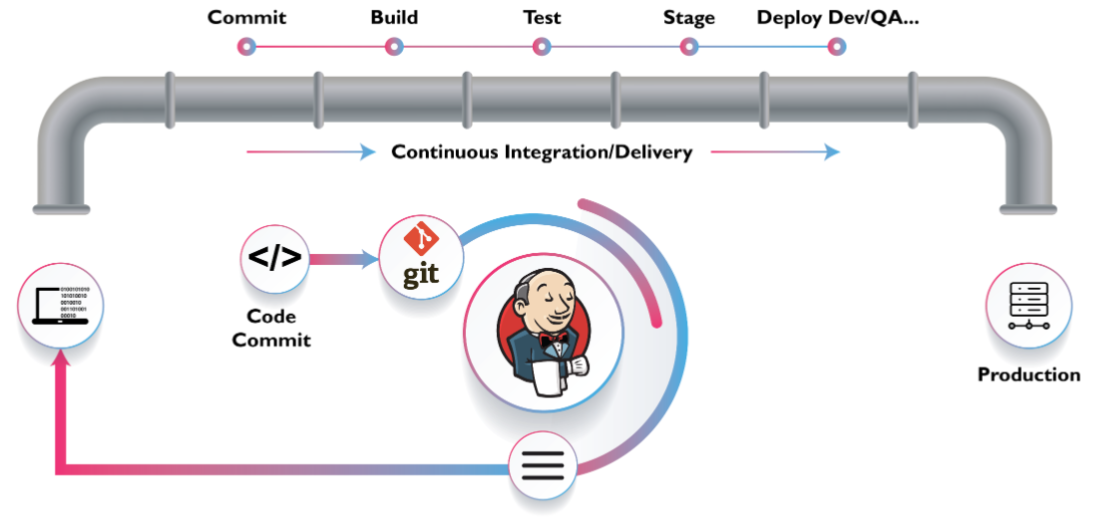
**Jenkins**

**Jenkins:** It's a continuous integration automation server, used to automate repetitive tasks and also for continuous integration.

Jenkins is an open-source automation server that helps automate parts of the software development process, such as building, testing, and deploying code. It's widely used for continuous integration and continuous delivery (CI/CD).

Roles of Jenkins in Devops :



**Essential Requirements for Installing Jenkins & Step-by-Step Guide to Installing Jenkins**

Run the commands below on your EC2 instance.

* **Update patches on Linux Machine:**

yum -y update

* **Install git:**

yum install -y git

* **Install Java:**

yum install -y java-21-amazon-corretto

yum install -y java-21-amazon-corretto-devel

[root@ip-172-31-41-161 ~]# java --version

openjdk 21.0.5 2024-10-15 LTS

OpenJDK Runtime Environment Corretto-21.0.5.11.1 (build 21.0.5+11-LTS)

OpenJDK 64-Bit Server VM Corretto-21.0.5.11.1 (build 21.0.5+11-LTS, mixed mode, sharing)

[root@ip-172-31-41-161 ~]#

[root@ip-172-31-41-161 ~]#

[root@ip-172-31-41-161 ~]# which java

/usr/bin/java

[root@ip-172-31-41-161 ~]#

[root@ip-172-31-41-161 ~]#

[root@ip-172-31-41-161 ~]# readlink -f /usr/bin/java

/usr/lib/jvm/java-21-amazon-corretto.x86\_64/bin/java

* **Install Maven:**

cd /opt

sudo wget https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz

sudo tar xvf apache-maven-3.9.9-bin.tar.gz

ls -ltr

cd

echo "export M2\_HOME=/opt/apache-maven-3.9.9" >> .bash\_profile

echo "export M2=$M2\_HOME/bin" >> .bash\_profile

echo "export PATH=$M2:$PATH" >> .bash\_profile

source ~/.bash\_profile

echo $M2\_HOME

* **Check versions of installed tools:**

git --version; java --version; /opt/apache-maven-3.9.9/bin/mvn --version

=============

* **Install Jenkins:**

sudo wget -O /etc/yum.repos.d/jenkins.repo \

https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

sudo yum upgrade

sudo yum install -y jenkins

sudo systemctl daemon-reload

service jenkins start

chkconfig jenkins on

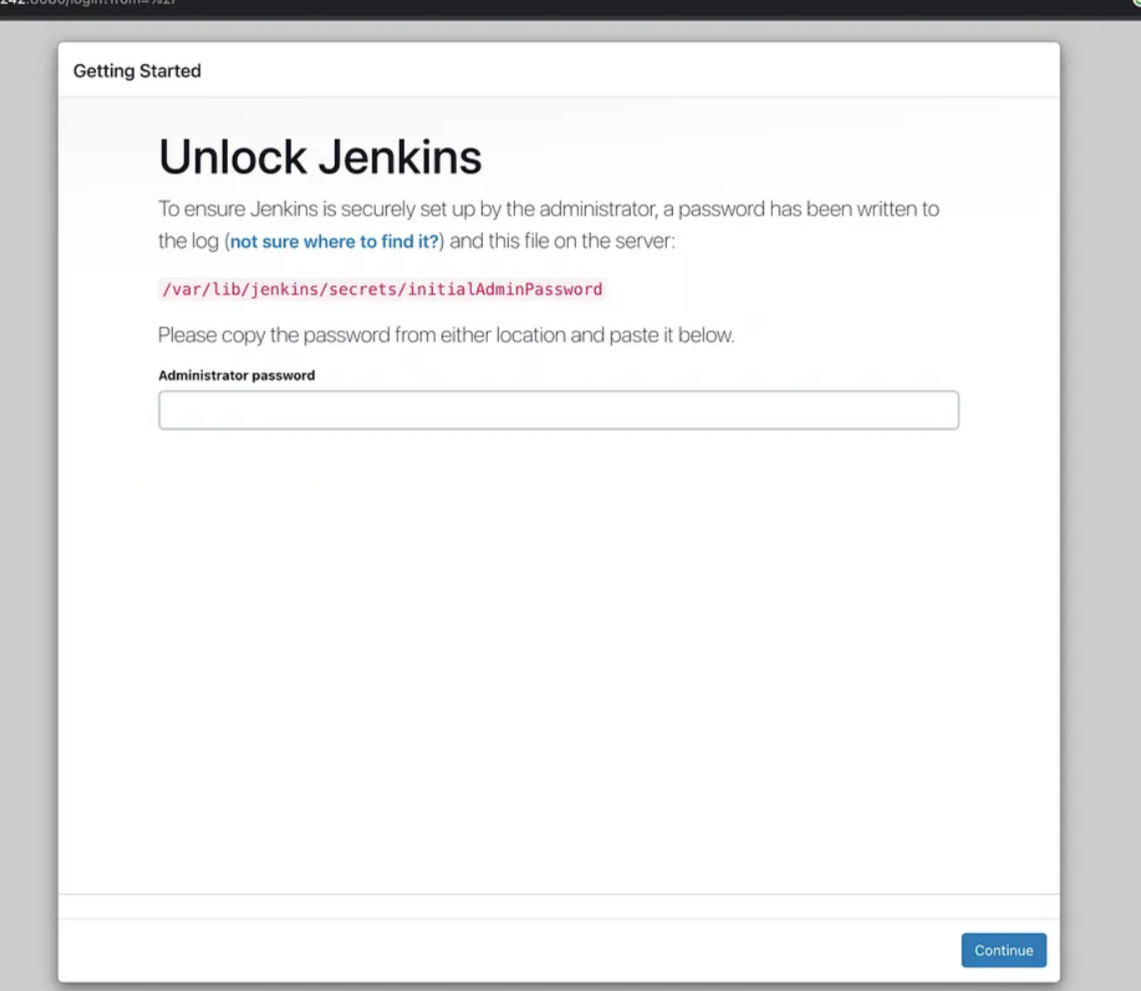
***Once the Jenkins installed on machine, follow the steps below to configure the Jenkins.***

* Open google chrome and type 15.206.165.149:8080 (Public IP address of your ec2 instance)

***Note : In your ec2 instance security group you have to whitelist the Port 8080 because Jenkins use Port 8080***

***Ensure your security groups are configured to allow traffic through port 8080. This is the default port Jenkins uses.***

* You will see a setup wizard and be prompted to enter the administrator password.

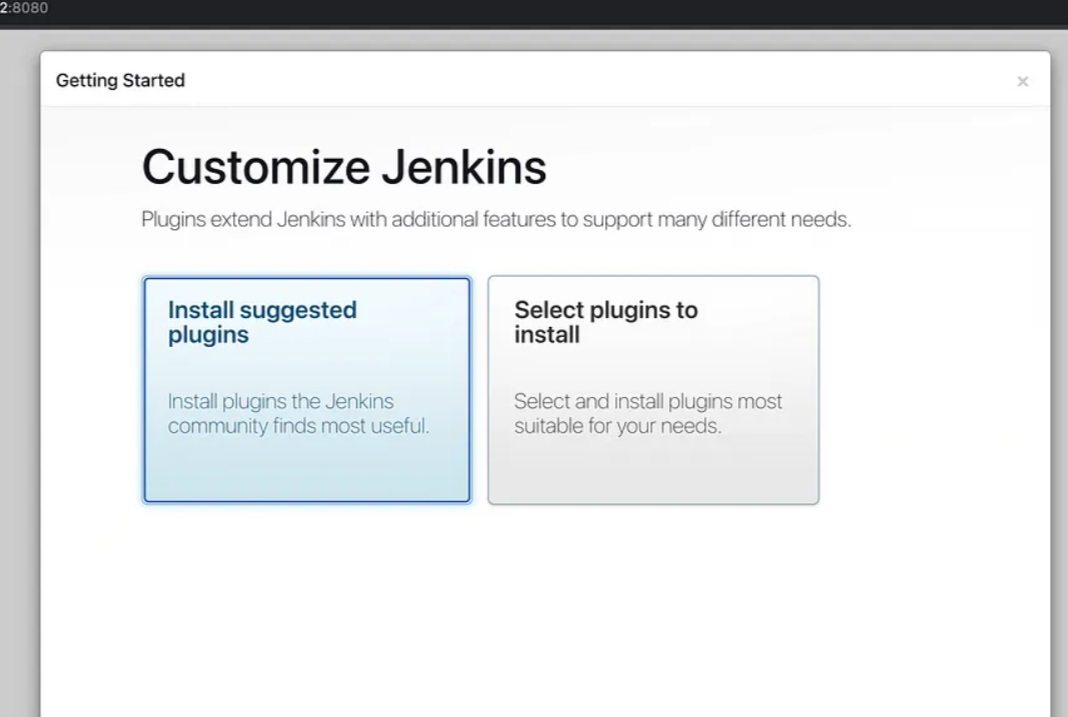


Retrieve the password with the following command:

