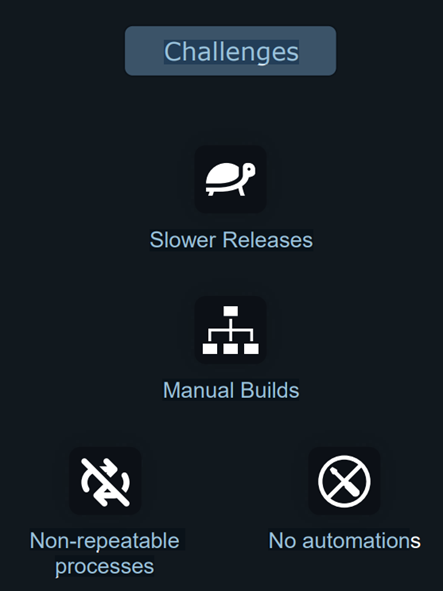
Setting up a CI/CD toolchain for each new project will lead to certain challenges like:

• Slower Releases

• Manual Builds

• Non-repeatable processes

• No Automations



Jenkins is the solution to those challenges.

It provides:

• Automated builds

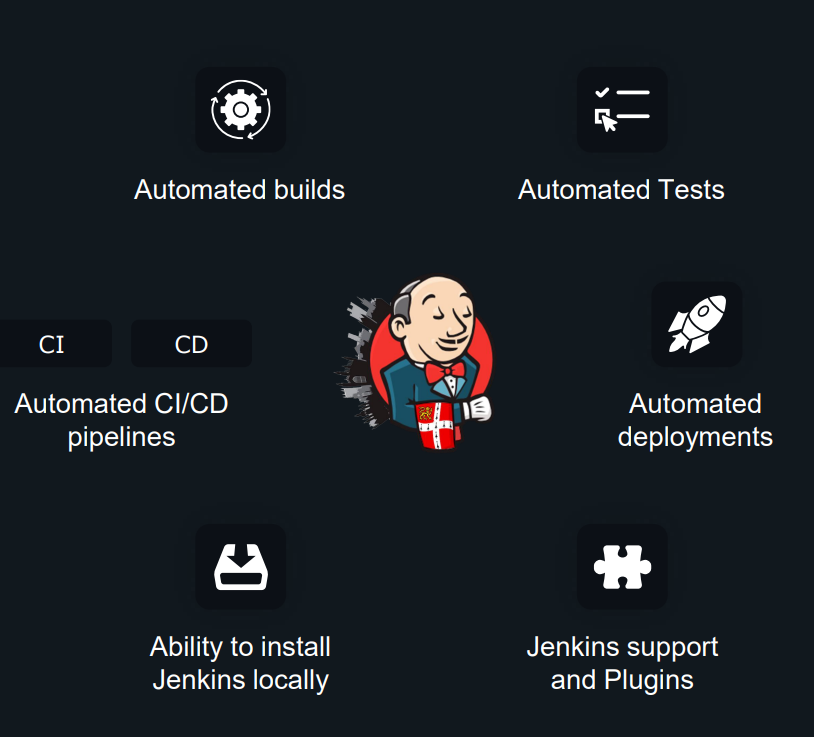
• Automated Tests

• Automated CI/CD pipelines

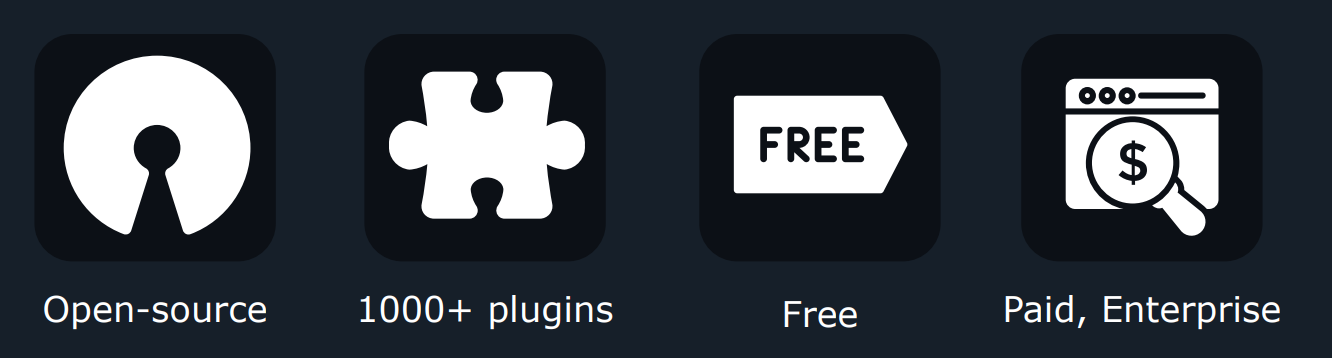
• Automated deployments

• Ability to install Jenkins locally

• Jenkins support and Plugins



* Open-source
* 1000+ Pluggins
* Free
* Paid/Enterprise

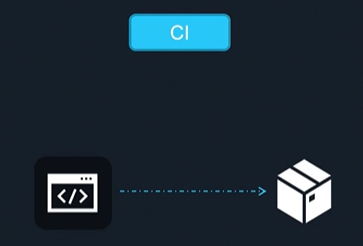


Jenkins is free and you don’t have to pay for anything. Jenkins can be hosted on a Virtual Machine, a container. Or even locally for development purposes

**CI/CD:**

**Continuous Integration:**

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.

