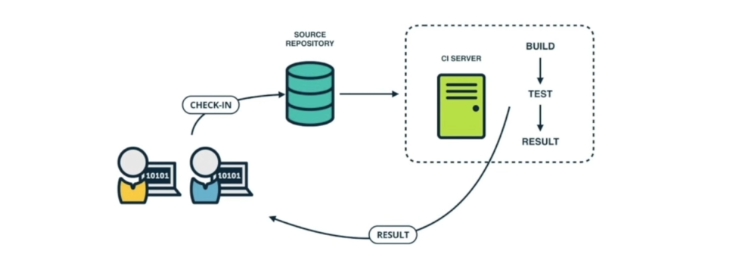
**Jenkins**

1. **Continuous Integration:**



Continuous Integration is a development practice that requires developers to integrate code into a repository several times a day.

Each check in is then verified by an automated build, allowing teams to detect problems easily.

If build is not green, system notify developer immediately. By this , developer can detect errors quickly , and locate them more easily.

1. **Stage of Adopting Continuous Integration.**

Continuous Integration is backed by several important principles and practices.

1. **Maintain a Single Source Repository**

Everyone on the team keeps their code in one central place, like a shared folder. This way, everyone works with the same code, and all changes are stored in one location. It helps everyone stay on the same page and reduces confusion about where the latest code is.

1. **Automate the build**

Instead of manually checking if the code works (like compiling it), a computer does it automatically every time someone changes the code. This is called "building" the code. It saves time and quickly catches mistakes, so they can be fixed before they cause bigger problems.

1. **Make your code self-testing**

Whenever the code is built, it automatically runs tests to check if everything still works as expected. This helps catch bugs early, so new code doesn’t break existing features.

1. **Make It Easy for Anyone to Get the Latest Executable Version**

The latest version of the software (like a ready-to-use app) is always available for anyone on the team to download and use. It makes it simple for team members to test, use, or show the latest version of the software without waiting for someone to prepare it.

1. **Everyone Can See What’s Happening**

Everyone on the team can easily see if the code is working, if the tests passed, and if there were any issues. It keeps the team informed about the project’s status, helps catch problems early, and promotes teamwork.

1. **How to do it**
2. **Developers Check Out the Code into Their Own Workspaces**

Each developer makes a copy of the code from the central repository to work on their own computer. This is called "checking out" the code. It allows developers to work on their own tasks independently without affecting the main codebase.

1. **When Done, Commit the Changes to the Repository**

After making changes to the code (like adding features or fixing bugs), the developer "commits" these changes back to the central repository. Committing changes regularly ensures that everyone has access to the latest code and reduces the chances of conflicts when merging different pieces of work.

1. **CI Server Monitors the Repository and Checks Out Changes When They Occur**

A CI server (like Jenkins) constantly watches the central repository. When it notices new changes, it automatically pulls those changes (checks them out) to start the CI process. This ensures that any new changes are immediately tested and integrated, so issues can be identified quickly.

1. **CI Server Builds the System and Runs Unit and Integration Tests**

The CI server compiles the code to make sure it works and then runs automated tests. These tests check if the individual parts (unit tests) and combined parts (integration tests) of the software work correctly. This step ensures that the code is always in a working state and that new changes don’t break existing functionality.

1. **CI Server Releases Deployable Artifacts for Testing**

If the build and tests are successful, the CI server packages the software into a "deployable artifact" (like an executable file or a Docker image) that can be used for further testing or deployment. This makes it easy to deploy the latest version of the software to a testing environment or even to production, ensuring that the software is always ready for release.

1. **CI Server Assigns a Build Label to the Version of the Code It Just Built**

The CI server gives this version of the software a unique name or "build label," like a version number. This helps keep track of different versions of the software.

1. **CI Server Informs the Team of the Successful Build**

The CI server lets the team know that everything went well with the build, usually by sending a message through email, Slack, or another communication tool.

1. **If the Build or Tests Fail, the CI Server Alerts the Team**

If something goes wrong—like the code doesn’t build correctly or a test fails—the CI server immediately alerts the team. This way, everyone knows there’s an issue that needs fixing.

1. **Continuous Deployment**

**Continuous Delivery** is a method used in software development to make sure that code changes are delivered smoothly and quickly to users. Here’s a simple breakdown:

* 1. **Code Changes:** Developers write and update the code.
  2. **Automated Build:** The code is automatically built and packaged into a deployable version.
  3. **Testing:** The new version of the code is automatically tested to ensure it works correctly.
  4. **Staging Environment:** The code is first deployed to a staging environment, which is a replica of the production environment. This is where Quality Assurance (QA) and other reviews happen.

**Example:** Before a new feature or fix is released to everyone, it’s tested in a staging environment to make sure it doesn’t break anything.

* 1. **Approval:** After testing in staging, if everything looks good, the code is approved for release.
  2. **Production Environment:** Finally, the code is deployed to the production environment where users can see and use the changes.

1. **Jenkins**

Jenkins is an open-source automation server widely used for continuous integration (CI) and continuous delivery (CD) in software development. It helps automate the parts of software development related to building, testing, and deploying, facilitating a CI/CD pipeline.

It is written in java

It is an automated tool, used to build and deliver the software product

Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

**Why Jenkins?**

* Code is built and test as soon as developer commits code. Jenkins will build and test code many times during the day.
* On successful build, jenkins will deploy the source into the test server and notifies the deployment team.
* On build failures, jenkins will notify the errors to the developer team

1. **Installing Jenkins On linux/Unix Cloud Machine**

Java must be install on host machine

Make sure Java 8 is installed **(Now you can use java 17 version too for jenkins installation)**

Verify java version: **java - -version**

**Steps**

1. **Install Java to execute Jenkins war files**

Sudo add-apt-repository ppa:webupd8team/java

Sudo apt-get update

Sudo apt install openjdk-8-jdk

1. **Install Jenkins on Machine**

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg

1. sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
2. sudo apt-get update
3. sudo apt-get install Jenkins