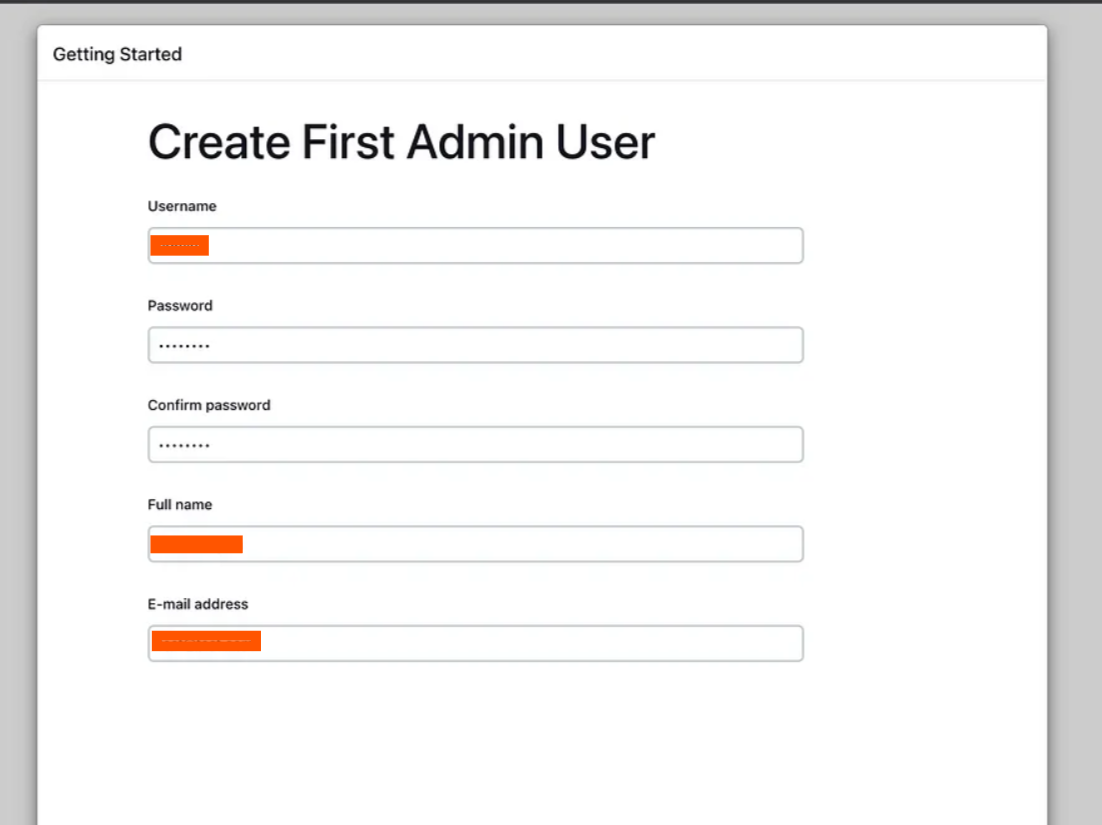
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Copy the password and paste it into the Jenkins setup wizard to continue.

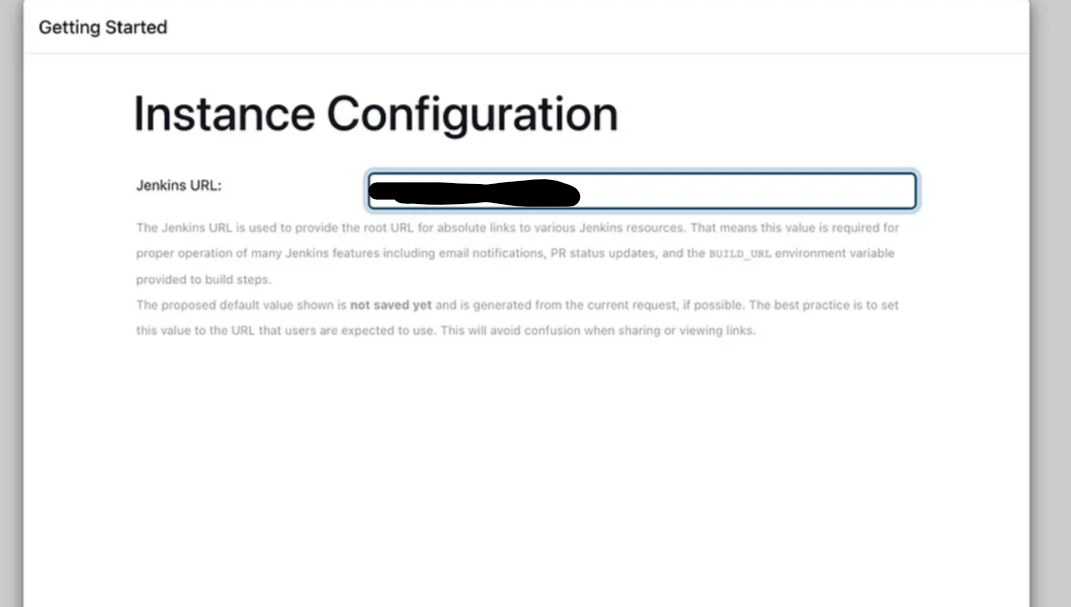
* Once done click on the install Plugins.



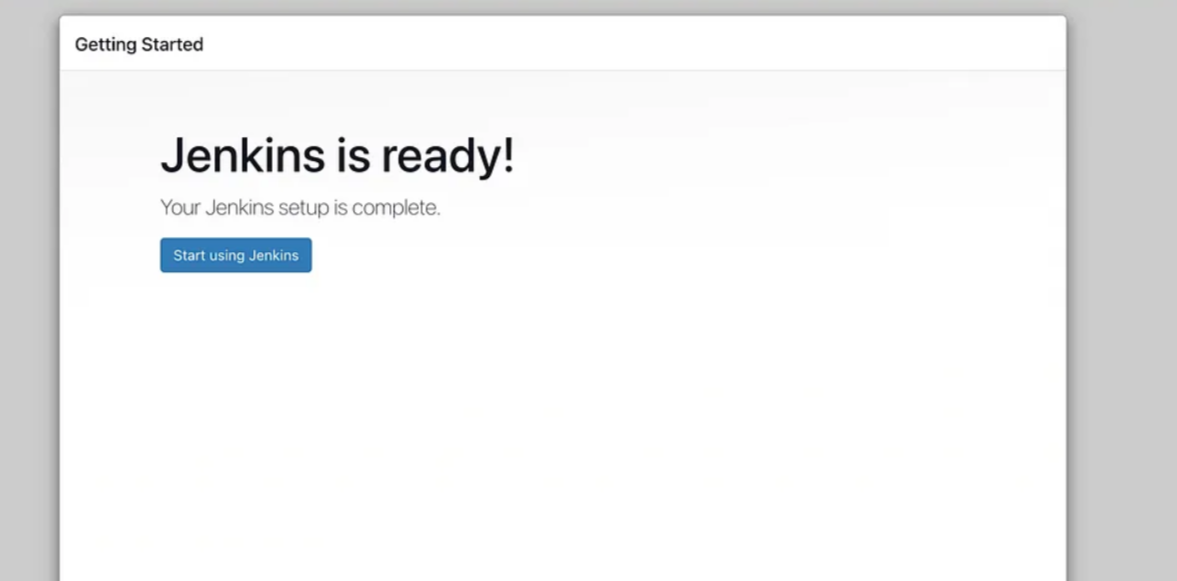
* Create the Admin user and click on save and continue.



* Confirm the Jenkins URL.  It should be set to your instance’s IP.