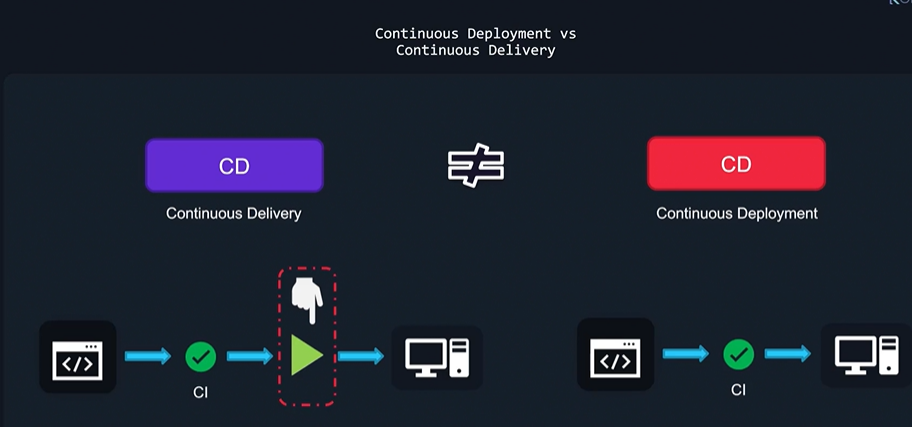
****

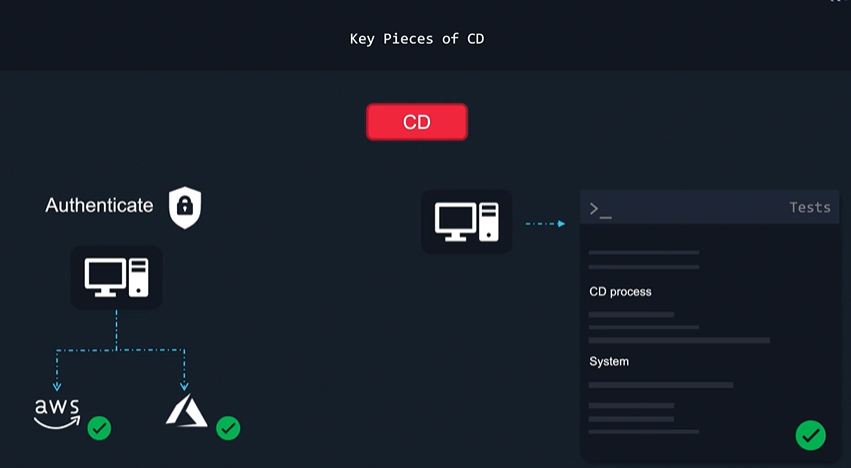
**Continuous Delivery:**

Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, without doing so manually. It aims at building, testing, and releasing software with greater speed and frequency.

**Continuous Deployment:**

Continuous deployment is a strategy for software releases wherein any code commit that passes the automated testing phase is automatically released into the production environment, making changes that are visible to the software's users.





**Install Jenkins:**

This is the process to install Jenkins on Ubuntu 22.04 which has created on AWS.

Step 1: Install Java on Ubuntu

$ sudo apt update

$ sudo apt install openjdk-8-jdk

Alternatively, install version 11:

$ sudo apt install openjdk-11-jdk

Step 2: Add the repository key to the system:

$ curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

Step 3: Once the key is added with no errors, append the Debian package repository address

$echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

Step 4: Run update to use new repository

$ sudo apt update

Step 5: Install Jenkins

$ sudo apt install jenkins

After successful installation let us start Jenkins

$ sudo systemctl start jenkins

$ sudo systemctl status jenkins

If its AWS ec2 with amazon linux

amazon-linux-extras install java-openjdk11 -y

6 amazon-linux-extras install java-openjdk11 -y

7 java -version

8 sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

9 rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key

10 yum upgrade

11 yum install jenkins -y

12 systectl enable jenkins

13 systemctl enable jenkins

14 systemctl start jenkins

15 systemctl status jenkins

update-alternatives --config java ( to change java with required versions)

<https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/>

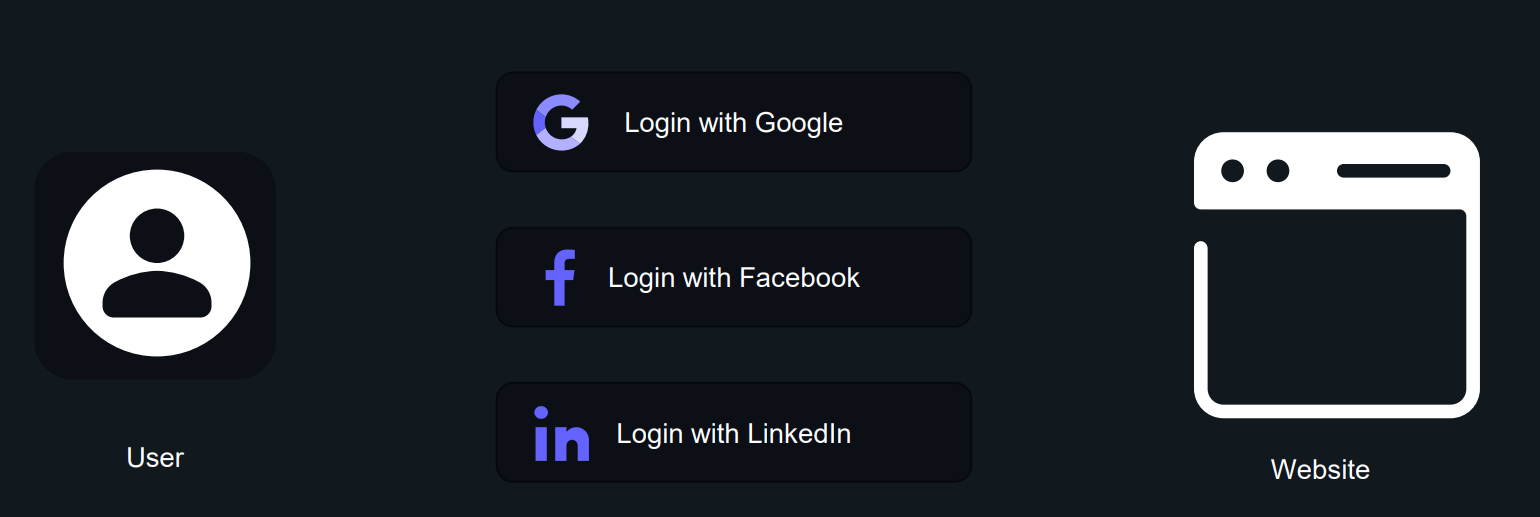
**Jenkins Plugins:**

Plugins are used in Jenkins to enhance Jenkins functionality and cater to user-specific needs.

