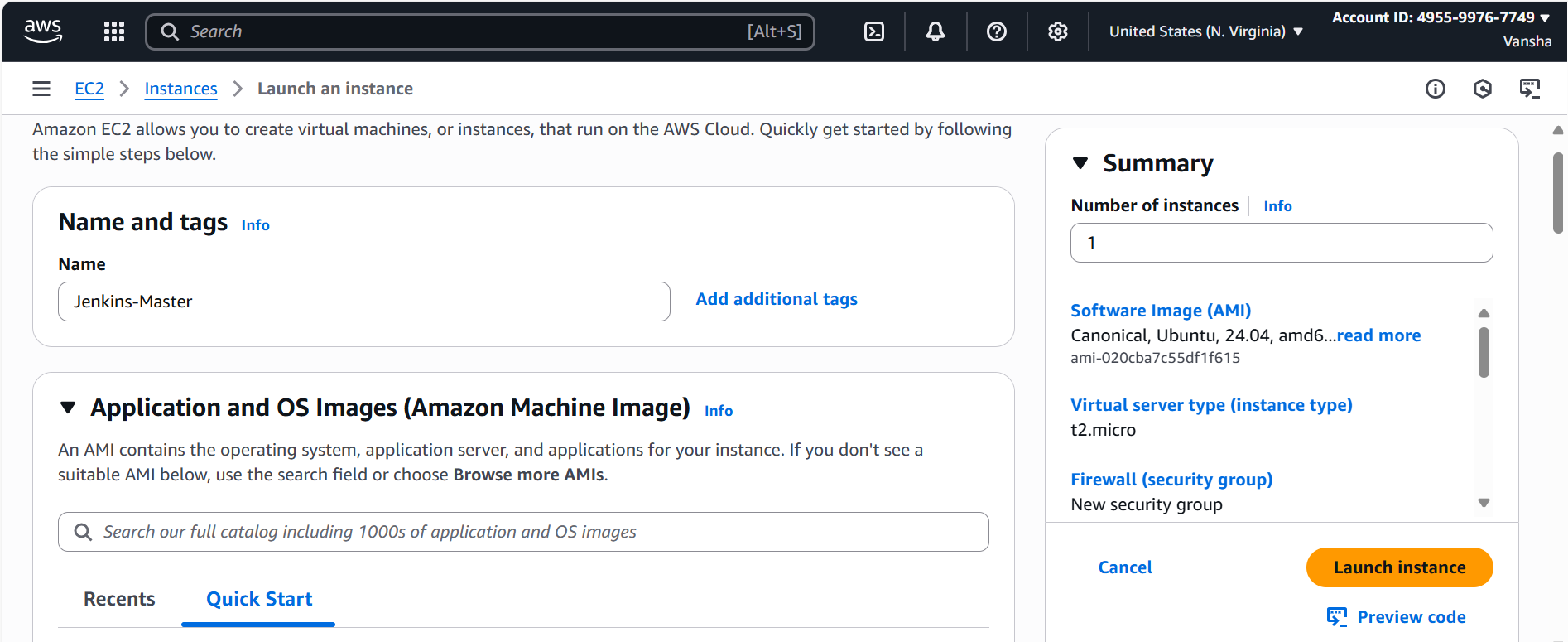
**JENKINS HANDS ON VIDEO**

**STEP1:**

1. Launch an EC2 instance. We are going to install Jenkins onto this Ubuntu OS EC2 Instance named “Jenkins Master”.



1. Install JAVA package and Jenkins binary onto this EC2 instance.

Copy the Public IP of the EC2 instance and try to connect it via your Git Bash/ Command prompt terminal.