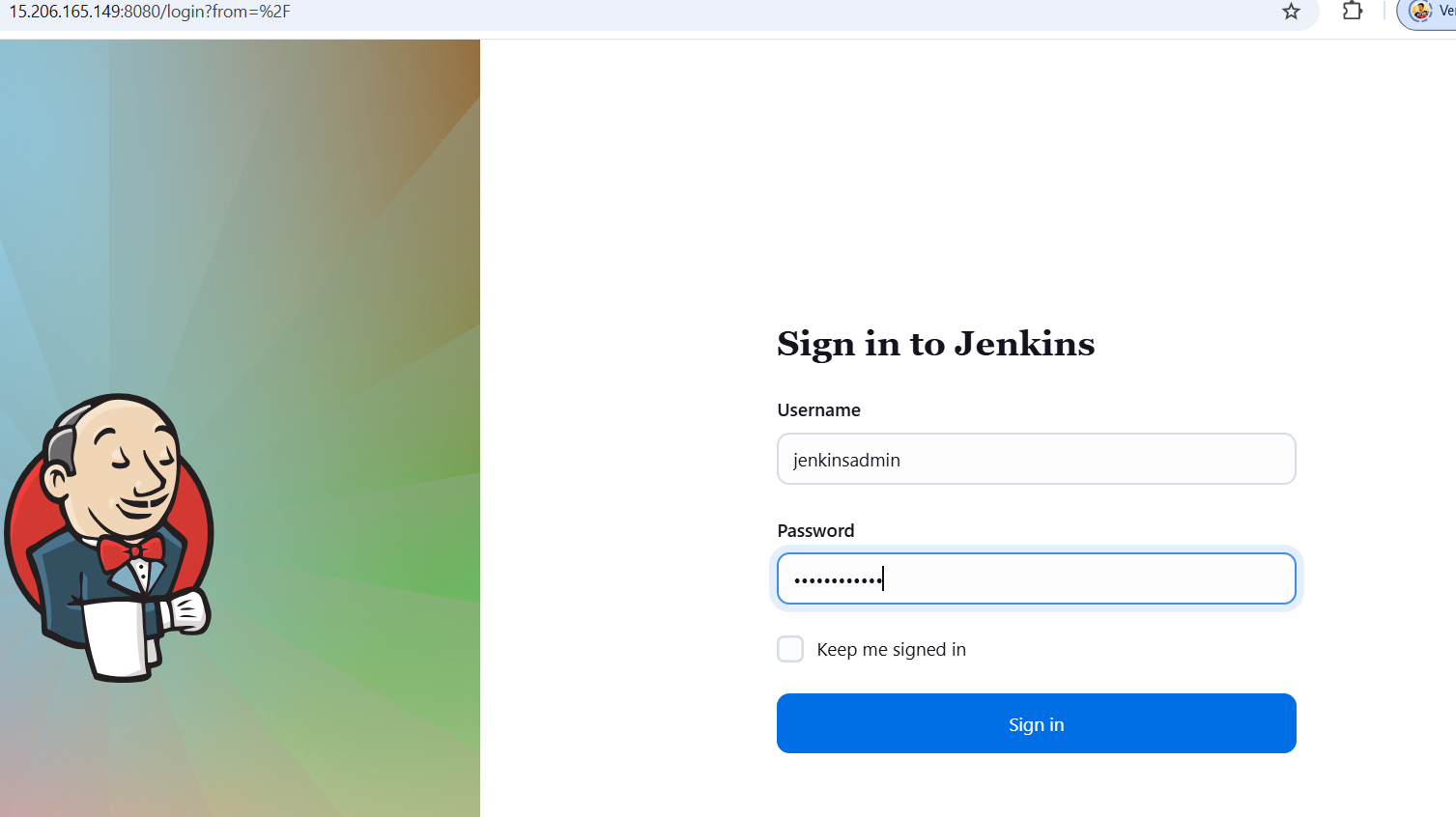
* Once done, start using Jenkins.



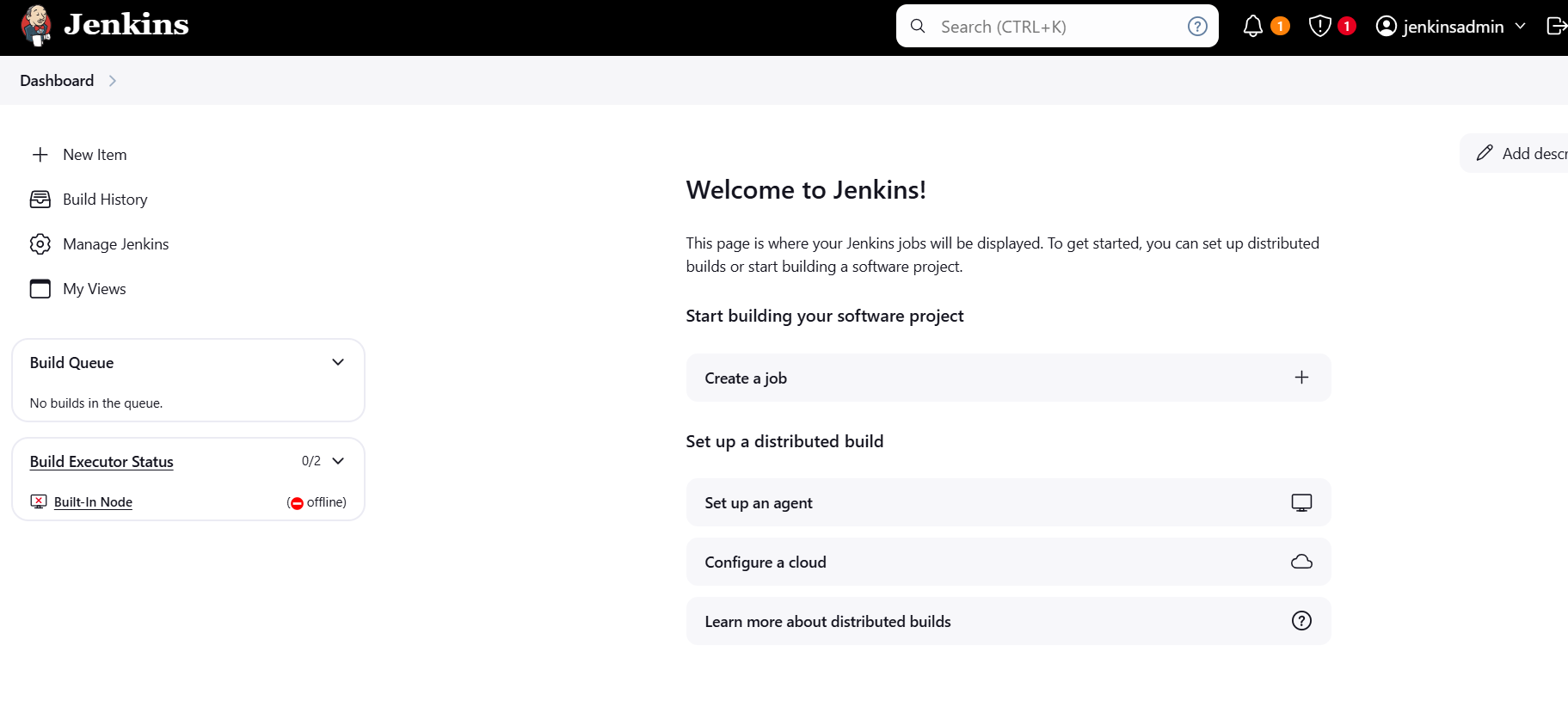
***Once the initial configuration done, you have to setup path for Java, Git and Maven in Jenkins.***

***Follow the steps below for the same.***

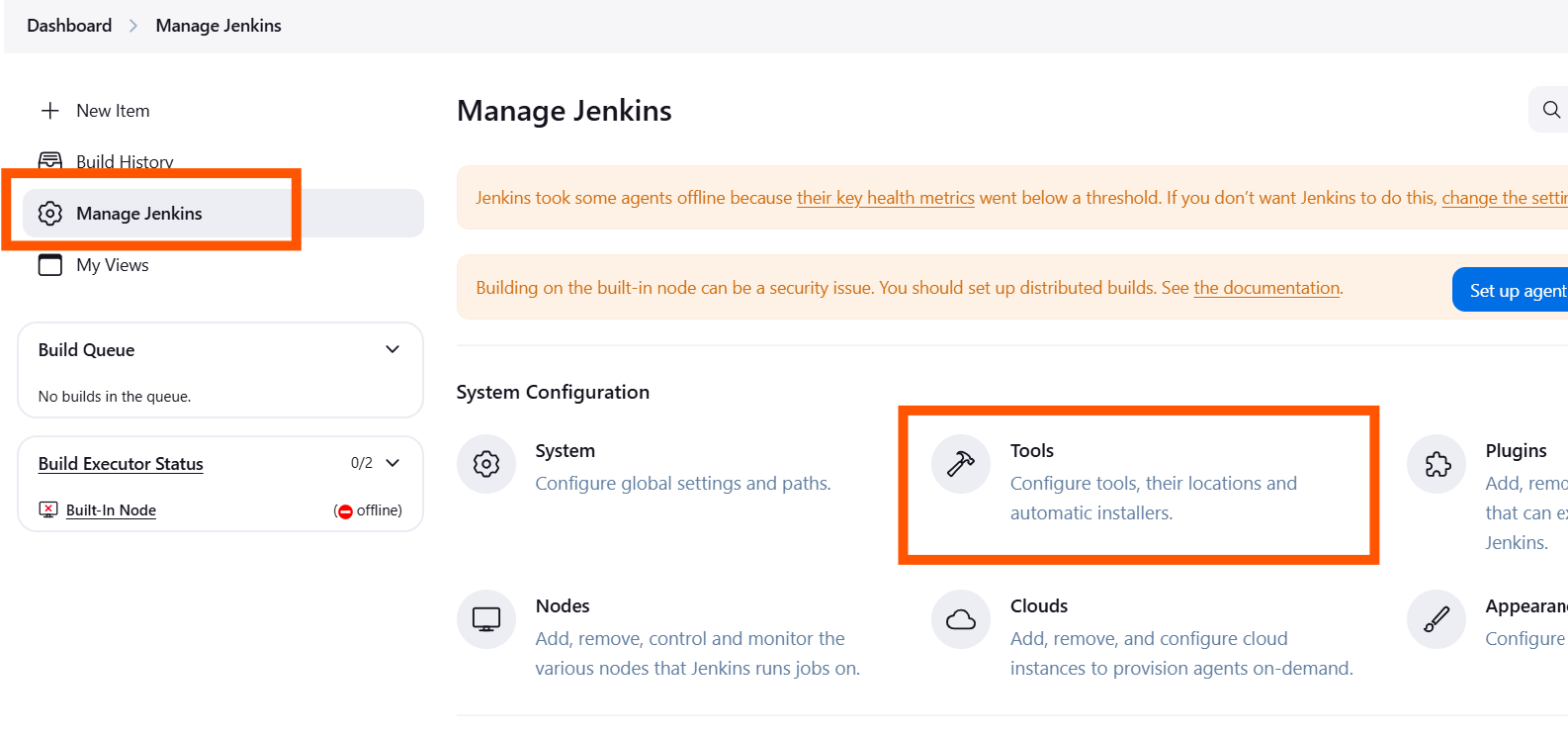
* **Login to Jenkins with username and password.**

