PROJECT-4 ;

JENKINS MASTER SLAVE CONFIGURATIONS

Master Slave Configurations:

In Jenkins, master-slave configuration (now referred to as controller-agent configuration) is a setup that allows you to distribute the build and test workload across multiple nodes (machines). This setup improves performance, scalability, and allows better resource utilization in CI/CD pipelines.

Advantages:

Better Resource Utilization:

- By distributing workloads, you maximize the resource usage of the available machines. The master only handles the management, while the agents run the actual builds, freeing the master from performing all tasks.

Improved Performance:

- Distributing tasks reduces the time needed to complete jobs. Agents can run builds in parallel, reducing the overall build time in CI/CD pipelines.

Scalability:

- Adding more agent machines helps increase the capacity of your Jenkins environment without requiring upgrades to the master machine. This makes Jenkins highly scalable to meet increasing project demands.

Separation of Concerns:

- You can isolate environments for different purposes (e.g., building for Android/iOS, Java, or Python). This separation ensures the jobs run smoothly without interfering with each other.

Fault Tolerance:

- If one agent goes down, the workload can be shifted to other available agents, ensuring minimal disruption to your build process.

Customizable Hardware for Specific Jobs:

- You can tailor specific agents to run certain jobs that require different hardware or software (e.g., a high-performance agent for resource-intensive builds).

Step-by-Step Setup:

Install Jenkins on the Master (Controller):

- Download and install Jenkins on your primary machine (the controller node).
- Start Jenkins and set up the basic configurations

Prepare the Slave Machine (Agent):

- Install the necessary software and dependencies on the slave machine (e.g., Java, Docker, or any other tools your jobs require).
- Ensure the slave machine can communicate with the master over the network (TCP/IP, usually on port 22 for SSH).

Create a New Agent in Jenkins:

- Go to **Manage Jenkins** > **Manage Nodes** > **New Node**.
- Provide a name for the agent and select the type of agent you are setting up (usually "Permanent Agent").
- Configure the connection details (remote directory, usage, launch method). Common methods include:
    i. SSH: If the agent is a Linux/Unix-based system.
    ii. Jenkins Swarm Plugin: If the agent machine uses a custom plugin to connect.
    iii. Windows Agent: For Windows machines, you may need to use the JNLP protocol.

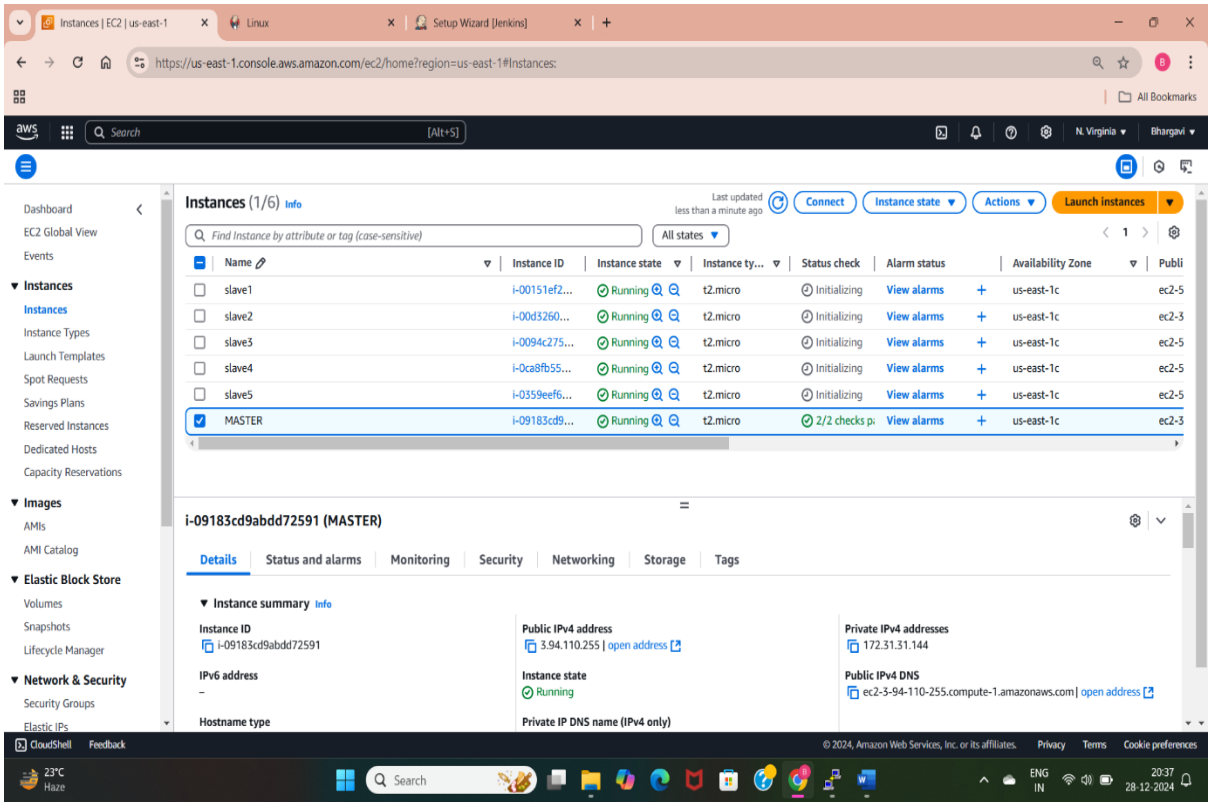
Launch the Agent:

- After configuring the agent, you can either launch it manually or wait for it to connect automatically based on the configuration.
- You can verify the connection from the Manage Nodes section, where the agent will appear as connected.

METHOD-1:

Create One Master And Five Slave Nodes And Install Nginx (Slave-1), Httpd (Slave-2), Sonarqube (Slave-3), Tomcat (Slave-4) And Deploy Any One Static Application In Any One Of The Web Servers (Httpd & Nginx) (Slave-5)

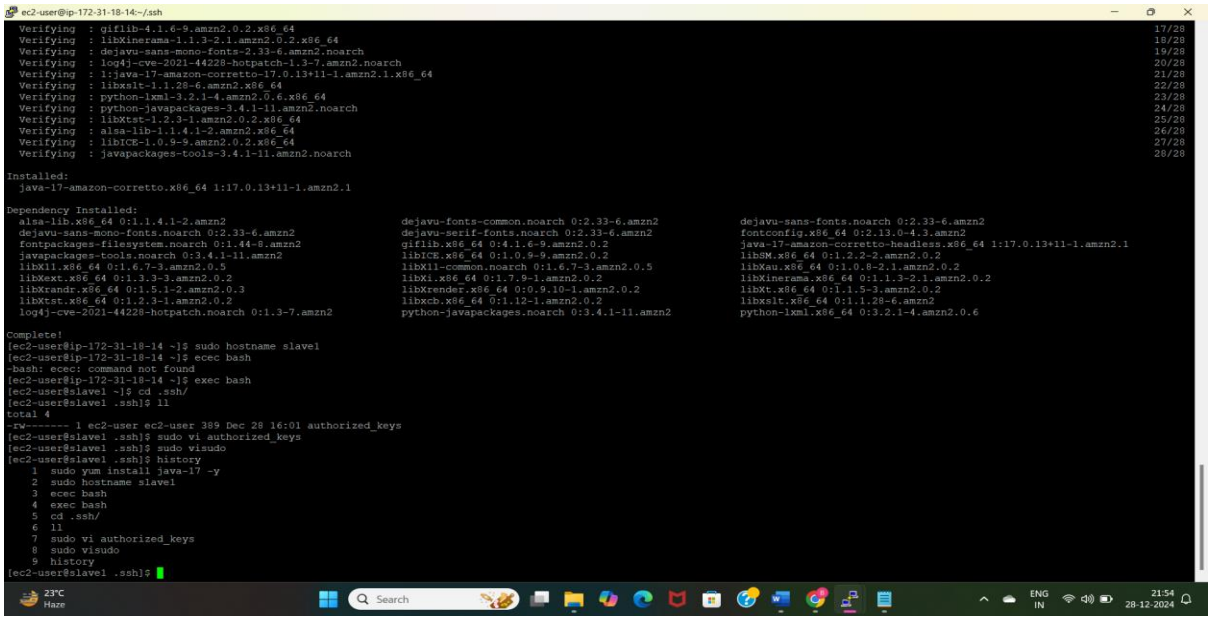
Launch an EC2 instance for the master server. Install Jenkins on the master (default port: 8080). Enable Jenkins service (sudo systemctl start jenkins).



Here I have created five slave servers to setup master slave configuration. I have taken 4 amazon-linux, with three t2.micro instances types, and only slave3 is with t2.medium.

- After that install the java 17 and Jenkins in master server & start the Jenkins.
- Generate a key in the master server.
- Use the public IP of instance and with Jenkins port 8080 paste in browser and we can see the Jenkins setup there.

Run the command ssh-keygen on the master server to generate a public-private key pair.

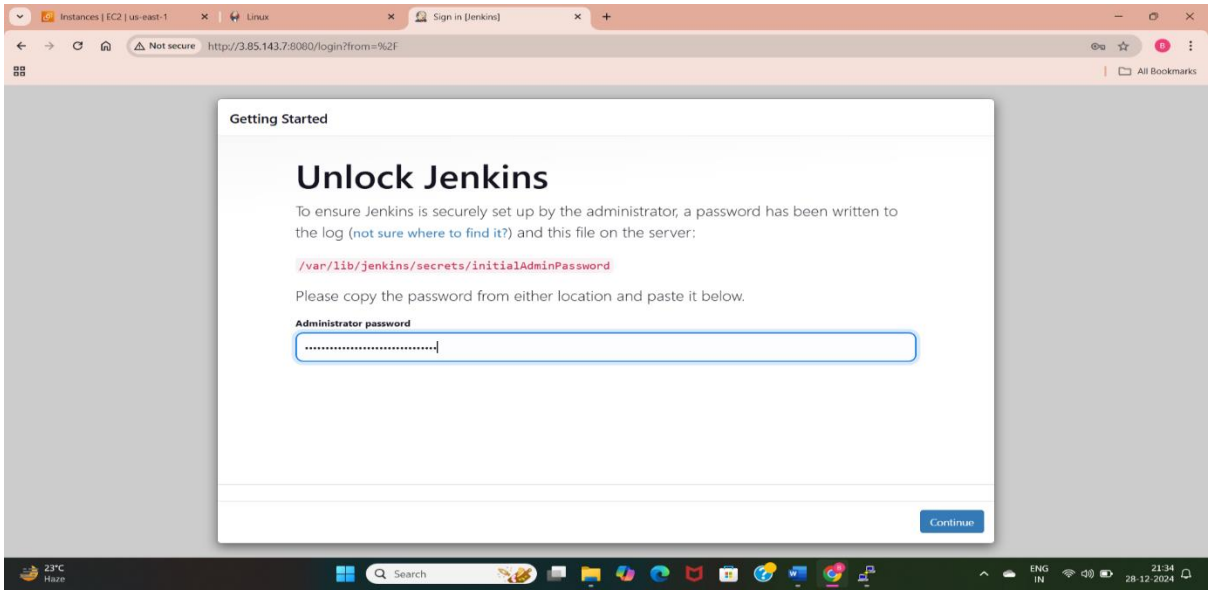


This creates:

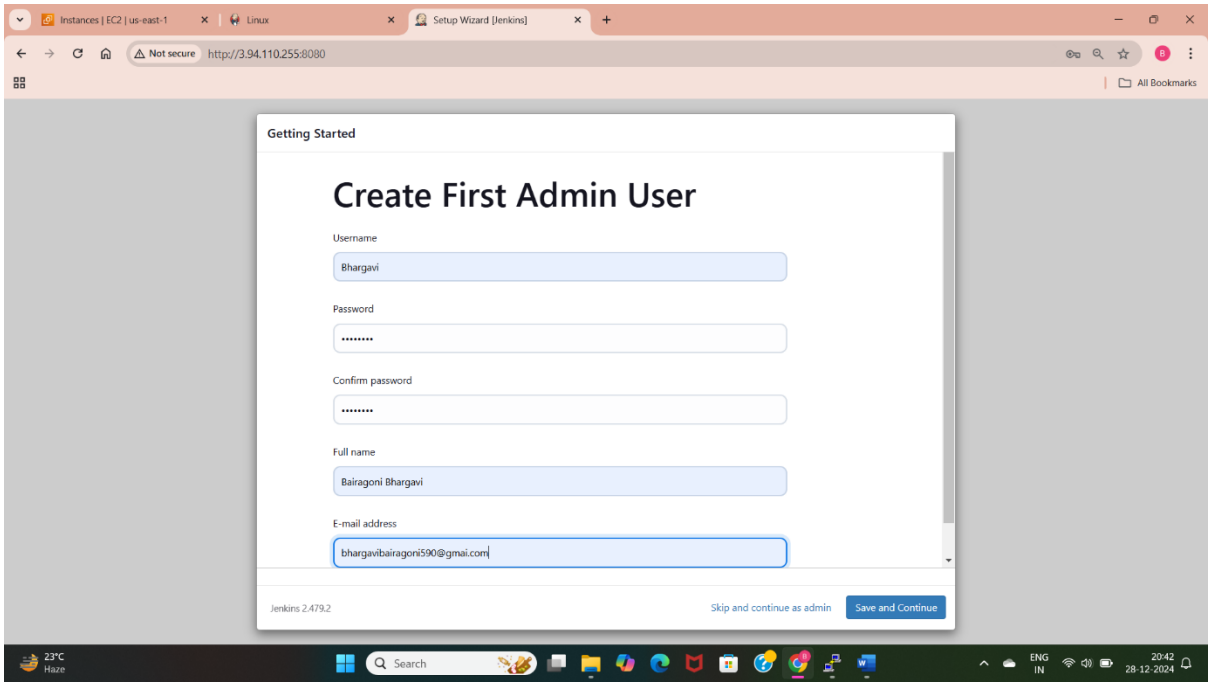
- Public key: Stored in ~/.ssh/id_rsa.pub.
- Private key: Stored in ~/.ssh/id_rsa.

Launch EC2 instances for the slave servers. Assign appropriate names using <sudo hostname slave4> and <exec bash>.

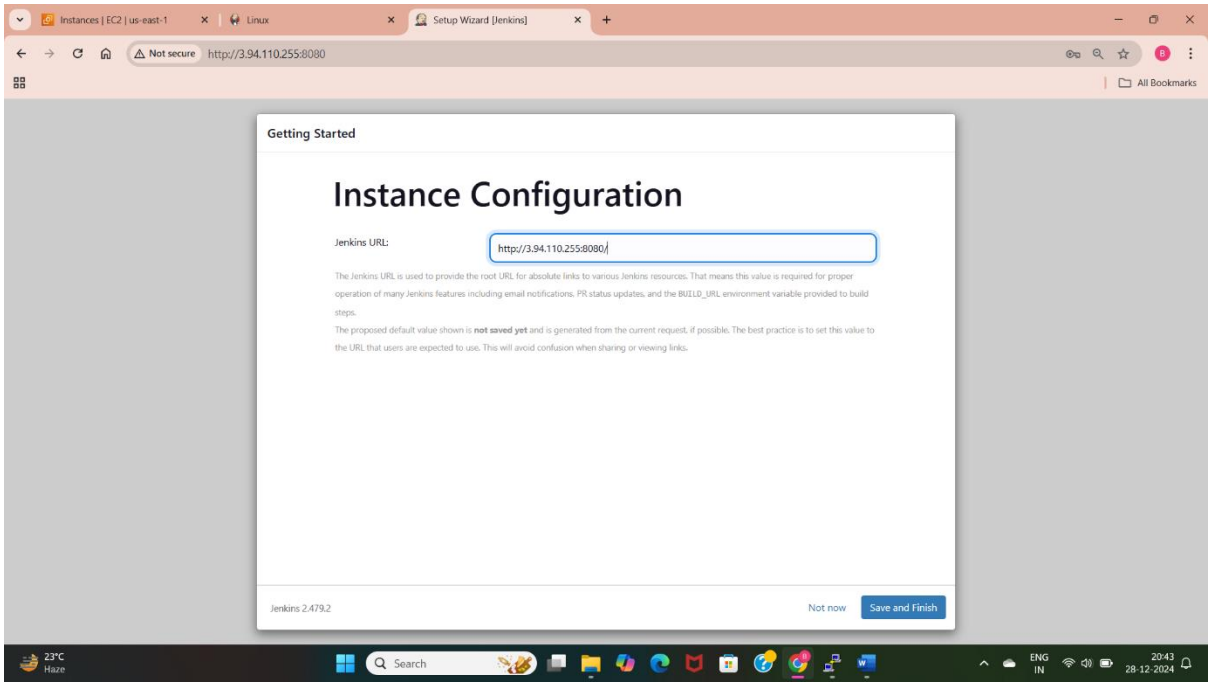
- On the master server, copy the contents of the public key <cat .ssh/id_rsa.pub>.
- On each slave server, paste the public key into the .ssh/authorized_keys file using < sudo vi .ssh/authorized_keys>. Use shift + A to append and paste the key, then save the file.
- On all servers (master and slaves), install Java as Jenkins requires it to run using <sudo yum install java-17-amazon-corretto -y>.
- On the master server, open Jenkins in your browser: http://<master-server-public-ip>:8080.



- First we need to give the password there using the above file path in the terminal as <sudo cat filepath> we will get a password there copy and paste in Jenkins.

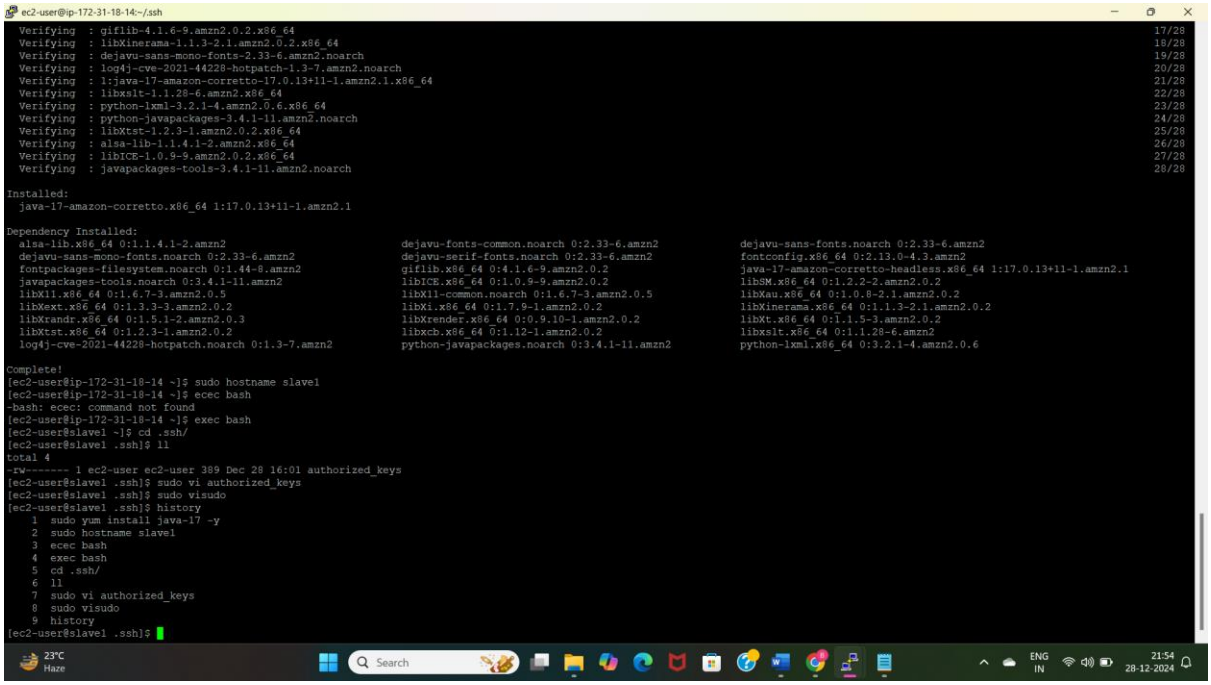


- Create a Jenkins user and password so we can build jobs using this Jenkins account.

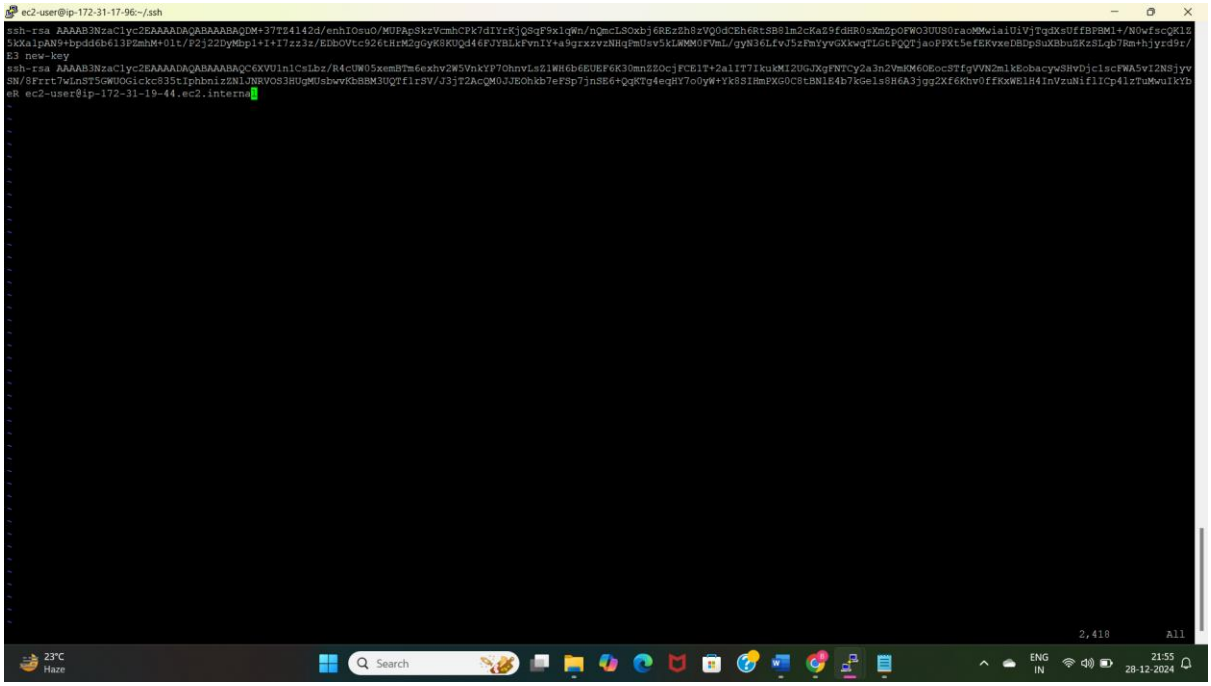


- This page is showing the Jenkins url which is “http://public IP: jenkins port(8080)”, save and finish the settings to start using Jenkins.

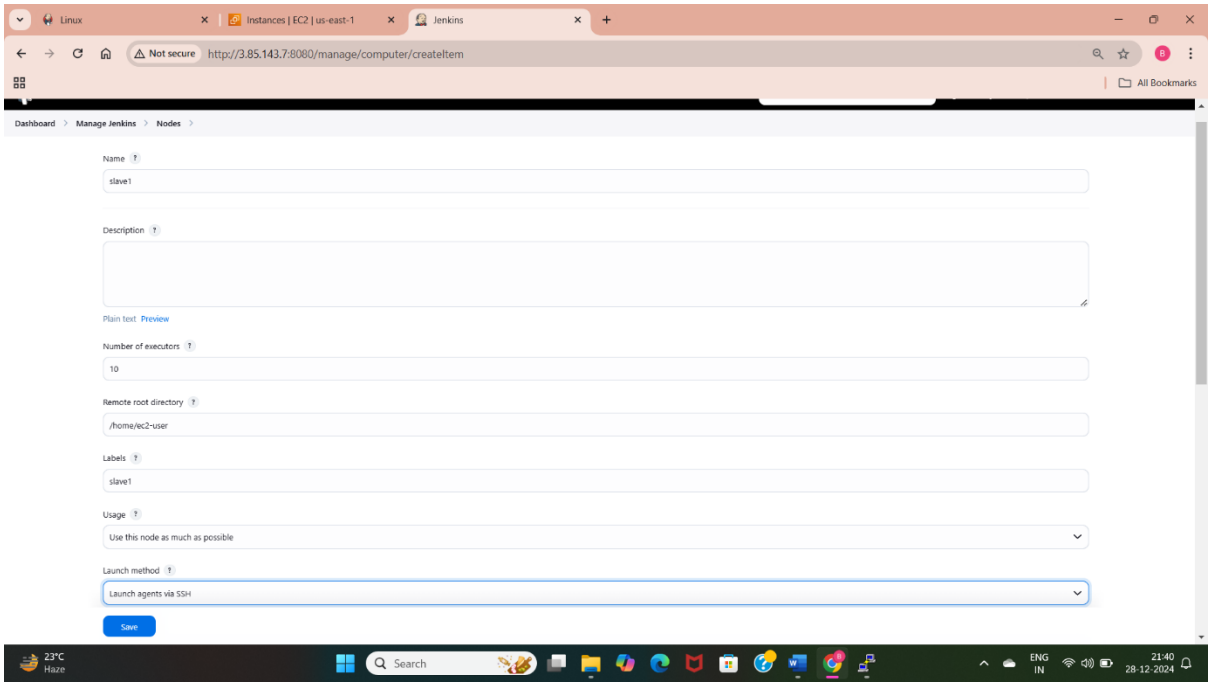
Slave-1:



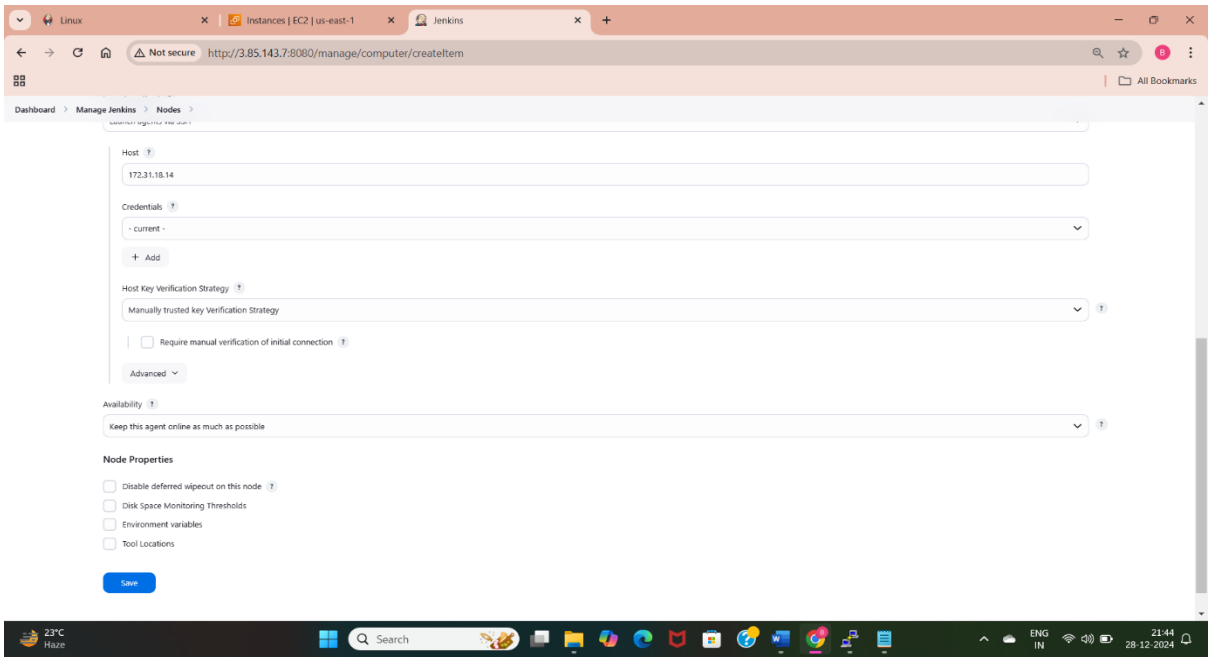
- Now I have connected the slave 1 server with the terminal. Install the Jenkins dependency (java 17) here to set the master slave configurations.
- Here I have changed the hostname in terminal to slave1 using <sudo hostname slave1> and updated the changes to terminal using <exec bash>.
- Now we need to connect the master with slave by copying the public key in the authorized key of slave1 server.