**(if youre using your linux unix on your local machine you can run Jenkins on localhost:8080**

**If youre using the cloud machine then you need to find the ip of this machine and access it with the ip)**

**(If youre using aws machine you need to open the inbound traffic with the help of your security group)**

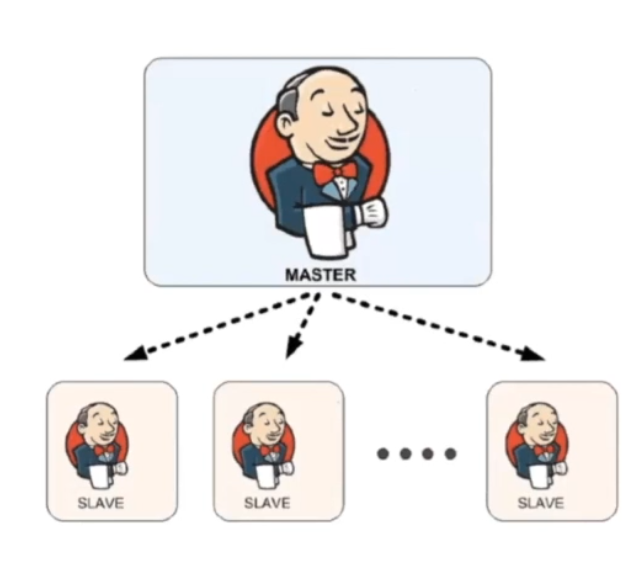
1. ip addr show

(get the id and use it in web browser. Suppose ip address is 192.168.0.210

**192.168.0.210 :8080** )

1. cat <the path given to unlock Jenkins> **(to get the initial admin password)(use the password to login)**
2. Install suggested plugins
3. Create first admin user(Make sure this account will be your admin account)

Run Jenkins:  
**java -jar jenkins.war**

1. **Jenkins Master and slave Architecture**

**Jenkins Master(Controller)**

1. **Scheduling Build Jobs**

The Jenkins Master is like a project manager. When you create a new task (a "build job"), the Master is responsible for organizing when and how that task will be done. It keeps track of all the tasks and decides the best time to start each one.

1. **Dispatching Builds to the Slave**

Once the Jenkins Master has scheduled a task, it doesn’t do the work itself. Instead, it finds a worker (called a "Slave" or "Agent") to do the job. The Master sends the task to the Slave, which is like assigning the work to an employee who has the right tools and skills to complete it.

1. **Monitoring the Slave and Recording the Results**

While the Slave is working on the task, the Jenkins Master keeps an eye on it to make sure everything is going smoothly. Once the job is done, the Slave reports back to the Master with the results. The Master then records these results so that you can review how the task went, whether it was successful or if there were any issues.

**Jenkins Slave(Agent)**

The Jenkins Slave’s main job is to do the work assigned to it by the Jenkins Master. It doesn’t decide what work to do on its own; it waits for the Master to give it a task.

1. **Jenkins Jobs**

In Jenkins, a **job** (or sometimes called a **project**) refers to a task or set of tasks that Jenkins will perform. This could be something simple, like compiling code, or something more complex, like running a full software build, testing, and deployment pipeline.

**Slave agents on slave**

**"Slave Agent on Slave"** refers to the setup where a Jenkins Slave (Agent) machine runs another Jenkins agent within itself.

* **Jenkins Slave Agent**: A small program running on a Slave machine that allows it to receive and execute jobs from the Jenkins Master.
* **Job Distribution**: The Jenkins Master assigns jobs to available Slaves to balance the workload and execute tasks.
* **Node**: Any machine in Jenkins, including both the Master and all Slaves, that is part of the Jenkins system.

**Jenkins** **Executor**: In Jenkins, an **executor** is a component that handles the execution of jobs on a Jenkins Slave (or Master, if it's running jobs directly). Think of an executor as a worker on a Jenkins Slave machine that runs the build tasks or jobs. Each executor can run one job at a time. If you have multiple executors on a Slave, that Slave can run multiple jobs simultaneously.