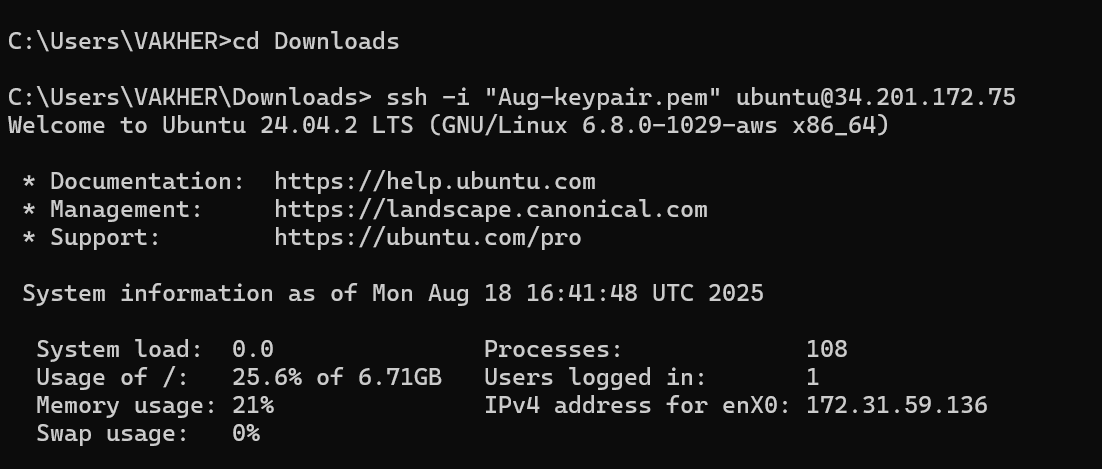
Public IP: 34.201.172.75

Keypair name : Aug\_keypair.pem

**Through Command Prompt:**

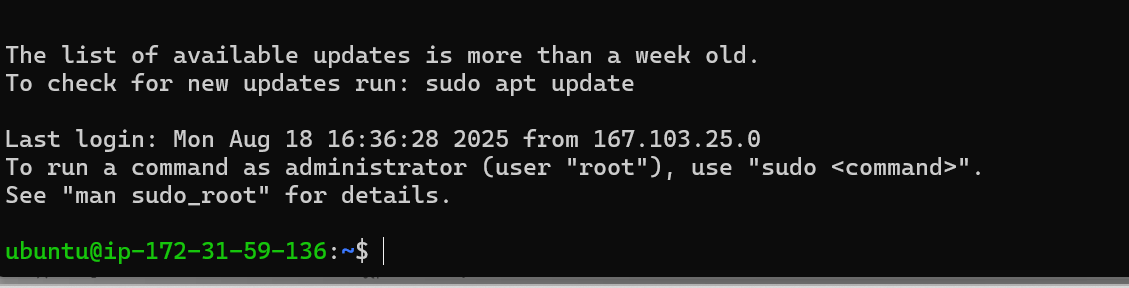
**The keypair is downloaded in Downloads folder.**

**Then go to Download folder from CMD.**

****

**Use command:**

**ssh -i “Aug\_keypair.pem” ubuntu@Public IP of EC2 instance.**

****

**b.**

**From GITBASH**

**Go to the path where the .pem file is present.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Go to the downloads folder in which .pem key is present.**

**A close up of words

AI-generated content may be incorrect.**

**Chmod 400 is used only in WSL/Bash (not in CMD).**

**A close up of words

AI-generated content may be incorrect.**

**A screenshot of a computer error

AI-generated content may be incorrect.**

**This way we can connect to the remote Ubuntu server via our personal laptop/Git Bash Tool.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Step2: Jenkins is a JAVA based application. So to install Jenkins, first requirement is to install JDK (JAVA DEVELOPMENT KIT).**