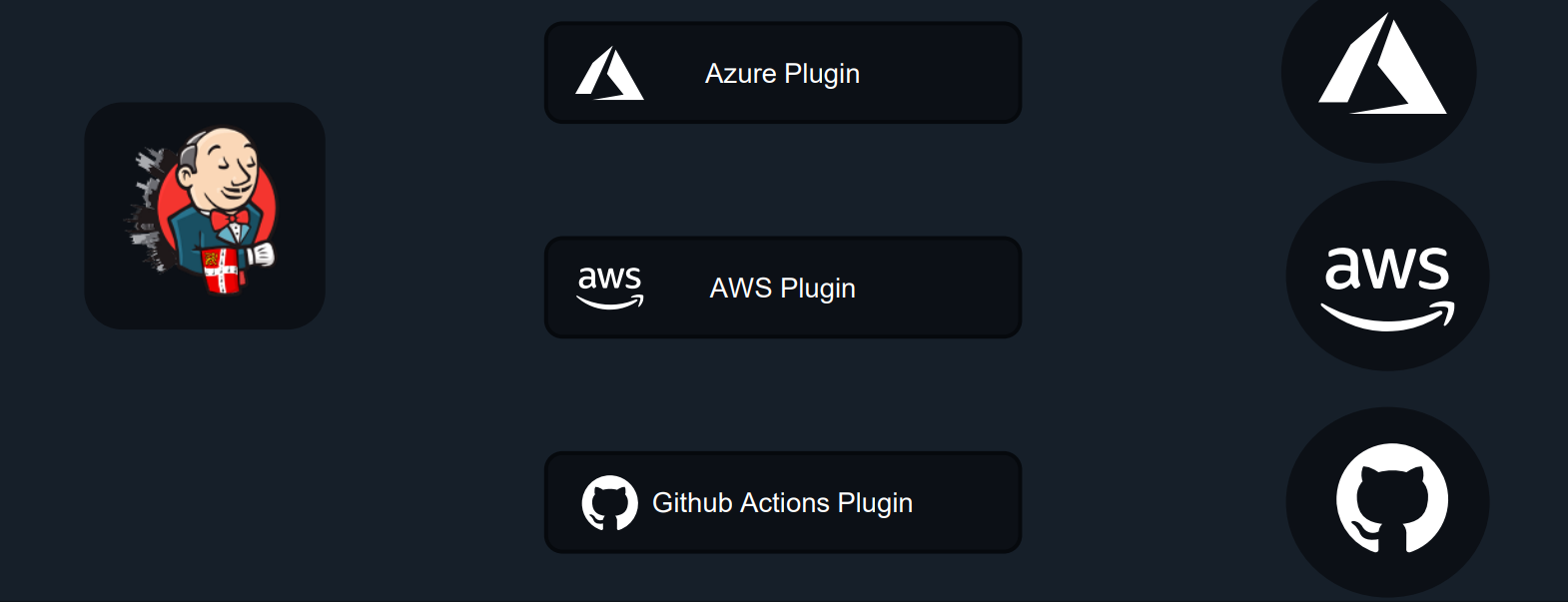
Just like how Gmail, Facebook and LinkedIn help you connect your one service to another,

plugins also work the same way and allow us to connect one service to other services and

work with other products.



For example, you want to connect to AWS from Jenkins you would need to download AWS Plugin which will allow you to connect to AWS at a programmatic level. Similarly, we can have other integrations with Azure, GitHub, etc using plugins.



**Install Plugins:**

In Jenkins we can install Plugins 3 ways.

1. **From GUI:**

To install a new plugin in Jenkins via GUI

1) Go to Manage Jenkins -> Manager Plugins

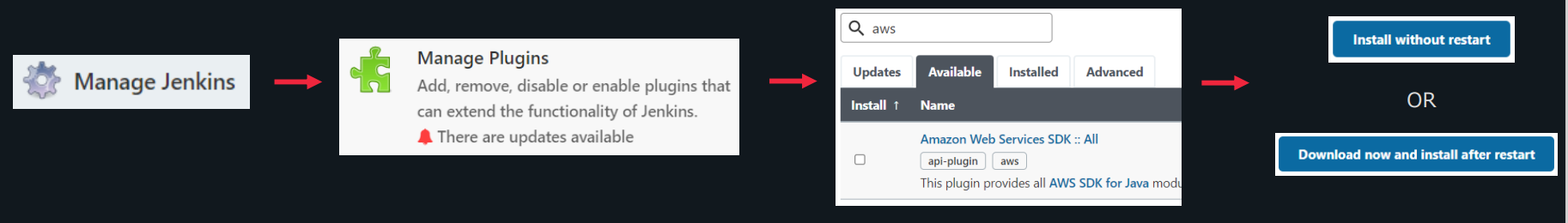
2) Click Available and search for the desired plugin.

3) Select the desired plugin and Install.

Note: Few plugins may need a restart

To restart Jenkins

$ sudo systemctl restart jenkins



1. **Browse Option from GUI:**
2. Download required Plugin in local machine (which is in .hpi format)
3. Go to Manage Jenkins -> Manager Plugins
4. Go to advanced option and browse the download plugin.
5. It will install respective plugin. If its required to restart you can restart the Jenkins.

**3. Copy Plugin in /var/lib/jenkins/plugin directory:**

1. Download required Plugin in local machine (which is in .hpi format)
2. Copy the same using any tool (like winscp) to jenkins server under location of /var/lib/jenkins/plugin location.
3. It will be effect from next restart of jenkins.

**Update Plugins:**

To update any existing plugin in Jenkins

1) Go to Manage Jenkins -> Manager Plugins

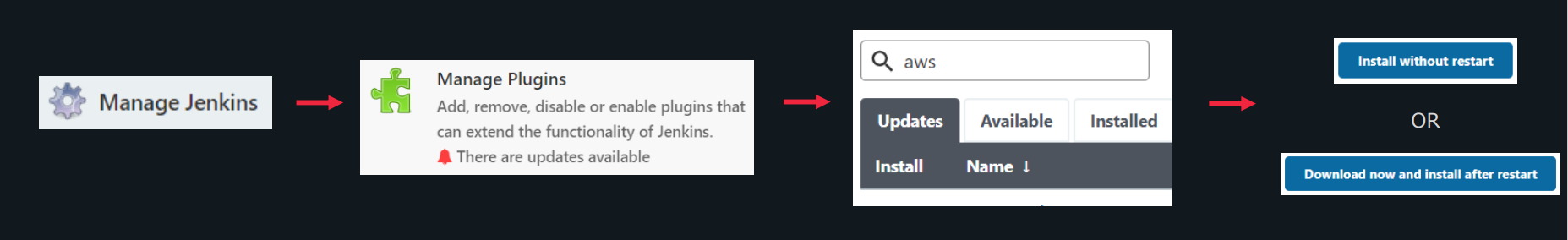
2) Click Updates and search for the desired plugin.

3) Select the desired plugin and Install.

Note: Few plugins may need a restart

To restart Jenkins

$ sudo systemctl restart jenkins



**Delete Plugins:**

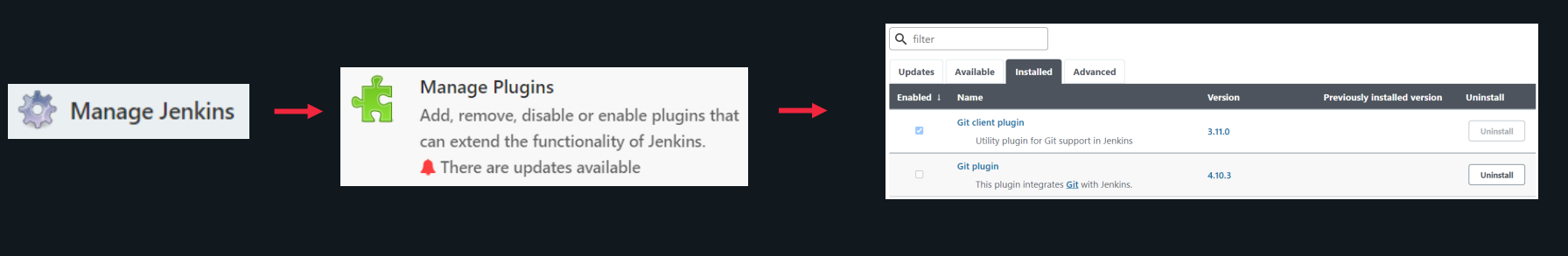
To delete any plugin in Jenkins

1) Go to Manage Jenkins -> Manager Plugins

2) Go to Installed and search for the desired plugin.