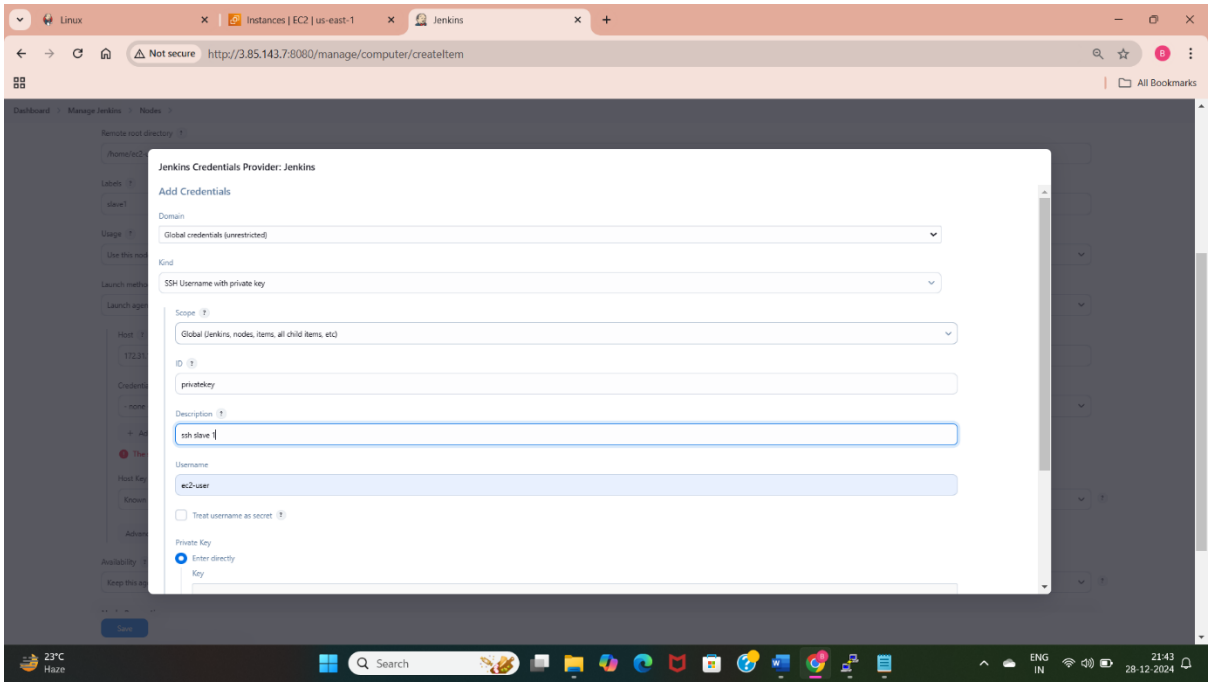
- To connect the master with the slave we need to copy the public key of instance and paste in slave server authorized_keys and save.



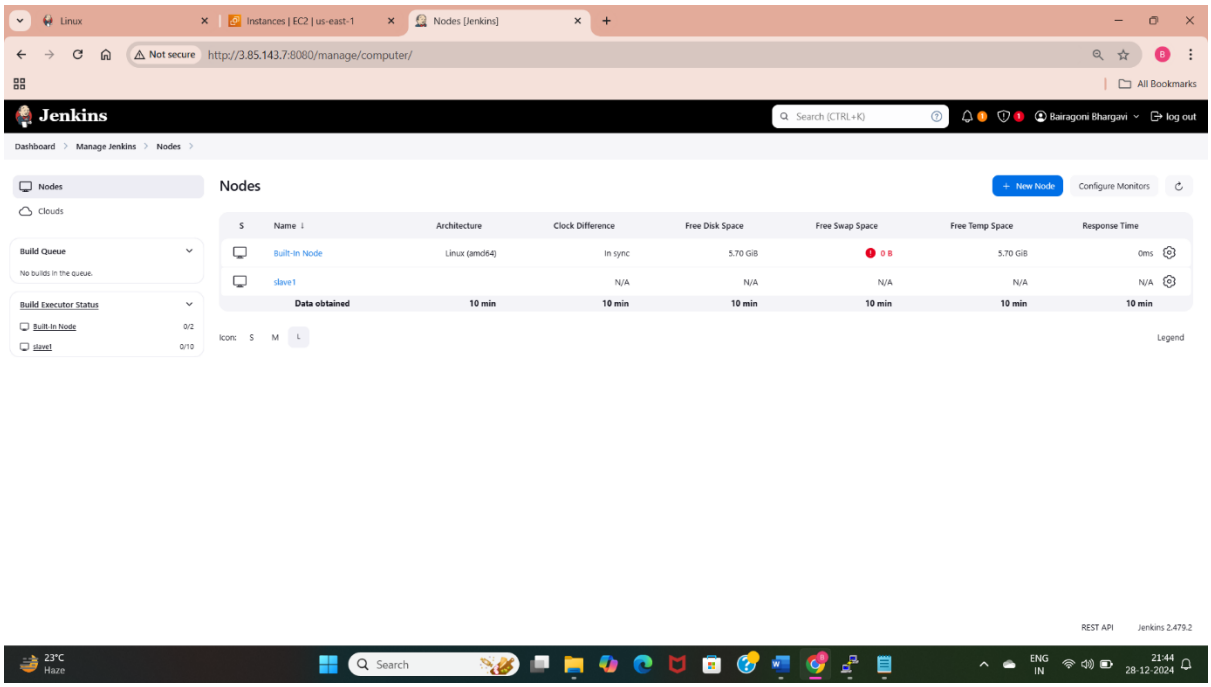
- ✓ Next we need to create a node in Jenkins using the above slave1 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave1”, select the number of executors(no. of builds) according to requirement.
- ✓ Give the remote root directory of the slave1 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



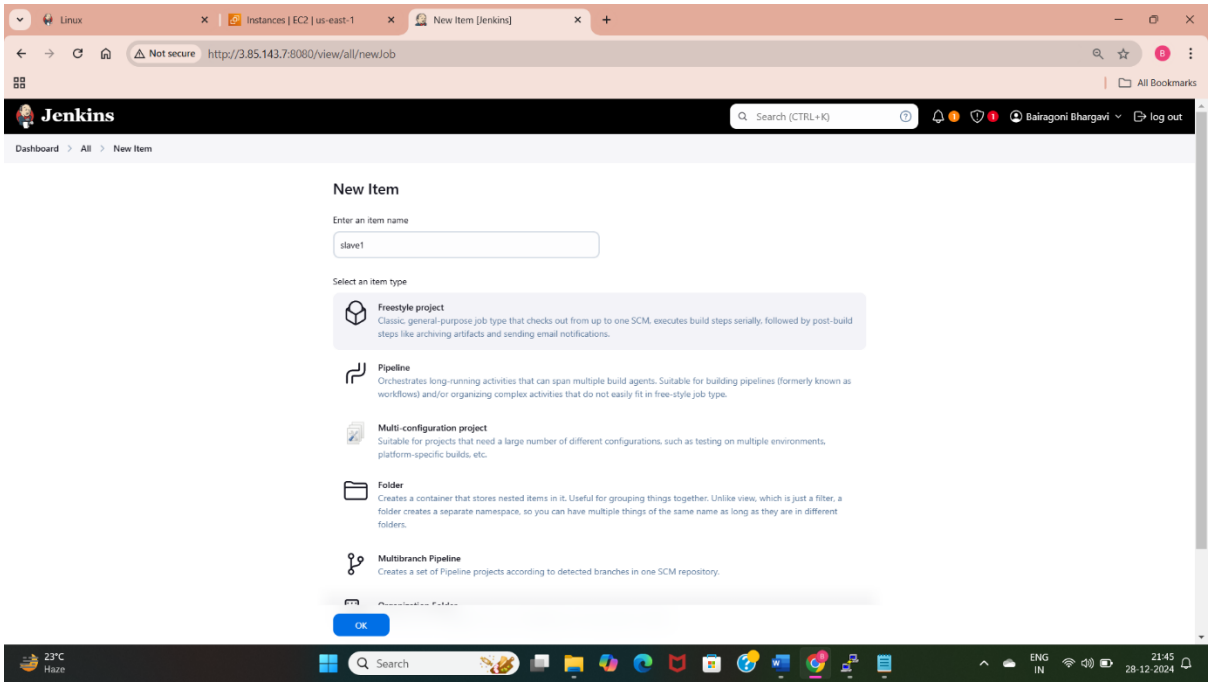
- Next select the host as private IP of slave1 server.
- Add credentials by following the below method.



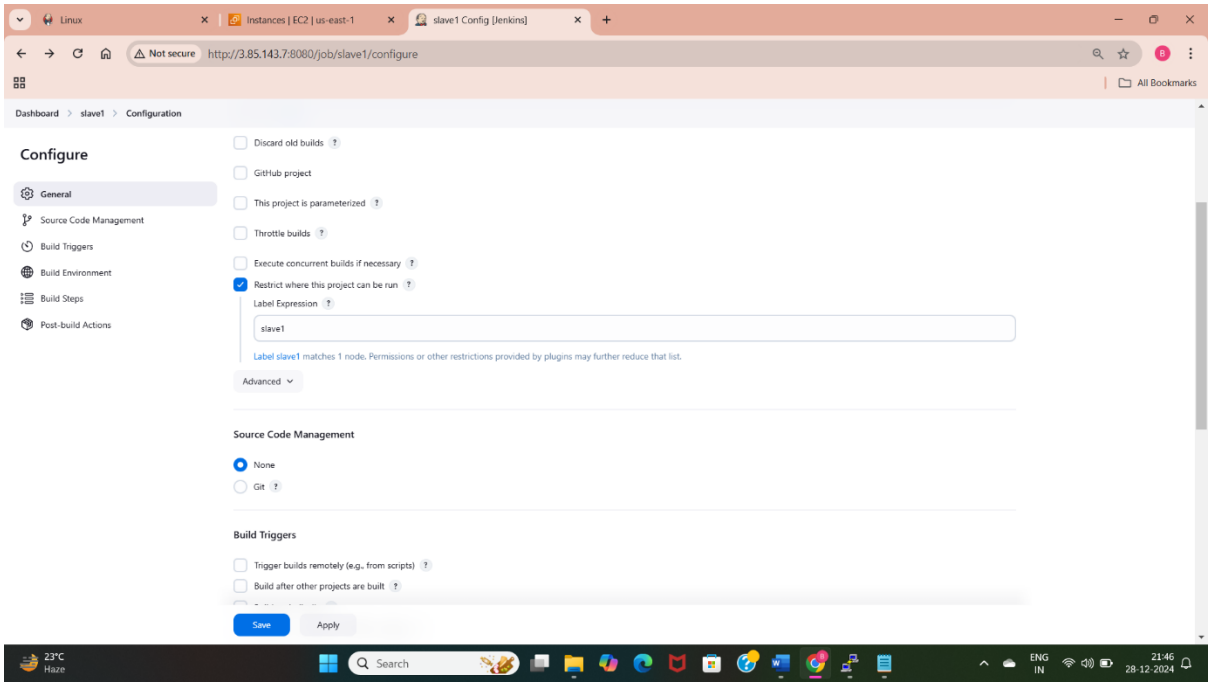
- Here am creating the credentials for this slave1 node. Select kind as “SSH Username with private key”.
- Next give any ID or description for this.
- Give the username. Here we need to give the username of our instance i.e ec2-user.
- Select “private key” and add the key using add option. Copy the private key which is stored in our local machine to Jenkins credentials, and save the changes.



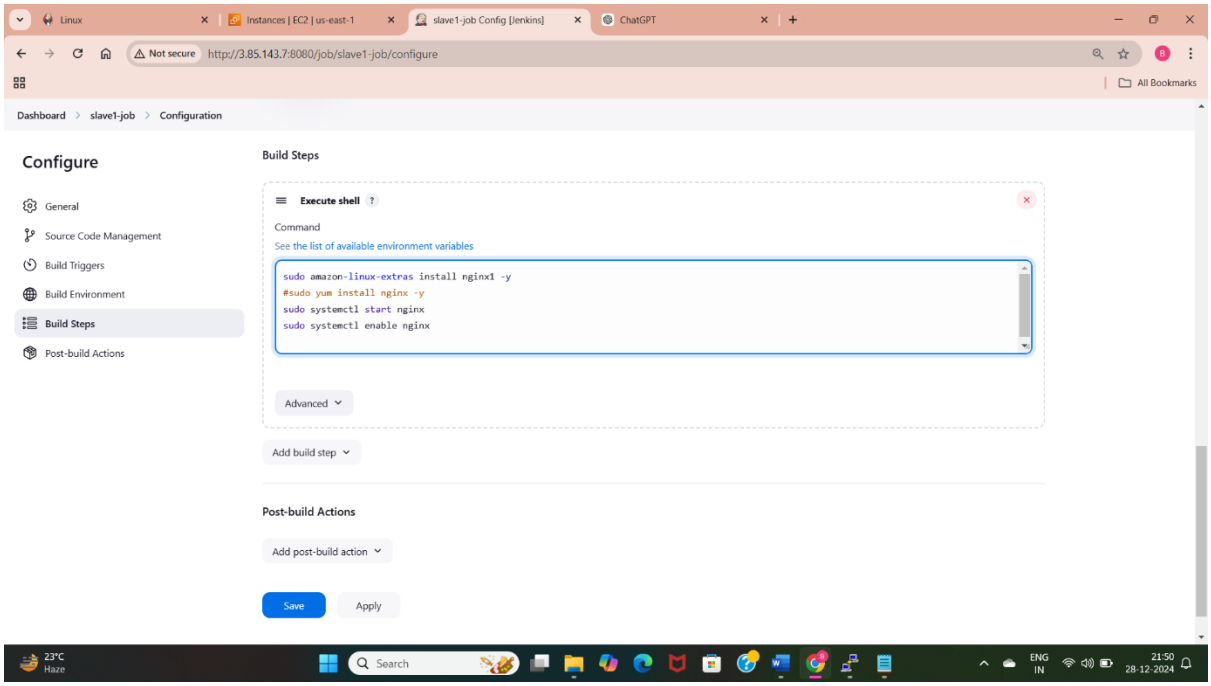
✓ This is the slave1 node which is created by me using master slave configurations with some configurations.



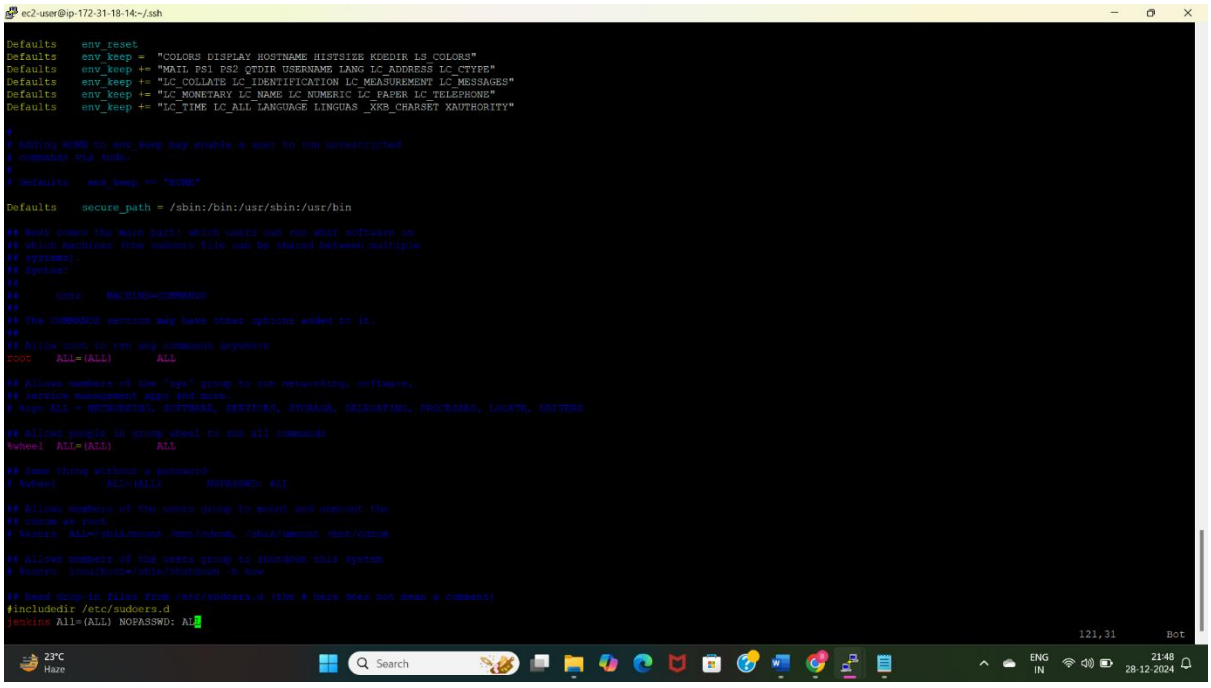
✓ Create a slave1 job by selecting new item, give the name for job ans select any one of the type like freestyle project.



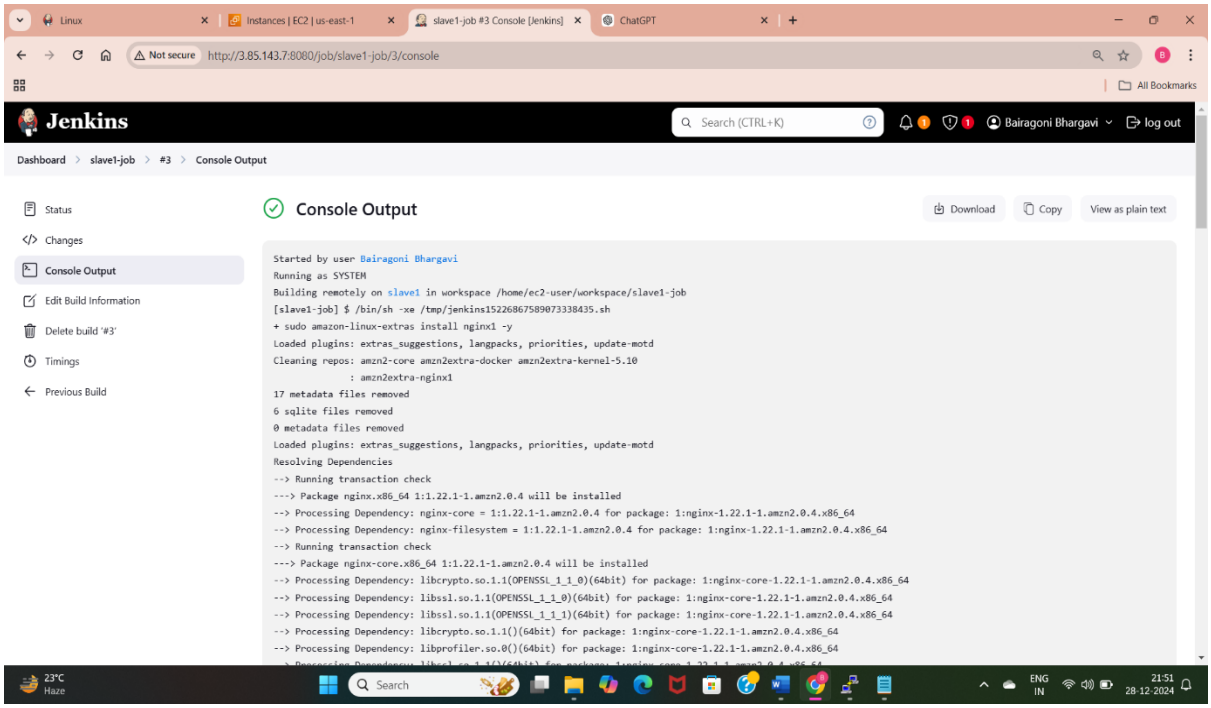
✓ Select the above created slave1 by using “Restrict where this project can be run” and select the label name which is created above.



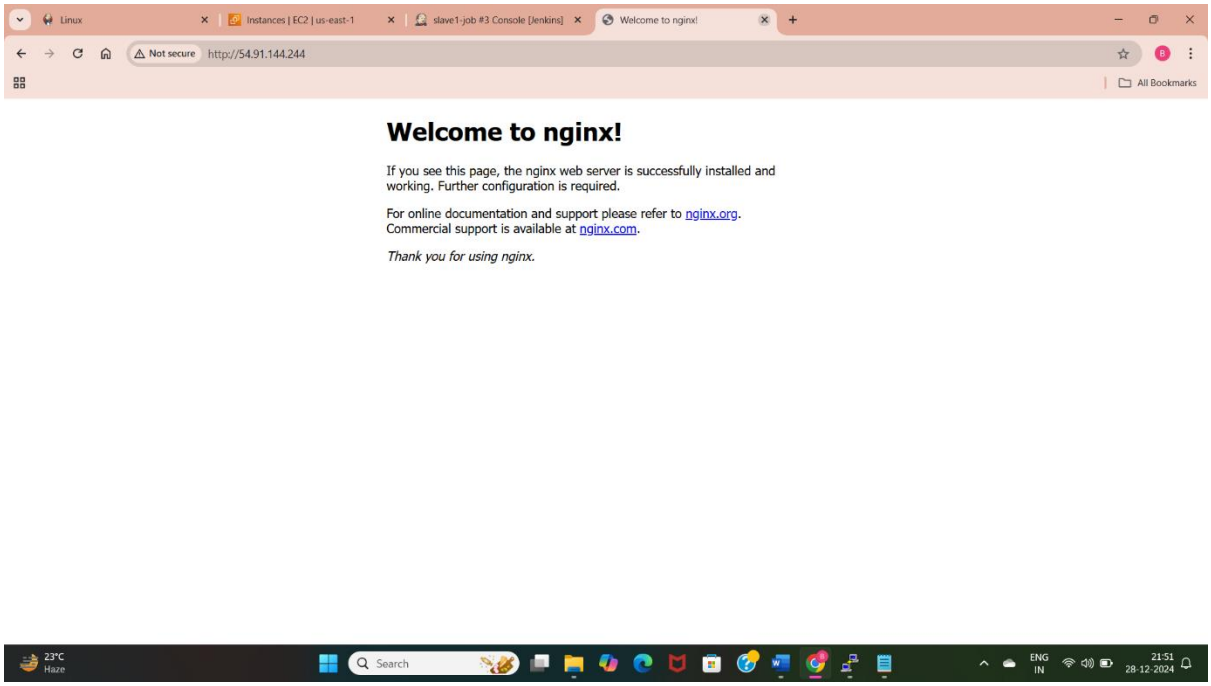
✓ Now go to build steps> select Execute shell and give the commands to install nginx , start nginx and save and build the job.



✓ Now give sudo permissions for Jenkins using “sudo visudo” and give the permission using “Jenkins ALL=(ALL) NOPASSWD: ALL”.

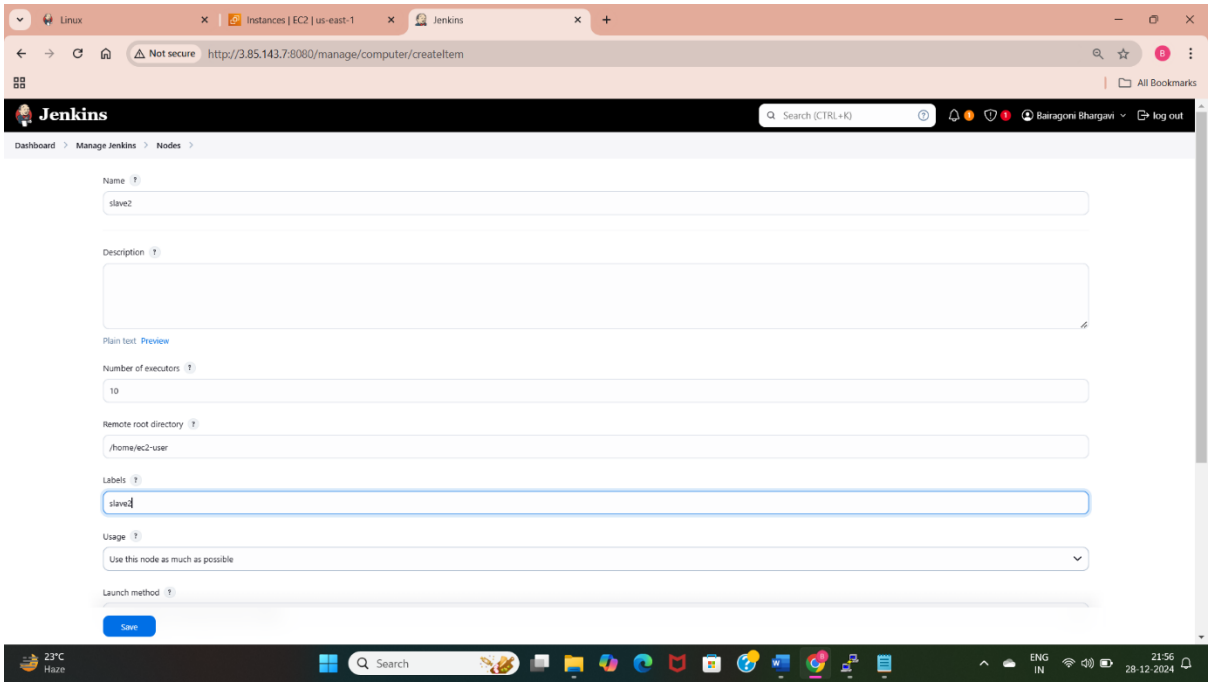


✓ Now we can see the build is started by the user Bairagoni Bhargavi. Building on the workspace of the Jenkins by moving into the job and running the above mentioned commands.

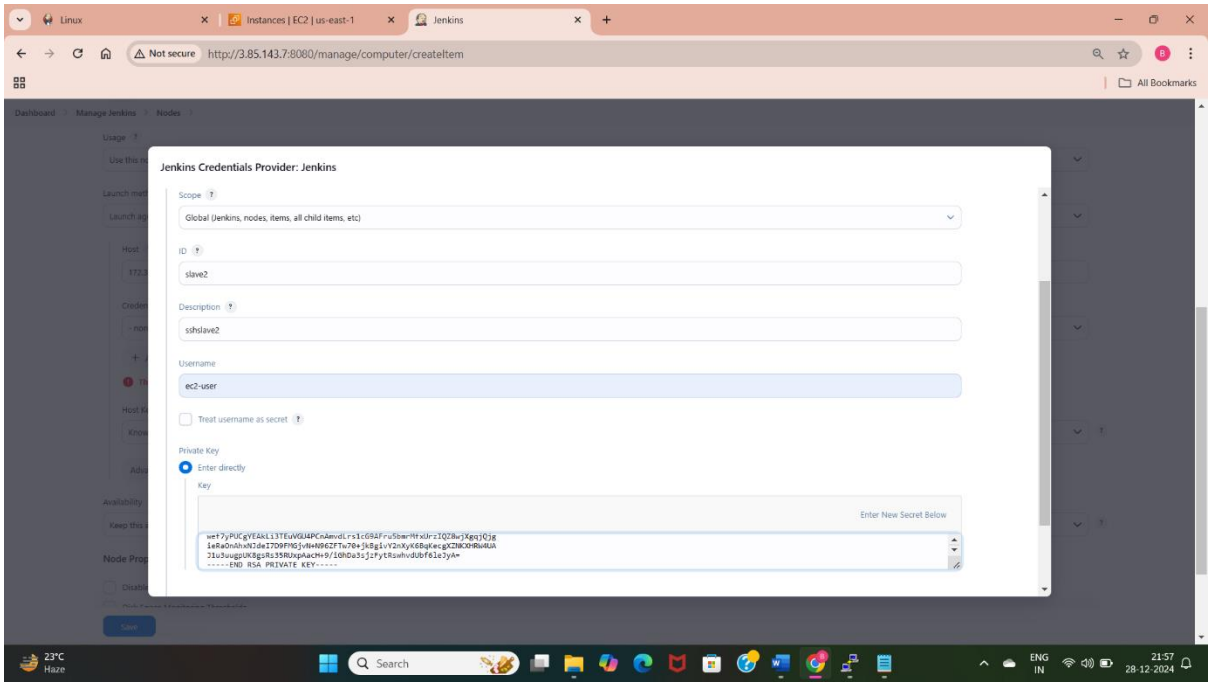


✓ After build is success we use the public IP address of instance in the browser so we can able to access the nginx webpage.