**First we are updating the apt packages, and then we are installing the JDK.**

**sudo apt update**

**sudo apt install openjdk-17-jre**

**Now, verify whether the JAVA has been installed or not on our VM:**

**Java -–version**

**Now, you can proceed with installing Jenkins (for Ubuntu Linux Distribution)**

**curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \**

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

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

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

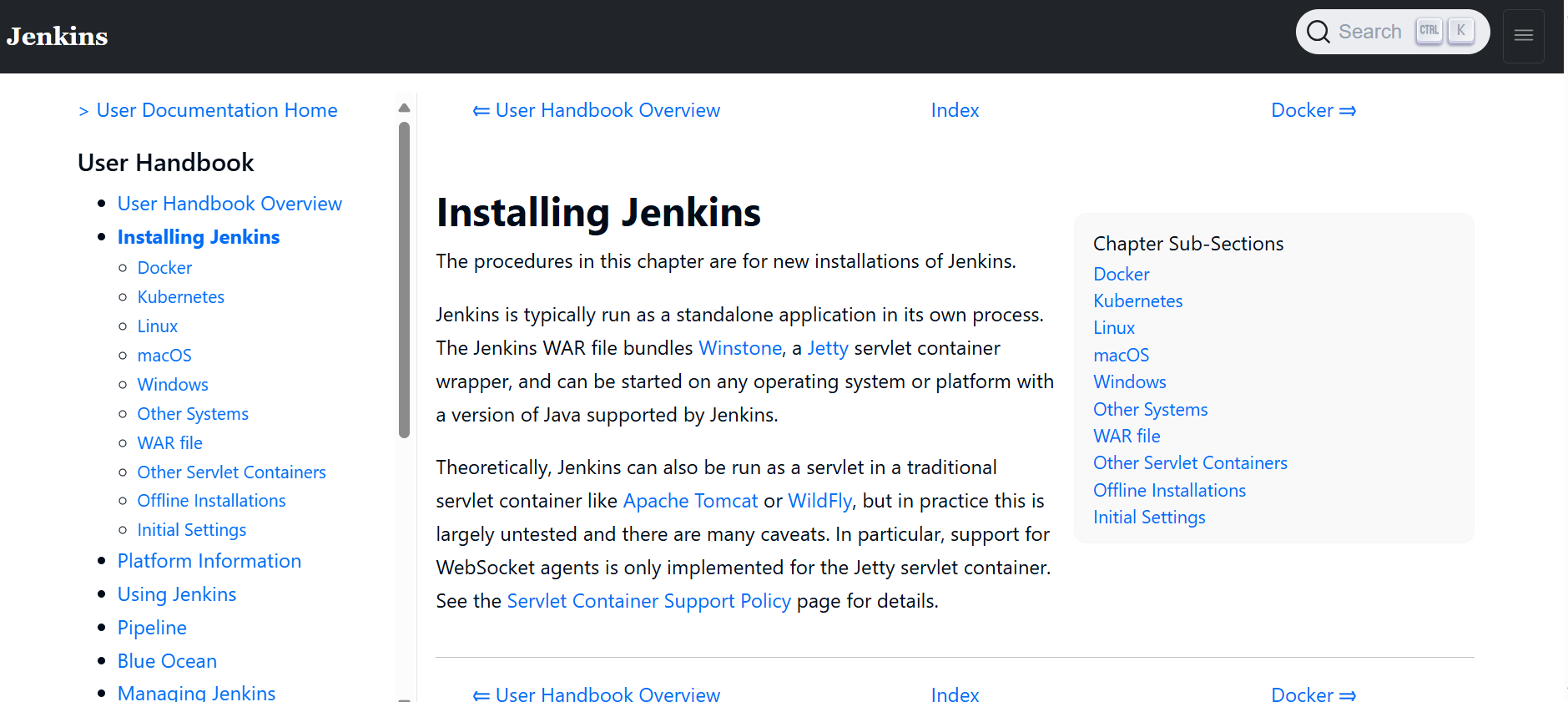
**/etc/apt/sources.list.d/jenkins.list > /dev/null**

**sudo apt-get update**

**sudo apt-get install jenkins**

Say you are on a different distribution of Linux such as CentOs, Debian etc, in that case the commands to install Jenkins will be different.

For that, go to Jenkins Official Website/Documentation:





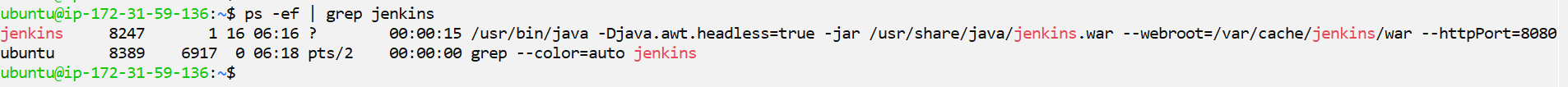
Jenkins is thus installed on the system.

Now to access this Jenkins application, we need to modify our EC2 Instance **Inbound Security Group Rules.**