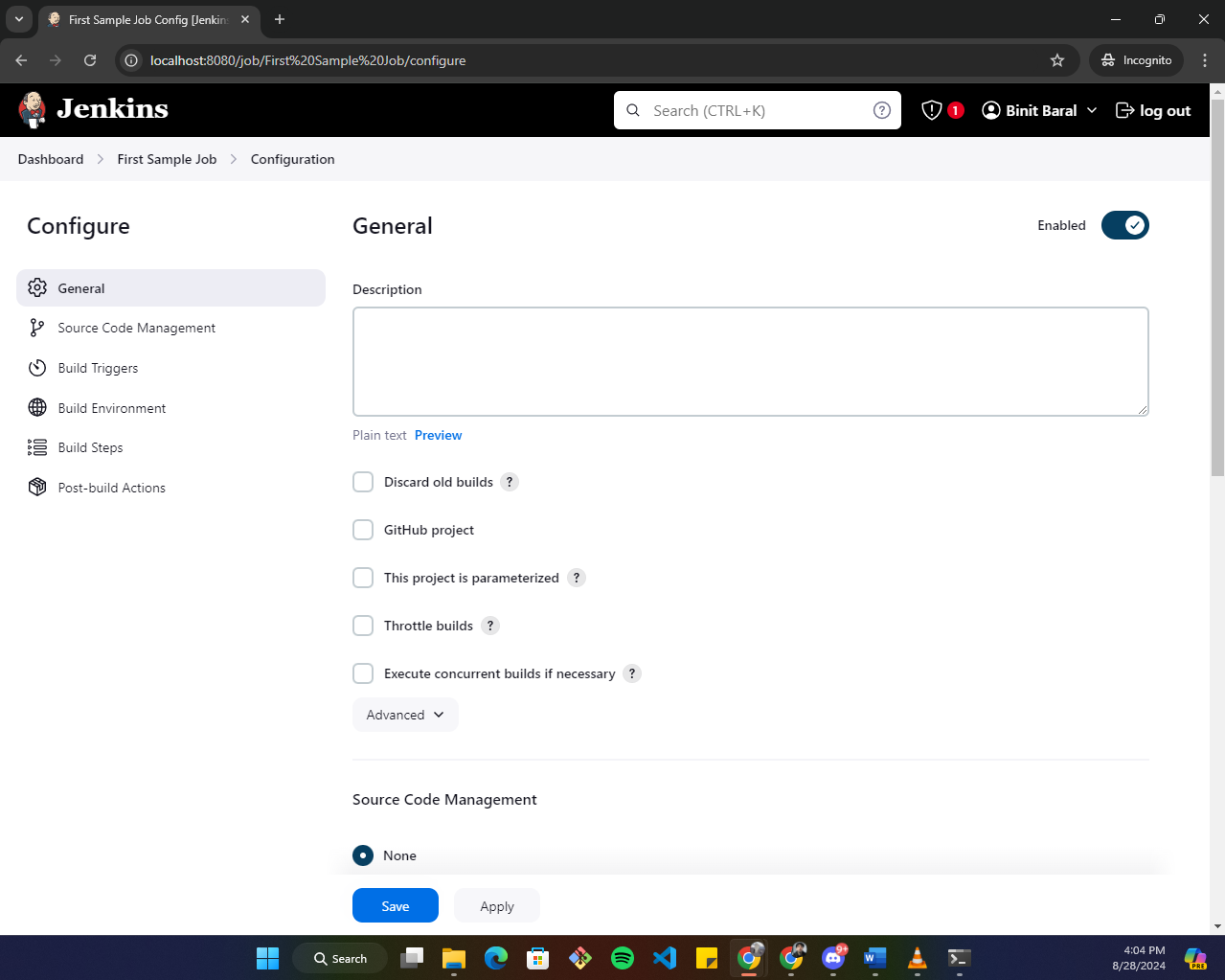
**Jenkins Plugins:**

Jenkins plugins are modules that can be installed to add new capabilities to the Jenkins server. They help Jenkins integrate with various tools and services used in software development, testing, deployment, and more.

1. **Creating Jenkins job**
2. **Create job**
3. **Select freestyle project**

* Freestyle Project: A basic job type where you can define simple build steps like compiling code, running tests, and deploying applications.
* Pipeline: A job type that allows you to define your entire build process as code using a script, making it easier to automate complex workflows.
* Multi-Configuration Project: A job type used to run builds in different environments or configurations, like testing on multiple operating systems or Java versions.
* Folder: A way to organize your jobs and other items into folders, making it easier to manage large numbers of projects.
* Multibranch Pipeline: A job type that automatically creates pipelines for each branch in your version control system, allowing you to run builds and tests on multiple branches simultaneously.
* Organization Folder: A job type that scans your organization’s version control repositories and automatically creates jobs for all the projects, managing them under one folder.

**Parameters:** In Jenkins, **parameters** are variables or inputs that you can define for a job, allowing you to customize how the job runs each time. They let you pass different values or options into the job when it starts, which can change its behavior based on those inputs.

**In General:** Basic settings for your Jenkins job, like naming the job and setting job descriptions or parameters.

* Discard Old Builds: Automatically delete old build data after a certain number of builds or days to save space
* GitHub Project: Link your Jenkins job to a specific GitHub repository, making it easier to track changes and view project information.
* This Project is Parameterized: Allows you to define parameters that can be customized each time the job is run, like input values or options for the build.
* Throttle Builds: Limit the number of builds that can run simultaneously, either globally or per project, to avoid overloading resources.
* Execute Concurrent Builds if Necessary: Let Jenkins run multiple builds of the same job at the same time, useful if different builds don’t interfere with each other.
* Advanced Options: Additional settings for more specific configurations, like setting custom workspace locations or blocking builds based on certain criteria.

**In source code management:** Where you connect Jenkins to your version control system (e.g., Git, SVN) to pull the source code for the job.

* None: No source code management is used; the job doesn’t pull code from a version control system.
* Git: Connects Jenkins to a Git repository to pull source code for the job.

**In build triggers:** Settings that determine when and how your job should be automatically started, such as scheduling builds or triggering on code changes.

* **Trigger Builds Remotely**: Start a build by sending a request to Jenkins from an external system or service.
* **Build After Other Projects are Built**: Automatically start this job after one or more other specified jobs have finished.
* **Build Periodically**: Schedule the job to run at regular intervals, like daily or weekly, using a cron-like syntax.
* **GitHub Hook Trigger for GIT**: Automatically start a build when a change is pushed to a GitHub repository, using a webhook.
* **Poll SCM**: Regularly check the version control system for changes, and start a build if new changes are detected.

**In build Environment:** Configuration options that prepare the environment before the build starts, like setting up variables or cleaning the workspace.

* **Delete Workspace Before Build Starts**: Remove any existing files from the workspace before starting the new build to ensure a clean slate.
* **Use Secret Text(s) or File(s)**: Access and use sensitive information, like passwords or API keys, securely during the build process.
* **Add Timestamps to the Console Output**: Include timestamps in the build logs to see when each part of the build process occurred.
* **Inspect Build Log for Published Build Scans**: Check the build log for detailed reports or scans that were published as part of the build.
* **Terminate a Build if It’s Stuck**: Automatically stop a build if it’s taking too long or seems to be hanging.
* **With Ant**: Configure the build environment to use Apache Ant, a tool for automating build processes, if you are using it in your project.

**In build steps:** The actual tasks Jenkins will perform to build your project, like compiling code, running tests, or deploying the application.

**In Post-Build Actions**: Actions that Jenkins should take after the build is completed, like sending notifications, archiving artifacts, or triggering other jobs.

1. **Jenkins Integration with Git and GitHub**

To build this project, we need to create a job that checkout this project executes the test cases and build this project.To checkout the project from the github we need github plugin.

Before installing the plugin make sure that github is installed on the local machine either it is cloud or not.(The latest version already installed github).

To install certain plugin, go to manage Jenkins, got to plugin and install the required plugin

To create the maven job, maven should also be installed where the Jenkins is hosted.

1. **Configure Jenkins to work with maven.**

Configuring Jenkins in tools is important because Jenkins is an **automation server** that helps manage tasks like building, testing, and deploying code automatically. To make this process smooth and effective, Jenkins needs to communicate with various **tools** (like Git, Docker, Maven, etc.). To do this, Jenkins needs to know where these tools are and how to use them.