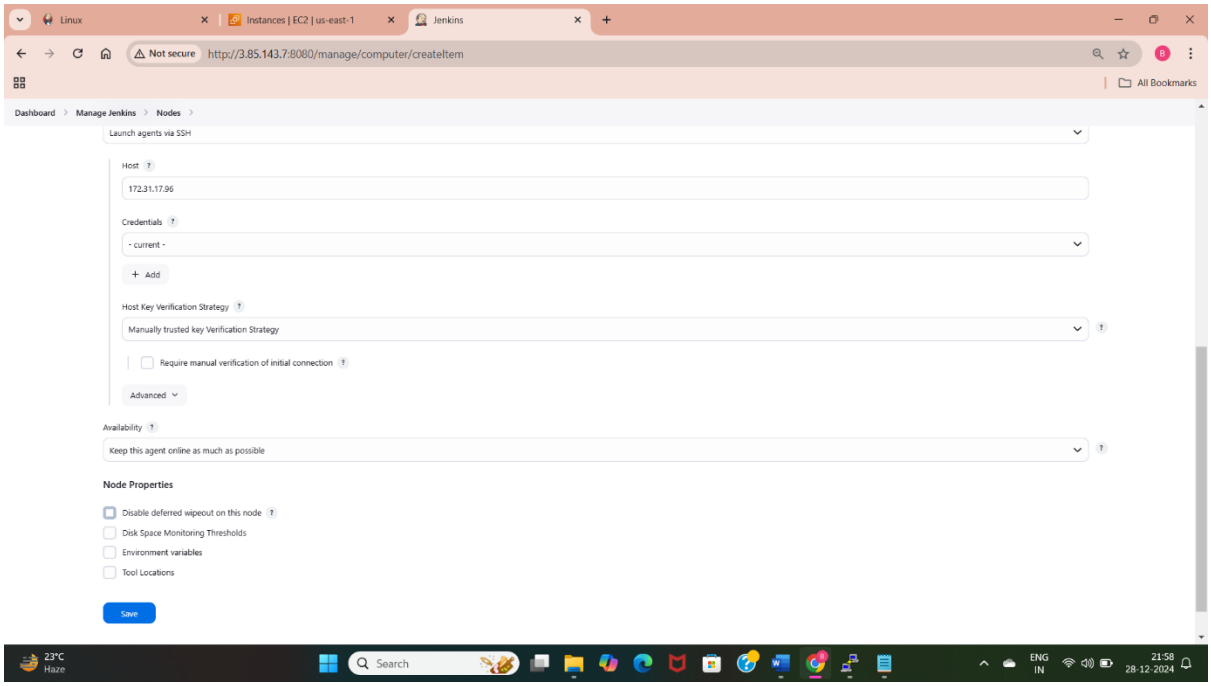
Slave-2:



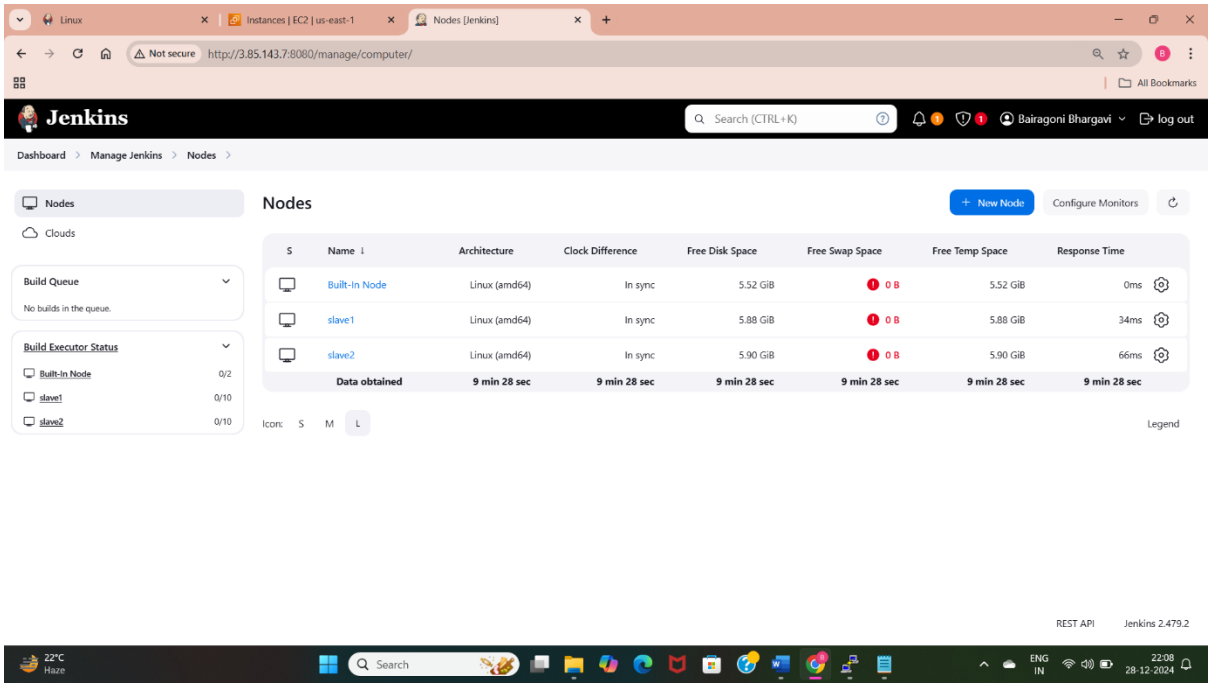
- ✓ Now am setting the master slave configurations between master and slave2. Here I have provided the name, selected 10 executers, root directory of the server, give any label name, select usage.



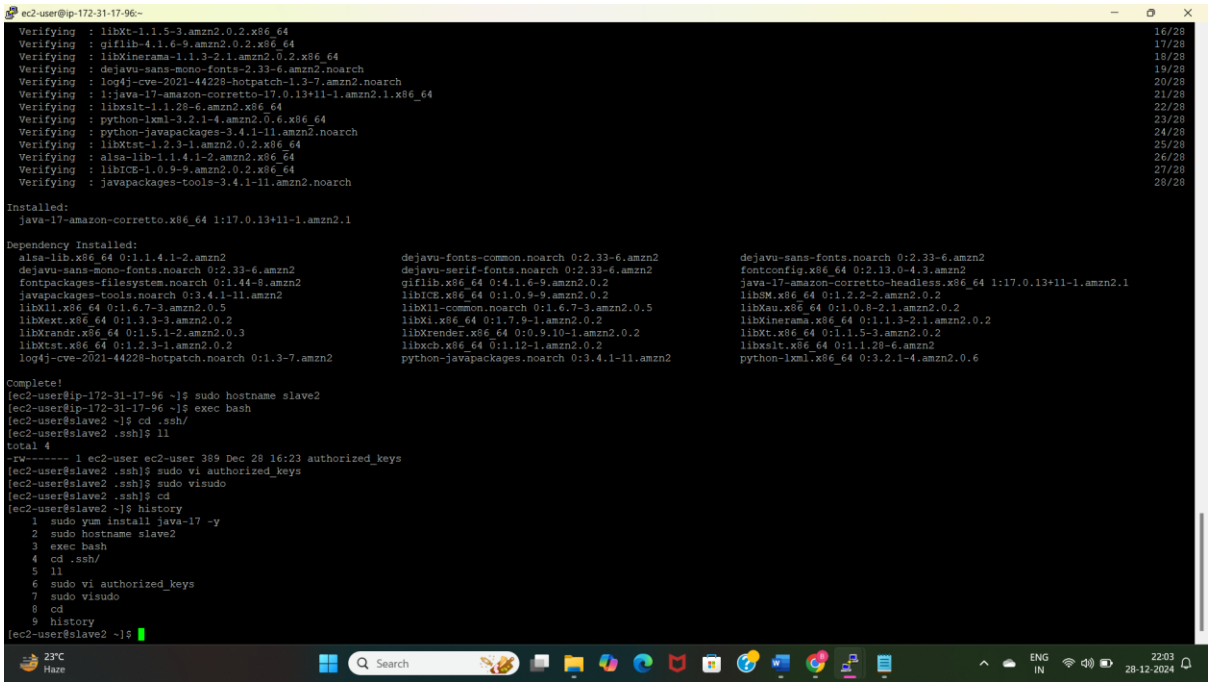
- ✓ Now provide the credentials to use the slave2 by using the SSH Username with Private key and provide private key there.



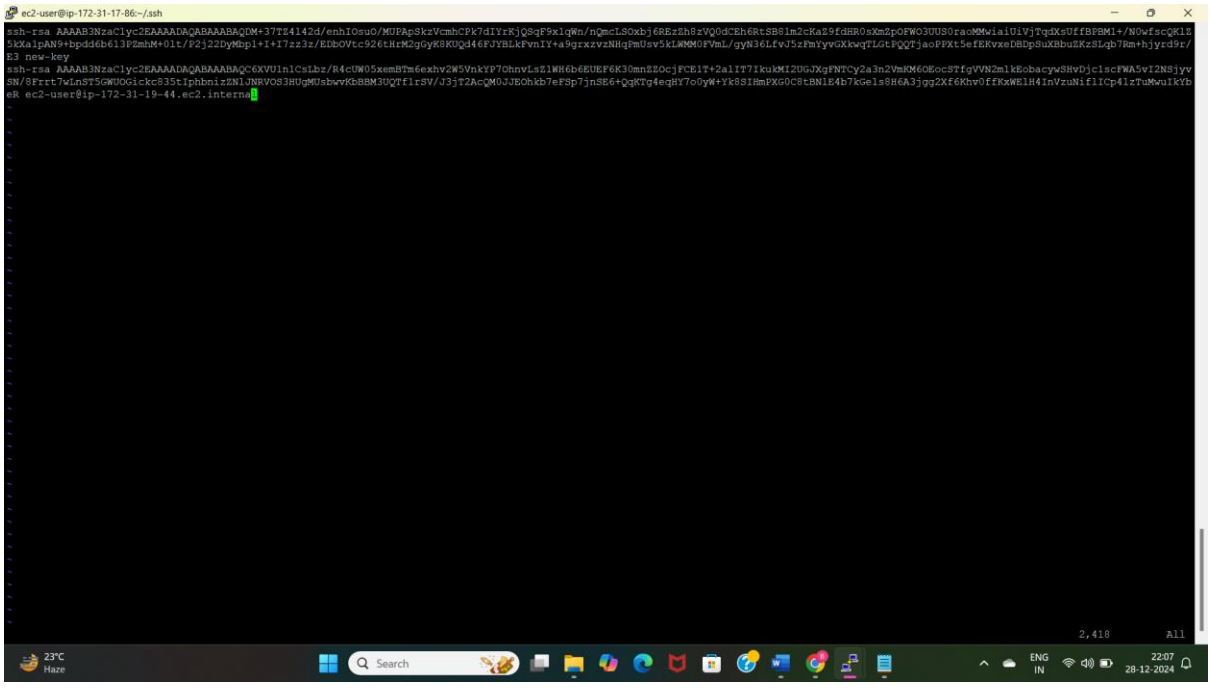
✓ Give the private IP address of the instance and add the created credentials, select host key verification strategy as “Require trusted key Verification Strategy”, select keep this agent online as much as possible.



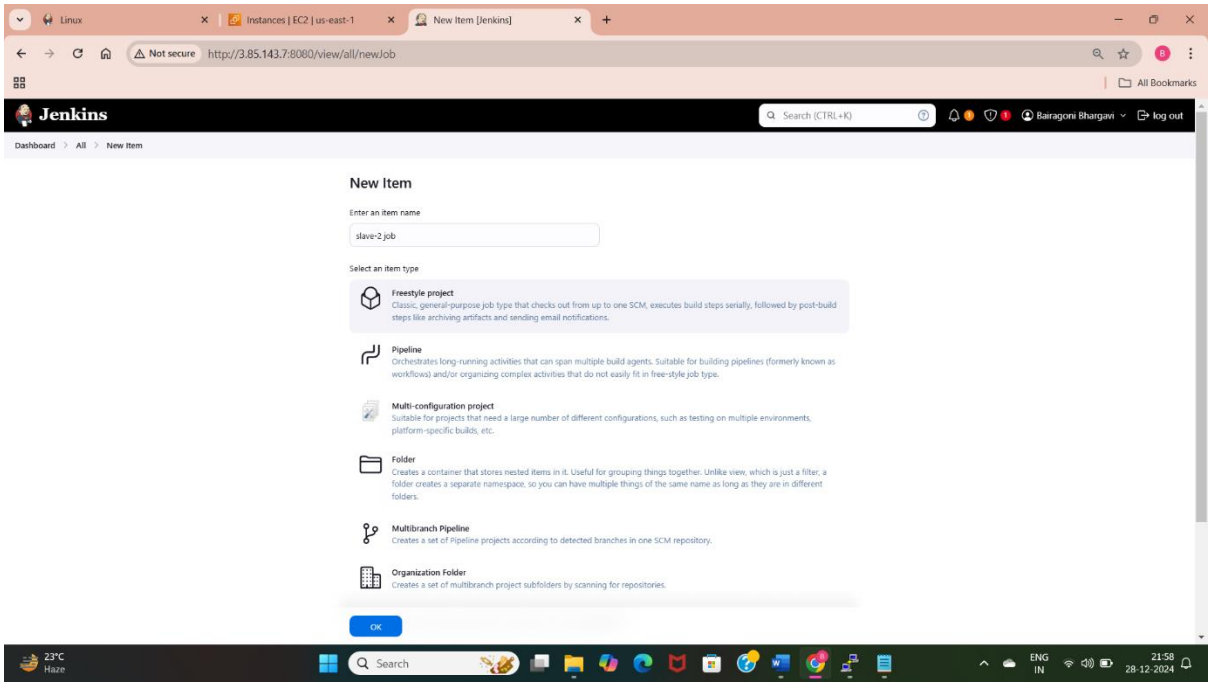
✓ We can see the slave2 node is created successfully with 10 executers.



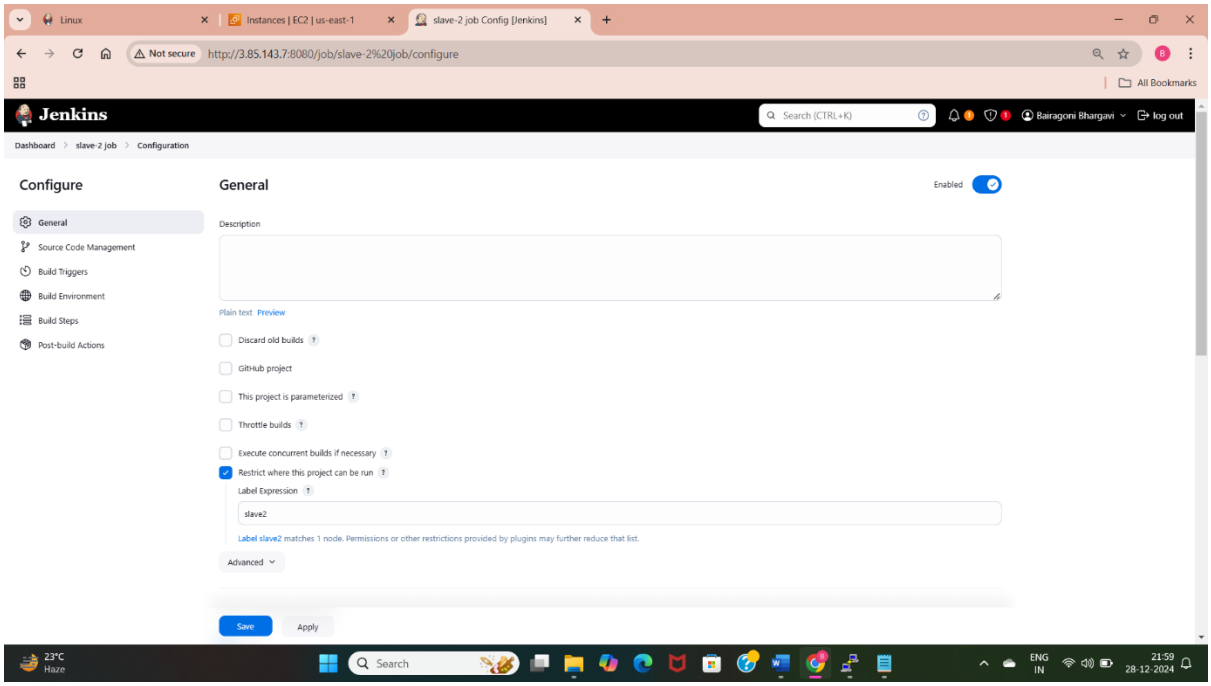
✓ Now am installing the java dependency in slave2 server.



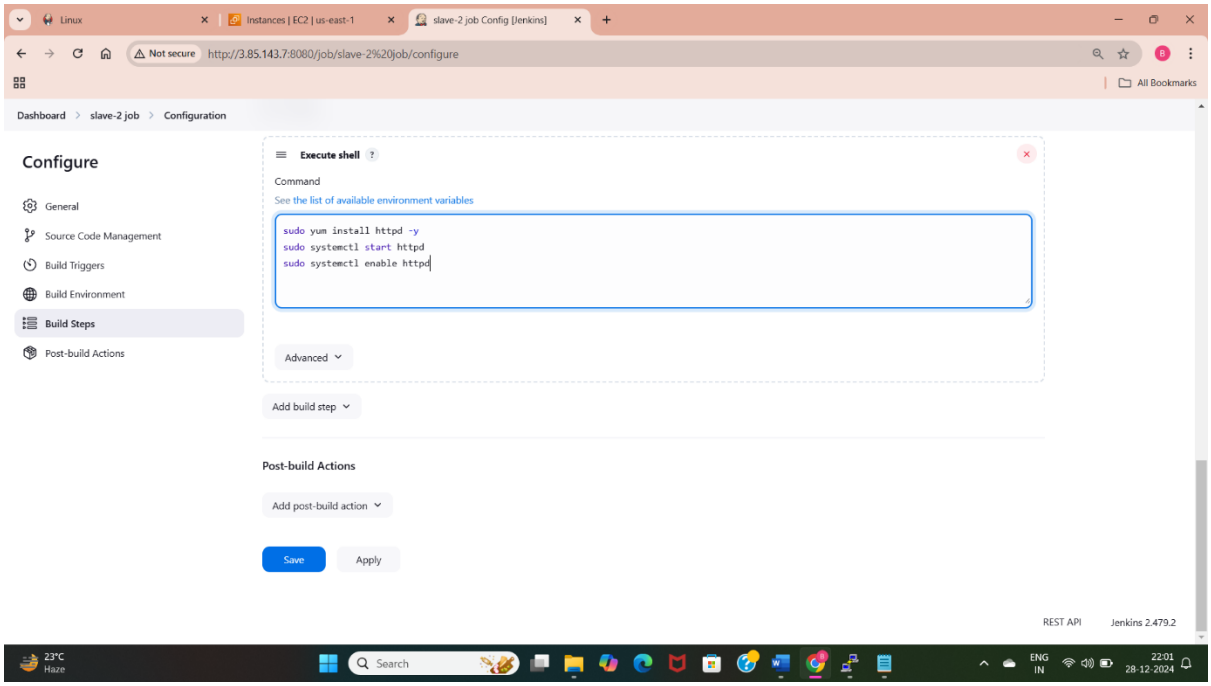
✓ Now am setting connection between master and slave by copying the public key of master into the authorized keys of slave2 server.



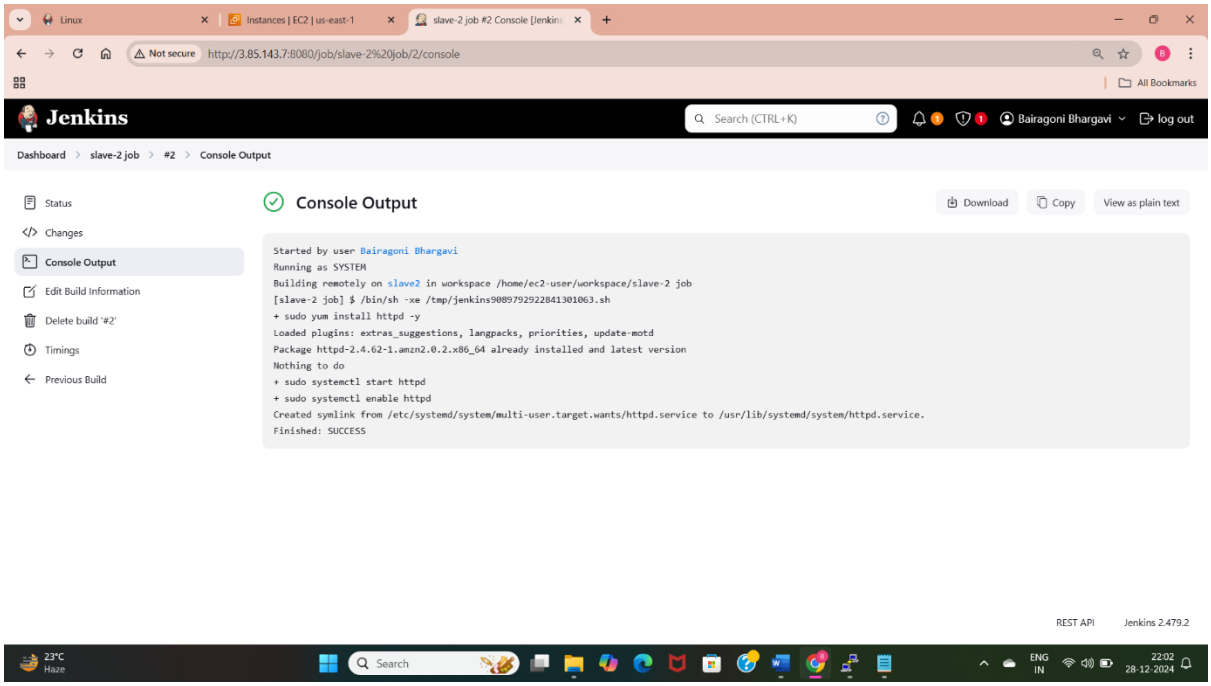
✓ Now am creating another job for slave2 using freestyle project and select ok.



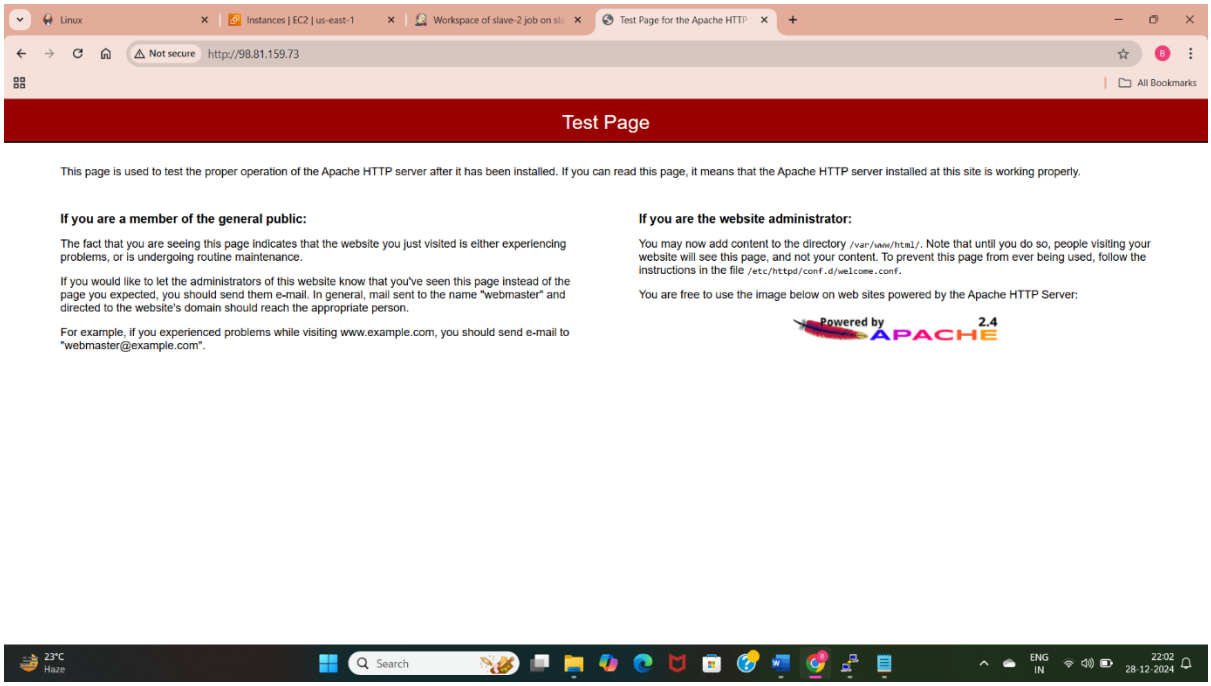
✓ Now select the where this job need to build. I am restricting the build by selecting slave2 label. Now whatever builds are executed are executed in this slave2.



✓ Now select the Build steps and select execute shell. Now give commands to install httpd using <yum install httpd -y>, start httpd using <systemctl start httpd> and save and build the job.



✓ Now we can see the build is success. We can also see the build is started by the user Bhargavi and executed the commands in Jenkins workspace where the job is stored.



- ✓ Now am able to access the httpd webpage using the public IP address of the instance.