**Jenkins is an application running on port 8080. So we need to open port 8080 to access this application.**

**To check this, run command:**

**ps -ef | grep Jenkins**

****

**Now go to Security Group of EC2 instance, Open the Inbound rules.**

**A screenshot of a computer

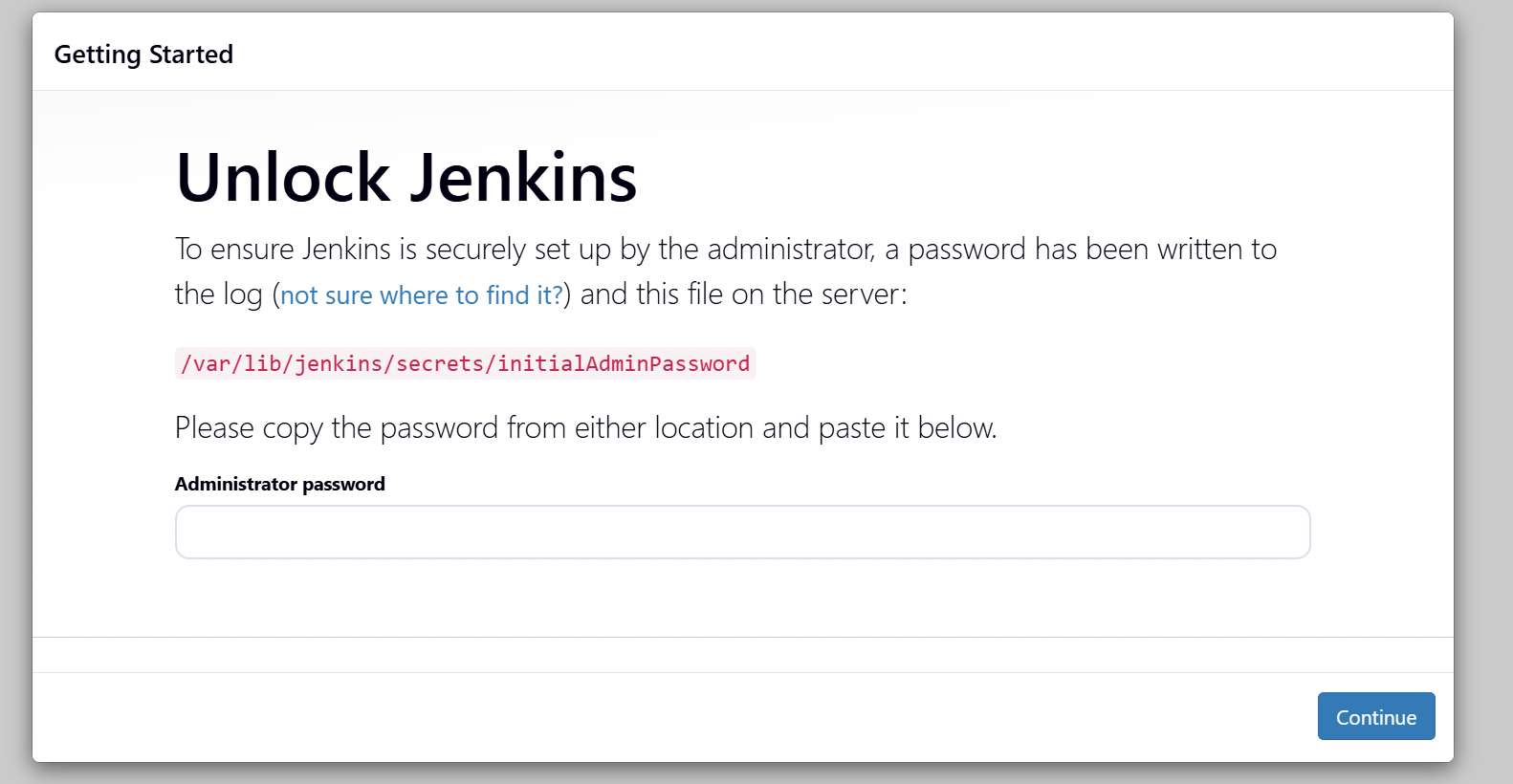
AI-generated content may be incorrect.**

**Click on: Save Rules.**

**Let us see if Jenkins instance is accessible from the outside world or not.**

**Open your browser:**

**Public IP of EC2 instance : 8080.**

****

After you login to Jenkins, - Run the command to copy the Jenkins Admin Password :