3) Click on uninstall button for the plugin you want to delete.

Click yes to proceed with the deletion.



**User creation:**

You can create users in Jenkins and assign proper role to that user. Follow below steps for the same.

1. Install [Role-based Authorization Strategy](https://plugins.jenkins.io/role-strategy) plugin if its not installed.
2. Enable Role-Based Strategy

Manage Jenkins -> configure global security -> under Authorization select Role-Based Strategy -> apply and save. You will get one addition feature Manage and Assign Roles.

1. Create user under Manage People.
2. Create role under Manage and Assign Roles - > Manage Role
3. Assign role to newly created user Manage and Assign Role -> Assign role.

Note: An user can be from any from below list. So Jenkins authentication can be integrated with a lot of varieties.

Azure Active Directory [?](http://52.66.245.68:8080/configureSecurity/)

Delegate to servlet container [?](http://52.66.245.68:8080/configureSecurity/)

Jenkins’ own user database [?](http://52.66.245.68:8080/configureSecurity/)

Allow users to sign up[?](http://52.66.245.68:8080/configureSecurity/)

LDAP [?](http://52.66.245.68:8080/configureSecurity/)

PWauth Authentication [?](http://52.66.245.68:8080/configureSecurity/)

Unix user/group database [?](http://52.66.245.68:8080/configureSecurity/)

None

**Jenkins Jobs**

Different types of jobs that can be created in Jenkins:

**1) Freestyle project**

This is a central feature of Jenkins. It will build the project, combine SCM with the build system. It can also be used for things other than building applications.

**2) Pipeline**

This is used to create a pipeline

**3) Multi-configuration project**

This is great if you need a large number of Jenkins configurations if you need multiple environments

like Dev/ UAT.

**4) Folder**

This creates containers and stores nested items. It is useful in grouping, creating a namespace, etc.

A **multi-branch pipeline** is a CI/CD pipeline strategy where a separate pipeline is automatically created and executed for each branch in a version control system (e.g., Git). This type of pipeline dynamically detects all branches in the repository and runs specific jobs based on the configuration defined in a pipeline script (e.g., Jenkinsfile, .gitlab-ci.yml). Each branch can have its own dedicated pipeline, which makes it particularly useful in managing different development workflows such as feature development, testing, and deployment.

pipeline {

agent any

stages {

stage('Build') {

steps {

echo 'Building...'

}

}

stage('Test') {

steps {

echo 'Testing...'

}

}

stage('Deploy') {

when {

branch 'main' // Deploy only if it's the main branch

}

steps {

echo 'Deploying...'

}

}

}

}

**Benefits of Multi-Branch Pipelines**

1. **Branch Isolation**:
   * Each branch has its own isolated pipeline, meaning builds, tests, and deployments can be handled separately for each branch. This prevents issues from one branch impacting others.
2. **Automatic Detection of New Branches**:
   * New branches are automatically detected and processed, which reduces manual work. Developers don’t need to configure new pipelines for every feature or bug fix branch they create.
3. **Consistency Across Environments**:
   * Since the pipeline is version-controlled alongside the code, every branch has a consistent CI/CD process. This makes it easy to ensure that development, testing, and production follow the same steps, with appropriate variations as needed (e.g., using mock services in testing environments).
4. **Efficient Feature Branch Testing**:
   * Multi-branch pipelines are ideal for **GitFlow** or similar branching models where multiple developers work on different features in parallel. Every feature branch gets its own pipeline for running tests, validating builds, and even deploying to test environments.
5. **Automatic Cleanup**:
   * Pipelines can be configured to automatically stop or clean up resources (e.g., test environments) when branches are merged or deleted, keeping the CI/CD environment clean.
6. **Branch-Specific Rules**:
   * You can configure different rules for different branches. For example:
     + master/main branch might trigger deployment to production.
     + develop branch might deploy to a staging environment.
     + Feature branches might only trigger build and test stages without deployment.

**Use Cases for Multi-Branch Pipelines**

1. **Feature Branch Development**:
   * When following a **GitFlow** or feature branch workflow, each branch (e.g., feature, hotfix, bugfix) gets its own pipeline. This allows teams to develop and test features in isolation from the main branch.
2. **Parallel Testing and Deployment**:
   * Multi-branch pipelines enable parallel testing across different branches, ensuring faster feedback for all team members.
3. **Environment-Specific Pipelines**:
   * Pipelines can be configured to perform different actions based on the branch (e.g., only deploy to production from the main branch or deploy to a staging environment from the develop branch).
4. **Continuous Deployment**:
   * With multi-branch pipelines, you can configure different branches to deploy automatically to different environments. For instance, main might deploy to production, develop to staging, and feature branches to isolated test environments.

**Advantages of Multi-Branch Pipelines**

* **Automation**: Automatically creates pipelines for new branches without manual intervention.
* **Flexibility**: Different branches can follow different CI/CD workflows, tailored to their needs.
* **Branch-Specific Testing**: Ensures that feature branches, hotfixes, and main branches all get tested appropriately.
* **Consistency**: Because pipelines are defined as code, there's consistency across branches and environments.

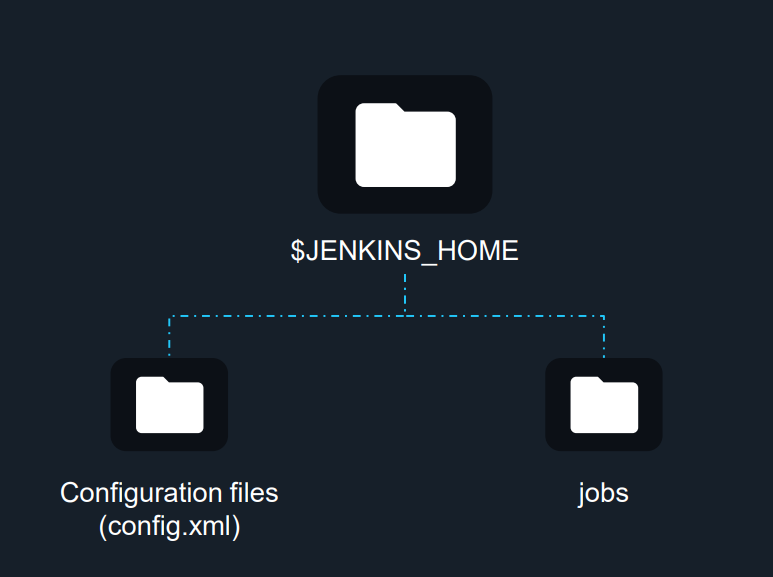
**Administering Jenkins**

* Backup
* Re-Store
* Monitor
* Scale
* Manage

**Backup and Restore:**

Backup can be Full backup, Snapshot backup.