Slave-3:

Sonarqube:

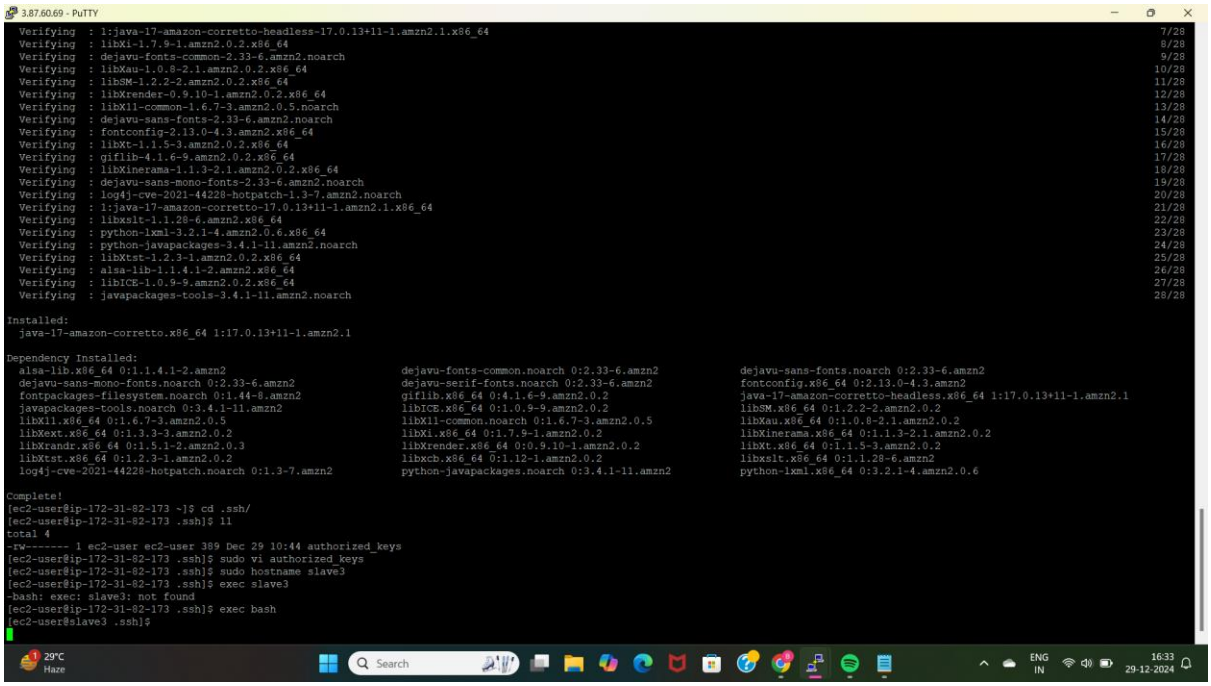
SonarQube is an open-source platform used for continuous inspection of code quality. It helps developers and teams to track, maintain, and improve the quality of their code over time by identifying potential issues, bugs, vulnerabilities, and code smells.

Here are the key features of SonarQube:

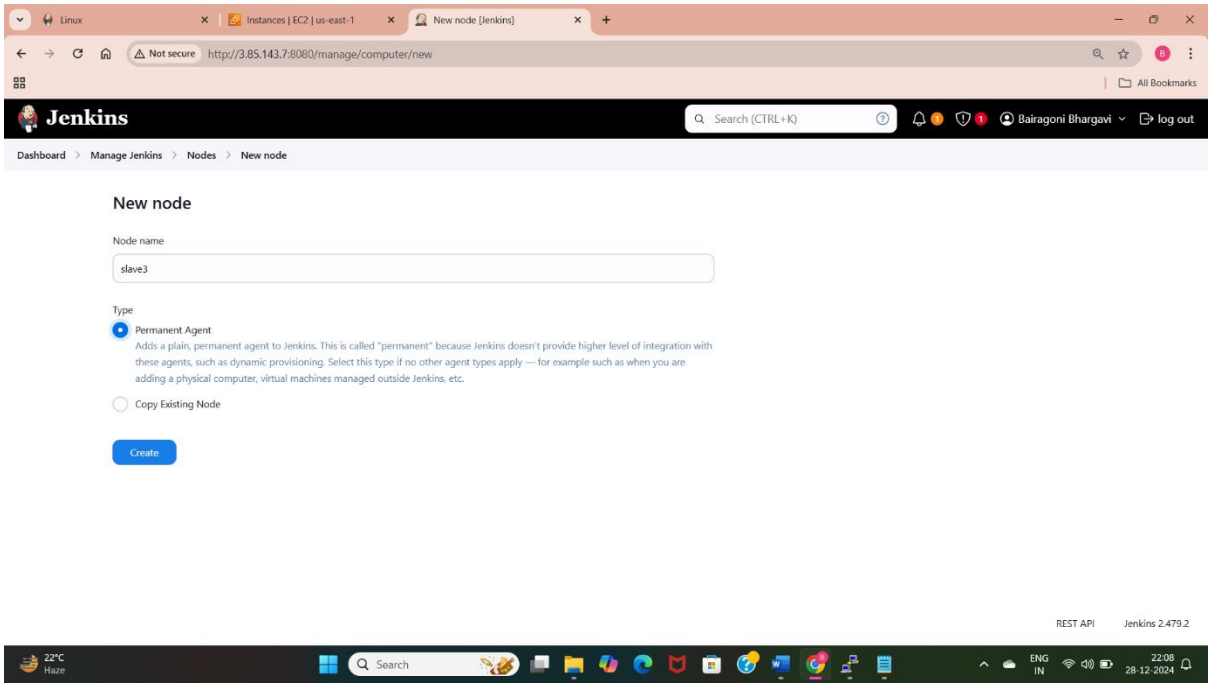
Code Quality Analysis

SonarQube performs an automated review of your code to check for quality issues. It analyzes various aspects of your code, such as:

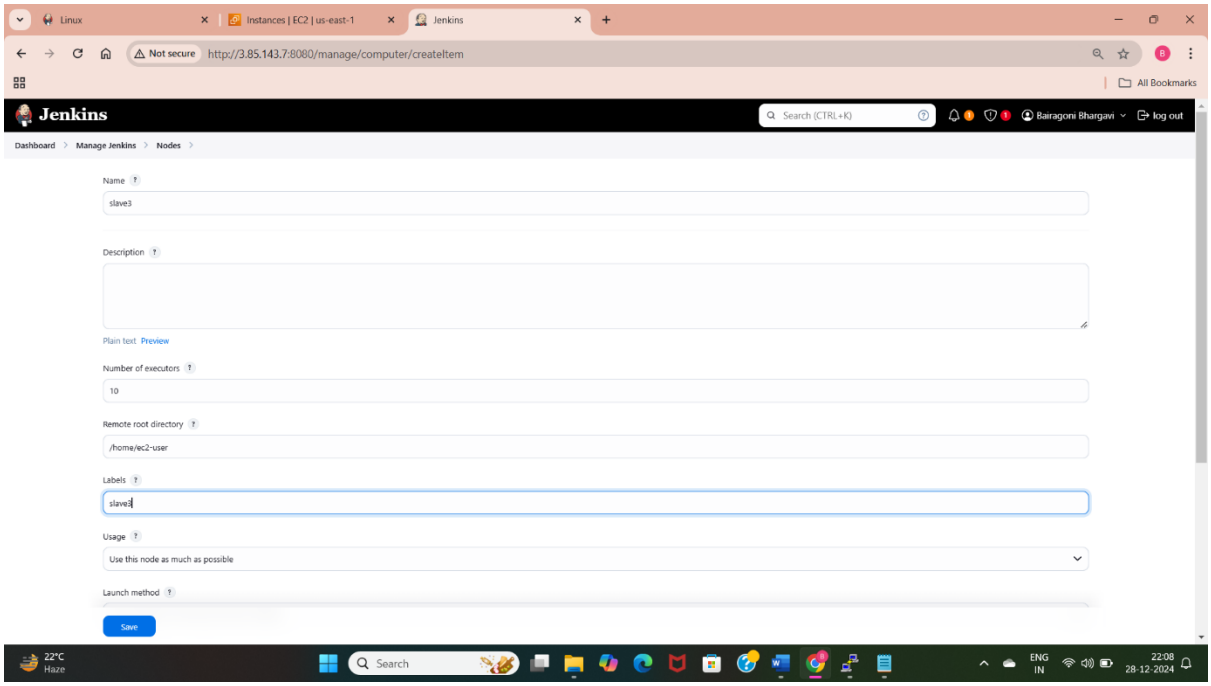
- ☑ Bugs: Defects that might cause errors or unexpected behavior.
- ☑ Vulnerabilities: Security issues that can lead to potential exploits.
- ☑ Code Smells: Areas where the code might be hard to maintain or understand.
- ☑ Duplications: Repeated code blocks that can be refactored.
- ☑ Test Coverage: Measures the percentage of your code covered by tests.
- ☑ Complexity: Evaluates the complexity of the code to help in maintaining it easily.
- ☑ SonarQube can be integrated with popular CI tools like Jenkins, GitLab CI, Azure DevOps, etc. After every build, it can analyze your code to catch issues early in the development process.



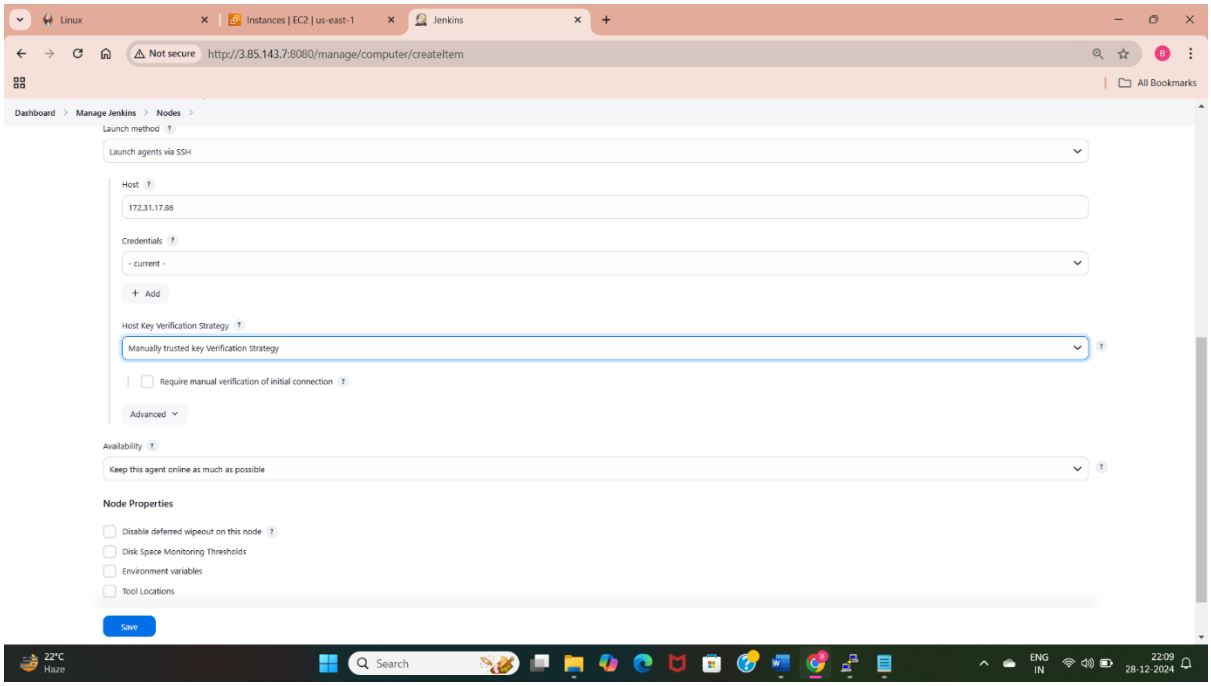
- Now connect the third server with terminal and install the runtime of Jenkins that is java 17. Copy the public key of master and paste it in the slave3 server authorized keys.



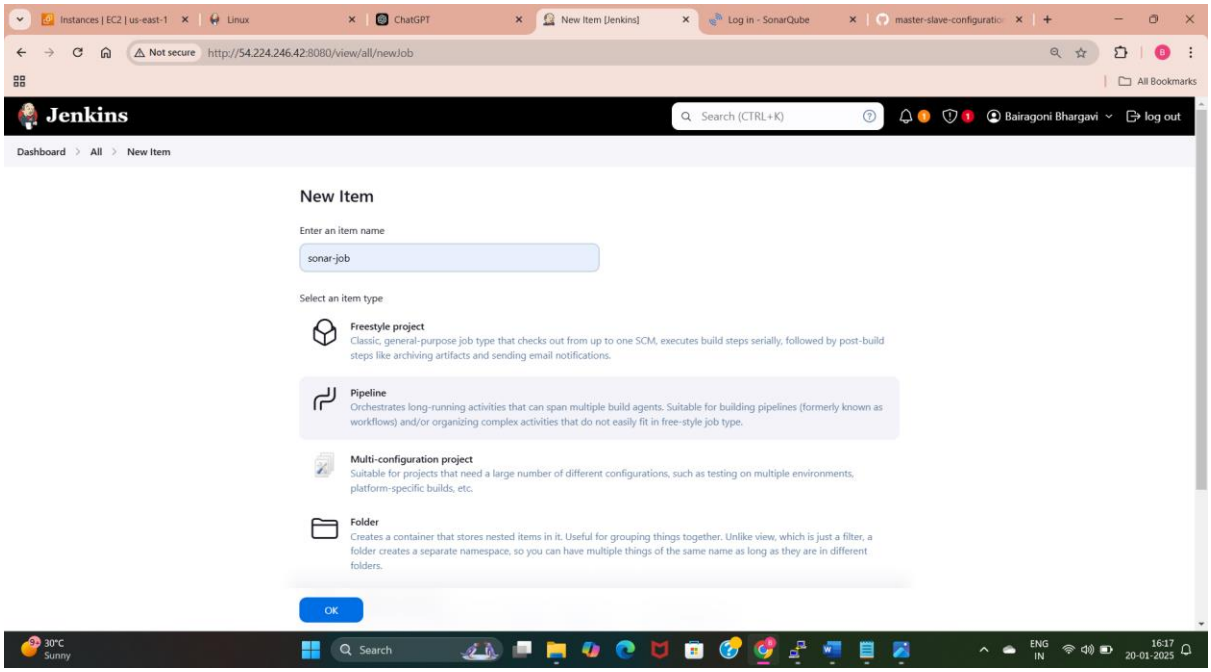
- Now create a slave3 node, by moving into Manage Jenkins> Nodes> new node name, and select permanent agent.



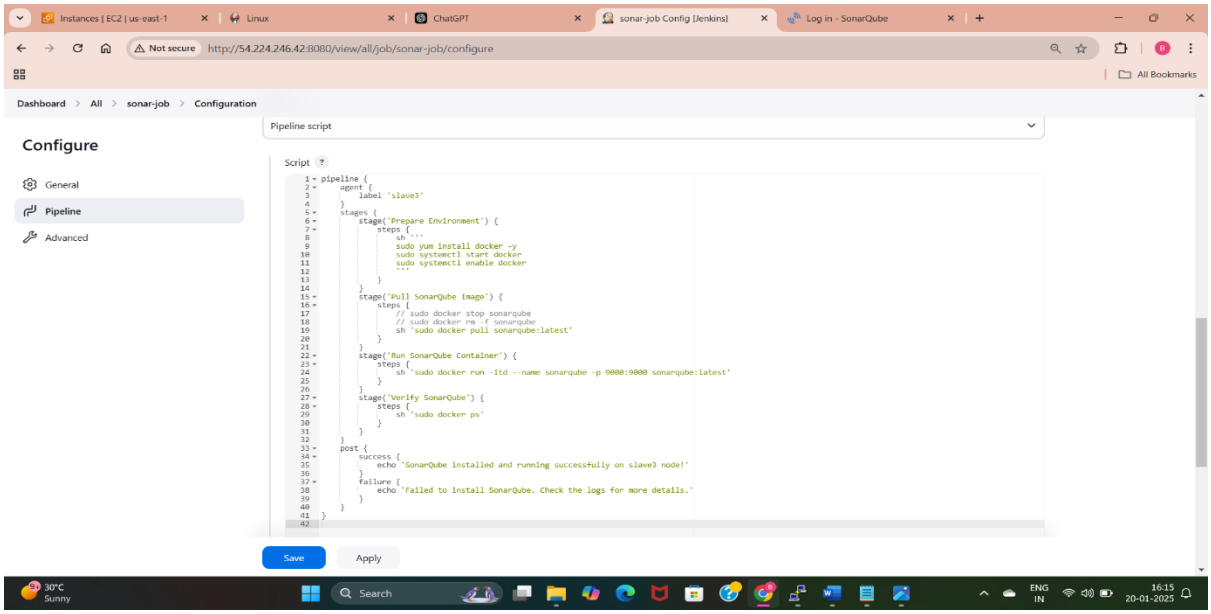
- Now give the name as slave3, select the number of executers, and provide root directory of slaver server, select use the node as much as possible as usage.



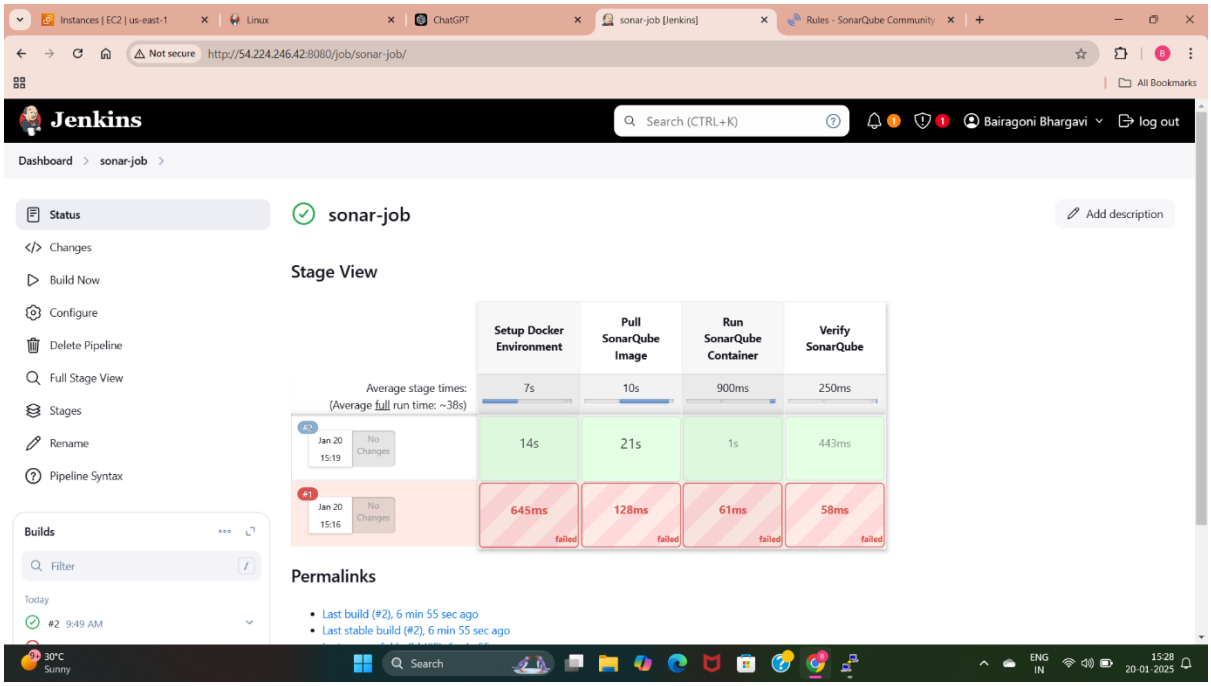
- In launch method select “launch agent via SSH” and provide private Ip address of slave4 instance and add credentials by using add option. Manually trusted key verification strategy is used as we are launching agent via SSH.

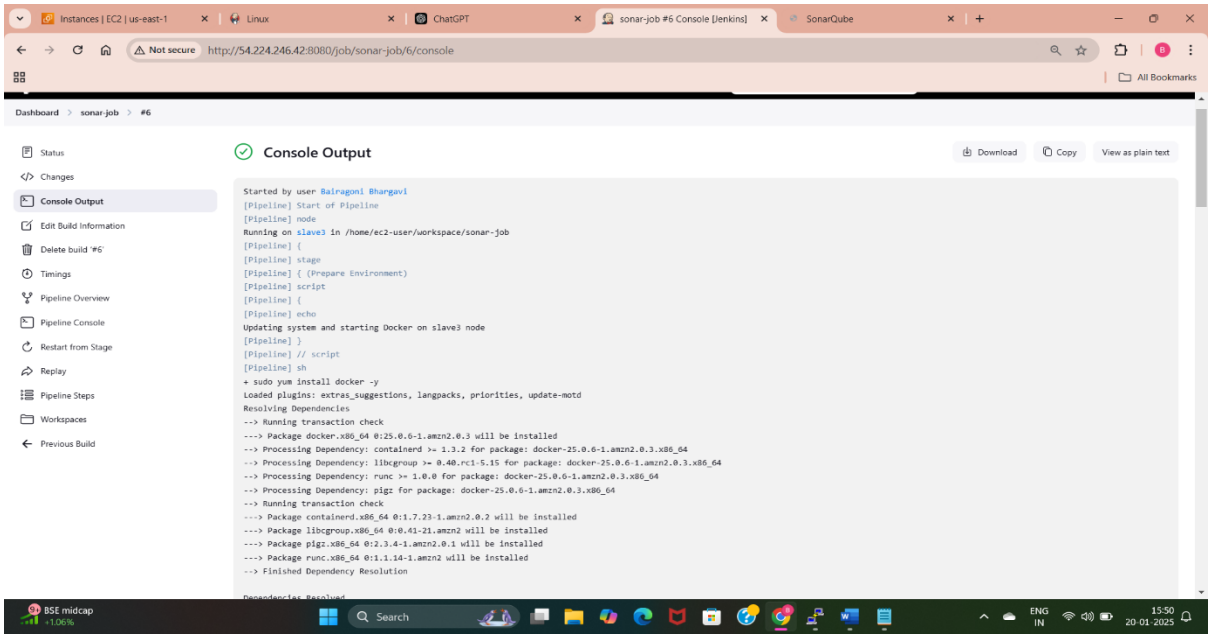


- Now create a new pipeline job for slave3 to install sonarqube and start using sonarqube webpage.

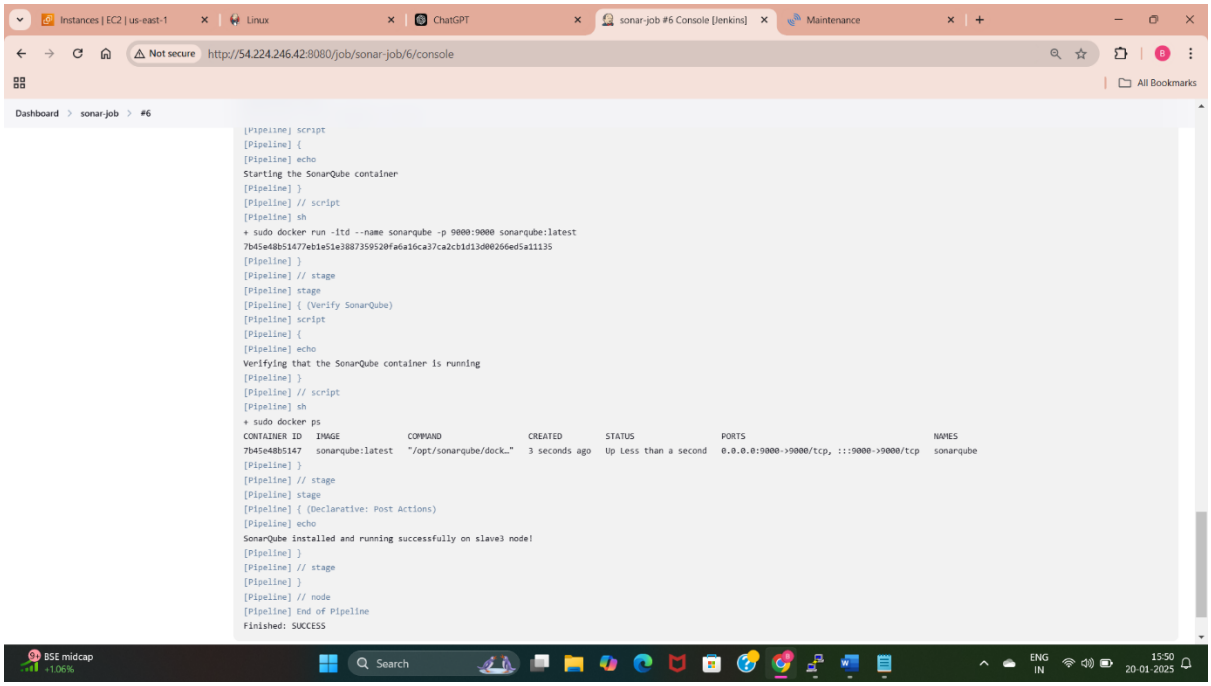


- Here am selecting the slave3 label which is created while creating slave3 node under agent in the pipeline. I have written the pipeline to install docker and pulled sonarqube image from docker.
- Now select build steps, in build steps select execute shell. In execute shell am providing commands to download the sonarqube.
- First am taking the sonarqube link from browser and downloading it using wget, downloaded as zip file.
- Now unzip the zip file of sonarqube.
- Am changing the downloaded and unzipped file name to sonar using mv command.
- Move into sonar file inside this we are having bin. Get inside the bin and there we are having linux-x86-64 file go inside this file and start sonar using “sh sonar.sh start”.

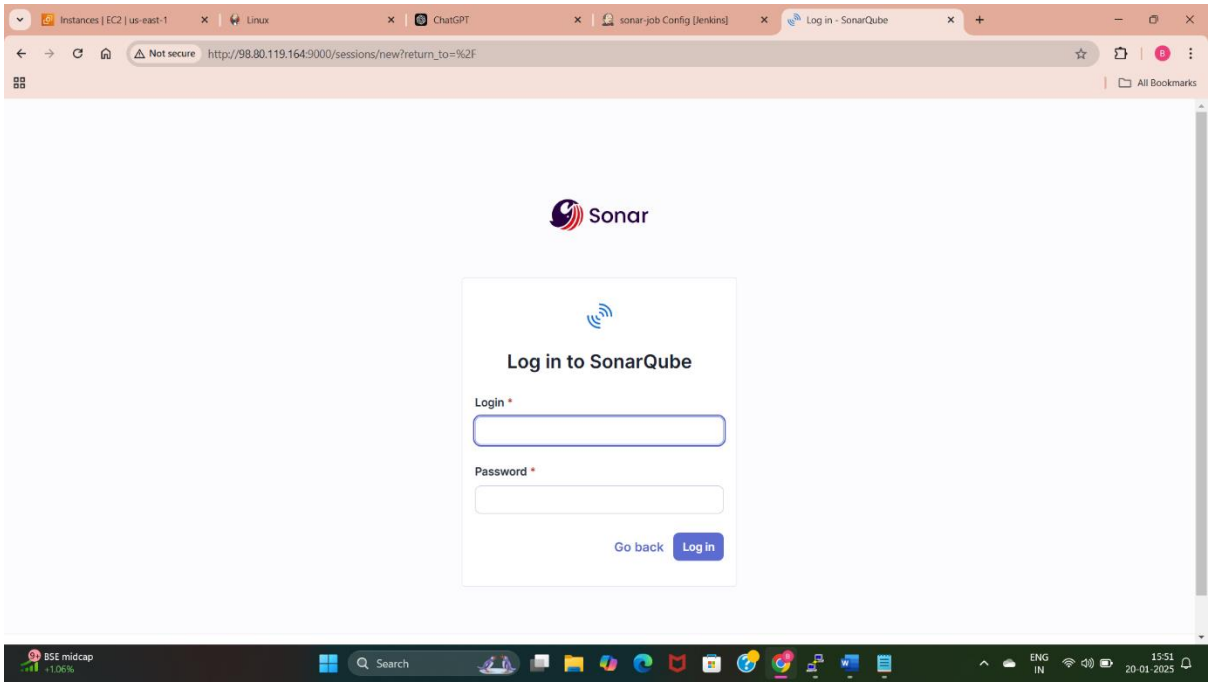




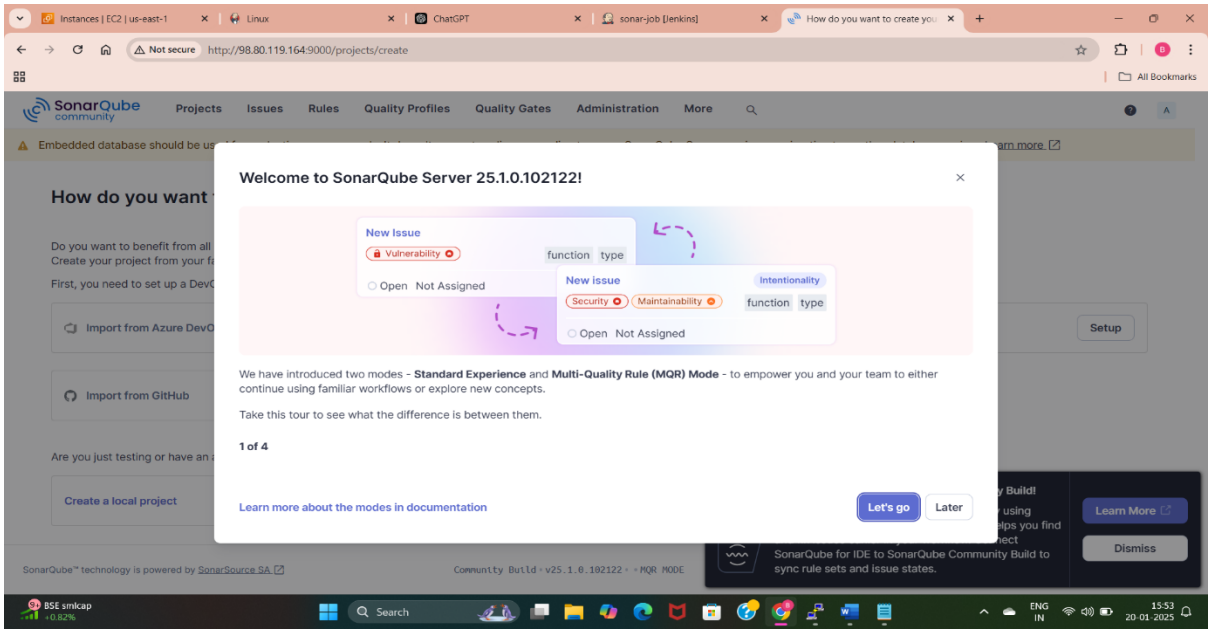
- Now after writing commands in execute shell save and click on “build now” option. Then the build will start and executes the commands. We can see the build is success.