****

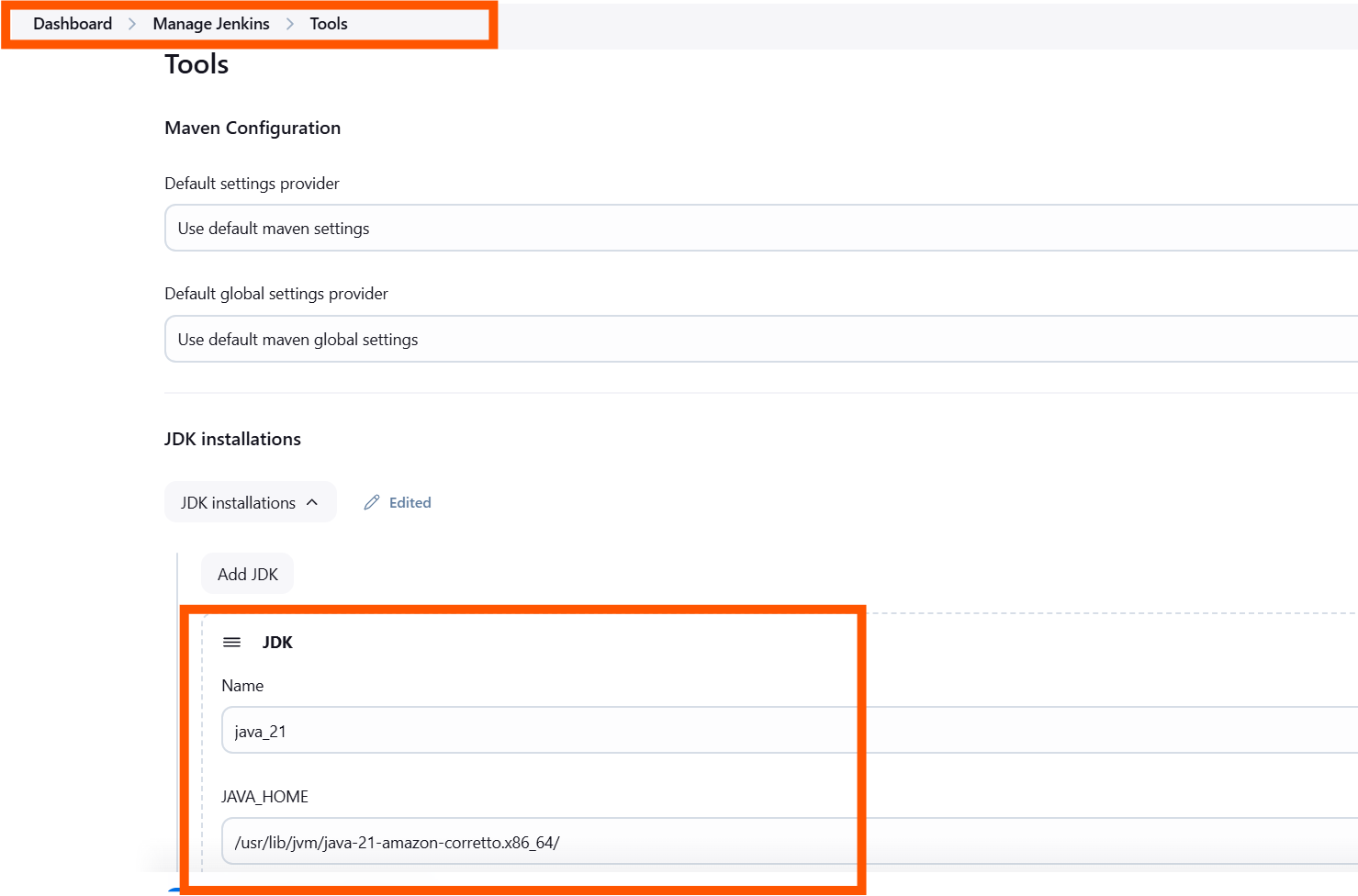
* Once you log in, Jenkins starting page will appear.

****

* Click on manage Jenkins and in System Configuration, Click on Tools.



* You need to specify the Java, Git, Maven installation path in tool.



Java : /usr/lib/jvm/java-21-amazon-corretto.x86\_64/

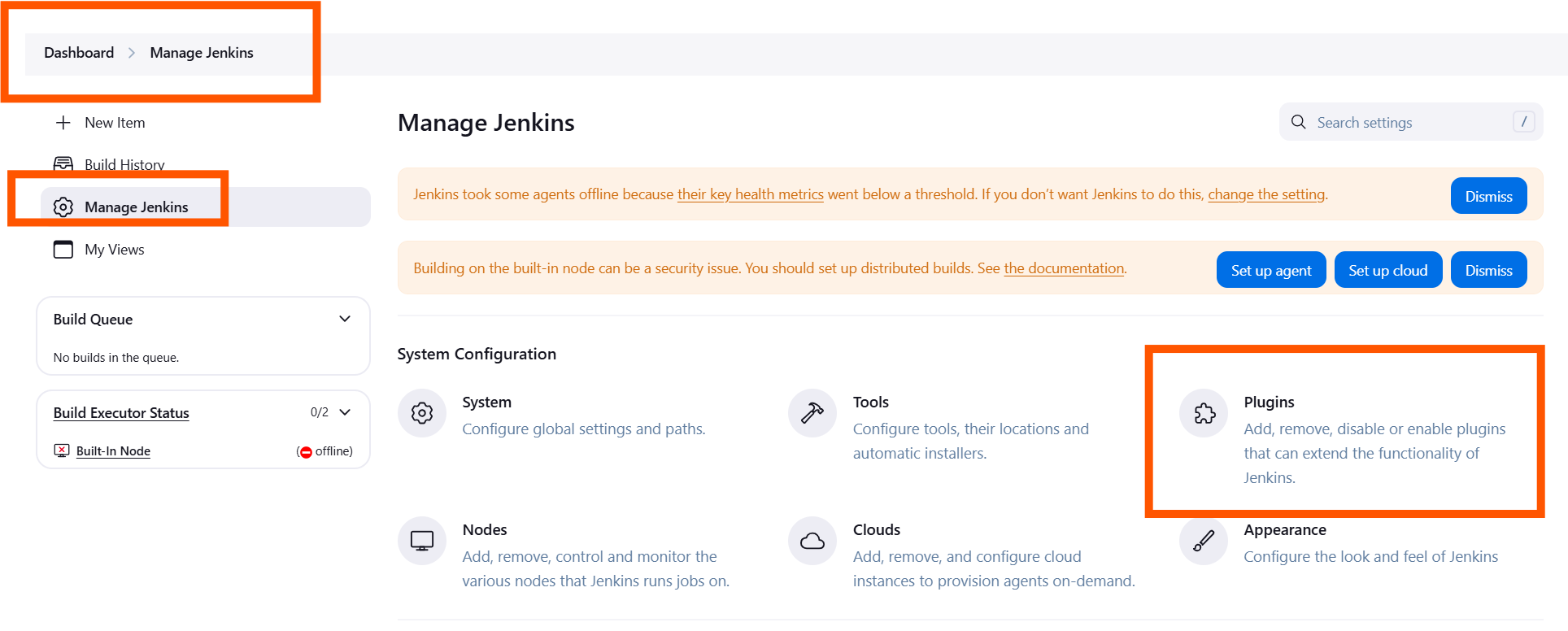
Git : /usr/bin/git

Maven : /opt/apache-maven-3.9.9

Once done click on save button.

* Install Plugins:

You can also install plugin in Jenkins as per your requirements.

****

**CI: (Continuous Integration):**

Continuous Integration (CI) is a software development practice where developers regularly merge their code changes into a shared repository, often multiple times a day. Each merge is automatically tested and verified, helping to detect and fix errors quickly. This ensures that the codebase is always in a deployable state and speeds up the development process.

**Example:**

Let's suppose I am a developer. I will create a feature or local branch from the dev branch, pull the latest changes from the dev branch, and start writing code in my local branch. Once I'm done writing the code, I will compile, package, and test it on my local system using maven if I am writing java code. Once testing is complete, I will push the code to the central repository (GitHub) from my local system.

Whenever I commit the code, the Jenkins pipeline will automatically trigger. It will clone the source code and fetch the code from the central repository (GitHub), then perform CPR (code compile, package, and run the code). Once done, an artifact will be created based on what we’ve specified in the pom.xml (either WAR or JAR). If we have specified a WAR file to be created in the pom.xml, then a WAR artifact will be created and stored in the artifact repository (such as Nexus or JFrog).

**Benefits of CI (Continuous Integration):**

***Early Detection of Errors*:** Bugs and issues are identified early in the development process, making them easier and cheaper to fix.

***Improved Collaboration:*** Encourages collaboration among team members, as changes are integrated and tested frequently.

***Increased Productivity:*** Automation of builds and tests reduces manual efforts, allowing developers to focus on writing code.

***Higher Quality Code:*** Automated tests and frequent integration ensure that code quality remains high.

**Jenkins Component:**

**Build Triggers:** Build triggers takes the responsibility of triggering Jenkins job.

1. **Build periodically:** Using the "Build periodically" trigger in Jenkins, you can schedule job to run at specified times, much like a cron job.

If you want any job to run at any certain frequency, you can use build periodically.

You have to specify cron tab in Build periodically.

1. **Poll SCM:**

Cron tab need to specify.

In poll scm, the time you have specified, the Jenkins job will go and check the GitHub repo we have specified, and if there is new commit, then it will download the commit only.

Job will poll/check the repository specified time (cron expression we specified at the time of job creation), if there is a new commit in GitHub repository then the job will run otherwise not.

1. **GitHub hook trigger for GITScm polling :**

GitHub hook trigger for GITScm polling" is a way to automatically check for new updates or changes in a GitHub repository.

Here’s how it works:

1. GitHub Webhook: A webhook is like a notification system. It tells other systems when something new happens on GitHub, like a new commit or a pull request.
2. GITScm Polling: This is a process where a system (like Jenkins, for example) checks (or "polls") the GitHub repository at regular intervals to see if there are any new changes, like code updates.

So, when you combine these two things, a GitHub webhook can "trigger" a job in your CI/CD tool (like Jenkins) to check for updates (poll) on GitHub and run the necessary tasks if there are any changes.

In short: It's a way to automatically keep track of updates from GitHub and trigger actions in another system when those updates occur!