Go to manage Jenkins- tools- edit the openjdk and give the path to where jdk is installed-edit maven and also give the path to where maven is installed

1. **Setup maven Project with Jenkins**

Maven: **Maven** is a powerful build automation and project management tool primarily for Java projects, which simplifies the process of managing project dependencies, building applications, and deploying software by using a standardized project structure and a pom.xml file to define project metadata, dependencies, and build configurations, allowing for easy integration and collaboration across development environments.

POM.xml: The pom.xml file in a Maven project is a **configuration file** that defines the project's metadata, dependencies, build process, and plugins required for the project, allowing Maven to manage the project lifecycle and automate tasks like compiling, packaging, and running the application.

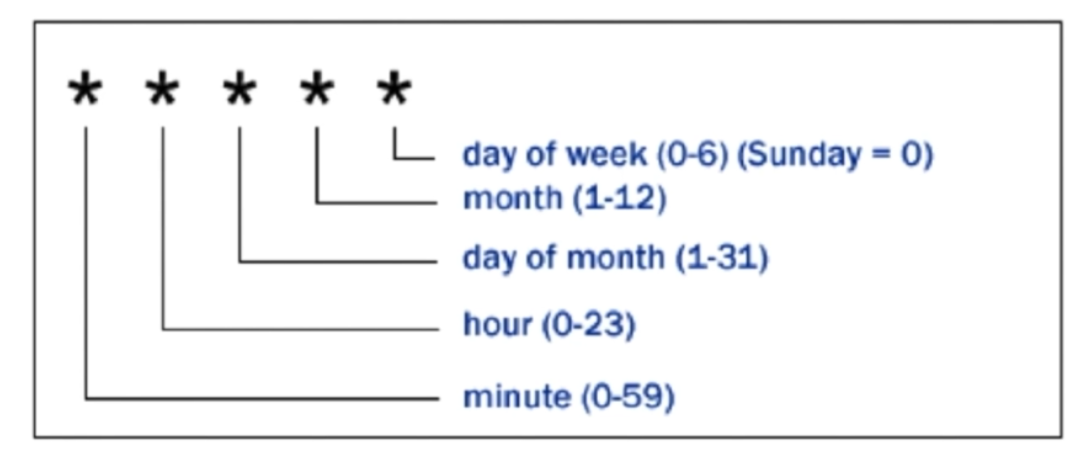
**Steps:**  
Go to new item- go to maven – copy paste the github repository- on build, add the path to pom.xml file and save it.-click o build now

1. **Source Code Polling**

**Source code polling in Jenkins** is a process where Jenkins periodically checks the version control system (e.g., Git) at specified intervals to detect changes in the source code repository, and if changes are found, it triggers a build automatically, ensuring continuous integration without manual intervention.

For that lets configure the job.

A white paper with black text

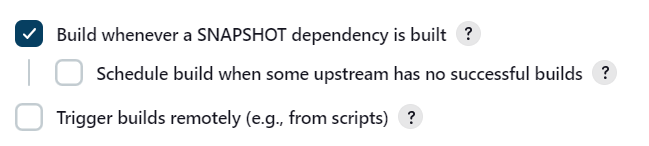
Description automatically generatedInside the build triggers,you change select on what basis the build should be triggered. If I choose poll SCM it periodically checks the source code repository (SCM) for changes based on a specified schedule, and if any changes are detected, it triggers a new build. This is configured using cron-like syntax to define the polling frequency.

1. **Remote Build Trigger**

Remote build triggers in Jenkins allow you to start a job remotely via an HTTP API by enabling the "Trigger builds remotely" option, using a unique token in the build URL (/build or /buildWithParameters), and ensuring security through proper authentication and HTTPS if exposed publicly.

User can call the build trigger by some script,api,or UI button click event

**Steps:**

* Go to the Jenkins job configuration and enable Trigger builds remotely in the Build Triggers section.
* Set an Authentication Token for secure access.I made my own TESTAUTH
* Use the trigger URL: http://<JENKINS\_URL>/job/<JOB\_NAME>/build?token=<TOKEN>.
* For parameterized builds, use buildWithParameters: http://<JENKINS\_URL>/job/<JOB\_NAME>/buildWithParameters?token=<TOKEN>&param1=value1.
* Test the URL in a browser or a script to ensure it triggers the job.
* Secure your Jenkins instance with proper authentication and HTTPS if exposed to the public.

1. **Archive Artifact in Jenkins**

Archive: **Archive** refers to the process of saving or storing specific files from a build outside workspace for future reference or use.

Artifact: **Artifacts** refer to the files or outputs that are produced during a build process, such as compiled code, binaries, JAR files, logs, or reports. These artifacts are essential files that result from running a build and are often archived for future reference, deployment, or sharing.

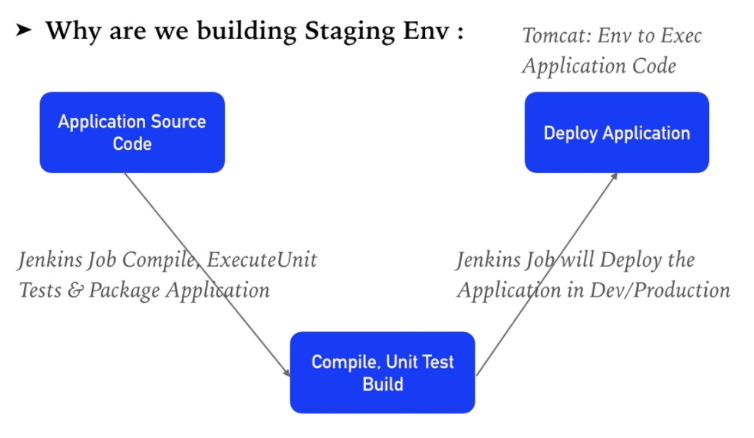
User can clean your workspace, run other builds and the file archived is safe