Which Files To Backup?



It is crucial to have adequate backups of your Jenkins instance. Backups are

used to recover from accidental configuration changes. Recovering a file that

has been mistakenly erased or has been corrupted. Or just to recover a

previous setup.

There are two ways we can backup Jenkins:

1) Using Plugins

2) Using custom shell script

3) take your jenkins FS snapshot

To backup Jenkins using a plugin, you will first need to install a backup plugin.

Some of the most commonly used plugins are

ThinBackup,

Periodic Backup,

Google cloud Backup.

For backing up using any of these plugins there are a few general steps that

must be followed:

1) Creating a backup directory with read and write access

2) Selecting files that need backup

Backing up using shell script

Please check out these popular repositories for your reference:

1) repository: <https://github.com/sue445/jenkins-backup-script>

2) gist: <https://gist.github.com/abayer/527063a4519f205efc74>

**Jenkinsfile**

Jenkinsfile is a text file that contains definitions. This could be

templates or instructions. It tells pipelines what they should be

doing and what services and plugins they should be interacting

with.

Components of Jenkinsfile:

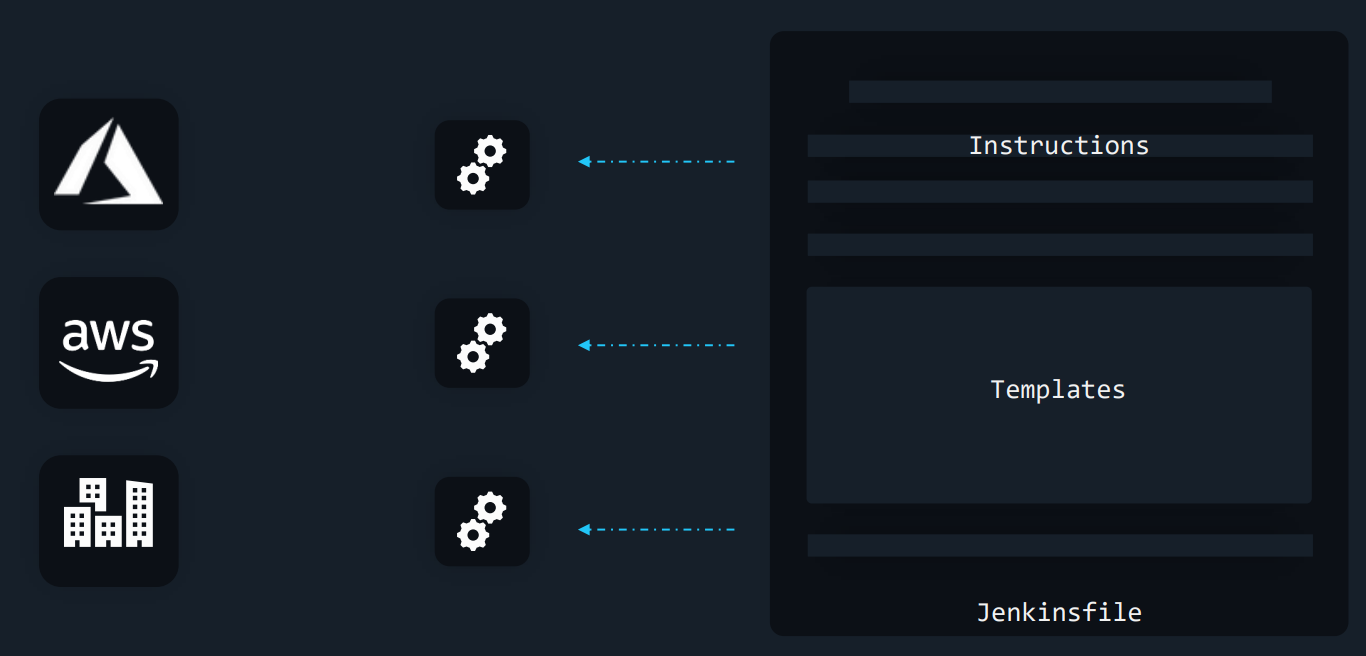
1) Pipeline – The task you are trying to accomplish