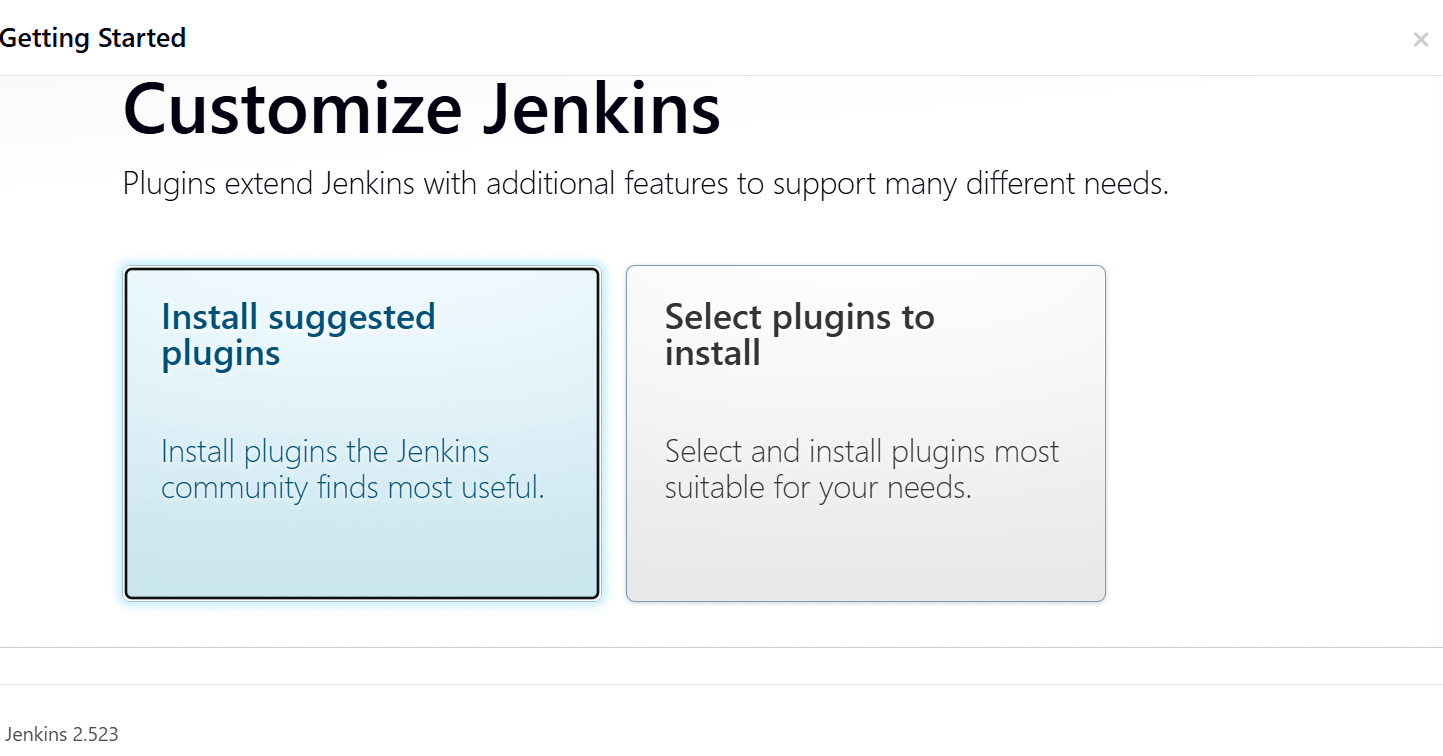
 sudo cat /var/lib/jenkins/secrets/initialAdminPassword –

36fdd3caf2d94f90becb0714fc7fa849

Copy Paste the Administrator password

A screenshot of a computer

AI-generated content may be incorrect.



**Install Suggested Plugins.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Kept the username as: admin**

**Password as: test@123**

**Full name as Devops and email id as** [**vanshapractice@gmail.com**](mailto:vanshapractice@gmail.com)**.**

**A screenshot of a computer

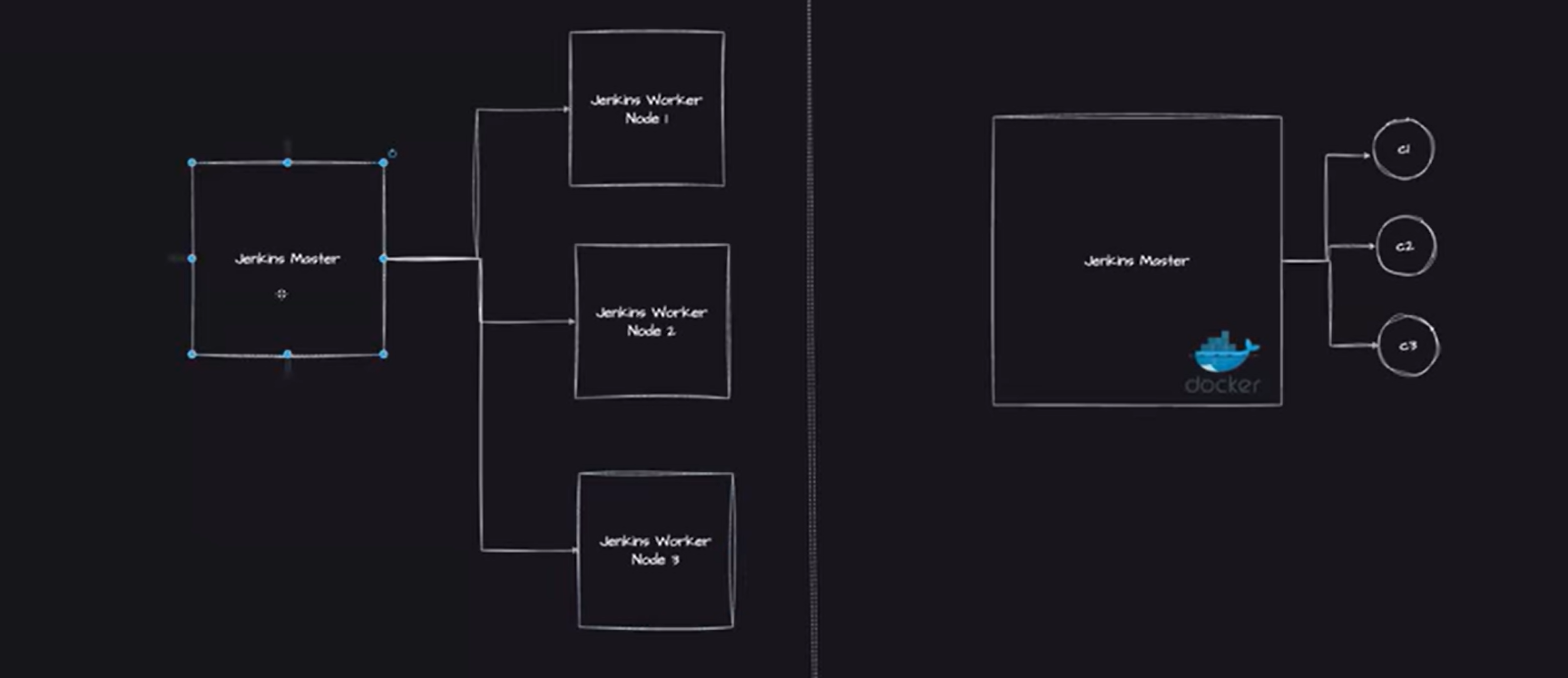
AI-generated content may be incorrect.**