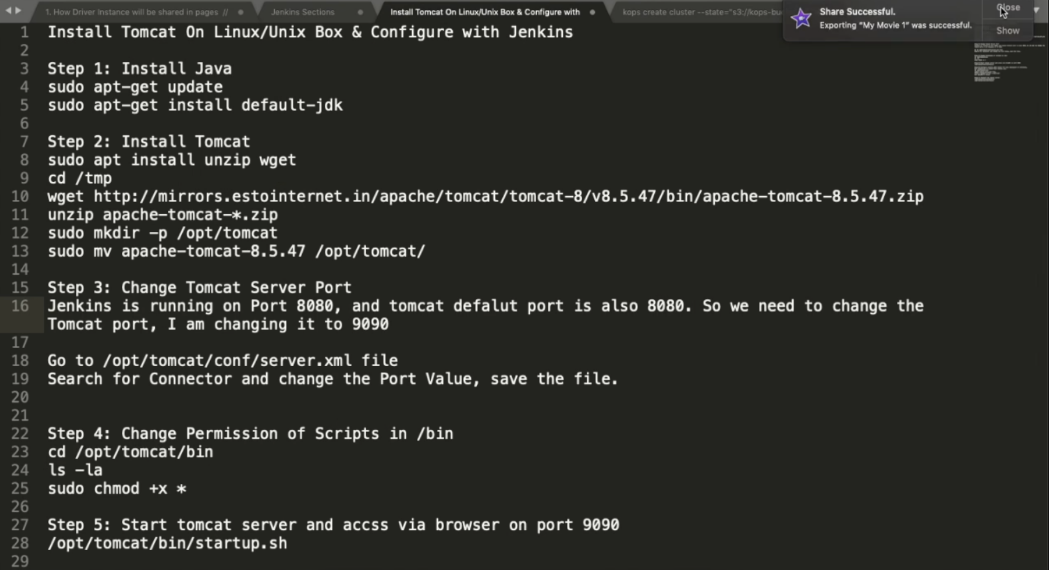
* A white rectangular object with a black line

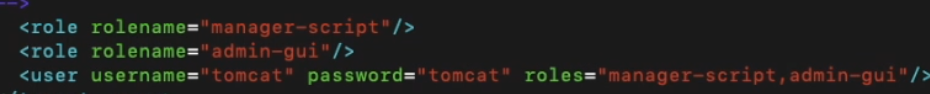
  Description automatically generated with medium confidenceFor this we need to edit the configuration and add post-build action :  
  the \*\*/\*.jar is :
* \*\*/ means "any directory or subdirectory."
* \*.jar matches all files with the .jar extension.  
    
  now after running the job, and go through the status,
* A screenshot of a computer

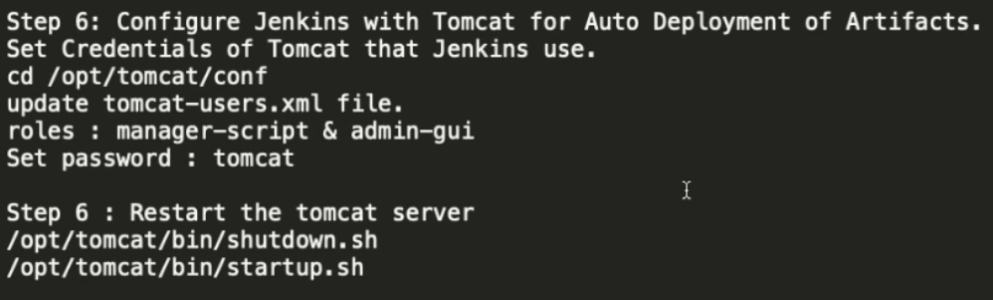
  Description automatically generated there will be a artifact at the top.

1. **Install and Configure Tomcat in Staging Environment**

Apache Tomcat is an open-source web server and servlet container that implements Java Servlet, JavaServer Pages (JSP), and other Java-based web technologies, allowing you to deploy and run Java web applications efficiently, often used for lightweight application hosting and development environments.

Steps to install Tomcat on linux :

A computer screen shot of a black background with purple text

Description automatically generatedAfter installing tomcat we need to configure the tomcat so that Jenkins can access the tomcat and deploy the application inside the tomcat.For that:  
  
move to the tomcat-user.xml file inside conf dir and edit it:  
  
to:  
  
 You need to **uncomment the role section in tomcat-users.xml** so that you can add users with specific roles (like admin). This tells Tomcat who is allowed to access Jenkins. Without it, Jenkins won't know which users can log in or have permission to use it.

1. **Deploy Staging Environment**
2. Create Job to produce Tomcat Deployable Artifacts.
3. Install Copy Artifact and Deploy to Containers Plugins

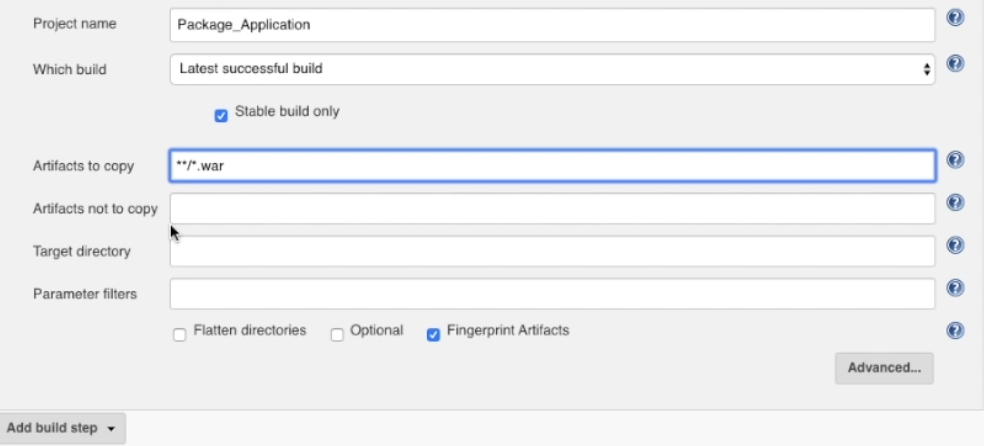
**Copy Artifact plugin**: This plugin lets you copy build artifacts (like files or results) from one Jenkins job to another, making it easier to share and reuse build outputs.