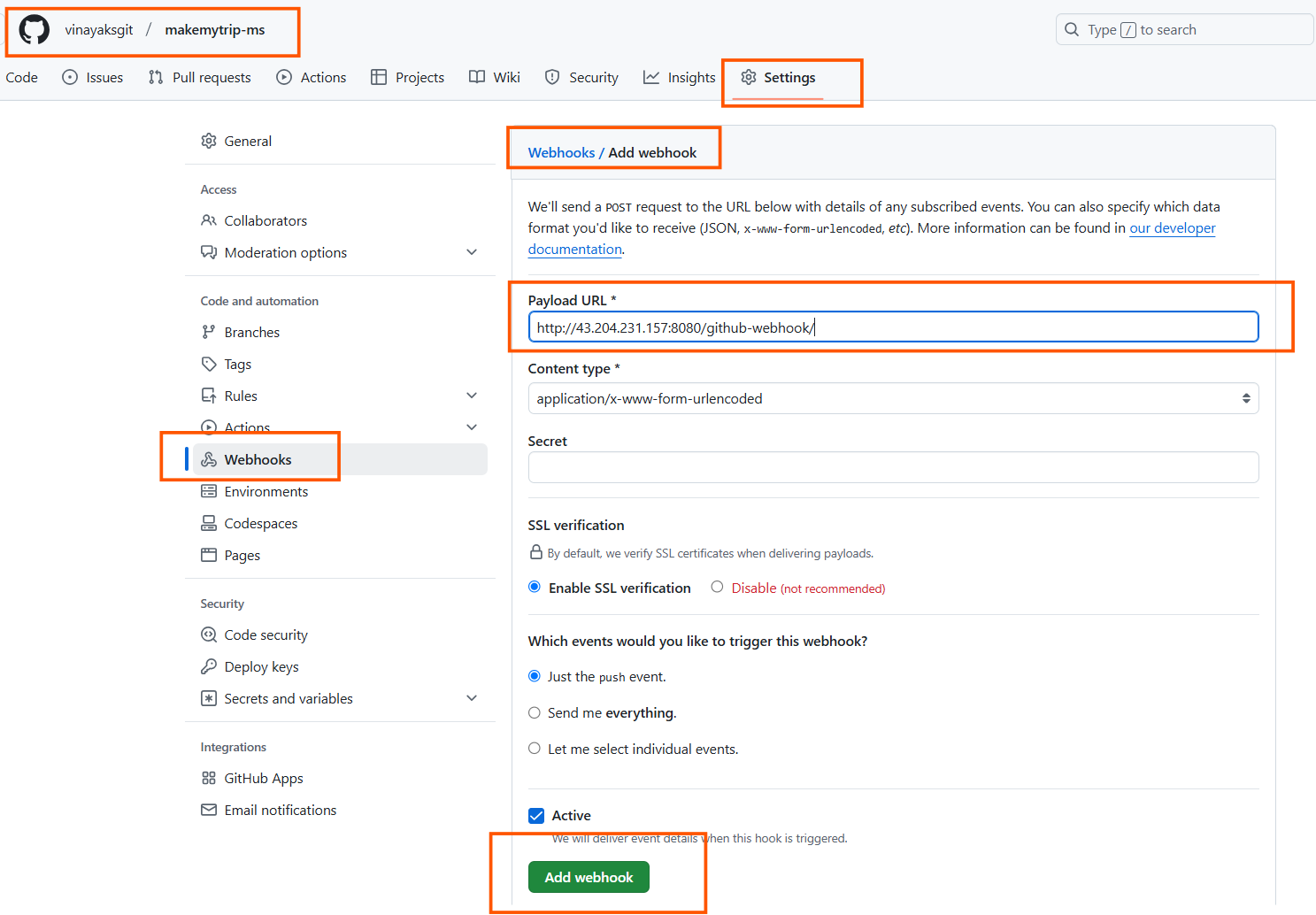
Once we made changes in specified branch in Job, Webhook will trigger the job.

Any changes in specified branch, will trigger the pipeline automatically.

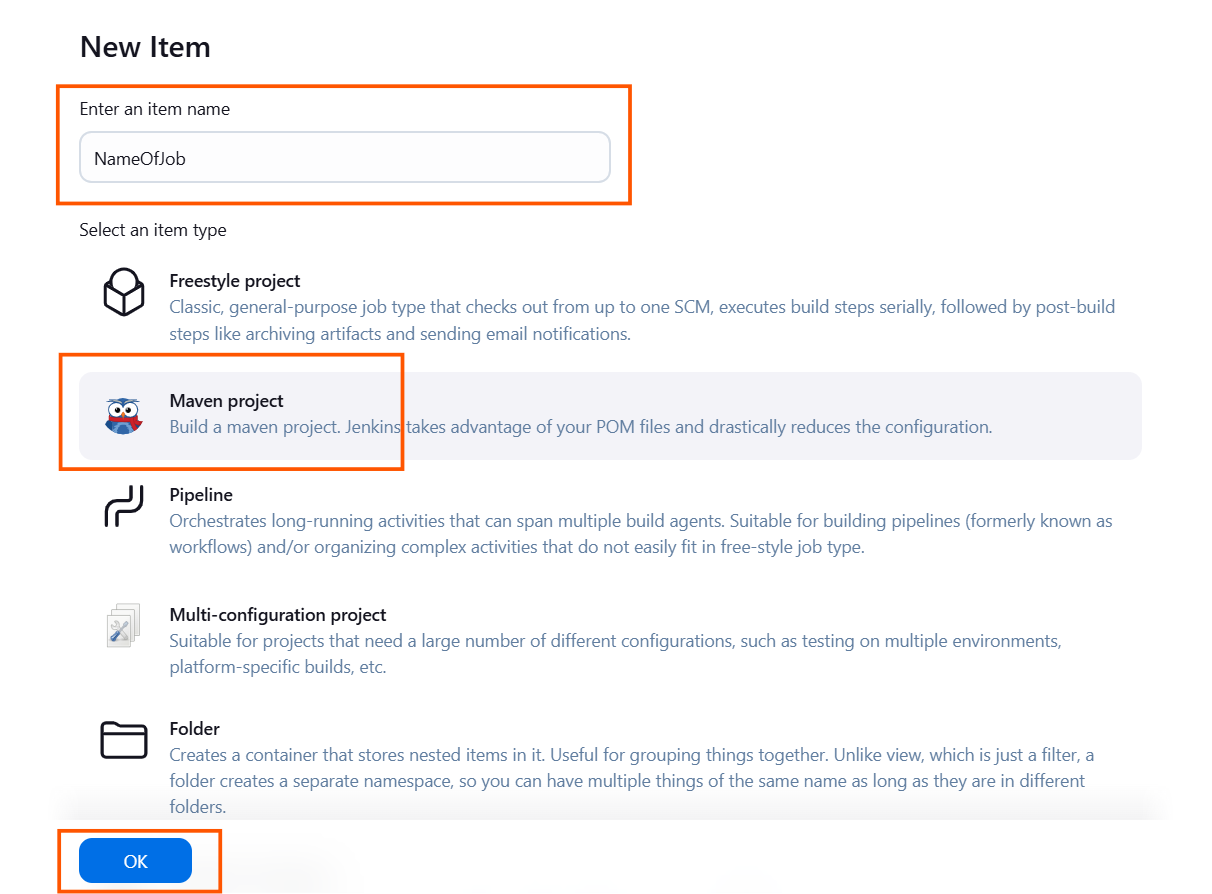
Its similar like poll scm, but since we are added webhook, code will be push to Jenkins.