- Now when build is success we can observe that the sonarqube is installed and started.
- After building the job we can check whether the sonarqube files are downloaded or not by moving into the Jenkins workspace and check with the process.



- After build is success copy the public ip of instance and paste it in the browser with sonarqube port number 9000 and check whether we can able to access the sonar page or not. Here am able to access the



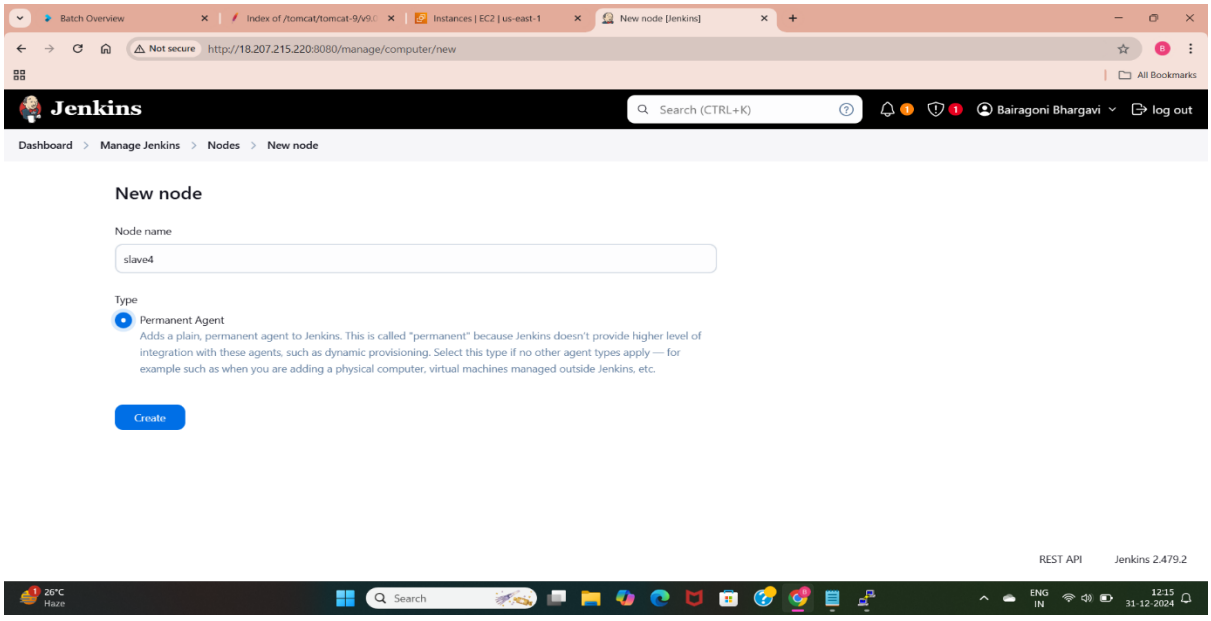
- This is the sonar page which is deployed using the Jenkins execute shell.

Slave-4:

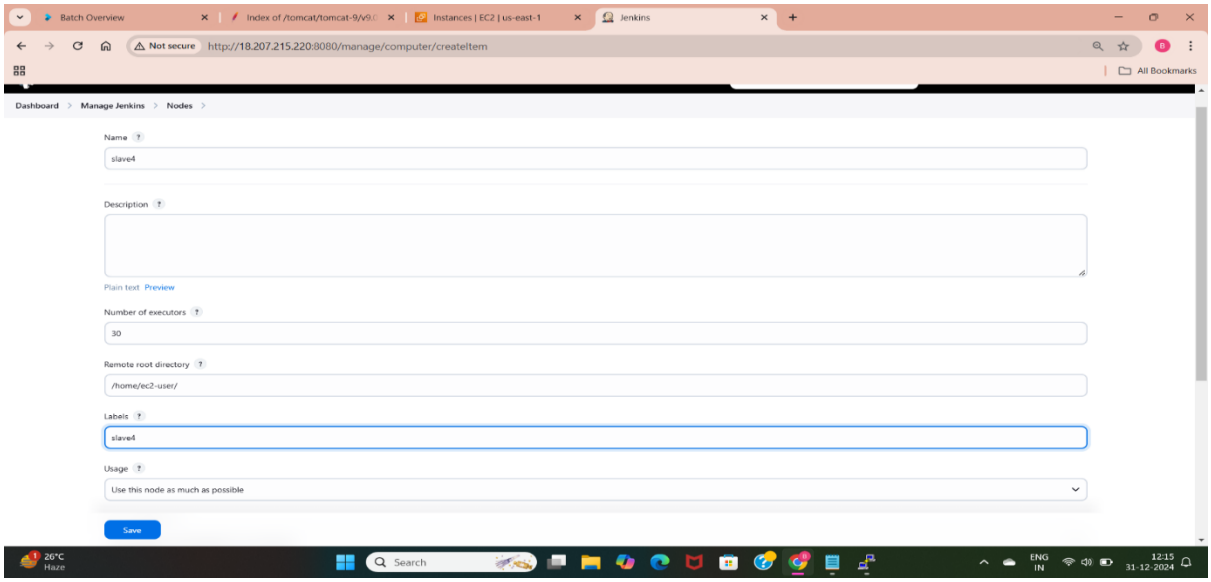
Tomcat:

Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation. It is primarily used to serve Java-based web applications and is often associated with running Java Servlets and JavaServer Pages (JSP).

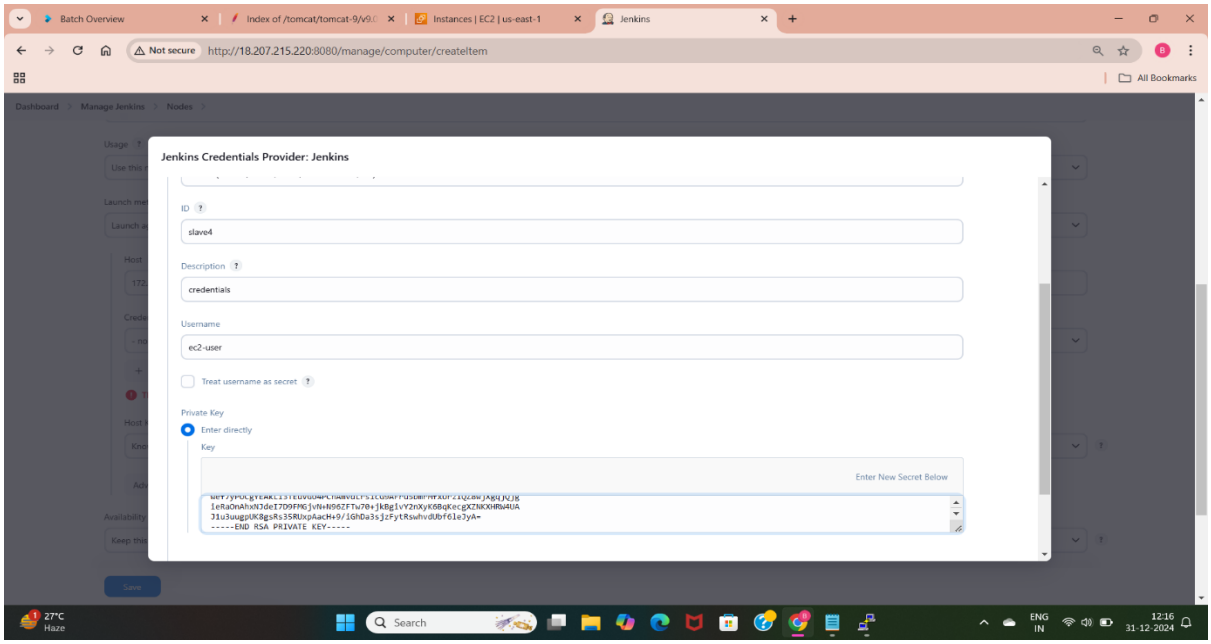
Go to Manage Jenkins > Manage Nodes and Clouds > New Node>Create Slave Node.



- ♣ Enter a name for the slave node and select Permanent Agent.
- ♣ Click OK to configure the slave node.

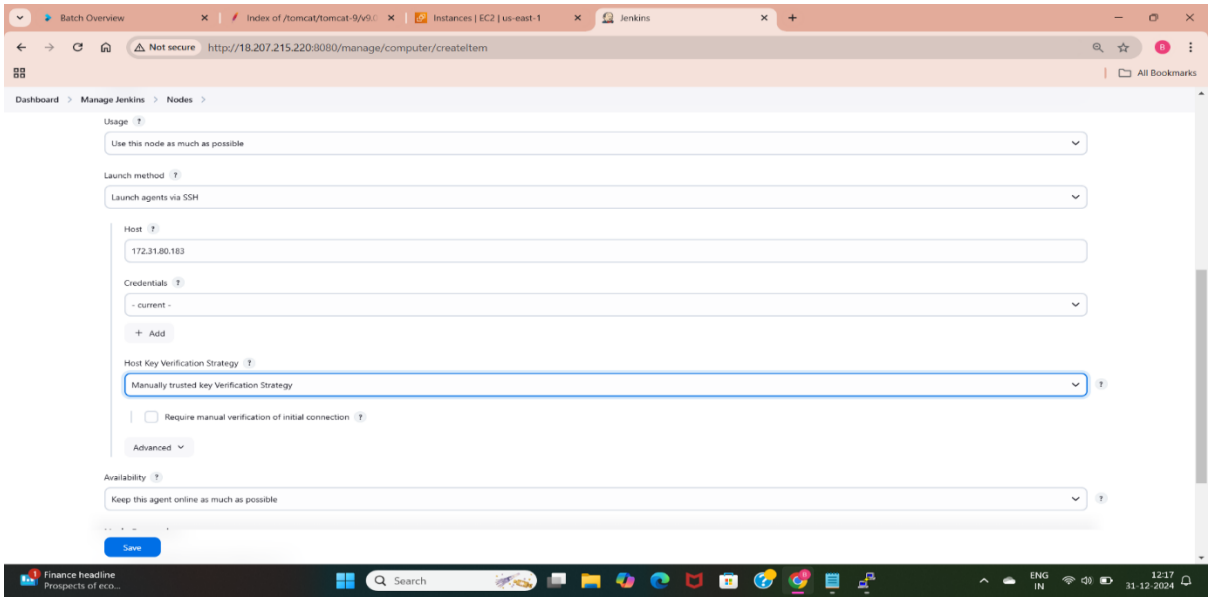


- ♣ Configure Node Settings:
- ♣ Number of Executors: Specify the number of builds the agent can execute simultaneously (e.g., 10).
- ♣ Remote Root Directory: Provide the directory where Jenkins will store job files on the slave (e.g., /home/ubuntu or /home/ec2-user).
- ♣ Labels: Assign a label for the slave (e.g., slave1, test-agent). This will help identify the node for specific jobs.
- ♣ Usage: Choose "Use this node as much as possible".



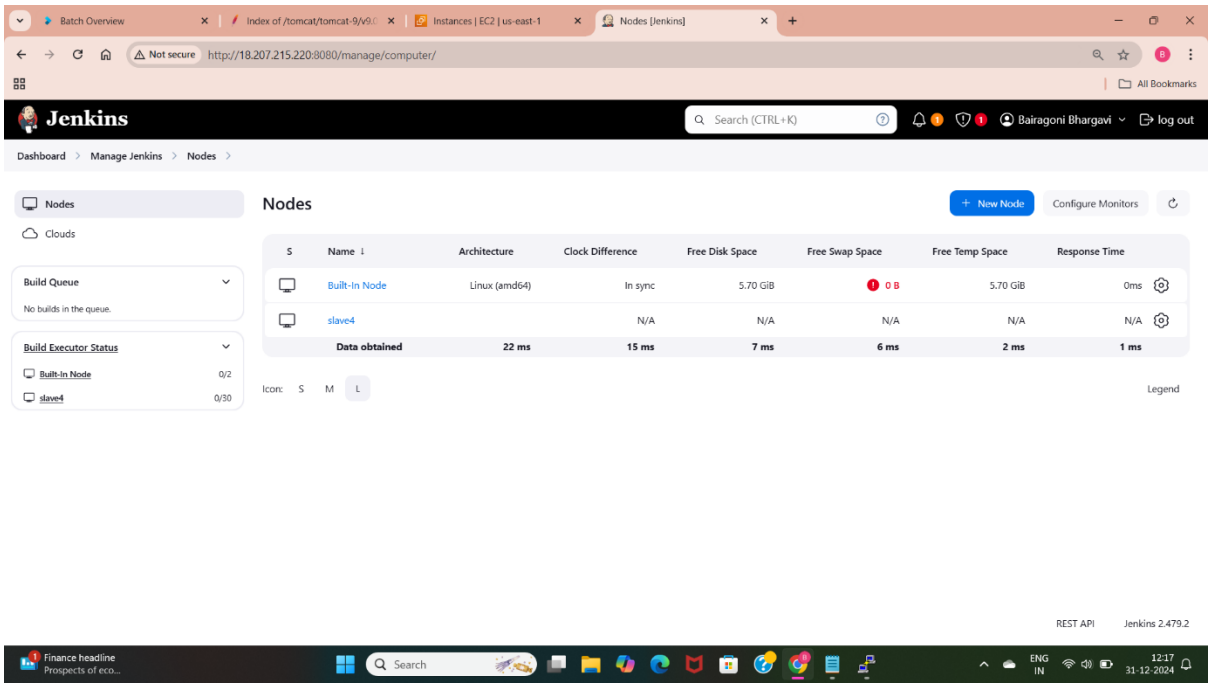
Set Launch Method:

- ♣ Select "Launch agents via SSH".
- ♣ Now provide the credentials to use the slave4 by using the SSH Username (ec2-user if you use amzazon linux) with Private key and provide created master private key there.



- Now select the launch method, provide private Ip address of slave4 server, add credentials using above created credentials, verify credentials by selecting the manually trusted key verification strategy.

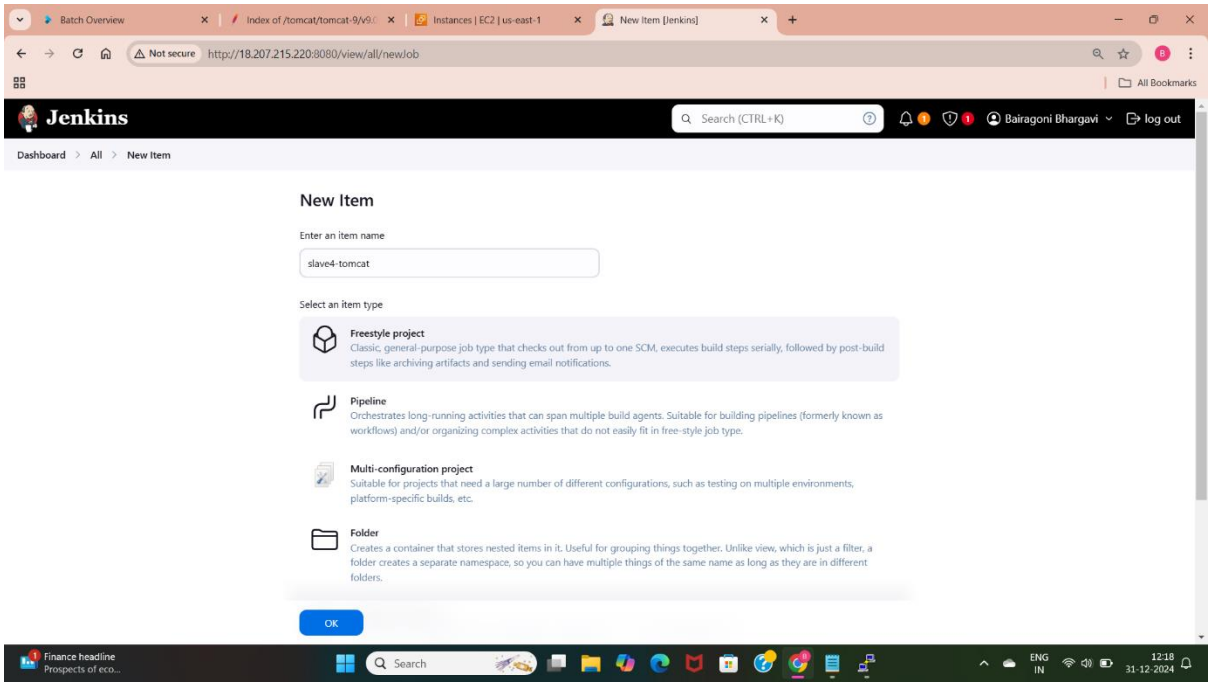
Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.



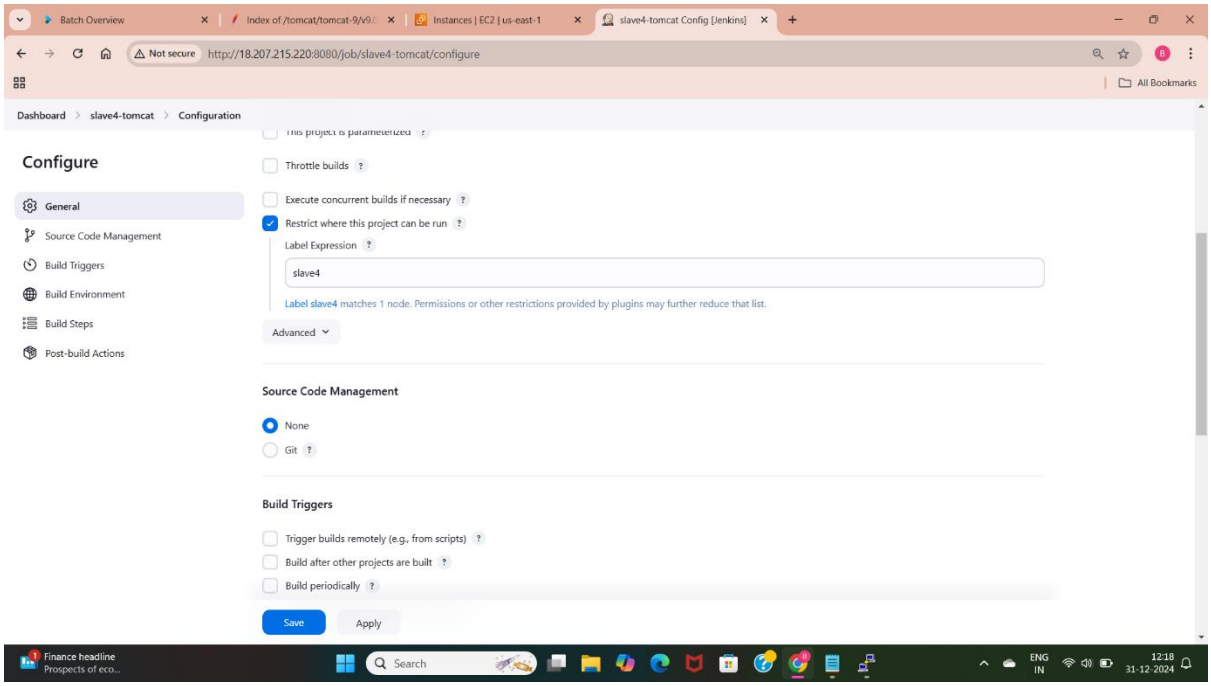
- Now we can see a new node is created with the name slave4. and Jenkins successfully connected with slave4.

Running Jobs on Slave Nodes:

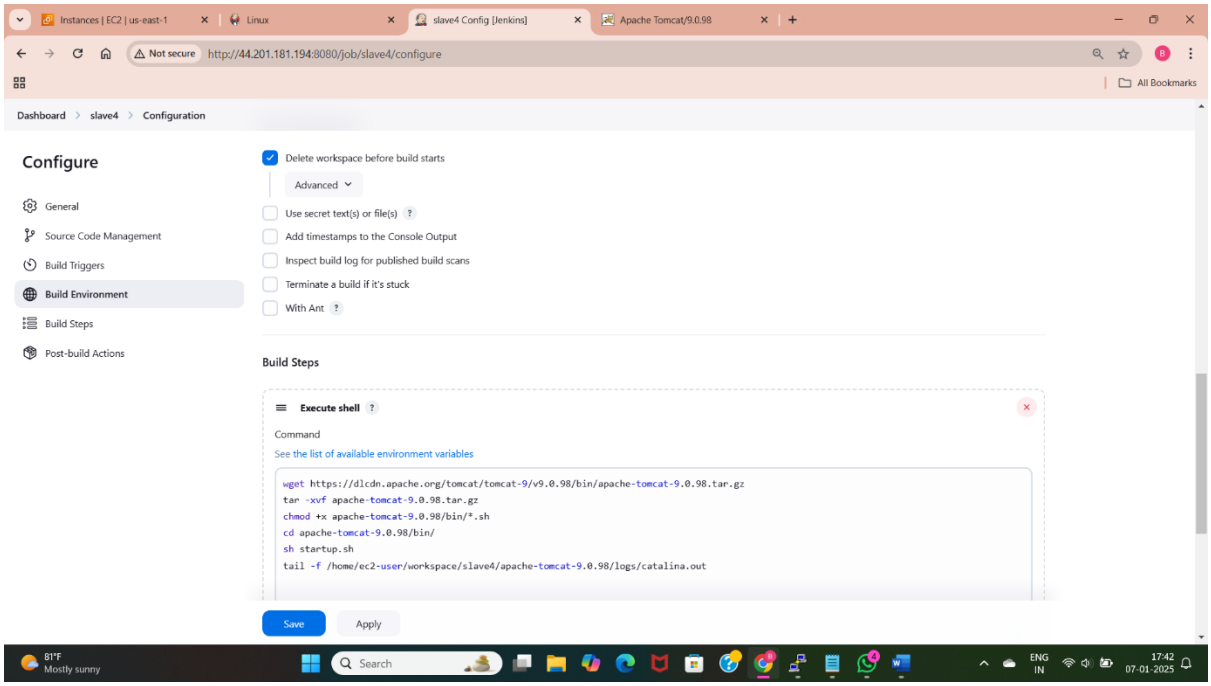
Go to Jenkins Dashboard and click New Item. Provide a name for the job and choose the desired job type (e.g., Freestyle Project).



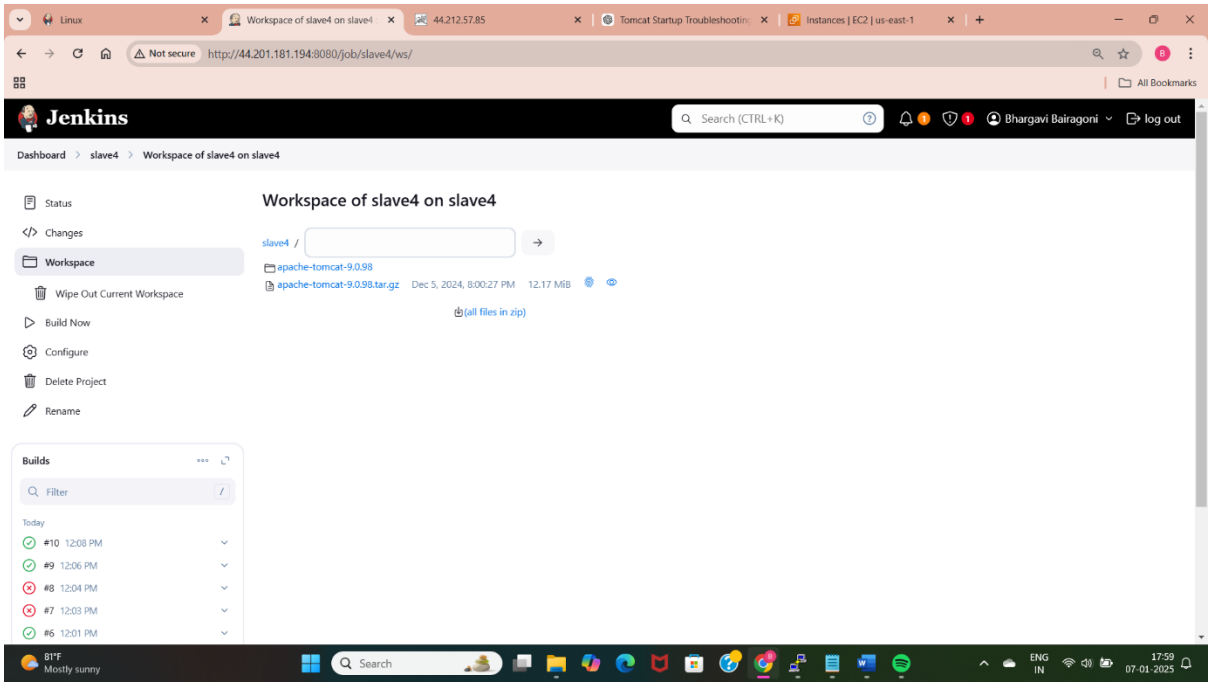
- Now am creating a new job by selecting new item at dashboard, give the name as slav4-tomcat for the new job.