**Deploy to Containers plugin**: This plugin allows you to automatically deploy applications (like WAR files) to application servers (e.g., Tomcat, JBoss) after a build is complete, streamlining the deployment process.

1. Deploy the application to Staging Env

**Steps for Deploying on staging Env:**

1. Create a tomcat view on the right side of all on dashboard . It is made to group specific jobs only related to tomcat so that we wouldn’t be searching the individual jobs in the future.
2. Create a new “Freestyle Project” job and edit the following configuration:
   * 1. Discard old builds:5 days and 1 build.
     2. Inside source code management: checkout the code from git
3. Add some build environment: “Delete workspace before build starts”, “Add timestamps to the console output”
4. A screenshot of a computer

   Description automatically generatedIn the build step add:Invoke top level maven target since the github directory contains pom.xml  
   (we need to globally configure the “LocalMaven” like ive told you above to use the maven project.)
5. In the post build action: Archive the artifacts: \*\*/\*.jar and save the job.
6. After building the job it will generate the java servlet war file.
7. After this install the two plugins i.e. Copy Artifact and Deploy to Container
8. Create a new job that will deploy the artifact generated above.
9. Remain the configuration of the job as it is(default) but in “build” section , there is a step called “Copy the artifact from the other project”
10. Here it is said to copy the artifact from Package\_application, \*\*/\*.war means copy every artifact of the projects
11. In the “add post-build action” there is “Deploy war/jar to a container”
12. I need to provide the war file to deploy, the context path is :\
13. A screenshot of a computer

    Description automatically generatedIn add container add your tomcat version(tomcat9.x) it will then ask for the credential and the tomcat url
14. The credential is the same which we have edited in the tomcat-user.xml. Also don’t forget to add http:// before url  
    (We now have two job.. Package-manager: It is producing the artifact …Deploy App: It is copying the artifact and deploying in tomcat)
15. Now edit the configuration of first job and add the post build action and add “build other project”

A screenshot of a computer

Description automatically generatedIt will automatically trigger once the package application job executes.

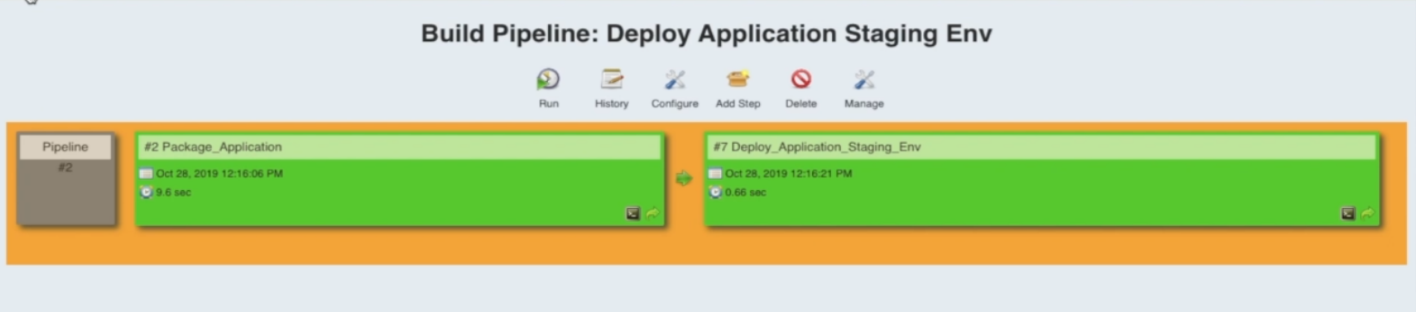
1. In build triggers add POLL SCM : \* \* \* \* \* so that the job executes every minute.
2. **Build Pipeline Plugin:**

The **Build Pipeline plugin in Jenkins** provides a visual representation of the build process, enabling sequential job execution, monitoring of stages, and manual approvals, making it easier to manage and troubleshoot complex build workflows.

The **Build Pipeline plugin in Jenkins** shows a visual map of the steps in your build process, making it easy to see each stage, trigger jobs one after another, and track success or failure, helping you manage the entire build workflow smoothly.

Pipeline: A **Pipeline in Jenkins** is a set of automated steps to build, test, and deploy code, defined in a script (Jenkinsfile), which allows continuous integration/continuous delivery (CI/CD) by managing complex workflows, supporting multiple stages, parallel execution, error handling, version control, and easy customization for repeatable and reliable software development processes.

Steps:

1. Create new view. You’ll see new option after installing the plugin “Build pipeline view”
2. Select the initial job which is package manager in case of above project.
3. After submitting you’ll see the view like this:
4. A screenshot of a computer

   Description automatically generatedIf you click on run it:  
   yellow means first job is in progress and blue means it is waiting for the previous operation to be complete.
5. **Deploy to Production.**

We will create a pipeline to deploy project in production.

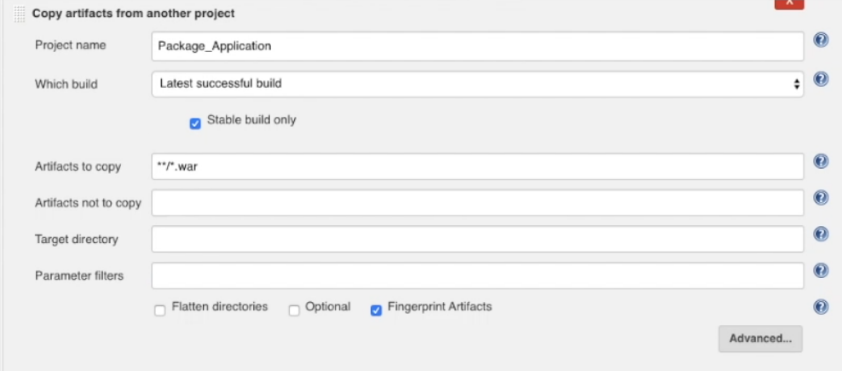
A blue rectangle with white text

Description automatically generatedSteps:

* 1. Shutdown the staging env of the tomcat and make the tomcat duplicate for production environment. Edit the connector of server.xml file of apache prod and change the port to 9091.(the port for them should be different)