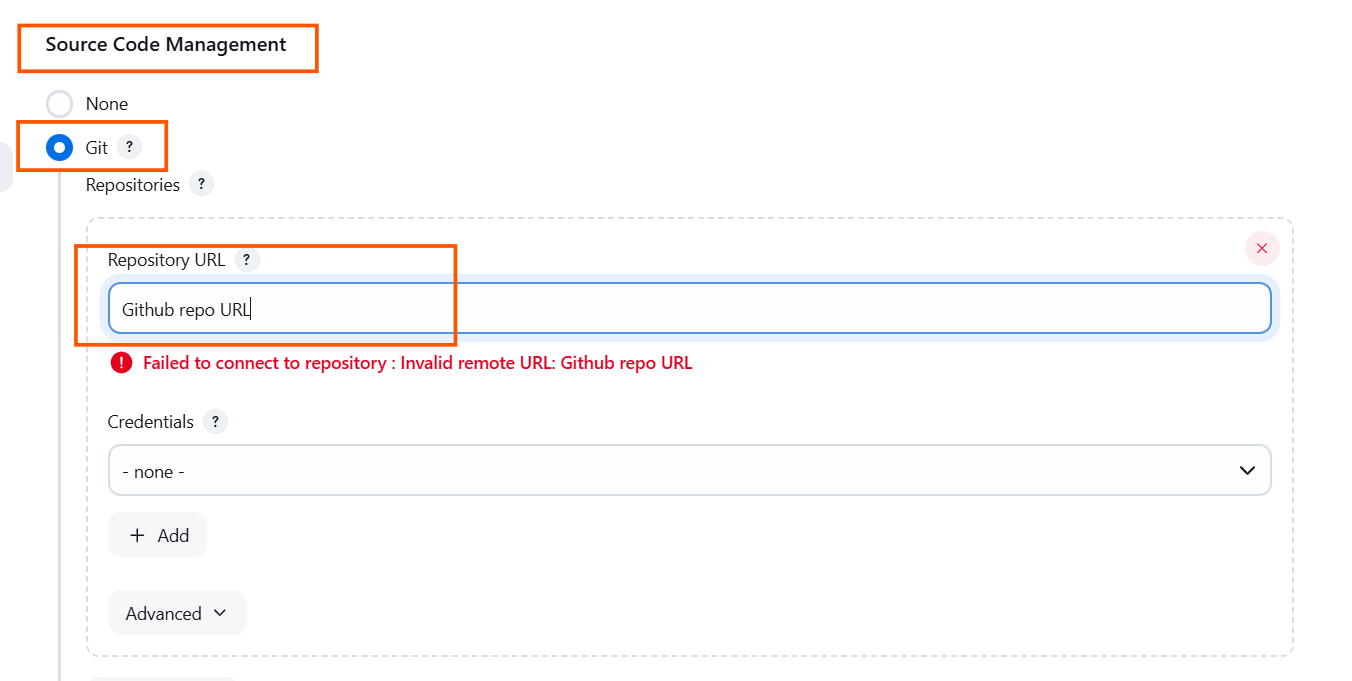
**Steps :**

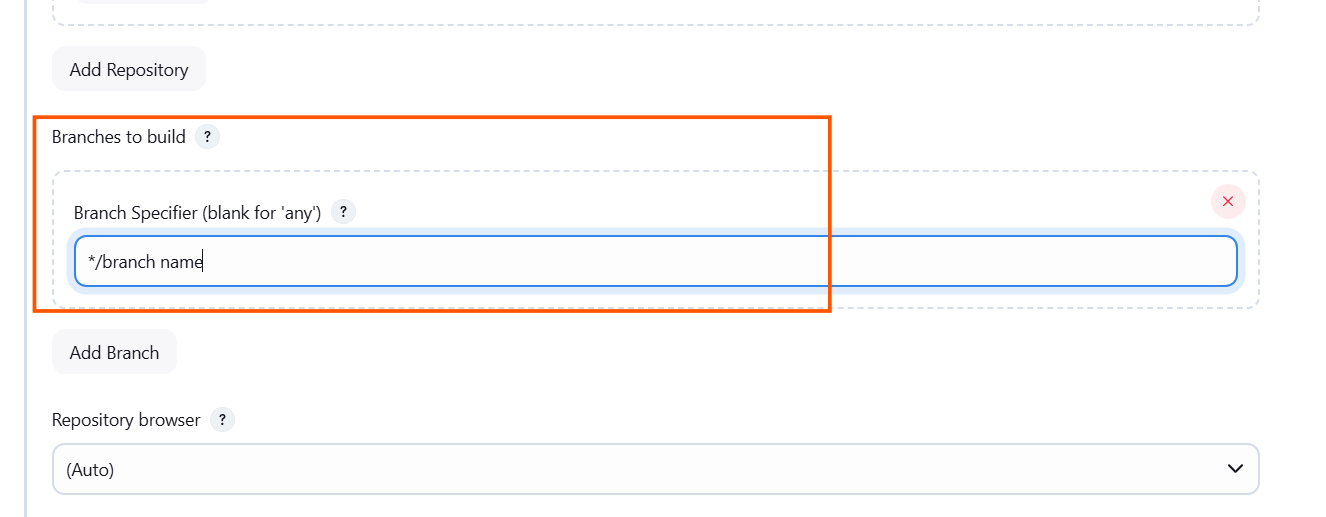
* **Create Webhook**

****

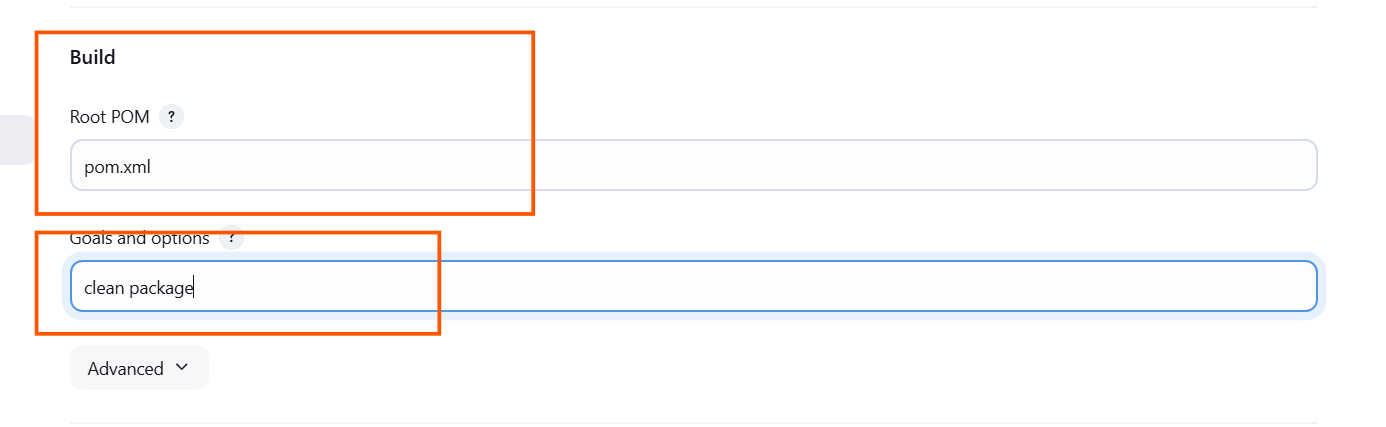
* Create Job in Jenkins:











Once these details are specified, click on save.

Jenkins Master Slave Architecture:

1**. Jenkins Master**

The Master is the main server that runs Jenkins.

It handles everything like scheduling jobs, managing settings, and providing a web interface where you can create and manage builds.

The Master doesn't always do the work itself; it tells other computers (called Slaves) to do the actual work.

2. **Jenkins Slave (Agent)**

The Slaves are other computers or machines connected to the Master.

These Slaves actually run the tasks (like building code or testing).

The Master assigns work to the Slaves when needed, and the Slaves send the results back to the Master.

**How It Works:**

You create a job in Jenkins (like "build this project").

The Master looks at all the available Slaves and gives the job to one of them.

The Slave runs the job, and then reports the result back to the Master.

This way, the Master doesn’t get overloaded, and Slaves can work on multiple jobs at once.

**Why Use This Setup?**

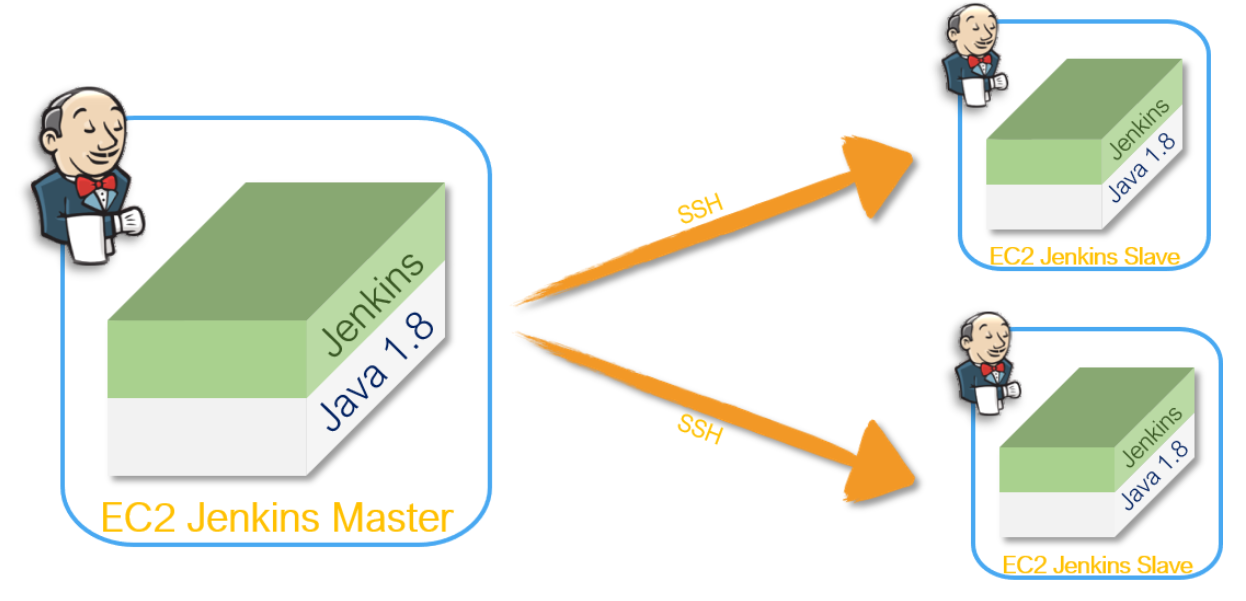
**Faster Builds**: More computers working together means Jenkins can handle more jobs at the same time.

**Flexibility**: You can set up different machines with different environments (e.g., Windows, Linux) to run specific jobs.

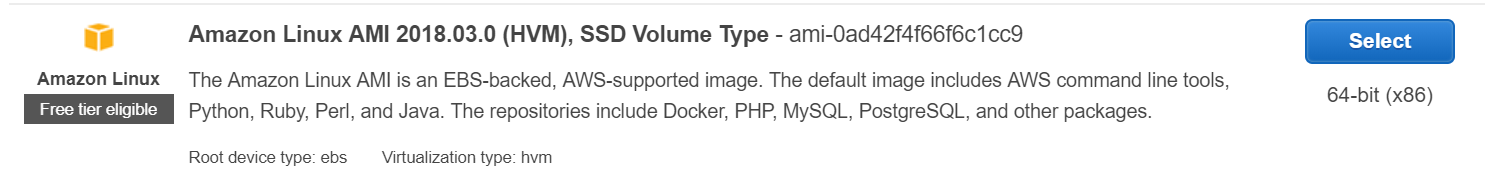
**Better Performance**: You don’t need to rely on just one computer; the load is shared.

Jenkins Master-Slave Installation:

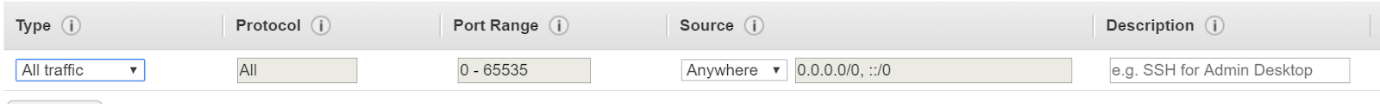
**Adding Jenkins Slave**



Step 1: Crete A EC2 as Jenkins Slave



Allow All traffic for this EC2 instance.