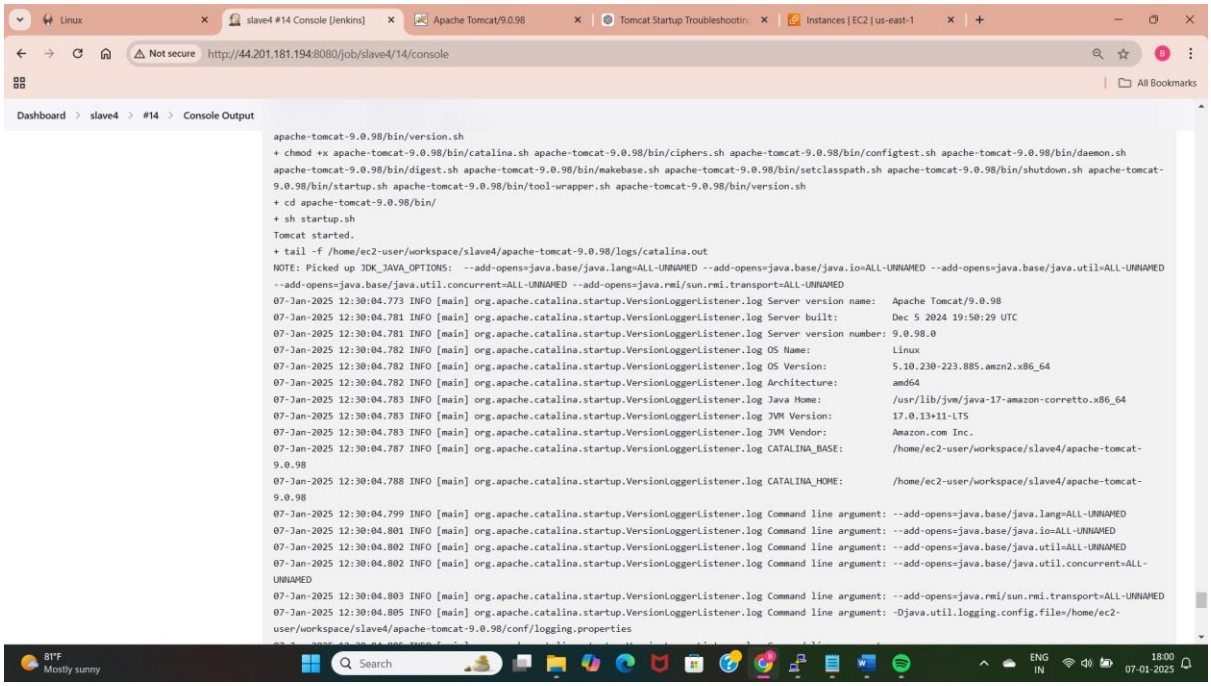
- Under the General section, enable "Restrict where this project can be run".
- Enter the label of the slave node where you want this job to execute.. Here am selecting the slave4 label to execute the builds.



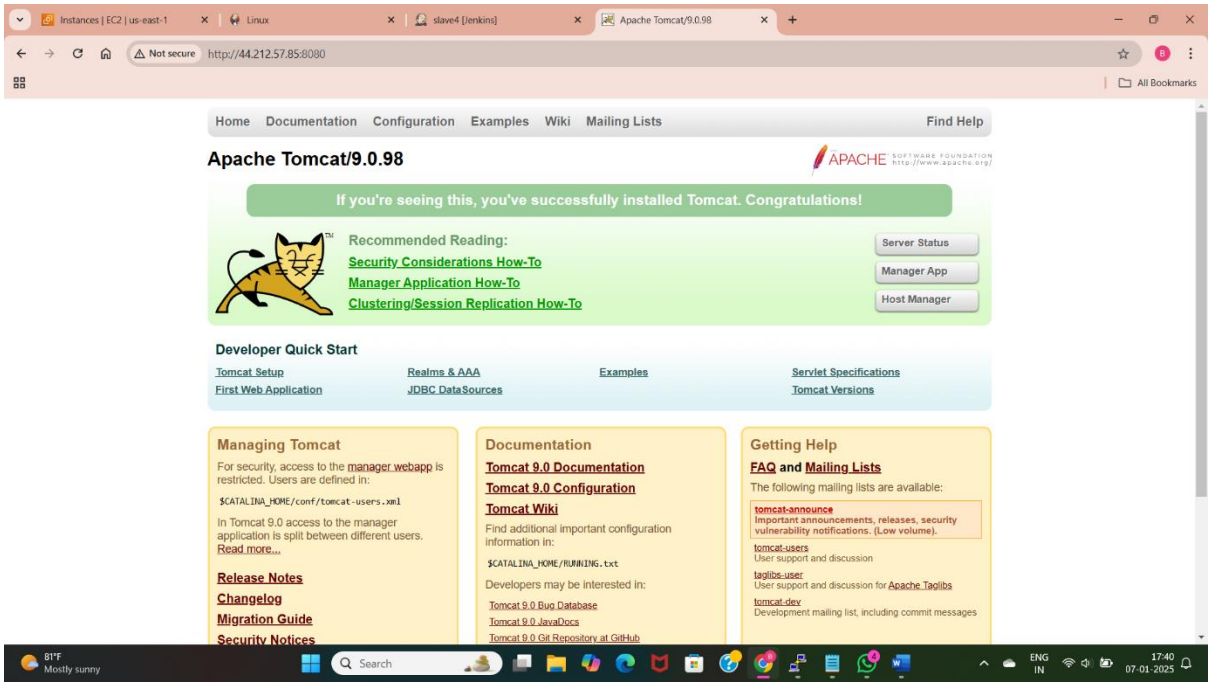
- Now in build steps select execute shell. Give the commands to download the tomcat using wget command. The file is downloaded as tar file, now extract the file using “tar -xvf filename”. Next move the path where tomcat is downloaded and move to bin folder inside the tomcat and start the tomcat.



- Now save and go to slave4 job and click on “build now”. Here we can see the build is success and the tomcat file is downloaded and extracted the file.

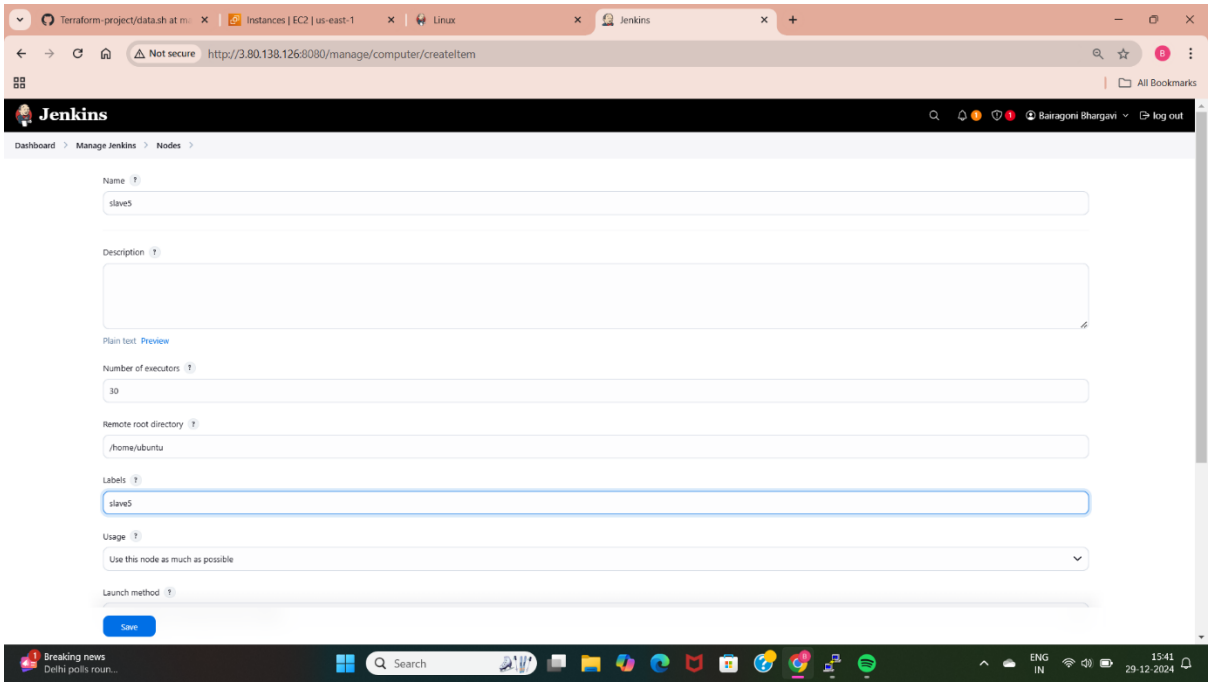


- We can see in the console output, showing that the tomcat is started and checking whether it is is started or not.

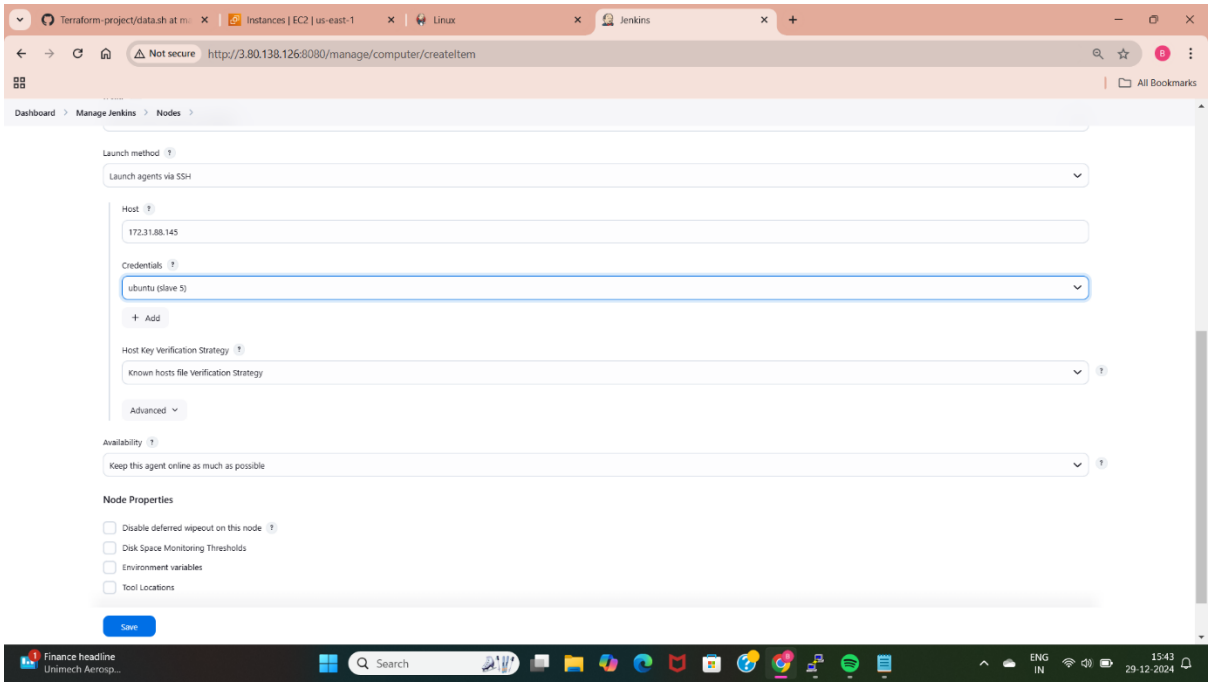


- This is the tomcat page which is deployed using Jenkins execute shell we can access this using the slave public ip address and tomcat port number 8080.

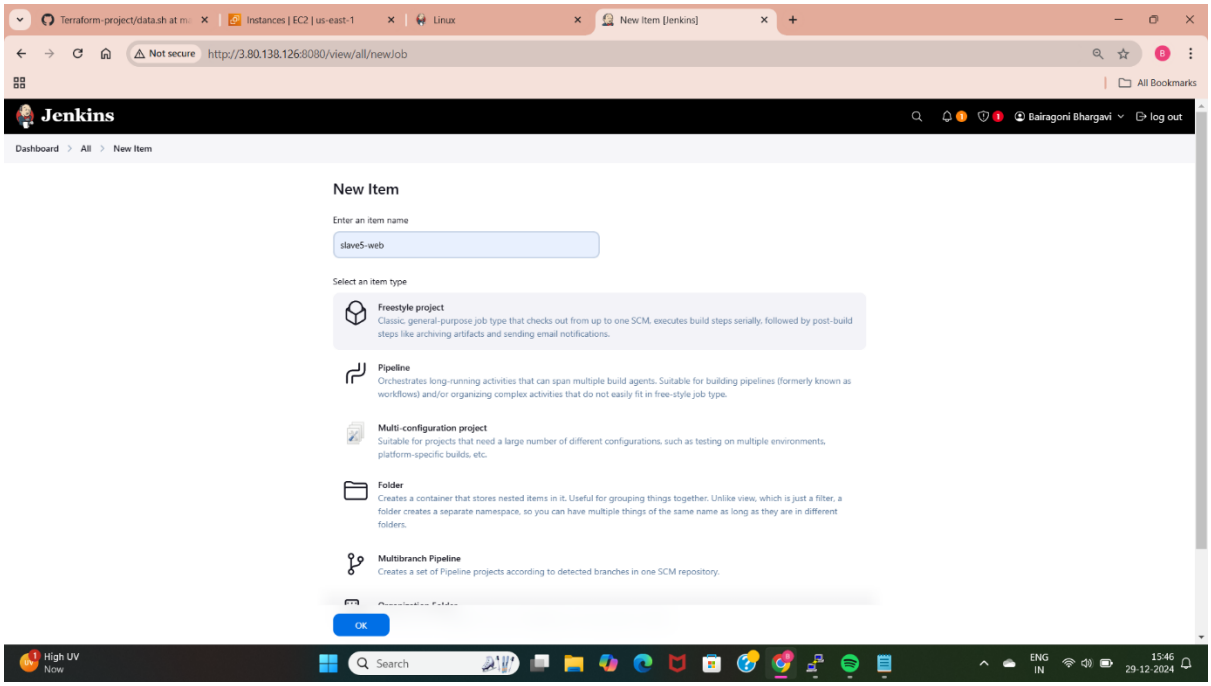
Slave-5:



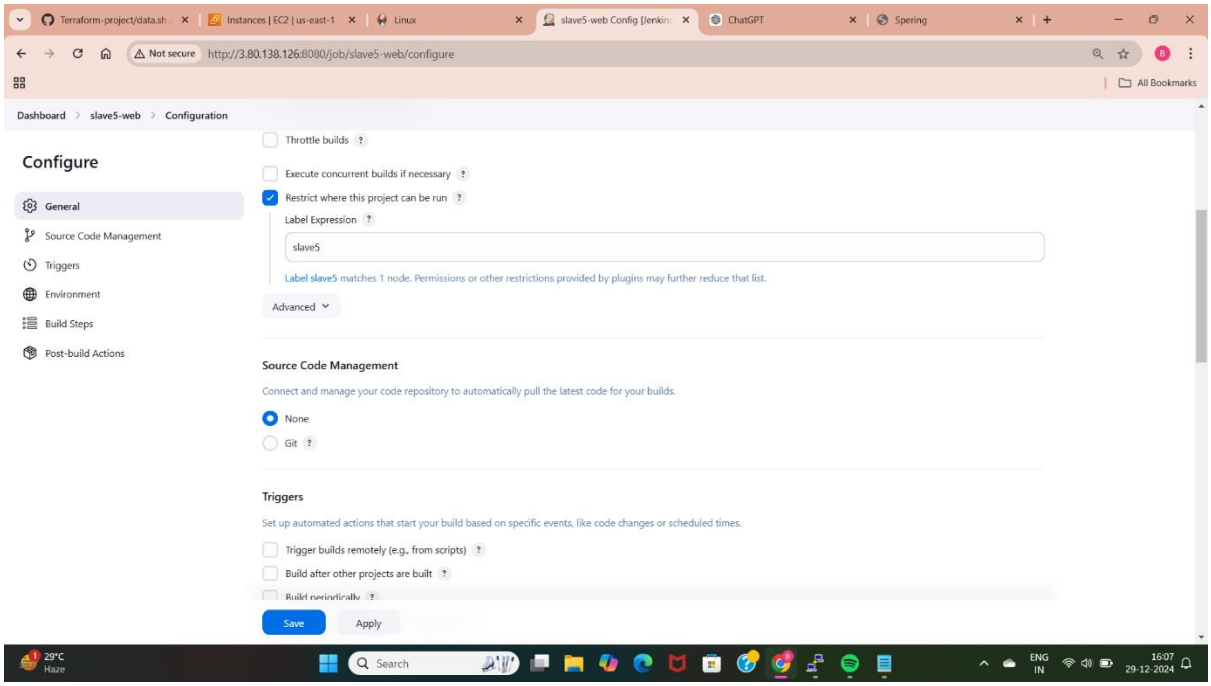
➲ Now create a new node with the name slave5 to deploy any static application. Now give the name as slave5, select the number of executers, and provide root directory of slaver server, select use the node as much as possible as usage and give any name for labels.



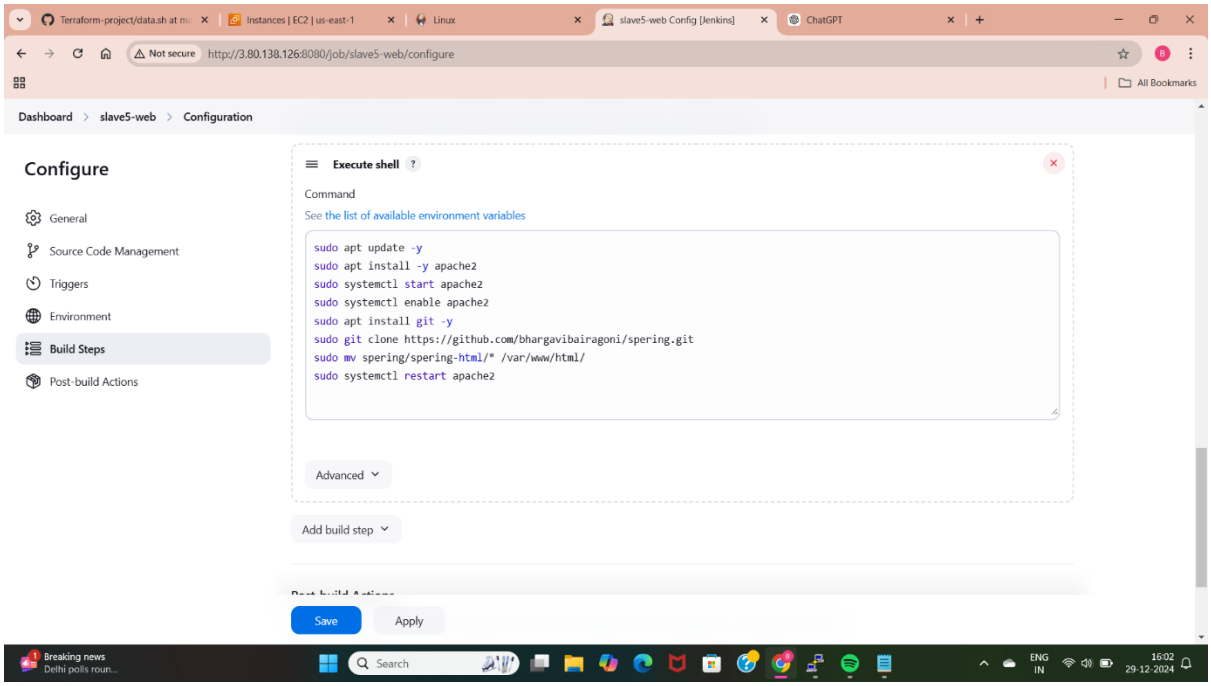
➲ Now select the launch method, provide private Ip address of slave5 server, add credentials using above created credentials, verify credentials by selecting the manually trusted key verification strategy.



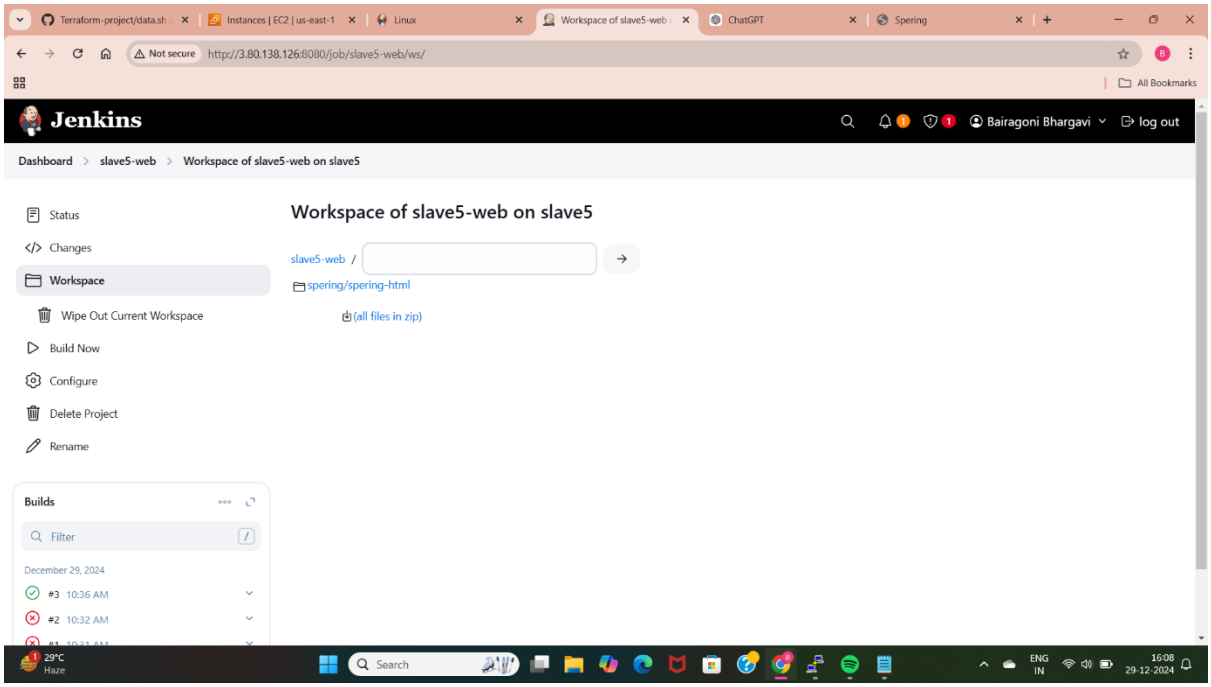
➲ Now create a new job to deploy static application. Here am creating a new job with the name “slave-web”.



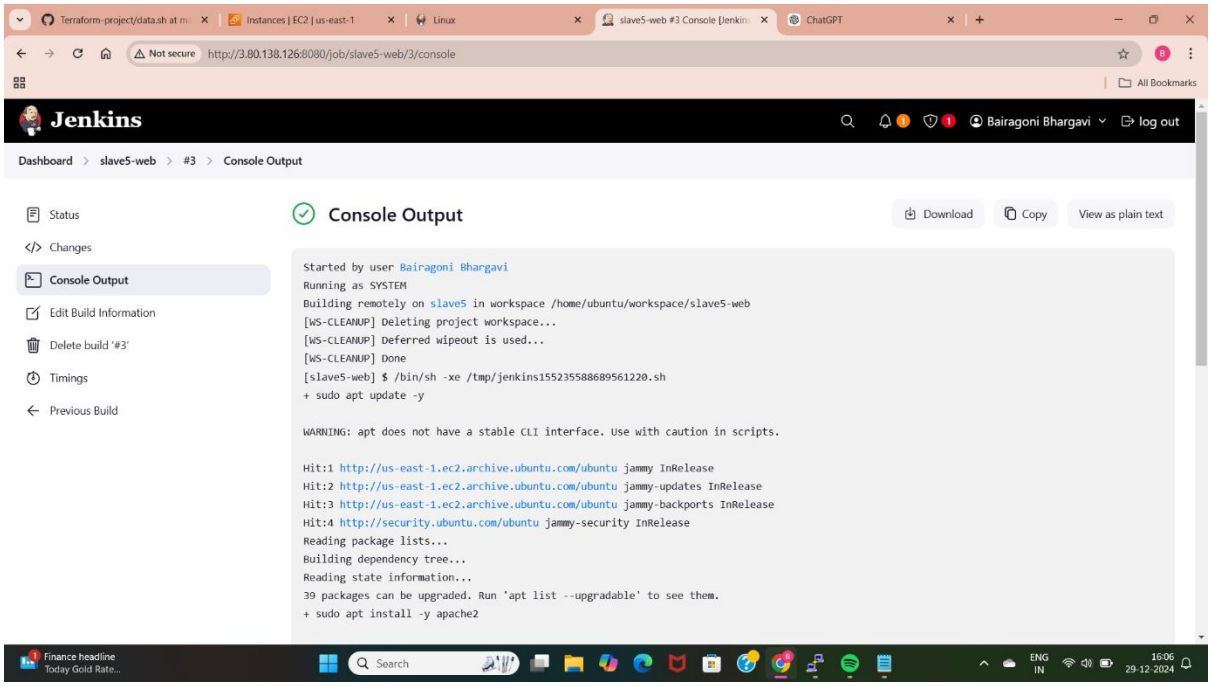
➲ Now go to configurations of slave5-web job and select the above created slave5 node label name to restrict where the project can run.



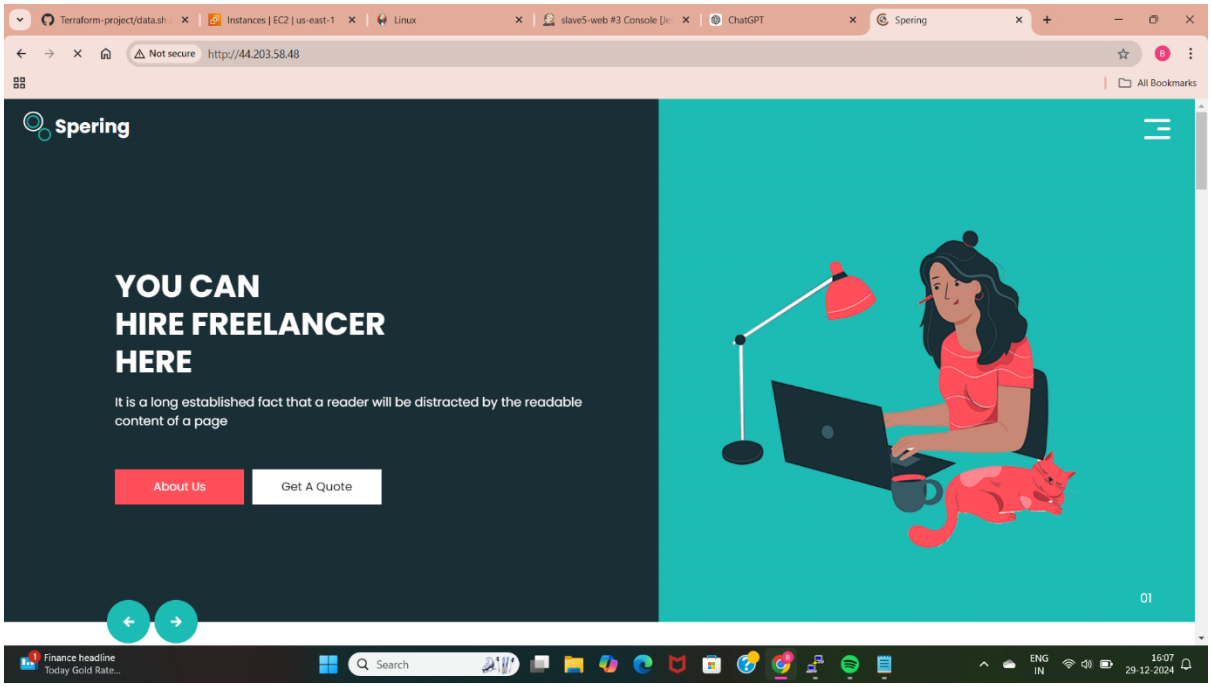
➲ Go to build steps select execute shell. Inside execute shell install httpd, git and start httpd. Take the code from github. Move the files from the repository to the httpd root directory, and restart httpd.



➲ After writing commands execute the commands by clicking on build now option. We can see that the build is success. Here the files are stored in the workspace.



➲ We can also check with the console output by clicking on green colored right mark. We can the build is started by the user Bhargavi and building on workspace, executed the commands to deploy the web application.



➲ We can also access the application by using the public IP address of the instance and here we are seeing the spering web application.