A black background with white text

Description automatically generated

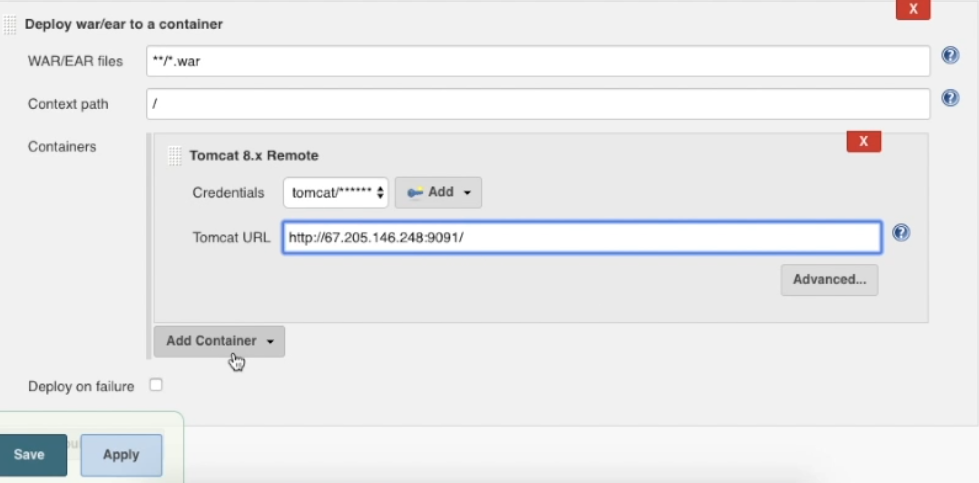
* 1. Start both the tomcat using startup.bat
  2. Create a job to deploy it on prod environment with some basic configuration.
  3. Add build step: Copy artifact from another project.
  4. The production should be dependent on the staging environment job. To make it go to the staging environment and manage configuration.
  5. A screenshot of a computer

     Description automatically generatedOn the “add post-build action” there’s and option called ‘build other project(manually)’ it doesn’t go automatically to the another job instead requires a manual interaction in order to host it in the production environment.
  6. Add another post build action to deploy war. :

\*\*/\*.war

Add the tomcat credentials.

Give the url of tomcat-prod env i.e. url of tomcat with port 9091.



The pipeline will now look like this:  
  
A screenshot of a computer

Description automatically generated

The package application will run itself cause poll scm is setup in it. Then it will go to the staging area automatically. But the prod env will not trigger.by hovering into the box you’ll see an icon to trigger on bottom right.in this way pipeline works.

1. **Infrastructure as Code.**

A white paper with black text

Description automatically generatedInfrastructure as Code (IaC) in Jenkins refers to the practice of managing Jenkins jobs, pipelines, and configurations as code rather than configuring them manually through the Jenkins UI. IaC allows these configurations to be stored, versioned, and maintained in a code repository, enabling teams to automate, standardize, and track changes across their Jenkins infrastructure.

1. **Jenkins Job DSL(Domain Specific Language)**

Jenkins DSL is a Groovy-based scripting language for automating, version-controlling, and managing Jenkins job configurations, enabling scalable, repeatable CI/CD setup through the **Job DSL Plugin**.

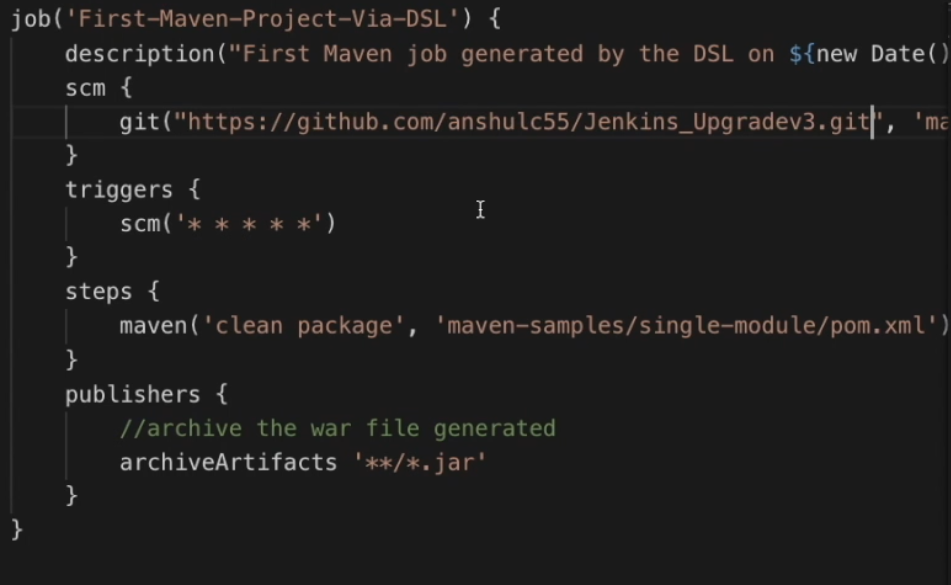
Jenkins DSL streamlines job management, promotes consistency, and reduces manual setup, ideal for scalable CI/CD workflows. However, it requires Groovy knowledge and may have limitations in flexibility compared to Jenkins Pipelines, especially for highly customized workflows.

You need to download the Job DSL plugin.

1. **Jenkins Job DSL with maven Project**

Seed Job: A seed job in Jenkins is a special job that pulls Jenkins DSL scripts from source control and executes them to automatically create or update other Jenkins jobs as defined in the scripts. To use the Job DSL plugin, you first need to create a seed job.

The seed job is a normal freestyle Jenkins job that you add the “process job dsl” build step.this step takes the dsl and generates the configured jobs.