Step 2: Install Java Maven Git on Jenkins Slave node as well .

yum update -y

yum install git -y

sudo dnf install java-17-amazon-corretto

sudo dnf install java-17-amazon-corretto-devel

cd /opt

sudo wget https://dlcdn.apache.org/maven/maven-3/3.9.4/binaries/apache-maven-3.9.4-bin.tar.gz

sudo tar xvf apache-maven-3.9.4-bin.tar.gz

ls -ltr

cd

echo "export M2\_HOME=/opt/apache-maven-3.9.4" >> .bash\_profile

echo "export M2=$M2\_HOME/bin" >> .bash\_profile

echo "export PATH=$M2:$PATH" >> .bash\_profile

source ~/.bash\_profile

echo $M2\_HOME

Step 3 :  Login to Jenkins EC2 Slave machine



Run below command on Jenkins Slave

 Create user and add the user to wheel group

#useradd jenkins-slave-01

# Create SSH Keys

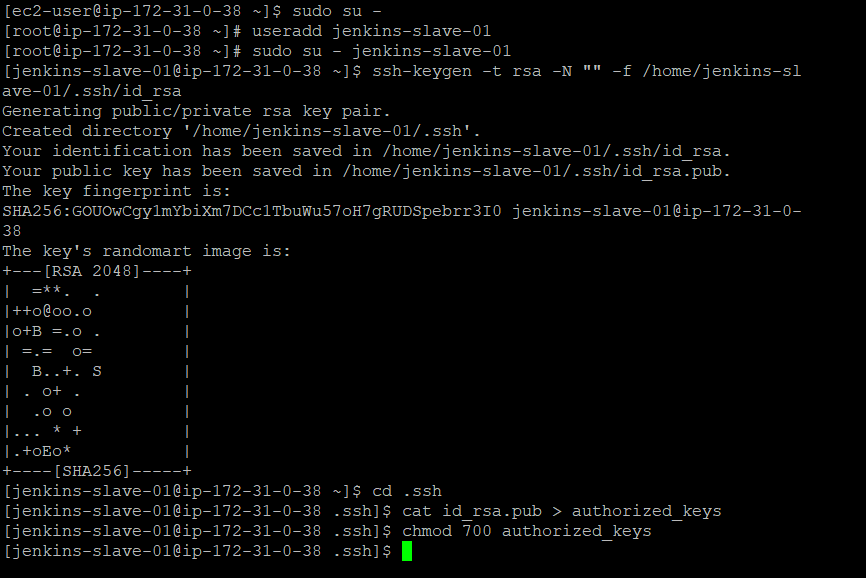
#sudo su - jenkins-slave-01

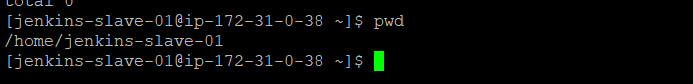
#ssh-keygen -t rsa -N "" -f /home/jenkins-slave-01/.ssh/id\_rsa

cd .ssh

cat id\_rsa.pub > authorized\_keys

chmod 700 authorized\_keys





Step 5: Login to Jenkinsmaster and scan the key for jenkinsS

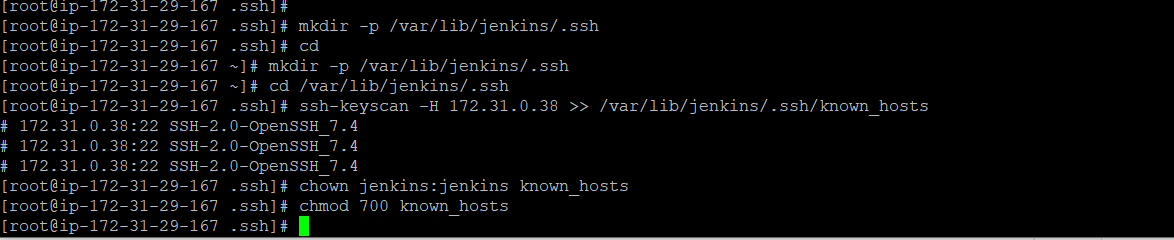
mkdir -p /var/lib/jenkins/.ssh

cd /var/lib/jenkins/.ssh

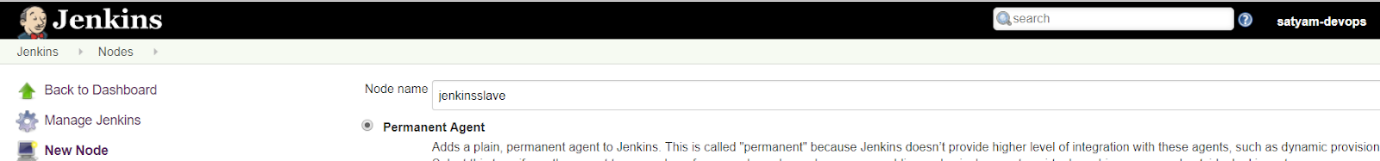
ssh-keyscan -H   172.31.45.2 >> /var/lib/jenkins/.ssh/known\_hosts

chown jenkins:jenkins known\_hosts

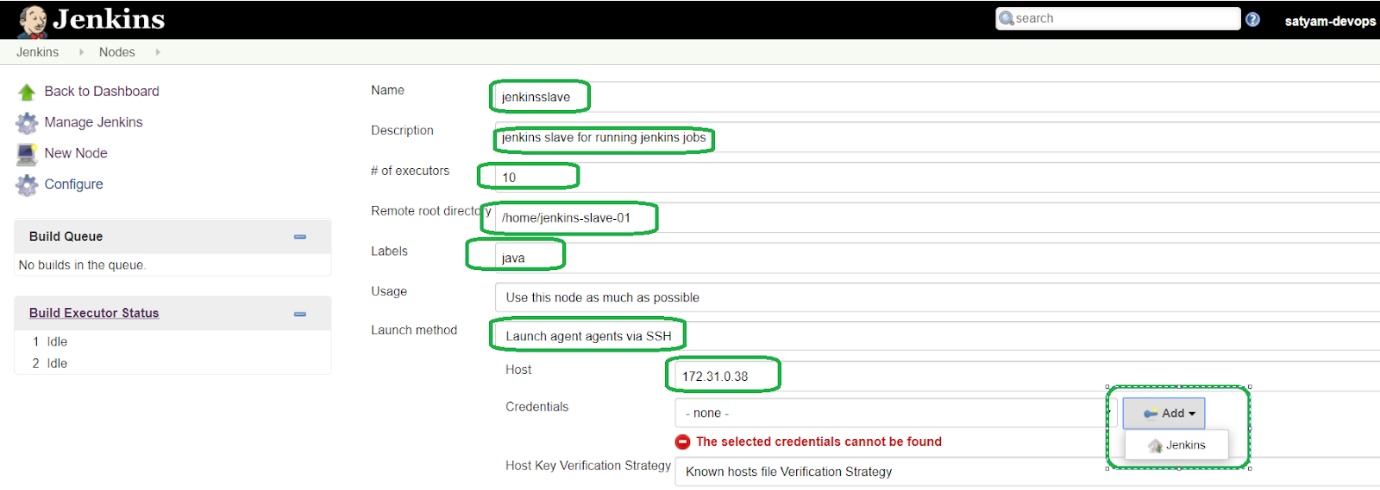
chmod 700 known\_hosts



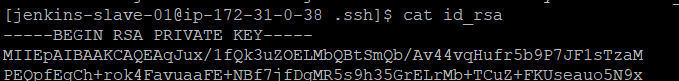
Step 6: Configure the Slave using Manage Jenkins