2) Build Agent –The place where you run your pipeline

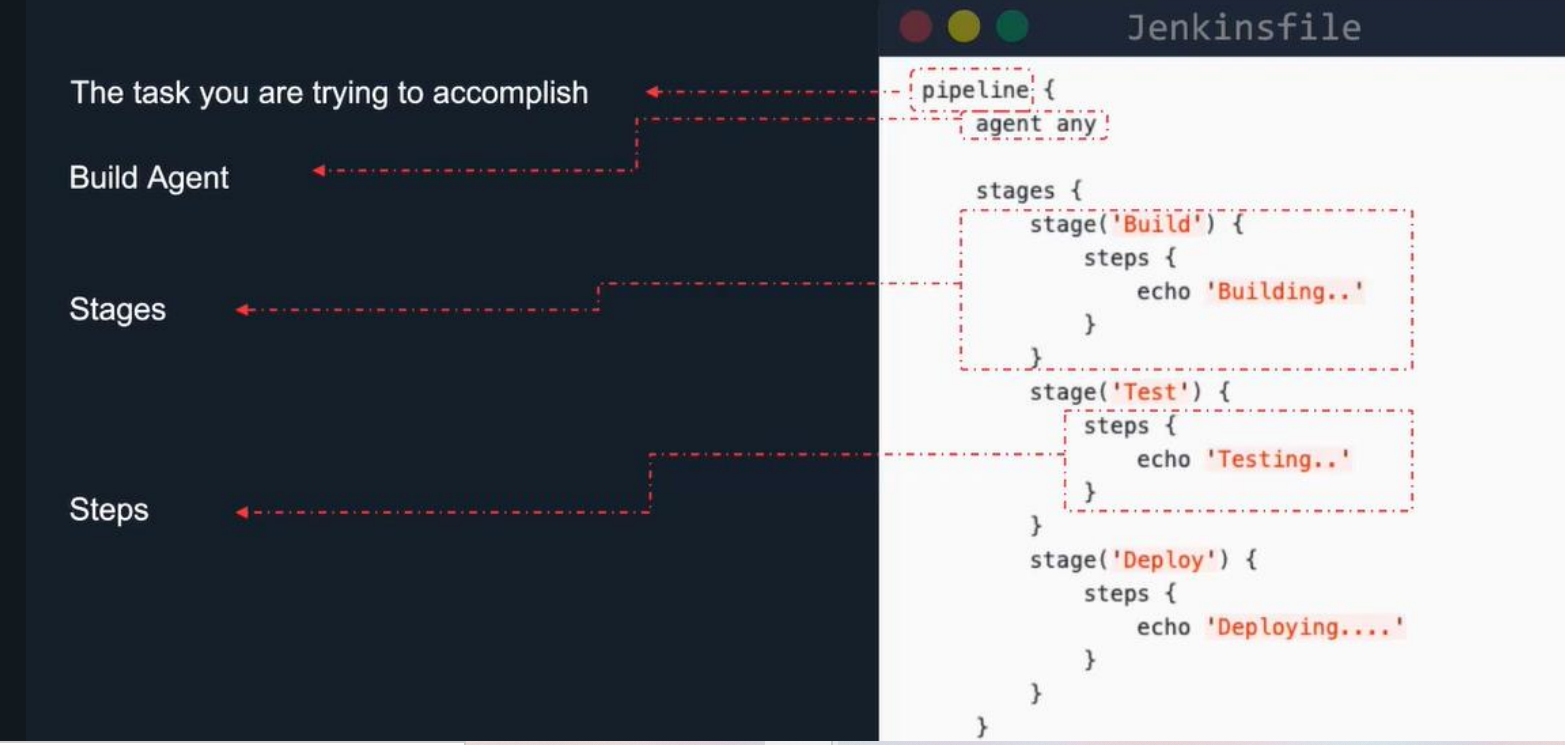
3) Stages – Staging/Production/UAT

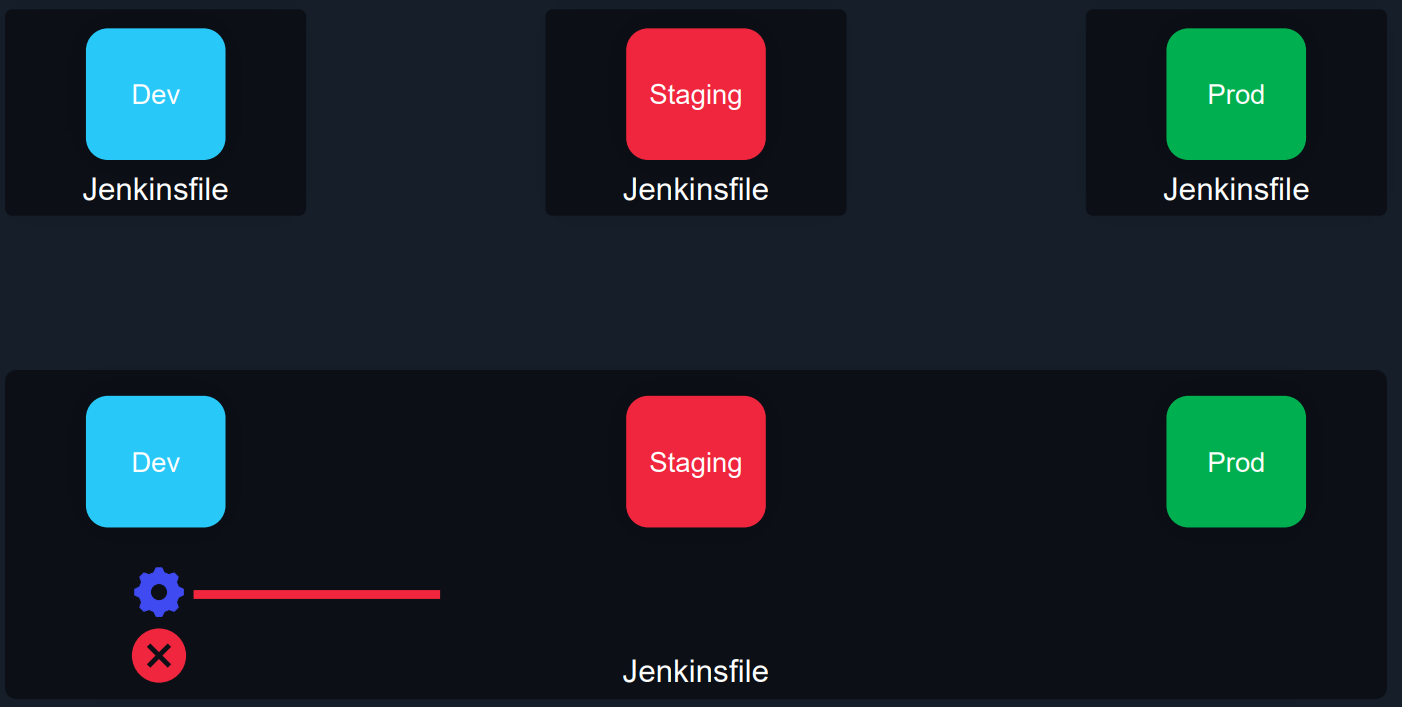
4) Steps –Work done in the pipeline

What is Jenkins file:

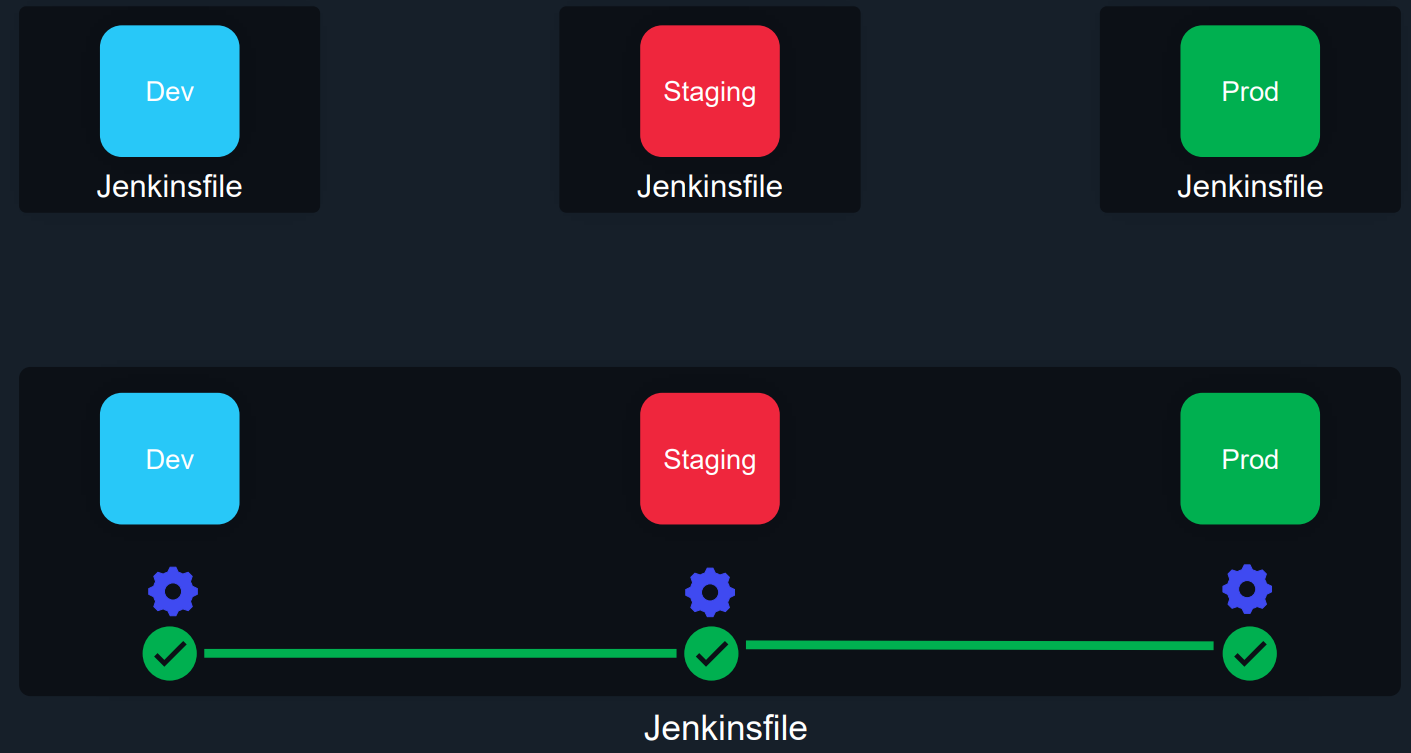


Components of Jenkinsfile:





Multi-Stage Pipelines:



**Tomcat Server Installation and Configuration:**

### **Pre-requisites**

1. EC2 instance with Java 11

sudo amazon-linux-extras install java-openjdk11

**Install Apache Tomcat**

Download tomcat packages from <https://tomcat.apache.org/download-80.cgi> onto /opt on EC2 instance  
Note: Make sure you change <version> with the tomcat version which you download.  
# Create tomcat directory

cd /opt

wget http://mirrors.fibergrid.in/apache/tomcat/tomcat-8/v8.5.35/bin/apache-tomcat-8.5.35.tar.gz

1. tar -xvzf /opt/apache-tomcat-<version>.tar.gz

give executing permissions to startup.sh and shutdown.sh which are under bin.  
chmod +x /opt/apache-tomcat-<version>/bin/startup.sh

1. chmod +x /opt/apache-tomcat-<version>/bin/shutdown.sh  
   Note: you may get below error while starting tomcat incase if you dont install Java  
   Neither the JAVA\_HOME nor the JRE\_HOME environment variable is defined At least one of these environment variable is needed to run this program

create link files for tomcat startup.sh and shutdown.sh  
ln -s /opt/apache-tomcat-<version>/bin/startup.sh /usr/local/bin/tomcatup

ln -s /opt/apache-tomcat-<version>/bin/shutdown.sh /usr/local/bin/tomcatdown

1. tomcatup

#### **Check point :**

access tomcat application from browser on port 8080

* http://<Public\_IP>:8080

Using unique ports for each application is a best practice in an environment. But tomcat and Jenkins runs on ports number 8080. Hence lets change tomcat port number to 8090. Change port number in conf/server.xml file under tomcat home

cd /opt/apache-tomcat-<version>/conf

# update port number in the "connecter port" field in server.xml

# restart tomcat after configuration update

tomcatdown

tomcatup

#### **Check point :**

Access tomcat application from browser on port 8090

* http://<Public\_IP>:8090

now application is accessible on port 8090. but tomcat application doesnt allow to login from browser. changing a default parameter in context.xml does address this issue  
#search for context.xml

1. find / -name context.xml

above command gives 3 context.xml files. comment () Value ClassName field on files which are under webapp directory. After that restart tomcat services to effect these changes. At the time of writing this lecture below 2 files are updated.  
/opt/tomcat/webapps/host-manager/META-INF/context.xml

/opt/tomcat/webapps/manager/META-INF/context.xml

# Restart tomcat services

tomcatdown

1. tomcatup

Update users information in the tomcat-users.xml file goto tomcat home directory and Add below users to conf/tomcat-users.xml file  
 <role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<user username="admin" password="admin" roles="manager-gui, manager-script, manager-jmx, manager-status"/>

<user username="deployer" password="deployer" roles="manager-script"/>

1. <user username="tomcat" password="s3cret" roles="manager-gui"/>
2. Restart serivce and try to login to tomcat application from the browser. This time it should be Successful

**Projects:**

**Pre requisites**

1. Github URL: [lakshmiprasad2019/myweb (github.com)](https://github.com/lakshmiprasad2019/myweb)
2. Maven installation: maven need to install on Jenkins server.

#yum install maven

#mvn -version

1. Please install maven plugin and deploy to container plugin in Jenkins dash board as shown in class.
2. Please configure maven in global tool configuration as shown
3. Please configure Tomcat directory permissions and ownership and jenkins user add in Tomcat server.

**Free Style project:**

As shown in Demo, Please use above Github link to create CI/CD Pipe line using Free Style project.

**Pipeline Project:**

As shown in Demo please use above same Github link and use below Pipeline to configure CI/CD Pipleline deployment using Pipeline project.

pipeline {

agent any

stages {

stage ('git\_clone') {

steps {

git 'https://github.com/lakshmiprasad2019/myweb.git'

}

}

stage (mvn\_build) {

steps {

sh "mvn clean package"

}

}

stage("deploy-dev"){

steps{

sh """

scp -o StrictHostKeyChecking=no target/myweb-8.2.5.war jenkins@172.31.38.152:/tomcat/apache-tomcat-8.5.79/webapps

ssh jenkins@172.31.38.152 /tomcat/apache-tomcat-8.5.79/bin/shutdown.sh

ssh jenkins@172.31.38.152 /tomcat/apache-tomcat-8.5.79/bin/startup.sh

"""

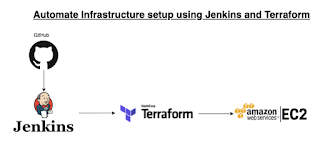
}

}

}

}

**Create Infrastructure using Jenkins+Terraform.**



pipeline {

agent any

stages {

stage('git-checkout') {

steps {

git branch: 'main', url: 'https://github.com/lakshmiprasad2019/my-tf-iac-aws-repo.git'