Steps to create a job through DSL:  
1. Write a groovy scripts in VSCode with all the required configuration in it. (you can find the scripts in documentation)

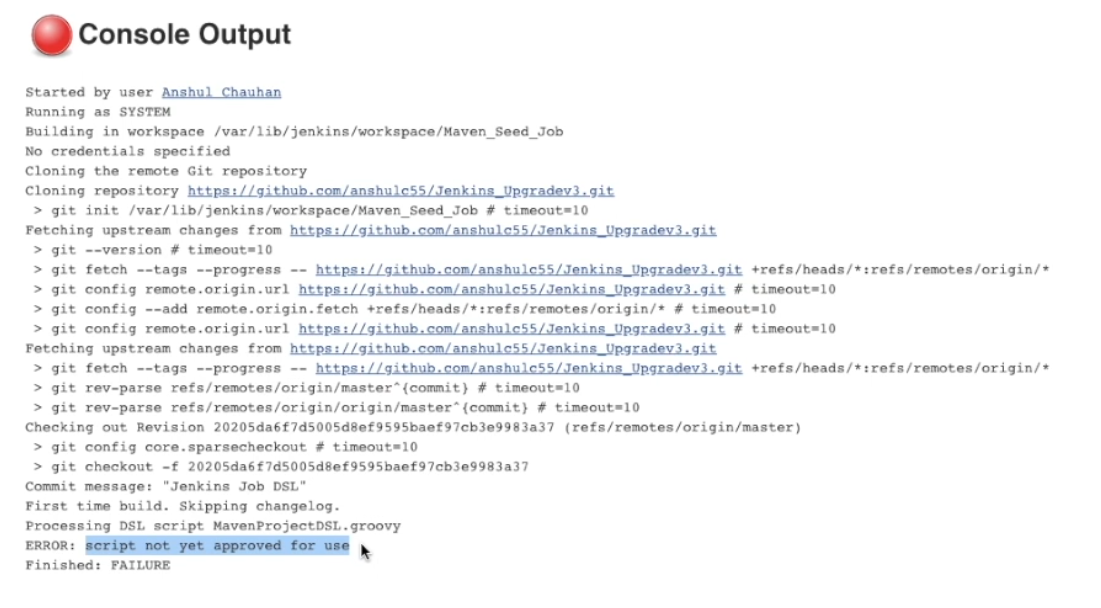
2. Create a freestyle Seed Job.

3. Configure by giving the github link consisting of the project along with the groovy file in it.

4. Inside the build option, there is an option called “Process job DSL”

A screenshot of a computer

Description automatically generated5. Inside DSL script we need to give path to the groovy file.

6. After running the seed job it will fail at first time because Jenkins doesn’t allow the script to execute until we approve it.

To fix this, go to manage Jenkins, there is an option called In-process approval which shows the pending approval.

7. Approve the script and run it again.

1. **Distributed Builds**

A white text on a white background

Description automatically generatedJenkins distributed builds, also known as Jenkins Master-Agent Architecture, allow Jenkins to run and manage builds across multiple nodes or servers, enhancing performance and scalability.

By using single node, it cannot handle multiple jobs which leads to system failure

**Jenkins Master**: Your main Jenkins server is the master machine. The tasks performed by the master are= scheduling build jobs, dispatching builds to the slaves for the execution, monitor the slaves

**Jenkins slave**= A slave is a java executable that runs on a remote machine.

It hears requests from the Jenkins master interface.

Slaves can run on a variety of operating system.

A diagram of a computer network

Description automatically generated

1. **Attach and configure Jenkins slave**

For this we need public ip of the master and slave.

Master node will start the slave agent on slave machine via SSH.

Steps:

1. Setup a slave server with Jenkins installed in it.
2. Create a dedicated user for Jenkins on the slave server. By creating a specific user just for Jenkins (jenkins\_slave), we limit what Jenkins can do on the slave server. This user will have only the permissions it needs, which reduces the risk of accidental changes or security breaches affecting other parts of the system.  
   sudo adduser jenkins\_slave
3. Set a password for this user and follow the prompts. Setting a password ensures that only someone with the correct credentials (username and password) can log in as this user, adding a layer of security.  
   sudo usermod -aG sudo jenkins\_slave
4. Configure SSH for Secure connection. To allow the Jenkins master to connect securely to the slave server, you need to set up SSH key-based authentication.
5. Generate ssh key pair on the master.SSH into your Jenkins master server and switch to the Jenkins user. The jenkins user is automatically created when Jenkins is installed on the master server. This user account is set up specifically for running Jenkins and managing Jenkins-related files and processes.

sudo su – jenkins

ssh-keygen -t rsa -b 2048 -f ~/.ssh/id\_rsa  
The command generates a new RSA SSH key pair (a private and a public key) with a key length of 2048 bits, saving the private key to ~/.ssh/id\_rsa and the corresponding public key to ~/.ssh/id\_rsa.pub, which is used for secure, password-less authentication between servers.

1. Copy the ssh public key to the slave server

ssh-copy-id -i ~/.ssh/id\_rsa.pub jenkins\_slave@<slave-server-ip>

1. Access the Jenkins dashboard on the master server..goto manage Jenkins> Manage nodes and clouds>click on new node and give it a name>Select permanent agentand ok.
2. Configure the node details.:  
   add a description for this slave node> Enter the home directory of the Jenkins\_slave user(e.g. /home/Jenkins\_slave)
3. Select the no of executor on the machine:  
   Jenkins executor: A **Jenkins executor** is a component of Jenkins that is responsible for running tasks on a specific node (whether it's the master or a slave).

By default,the Jenkins master node has one executor that can run one job at a time.when you add a slave node,you can assign multiple executors to it.allowing Jenkins to run multiple tasks in parallel on that node.

1. Then on launch method theres an option called launch via execution of command on the master  
   there you ssh the ip of the slave server   
   to execute it write this full command:  
   ssh root@<ip> java -jar /root/bin/slave.jar
2. Save the node configuration.
3. **A screenshot of a computer

   Description automatically generatedConcurrent build Execution**

In Jenkins, **concurrent build execution** refers to the ability to run multiple builds simultaneously. This is crucial in projects where you need to run various tasks in parallel, such as compiling different modules or running tests across multiple environments. To enable this, Jenkins uses a **master-slave architecture**.

For that we need to have master slave architecture. Inside a view create some jobs and we can execute it in parallel way. We can also label it using label property during configuring slave and using the label in the job configuration in order to run the specific job by the slave with same label in it.