**Jenkins URL is:** [**http://PublicIP**](http://PublicIP) **of EC2 instance: 8080/**

**This way, Jenkins Installation part is being done.**

**The Ec2 instance on which the Jenkins is installed is considered as the Master Jenkins Node.**

**Jenkins Master can be used for the scheduling purpose. We can create Jenkins Worker Nodes to run the jobs. (in order to avoid dependency conflicts and overloading of the master node).**

****

**Sometimes, it happens that certain applications need less upgrade or changes. So, most of the times, worker EC2 nodes get unused and wasted.**

**To eradicate this issue of wasting the resources in the form of unused worker nodes (EC2 instances), we use “Jenkins with Docker as Agents”**

**i.e we will try to run Jenkins pipelines on Docker containers.**

**Docker Containers are very light in weight as compared to the Ec2 instances/machines.**

**It is easy to spinup and tear down the docker containers and it is also very cost effective.**

**Configuration of Docker with Jenkins Pipelines:**

**Let us configure Docker on the same EC2 instance (where Jenkins have been installed).**

**A screenshot of a black screen

AI-generated content may be incorrect.**

**Docker usually runs on a single process called as the Daemon Process, and this Daemon process is by default not accessible to other users.**

**Since we installed Docker (using Root : using Sudo command) hence, so Daemon is only accessed by the Root user.**

**Hence we need to grant the Daemon permission to both the Ubuntu user as well as the EC2 user.**

**First become a root user :**

**sudo su –**

**Once you be a root user, execute the below commands:**

**usermod -aG docker jenkins**

**usermod -aG docker ubuntu**

**systemctl restart docker**

**Usermod is used to give permissions to Jenkins and the Ubuntu user as a part of the docker group. So who ever wants to access the docker have to be a part of the Docker group.**

**systemctl restart docker : will restart the docker daemon.**

**Use :**

**su - jenkins**

**The command su - jenkins is used on Unix-like systems to switch the current user to the jenkins user account, starting a login shell.**

**Now you are in Jenkins user.**

**Now run a Docker hello world command as below:**

**A black background with white text

AI-generated content may be incorrect.**

**docker run hello-world**

**This command shows that the Jenkins user is also able to create and run the containers.**

**NOTE: Sometimes Jenkins will NOT pick up these changes, hence it is important to restart the Jenkins.**

**Go to Jenkins Login page/ browser.**

**Change the URL to :**

**restart.**

**Then it will ask for login and passoword used above.**

**NEXT IMP STEP IS TO INSTALL THE DOCKER PIPELINE PLUGIN IN JENKINS:**

**This is to make sure that whenever Jenkins is running a job then if the user provides to run a specific job on docker, that way Jenkins should have that configuration ready hence installing the docker pipeline plugins are important.**

**Go to Jenkins Dashboard : Manage Jenkins: Manage Plugins**

**Install the Docker Pipeline Plugin (in Available Plugin) section.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Download and install after restart.**