}

}

stage('Terraform Init') {

steps {

sh ('terraform init')

}

}

stage('Terraform Action') {

steps {

echo "Terraform action is --> ${action}"

sh ('terraform ${action} --auto-approve')

}

}

}

}

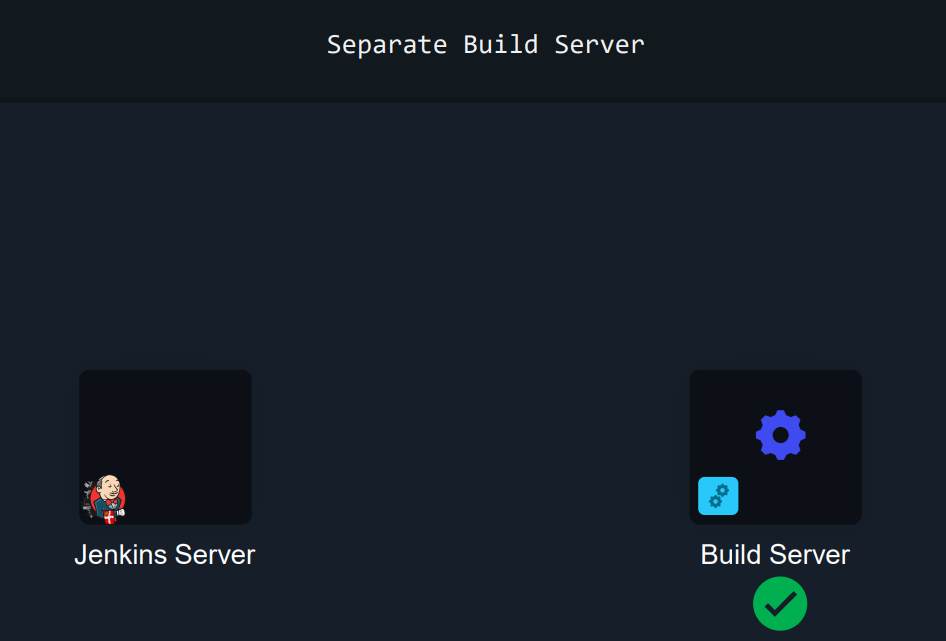
**Build Agent:**

Build Agents are systems that run the processes throughout the

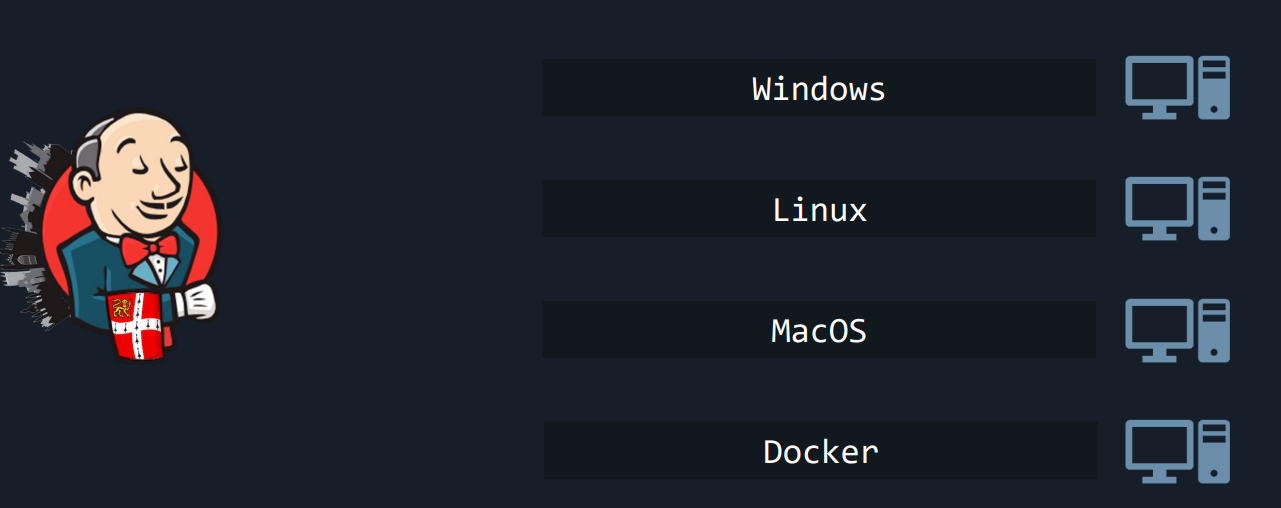
pipelines.

Build agents help in building codes, deploying, and running

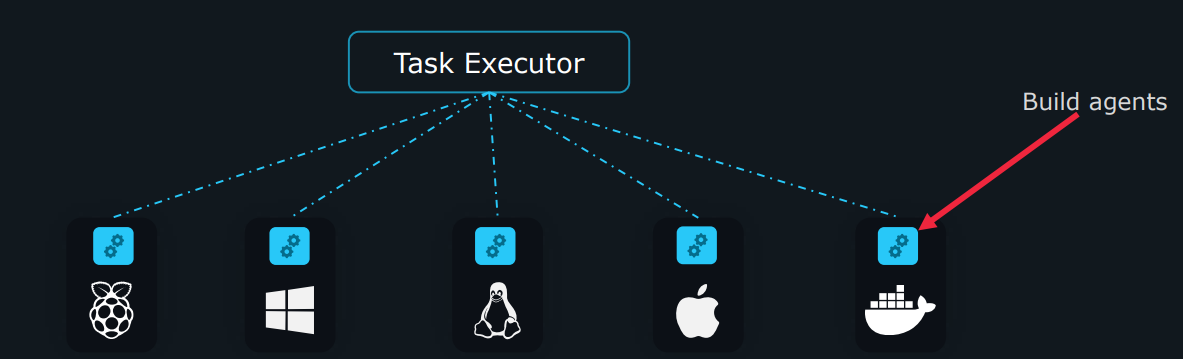
automated tests. It is a system that runs the entire workload



A build agent can be any one OS server and even you can use docker container as a image also.



Functionality of Build agent



As shown in class, please configure build agent and run below different piple lines.

pipeline {

agent {

label "node01"

}

stages {

stage('Hello') {

steps {

echo 'Hello World'

}

}

}

}

Docker, dokcer slave and dokcer pipeline plugins need to install

chmod 666 /var/run/docker.sock

pipeline {

agent {

docker { image 'golang:latest' }

}

stages {

stage('Development') {

steps {

git 'https://github.com/AdminTurnedDevOps/go-webapp-sample.git'

sh 'go version'

}

}

}

}