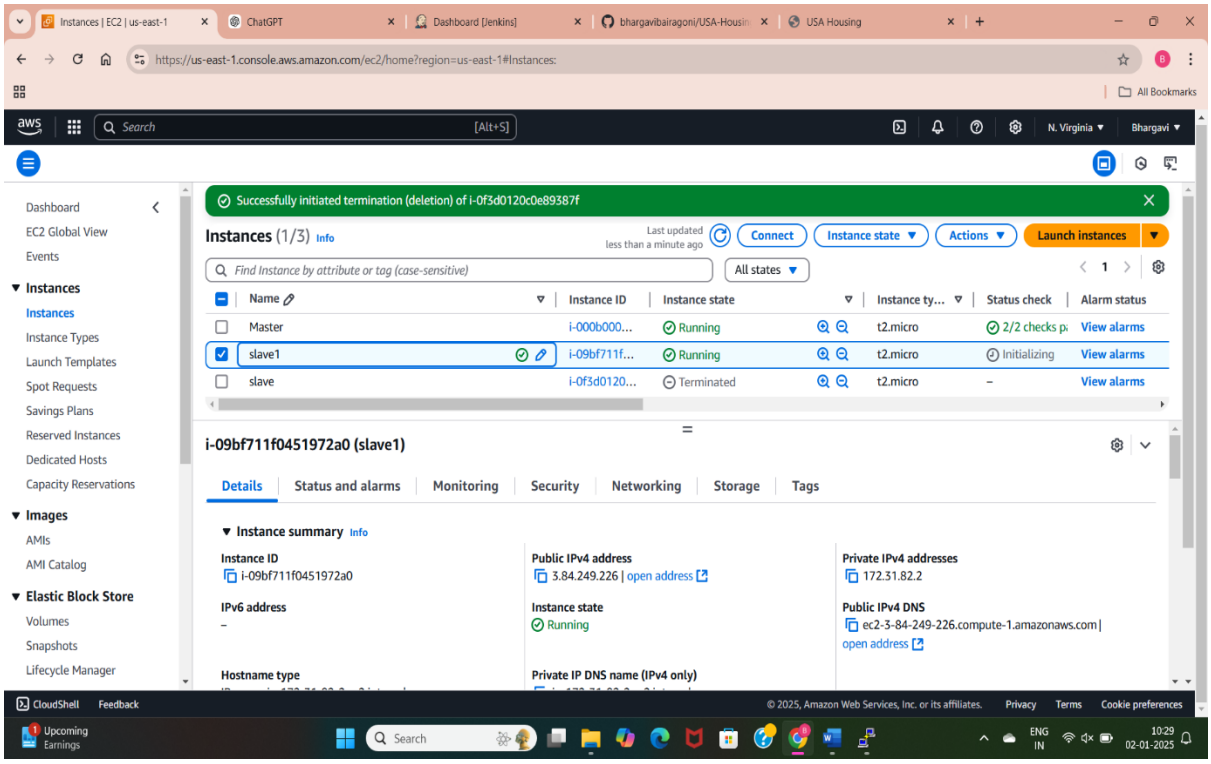
METHOD-2:

Create One Master And Five Slave Nodes And Deploy Python Web Applications Through Manually (Slave-1), Bash Script (Slave-2), Terraform (Slave-3) And Deploy Java-Based Application Through Manually (Slave-4) And Bash Script (Execute Shell) (Slave-5).

In method2 we are creating master slave configurations to deploy python web application, java based application through manually and also using bash script.

Here first we need to create a master server in aws management console. After creating master need to install Jenkins dependency i.e java 17 using <sudo yum install java-17 -y> After that we need to install Jenkins repository, token which is having password, install Jenkins and start Jenkins.

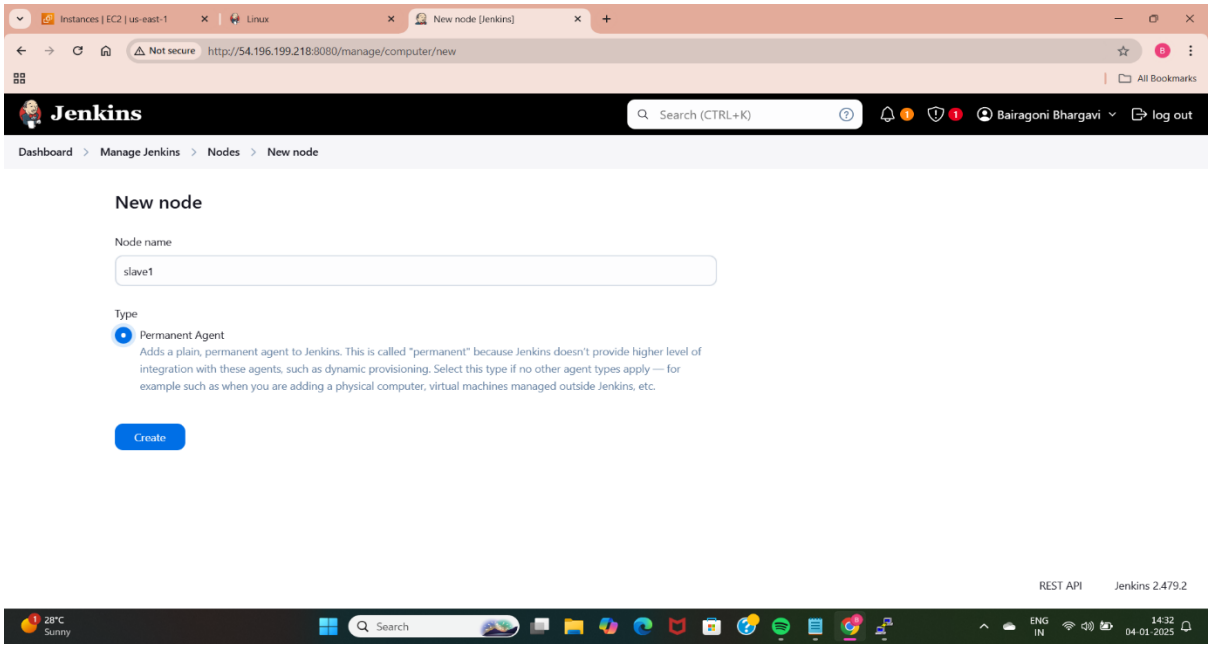
Here in master server we need to generate key pairs so we can use it in slave servers. Take the public key of master using <sudo vi id_rsa.pub> and copy the key without any empty spaces.



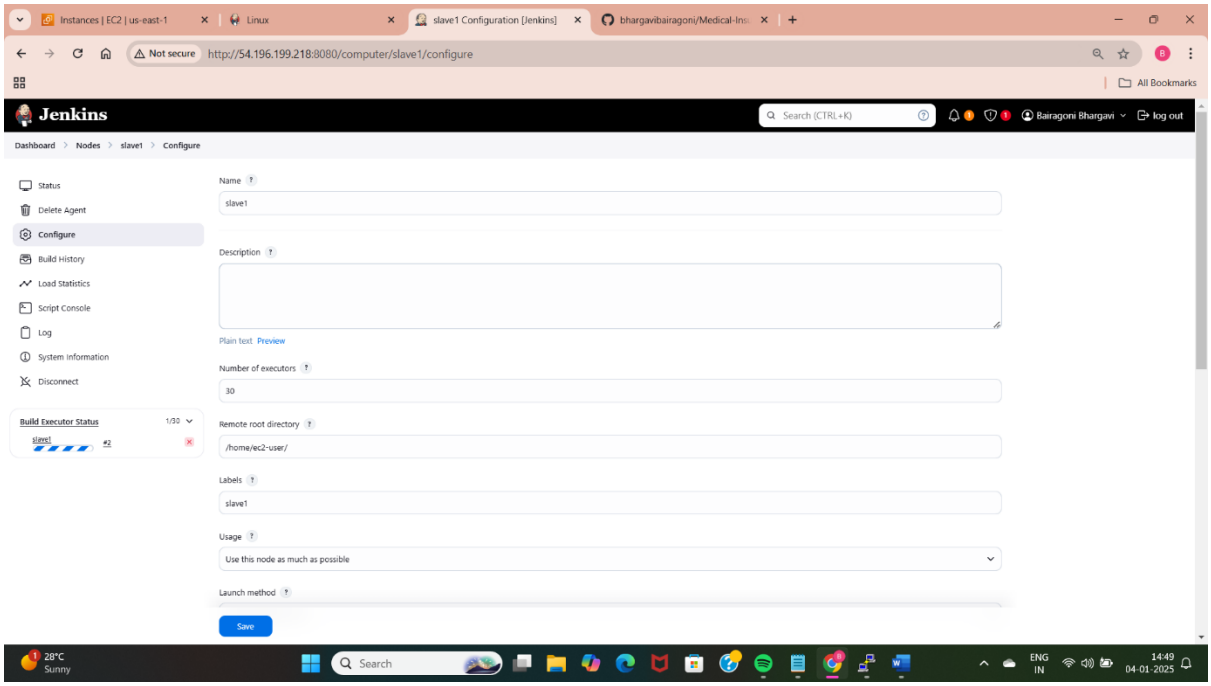
Here am launching one master and slave1 server. After launching slave1 server we need to connect it with master by pasting the masters public key in the slaves authorized keys using <sudo vi authorized_keys> don't select insert mode instead select shift + a, enter to next line and paste the key in slave server and save.

Next install git in slave1 server and take the code from the github by using <git clone> or by adding the github repository to the server using <git remote add origin url> , pull the files from the github.

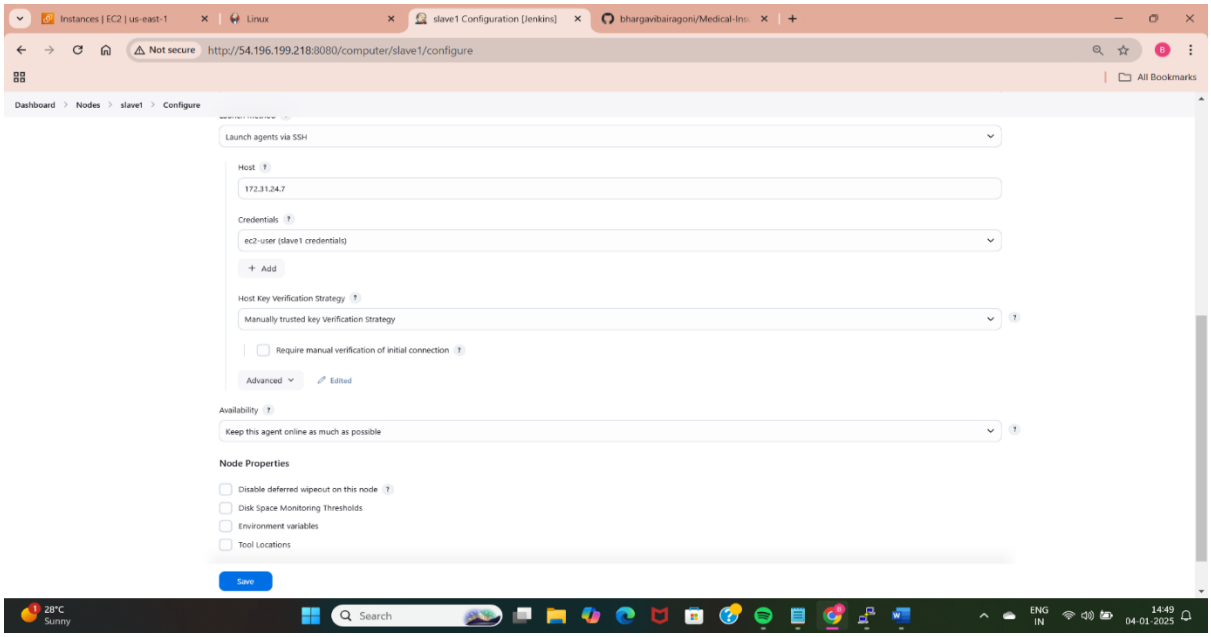
Slave1 - Deploy Python Web Application:



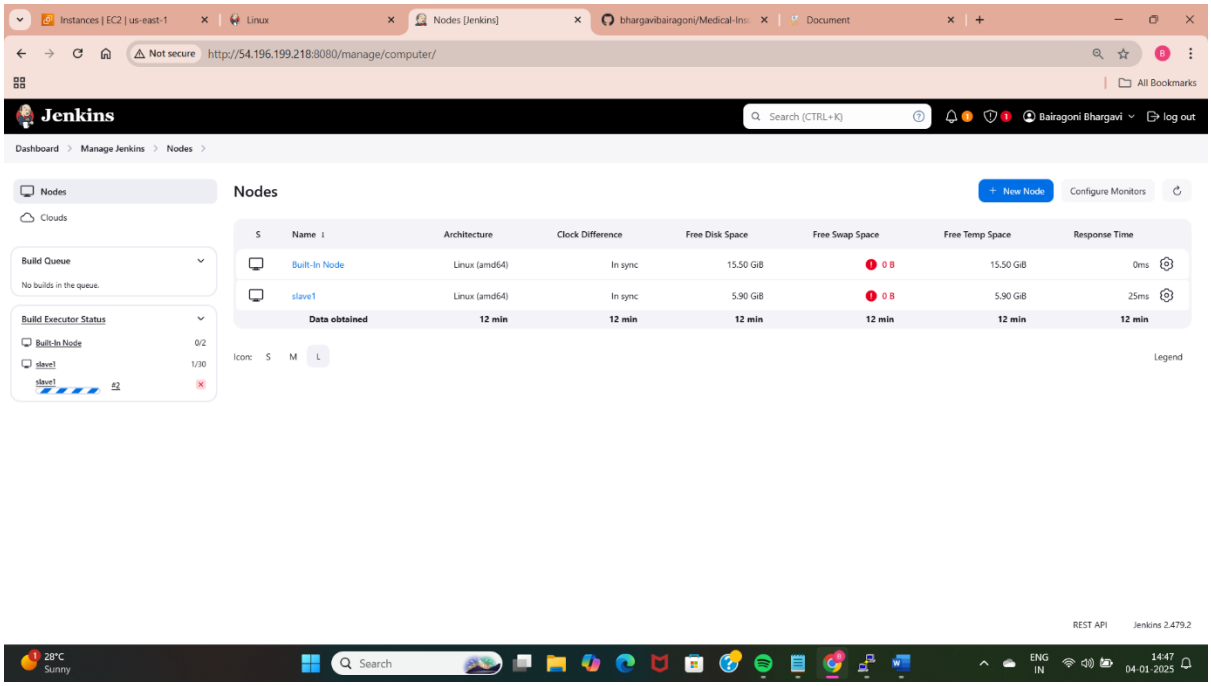
- Go to Manage Jenkins > Manage Nodes and Clouds > New Node, give name for the node and select permanent agent and clik “ok”.



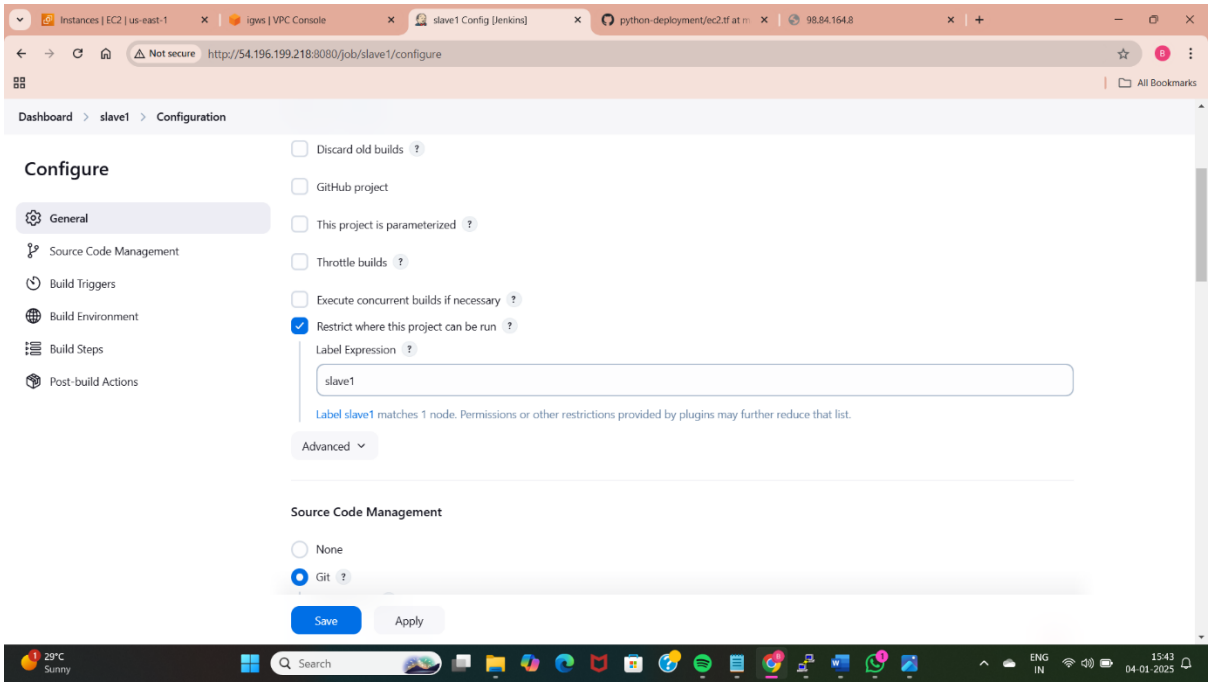
- Next we need to create a node in Jenkins using the above slave1 server.
- Here first we need to go to manage Jenkins>Nodes.
- Now I have given the name as “slave1”, select the number of executors(no. of builds) according to requirement.
- Give the remote root directory of the slave1 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”.
- Give the name for label. We will use this label while running builds.
- Select usage as “use this node as much as possible” to execute builds.



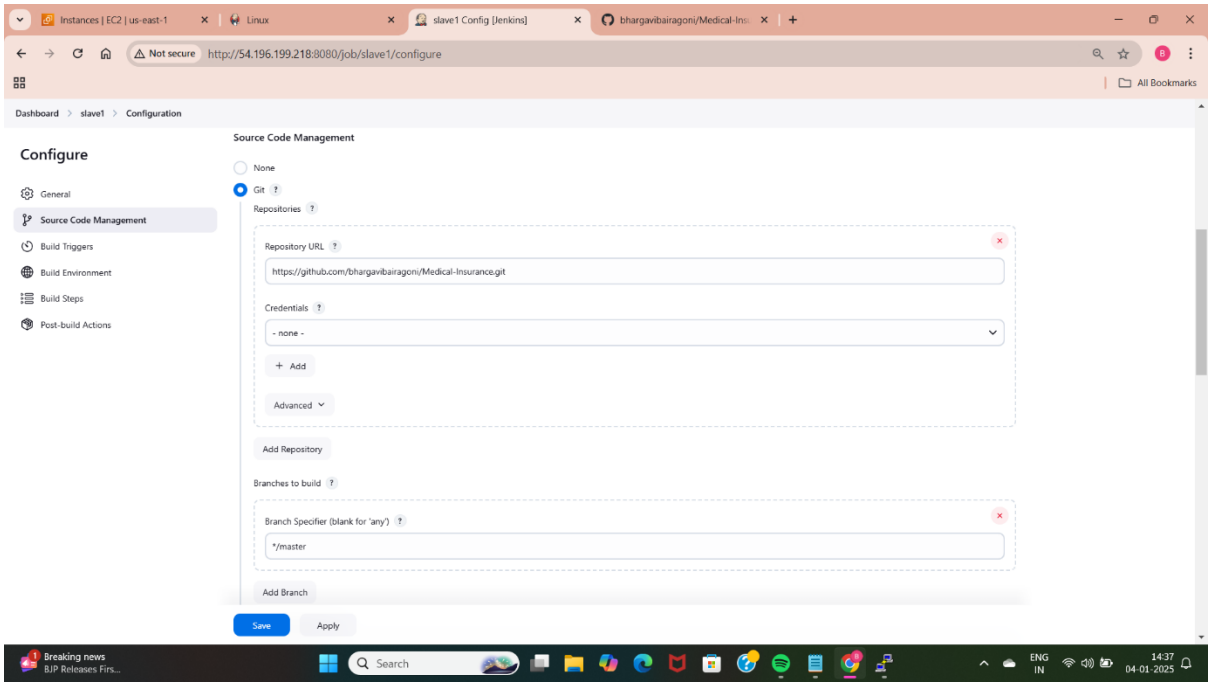
- I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



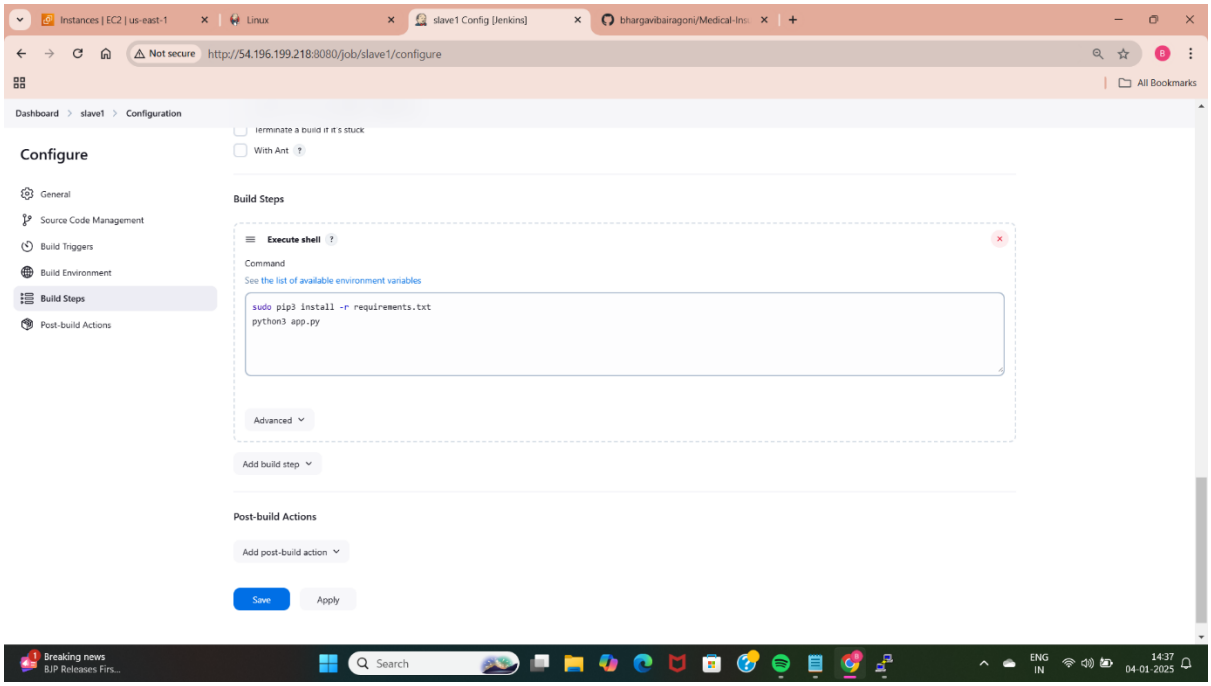
- Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.



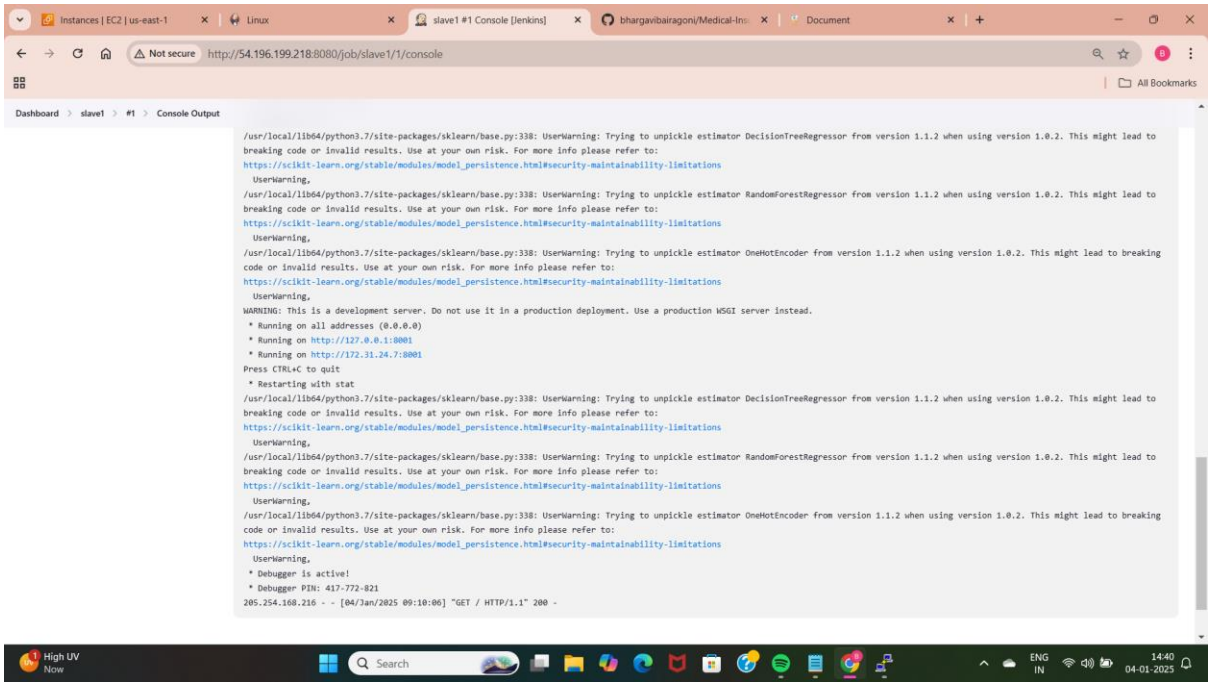
- Select the above created slave1 by using “Restrict where this project can be run” and select the label name which is created above.



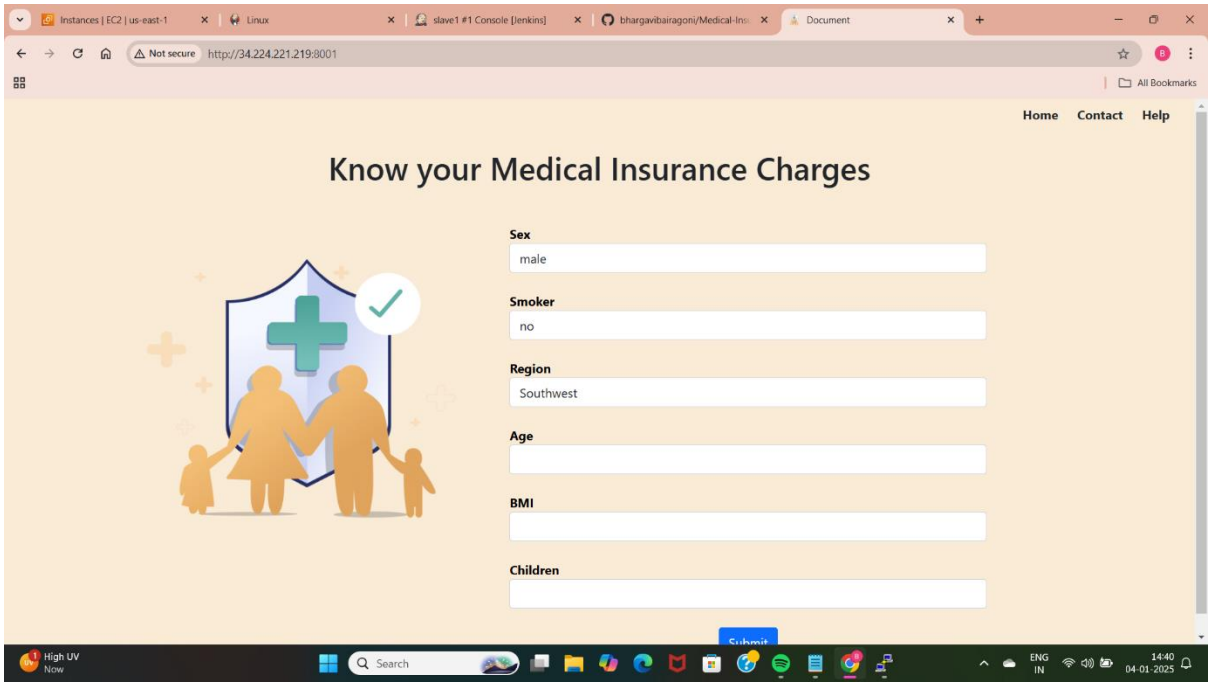
- Go to source code management in Jenkins and select git, give the github repository url here and select branch where the files are stored.