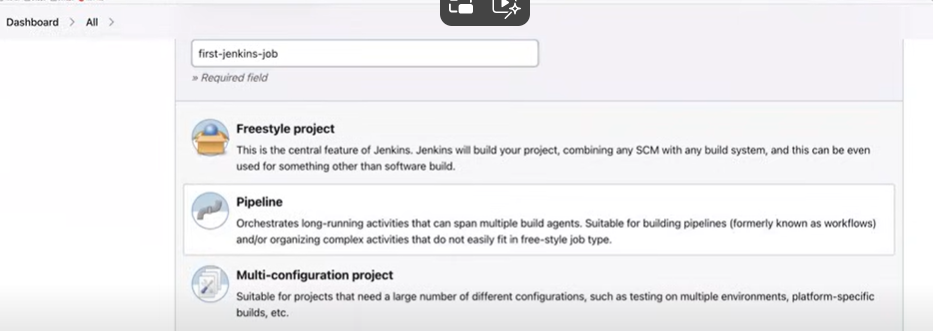
**Then click on: Restart Jenkins.**

**Again provide the user name and the password.**

**Let us write our first Jenkins pipeline.**

**Go to Jenkins Dashboard: New Item: Pipeline**

**Jenkins has come up with a pipeline approach. Give the name as : first-jenkins-job and select the PIPELINE.**

****

**In this, we can write scripted / declarative pipelines. Ie we can write the pipeline as a code here.**

**In this, we can write, modify our own code, storing the code in GitHUB. and hence share it with our peers as well.**

**Now, write a simple Jenkins pipeline to verify if the Docker agent configuration is working as expected OR NOT?**

**The pipeline will have agent and stages configuration.**

**pipeline {**

**agent {**

**docker { image 'node:16-alpine' }**

**}**

**stages {**

**stage('Test') {**

**steps {**

**sh 'node --version'**

**}**

**}**

**}**

**}**

**Explanation:**

**Jenkins will pick up the code from your local Github or VCS and delivering it to Production/ staging environment by automating all the stages/steps in between.**

**Say your job is to automate 10 stages on Jenkins so it works as an orchestrator.**

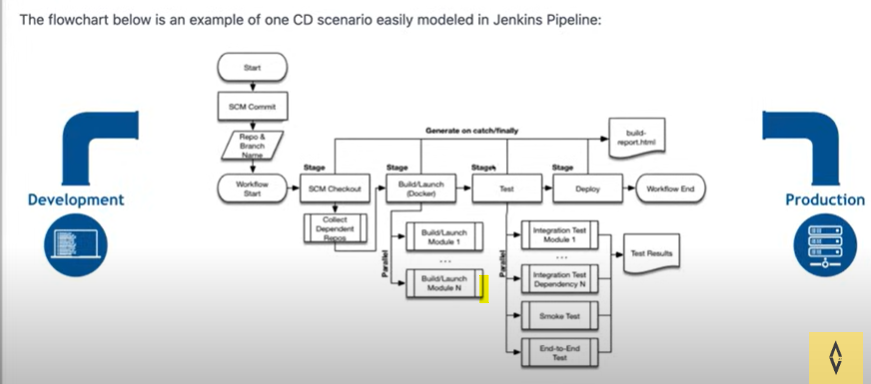
**When we write the Groovy Scripting in Jenkins, it includes a number of stages:**

**1, The first stage is the build stage where it checkout the code from a Github repo.**

**2.Then it is executing the MAVEN target.**

**3.Then it is checking that the JNUIT coverage (unit testing for JAVA) is ok or not.**

**4. Then archiving the JUNIT report artifacts. Storing the .jar files. (generated as an end product of the Build process file).**

****

**End to End Jenkins Flow.**

**In Stages, there can be multiple STEPS (steps means the commands in bash to be executed).**

**Jenkins also provide the Pipeline Syntax. That will provide you with the scripting for STEPS required in Stages.**

**Say we want to check out the code from the GITHUB repo.**

****

**Then give the URL of the GUTHUB where the code is hosted.**

**Give the branch name : say (main)**

**Now, GENERATE PIPELINE SCRIPT…**

**Copy the script and use in your steps**

1. **Say we want to run a shell script (test.sh) present in a folder foo**

**A blue and white line

AI-generated content may be incorrect.**

**A white rectangular object with blue lines

AI-generated content may be incorrect.**

**It will give a pipeline script…**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Now go to Jenkins dashboard.**

**Select: Pipeline Script from SCM. (Github Source Code management Repository)**

**A screenshot of a computer

AI-generated content may be incorrect.**

**SCM : Git**

**Give the URL of the repository : (paste here).**

**If u have a private repository, Give your credentials.**

**Give the branch name : \*/main**

**Copy the URL or the path where the Jenkins pipeline is present.**

****

**SAVE.**

**Where is this Jenkins pipeline going to RUN?**

**Click on Left side: BUILD NOW.**

**Jenkins will start running. It will first fetch the code from the GitHub. Access the Jenkins file there.**

**Now it Is pulling an Docker image : Node 16 alpine.**

**First it will check if there is any docker image or container on master Ec2 instance. It did not find it. Then it started pulling the mode js 16 container image.**