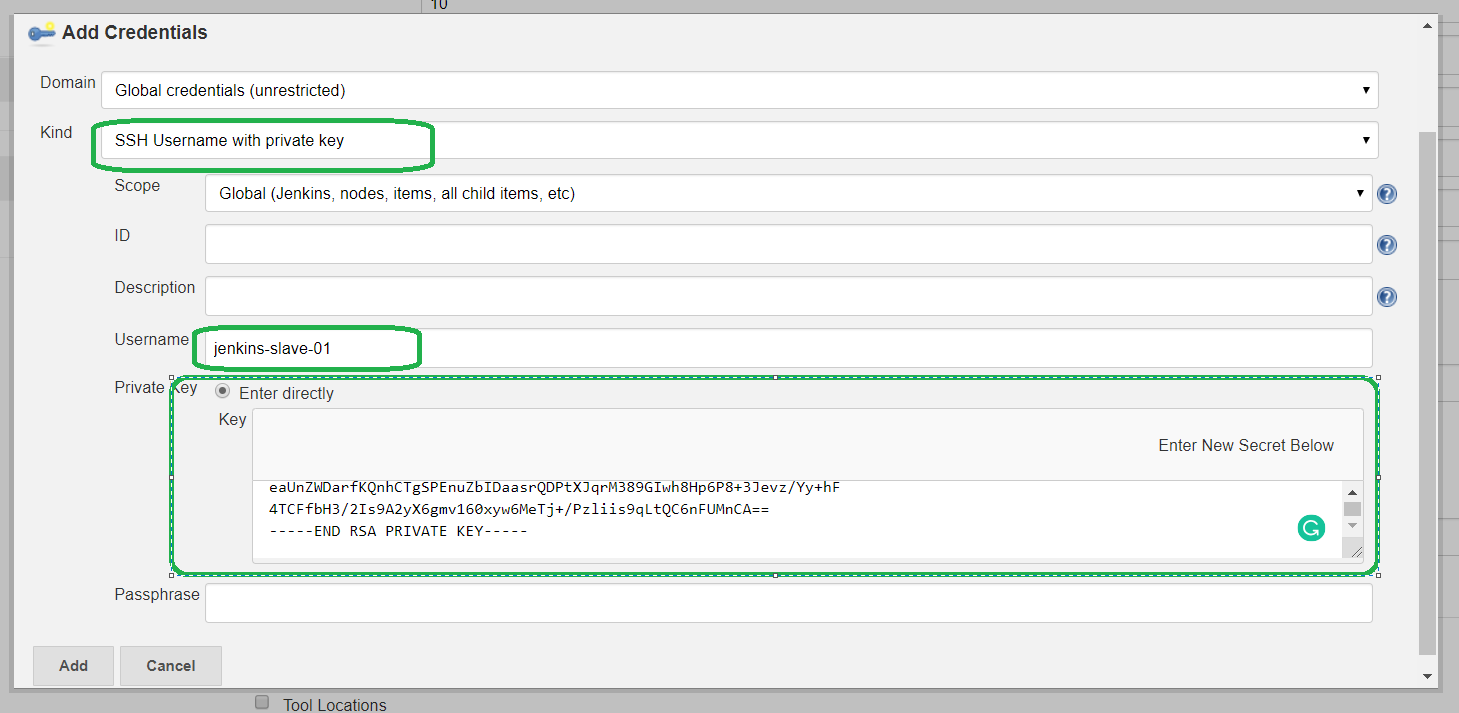


Step 7: Start filling the details

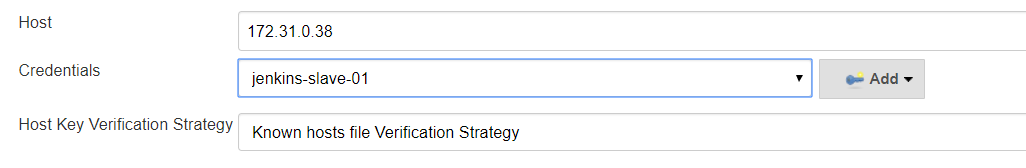


Step 8 : Add the credential’s , Find the key on Jenkins

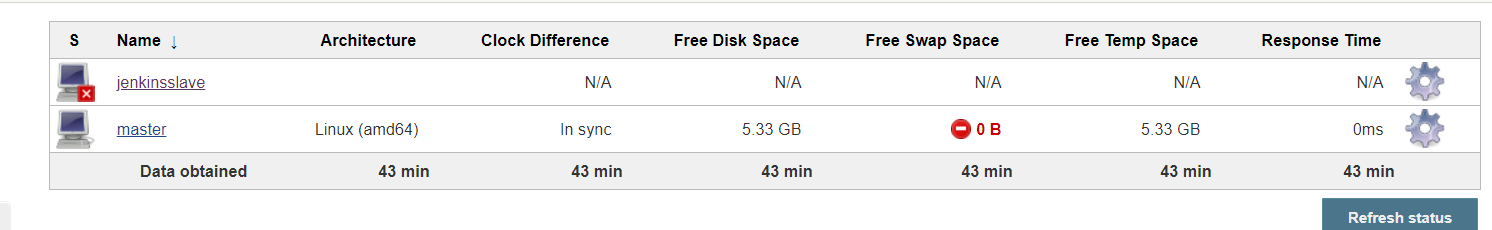


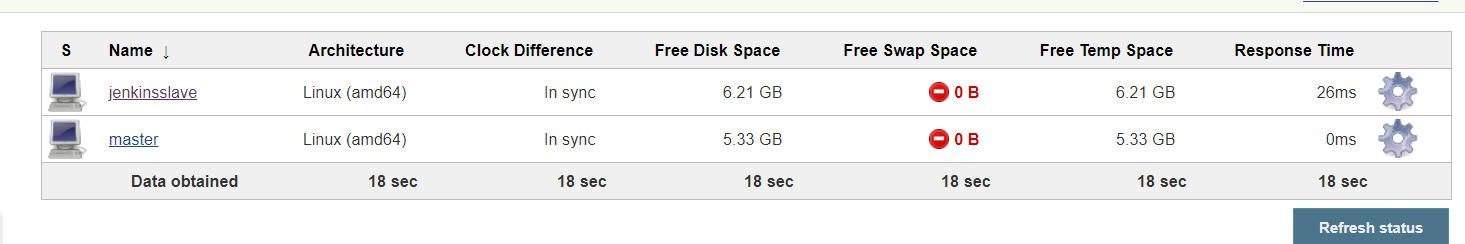


Step 9 :  Select the User Id and for password we are using key then save it



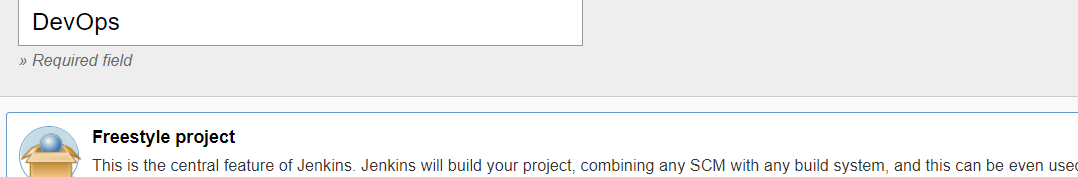
Step 10:  Once you save it Jenkins Master will start taking to Jenkins Slave if everything is ohk, he will add



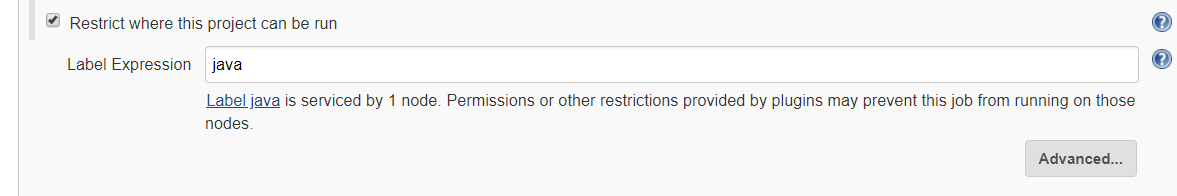


**Create a hello world jobs that run on our slave Node.**

Step 1 :  Create Jobs in Jenkins use free style project .



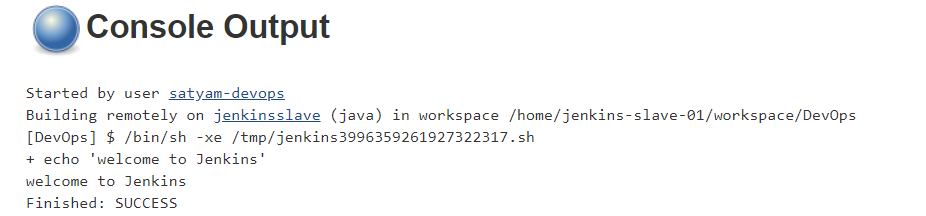
Step 2: Under General tab of jenkins select



Step 3: Under build tab



Step 4 : Save apply and click on build now.



**Jenkins Pipeline Job:**

**Jenkins Pipeline:**

It’s a linear sequence of stages.

Pipeline can have multiple stages.

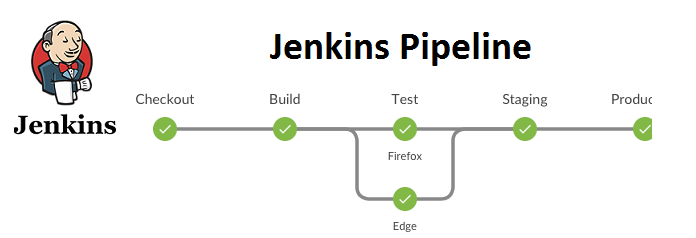
There are 2 Type of create jobs (Pipeline) in Jenkins.

1. Scripted Method: Old method and very Less use in industry
2. Declarative Method: It’s a new way of creating pipeline.

Have you created any pipeline in Jenkins?

Have created pipeline in Jenkins using declarative method.

**Pipeline as a code : Creating pipeline with the help of Code, is called as Pipeline is a code.**



**Pipeline Syntax:**

pipeline{

Agent any ()

stages {

stage (code comiplation) {

steps {

sh "mvn clean compile"

}

}

stage (junit test) {

steps {

sh "mvn clean test"

}

}

stage (code package) {

steps {

sh "mvn clean package"

}

}

}

}

**What is Blue Ocean in Jenkins?**

**Blue Ocean** is a **new user interface** (UI) for Jenkins that makes it easier to work with Jenkins, especially when you're using **pipelines** (a set of steps that automate building, testing, and deploying code).

**Key Things Blue Ocean Does:**

1. **Better Visuals**: It shows your pipeline (the steps that run automatically) in a **clear, easy-to-understand** way with nice colors and progress bars.
2. **Easier Pipeline Creation**: You can create pipelines without needing to write complicated code. It's much simpler, especially for beginners.
3. **Shows Everything in One Place**: You can see **builds**, **test results**, and **logs** all in one view, making it easier to see what's going on with your jobs.
4. **Real-Time Updates**: You can see your build progress live. If something goes wrong, Blue Ocean highlights it so you can easily fix it.
5. **Pull Request Integration**: If you're using GitHub or Bitbucket, Blue Ocean can show the **status of pull requests** (changes you want to merge into your project) directly in Jenkins.

**Why Use Blue Ocean?**

* **It's easier** to understand your builds and pipelines.
* **Better design** makes Jenkins look modern and user-friendly.
* **More insights**: You can see detailed progress, errors, and logs in a single view.

**How to Use It?**

1. **Install the Blue Ocean Plugin** in Jenkins.
2. **Go to the Blue Ocean link** in Jenkins, and you'll see a new, cleaner interface.
3. **Create or manage your pipelines** using the simple tools and visuals provided.

P**arameterized pipeline**:

P**arameterized pipeline** in Jenkins allows you to run the same Jenkins job with **different inputs** each time. This is useful when you want to customize the build process based on different values, like choosing a different branch of code or setting specific options for your build.

**How It Works:**

1. **Define Parameters**: You set up the parameters in your pipeline. For example, you can ask for:
   * Which **branch** to build (e.g., develop or master).
   * Which **environment** to deploy to (e.g., staging or production).
2. **User Input**: When you trigger the job, Jenkins will ask you to provide the values for these parameters before it runs.
3. **Run the Pipeline with Different Values**: Based on the inputs you give; Jenkins will adjust the pipeline's behavior. For example, it will build the correct branch of the code or deploy to the right environment.

**Why Use Parameterized Pipelines?**

* **Flexibility**: You can run the same pipeline in different ways without needing to modify the code each time.
* **Customization**: It lets you pass in values to control how the pipeline behaves (like the branch or environment).
* **Reusability**: One pipeline can handle multiple use cases.