**It executed the entire pipeline as a script and gave the ouput as a docker image node version ( to verify it is correct or not)..**

**NOTE: Once the jenkins execution is done, delete the docker container.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**If we do: docker ps, you will see that there are NO running containers.**

**Now execute,**

**Docker ps -a**

**The command docker ps -a is used to list all containers on your system — both running and stopped.**

**We can observe that NO docker processes are running or docker containers are available.**

**Jenkins requested Docker to create one container : (using docker pipeline plugin)..It requested Docker to give it one container so that it can run its pipeline, that is a node.js related application.**

**Docker told ok take a container C1 (say), it ran the pipeline on it and as soon as the execution is finished, it deleted the container.**

**There are NO background running containers as such, whenever there is a process then only these containers start running and executing the pipelines.**

**There is reduced cost and no dependency of managing the dependencies and upgrades (as it used to be in case of VMs). No headaches of upgrading the worker nodes on different versions.**

**But using the docker, you donot need to login to the VMs and manually download the recent packages or plugins. (and additionally download the security patches n all).**

**Just go to Jenkins file and change the node js version from 16 to 17.**

**In this way, single agent single stage Jenkins Pipeline can be created with Docker as Agent.**

**Let us continue with Multi Agent Multi Stage Jenkins Pipeline:**

**Say we have a three-tier application: Front-end, Backend and the Database.**

**Say: Database related CICD needs to be performed on a VM that has CentOS , where as Frontend and backend applications needs to be implemented on a VM that has UBUNTU in it.**

**For this type of scenarios, we go for Multi-Stage Jenkins architecture**

**pipeline {**

**agent none**

**stages {**

**stage('Back-end') {**

**agent {**

**docker { image 'maven:3.8.1-adoptopenjdk-11' }**

**}**

**steps {**

**sh 'mvn --version'**

**}**

**}**

**stage('Front-end') {**

**agent {**

**docker { image 'node:16-alpine' }**

**}**

**steps {**

**sh 'node --version'**

**}**

**}**

**}**

**}**

**Here we have mentioned the agent as NONE.**

**means that no global agent is assigned for the entire pipeline. Here's what that implies:**

**🔍 What does agent none do?**

* **It disables the default agent for the pipeline.**
* **You must then explicitly specify an agent at the stage level for any stage that needs to run on a node.**

**✅ When to use it?**

**Use agent none when:**

* **You want fine-grained control over which stages run on which agents.**
* **Different stages need to run on different types of nodes (e.g., Linux, Windows, Docker).**

**In Jenkins, an agent is a directive that tells Jenkins where and how to execute the pipeline or a specific stage.**

**An agent can be:**

* **A node (physical or virtual machine) in your Jenkins environment.**
* **A Docker container.**
* **A label that matches one or more nodes.**
* **Or even none, if you want to control agents at the stage level.**

**Example:**

**stage('Back-end') {**

**agent {**

**docker { image 'maven:3.8.1-adoptopenjdk-11' }**

**If this is a backend application, use Maven or JAVA JDK as the agent.**

**If it s a frontend use agent as node js or react js docker image.**

**For DB, run mysql related docker image.**

**What are Steps?**

* **Steps are the actual commands or actions executed inside a stage.**
* **Examples:**
  + **echo 'Hello World'**
  + **sh 'npm install'**
  + **checkout scm**
  + **script { ... } for scripted logic**

**When we run this Jenkins pipeline, two targets will be running : one will be running on Maven Target and another will be running on Node.js targets.**

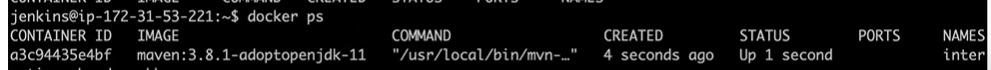
**On Jenkins Dashboard: Go to Configure:**

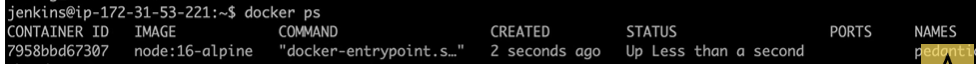
**Change the SCM path and the folder path**

**Click on : Build NOW**

**Go to the Ubuntu Master terminal and run:**

**Docker ps**

****

****

**Both the stages got executed.**

**After execution, both the docker containers got automatically deleted.**

**Say if we want to create these containers on VM, then we have to create two worker nodes for the same.**

**There will be a lot of maintenance activity.**

**To do: Fork the repo : instead of steps to execute as getting the outcomes of the version maven n node js; change in the scripting of the steps**

**Submit a java hello world application**

**OR**

**Submit a node js hello world application.**

**Then use mvn install OR npm install.**