- Here I have pulled the files from the github and installing the requirements related to this python application. Save the job configuration and click Build Now.

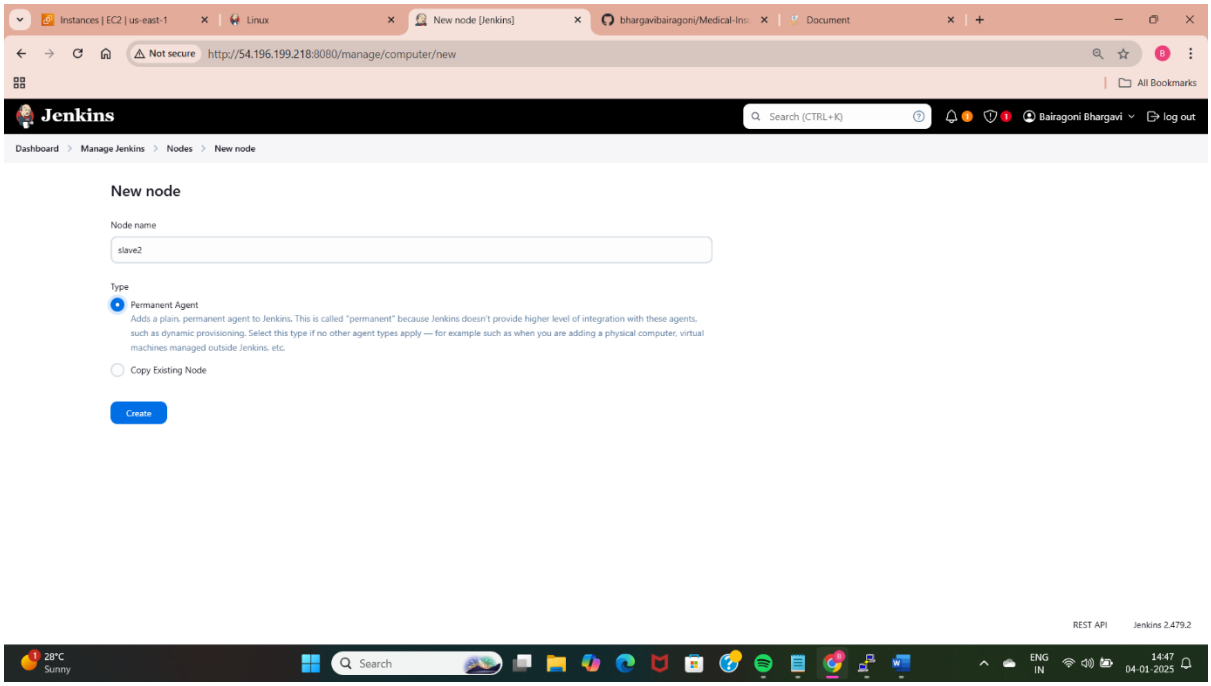


- Monitor the job's output to ensure the application is deployed on Slave-1. After installing requirements in the slave server we need to deploy the python application manually by using the command <python3 app.py>. Here we can see the url that means our application is deployed successfully without any error.

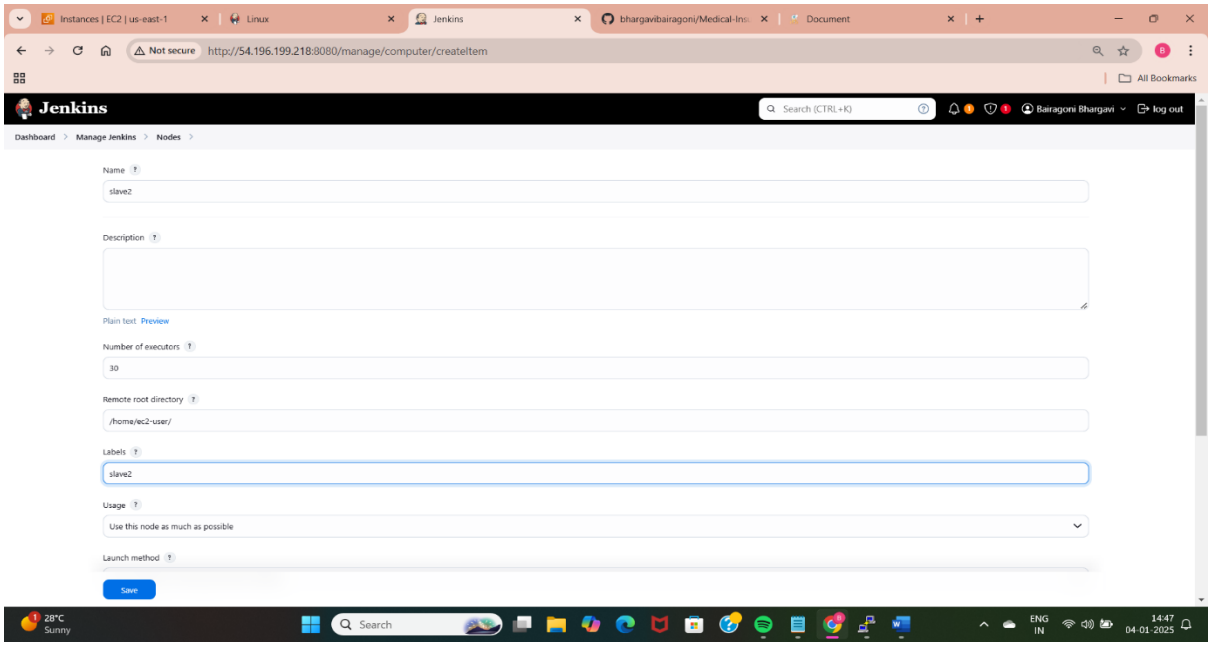


❖ After deploying the application we need to access the application to check whether it is working or not by using the public IP address of the slave1 server with application port number in the browser.

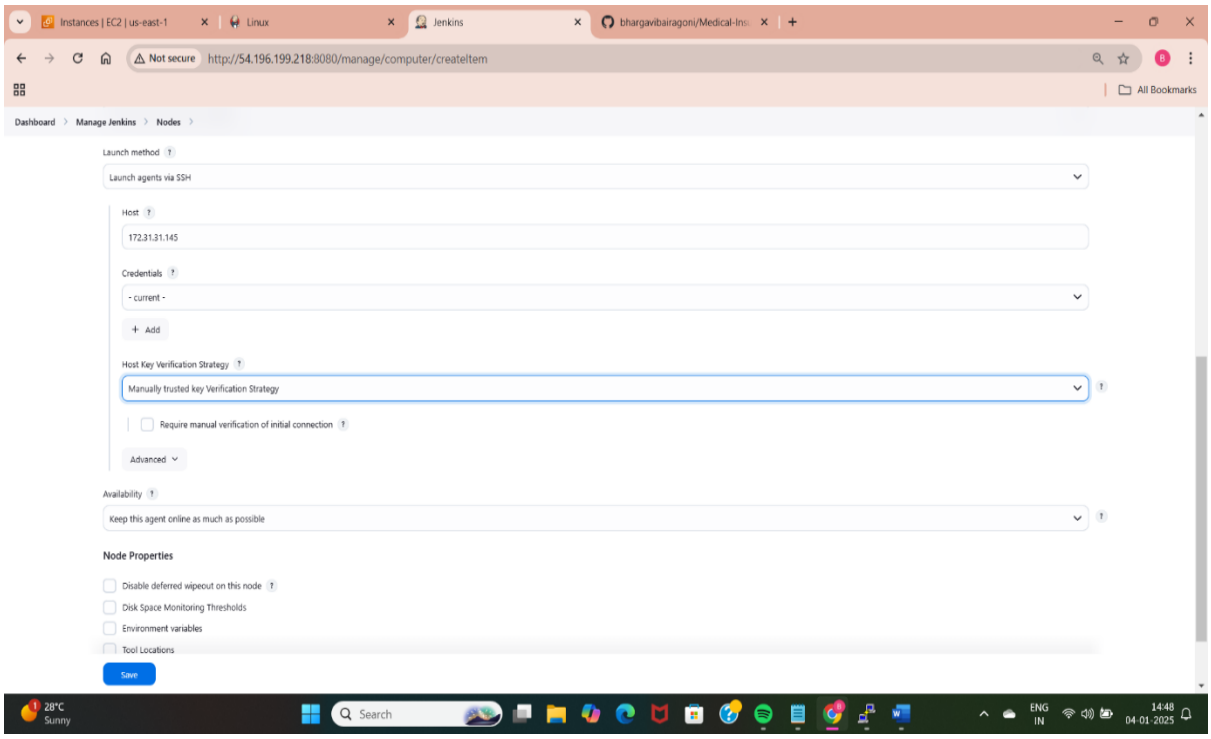
Slave-2: Deployment Using Bash Script



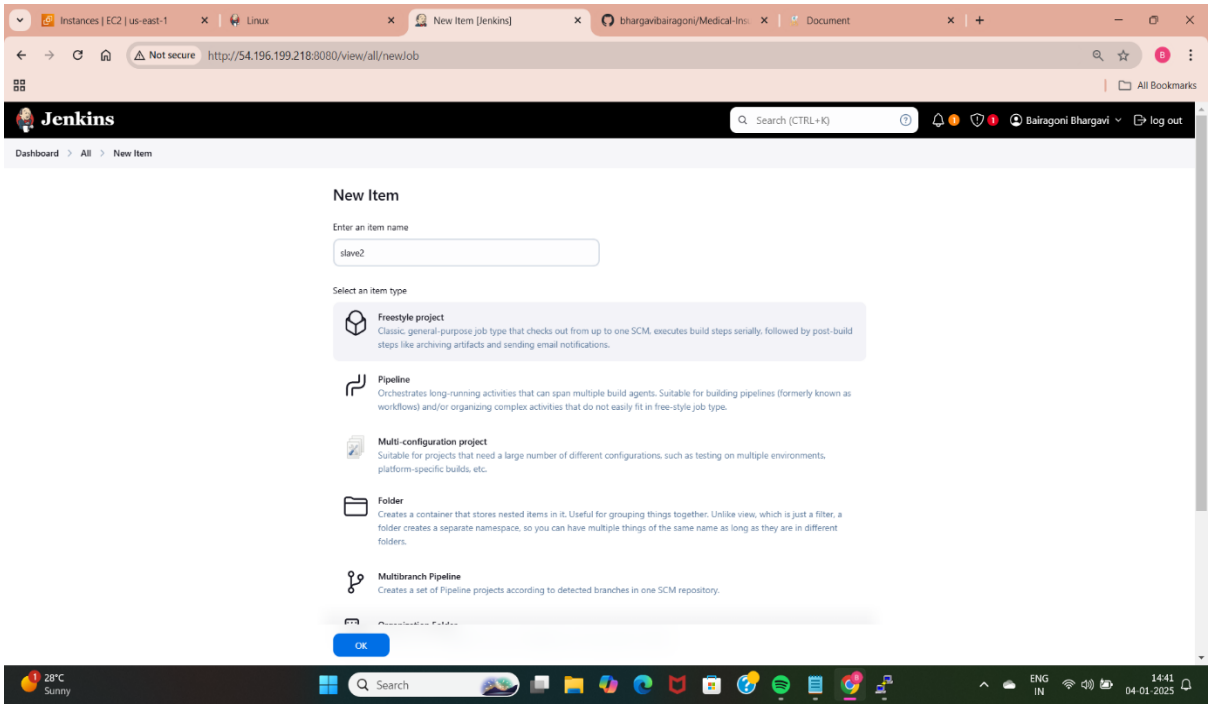
➔ Go to Manage Jenkins > Manage Nodes > New Node, give name for the node and select permanent agent and click “ok”.



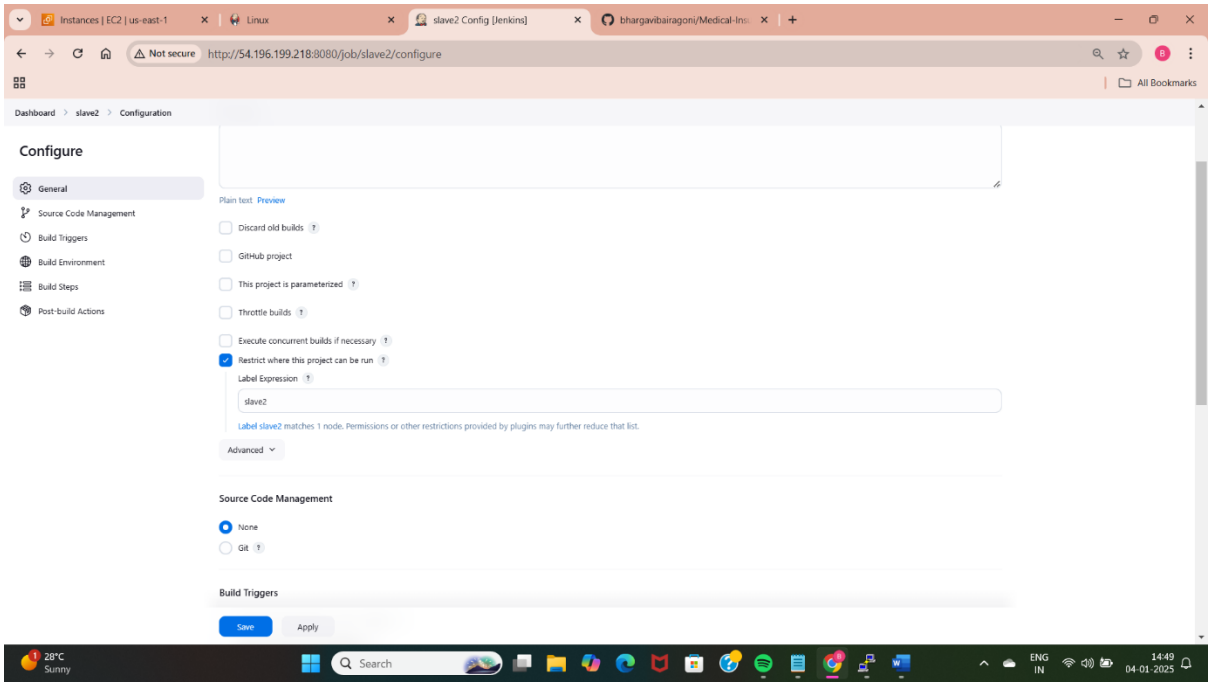
- ➔ Next we need to create a node in Jenkins using the above slave2 server.
- ➔ Here first we need to go to manage Jenkins>Nodes.
- ➔ Now I have given the name as "slave2", select the number of executors(no. of builds) according to requirement.
- ➔ Give the remote root directory of the slave2 server i.e I have taken amazon-linux so root directory is "/home/ec2-user/".
- ➔ Give the name for label. We will use this label while running builds.
- ➔ Select usage as "use this node as much as possible" to execute builds.



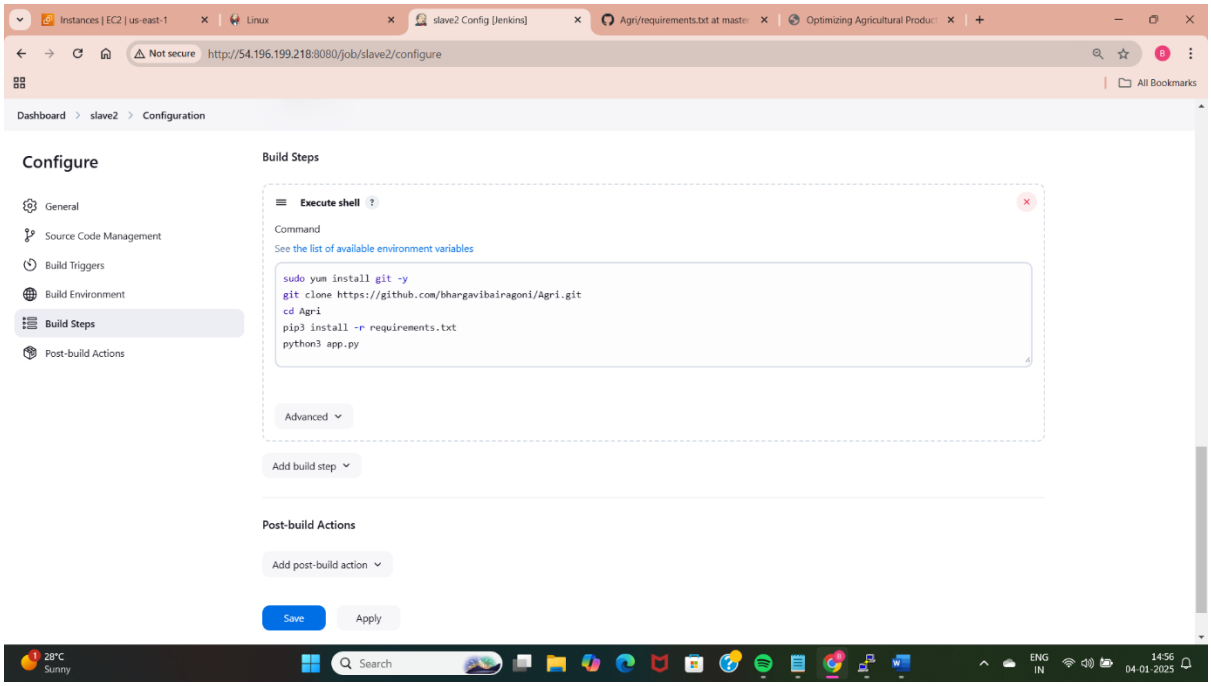
- ➔ I have selected the launch method as "launch agent via SSH" and provide the private keys there for authentication purpose.



➔ Now am creating a new job by selecting new item at dashboard, give the name as slave4 for the new job.

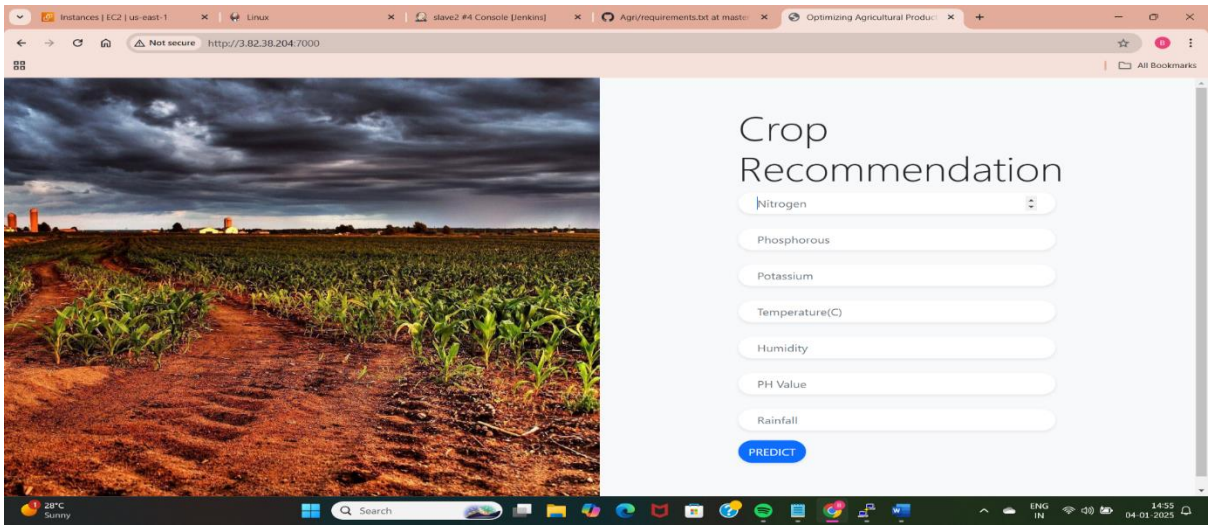


➔ Select the above created slave2 by using “Restrict where this project can be run” and select the label name which is created above.



After launching the slave2 server we need to deploy the python application using the bash script. Here I have written the script.

- ➔ First we need to install the git in the server using <sudo yum install git -y>.
- ➔ We need to take the code/python files from github using git clone.
- ➔ Go inside the repository where we are having files replated to application.
- ➔ Install the requirements in the root directory of the server.
- ➔ Deploy the application using “python3 app.py”.



- ➔ After executing the bash script we need to access the application by using the public IP address of the instance with the application port number.

Slave-3: Deployment Using Terraform

Terraform is an open-source Infrastructure as Code (IaC) tool created by HashiCorp. It allows you to define, provision, and manage infrastructure using simple configuration files written in HashiCorp Configuration Language (HCL) or JSON. With Terraform, you can automate the deployment and management of resources across multiple cloud providers or on-premises systems.

Key Features of Terraform

1. **Declarative Syntax:**

   You declare the desired state of your infrastructure (e.g., the number of servers, networking setup, etc.) in configuration files, and Terraform ensures it matches that state.

2. **Provider Support:**

   Terraform supports multiple cloud providers like **AWS, Azure, Google Cloud**, and on-premises solutions like VMware, OpenStack, and Kubernetes.

3. **State Management:**

   Terraform maintains a **state file** to track resources it manages, enabling it to compare the current state of your infrastructure with the desired state and make updates accordingly.

4. **Plan and Apply Workflow:**

   Plan: Terraform generates an execution plan, showing what changes it will make to achieve the desired state.

   Apply: Executes the plan to provision or update the infrastructure.

5. **Modular Infrastructure:**

   Terraform supports reusable code modules, making it easier to manage complex configurations.

6. **Immutable Infrastructure:**

   Instead of modifying existing resources, Terraform often creates new resources and replaces the old ones, ensuring consistency and reducing errors.

Core Terraform Concepts

1. **Providers:**

   Plugins to interact with specific APIs, like AWS, Azure, Google Cloud, Kubernetes, etc.

2. **Resources:**

   The components you create and manage, such as EC2 instances, S3 buckets, or databases.

3. **Variables:**

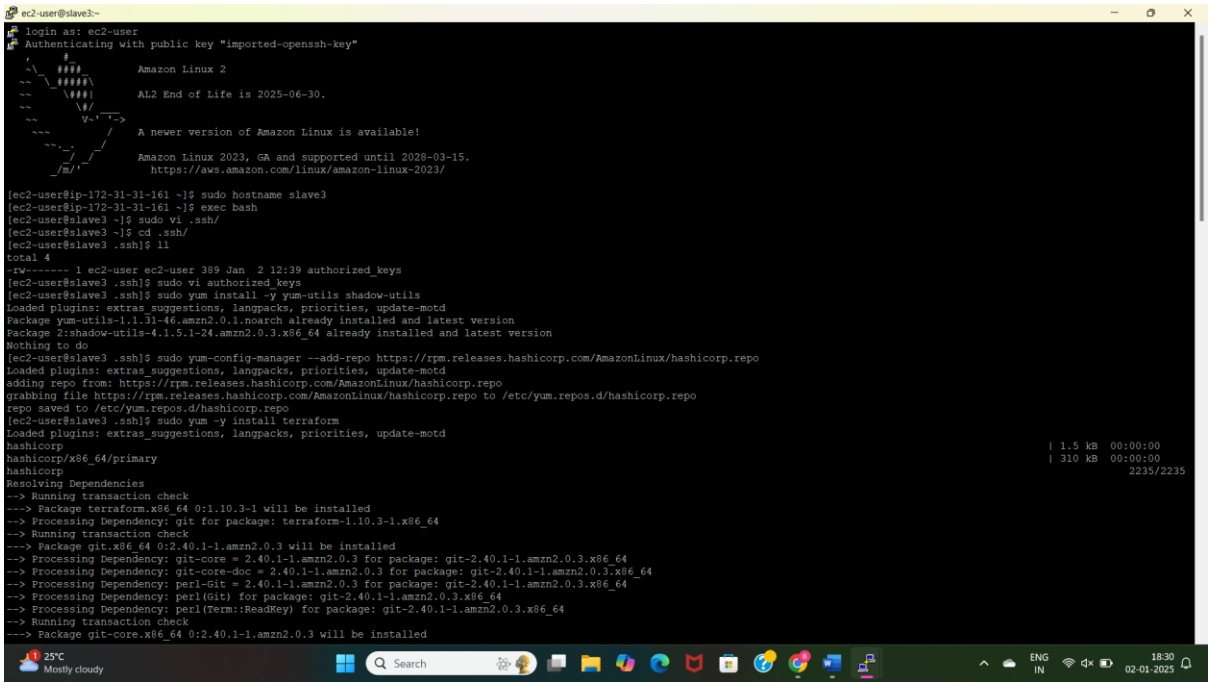
   Allow parameterization of configurations for reusability and flexibility.

4. **Modules:**

   Logical groupings of resources to simplify code and promote reuse.

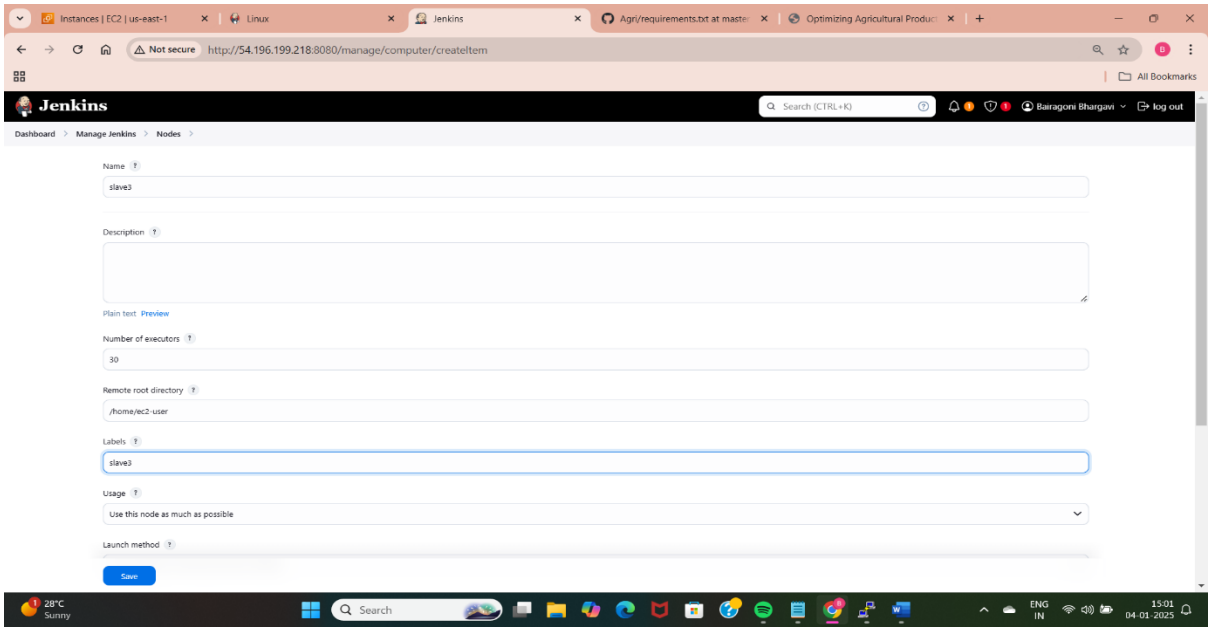
5. **State:**

   A file (terraform.tfstate) that keeps track of resources and their relationships.



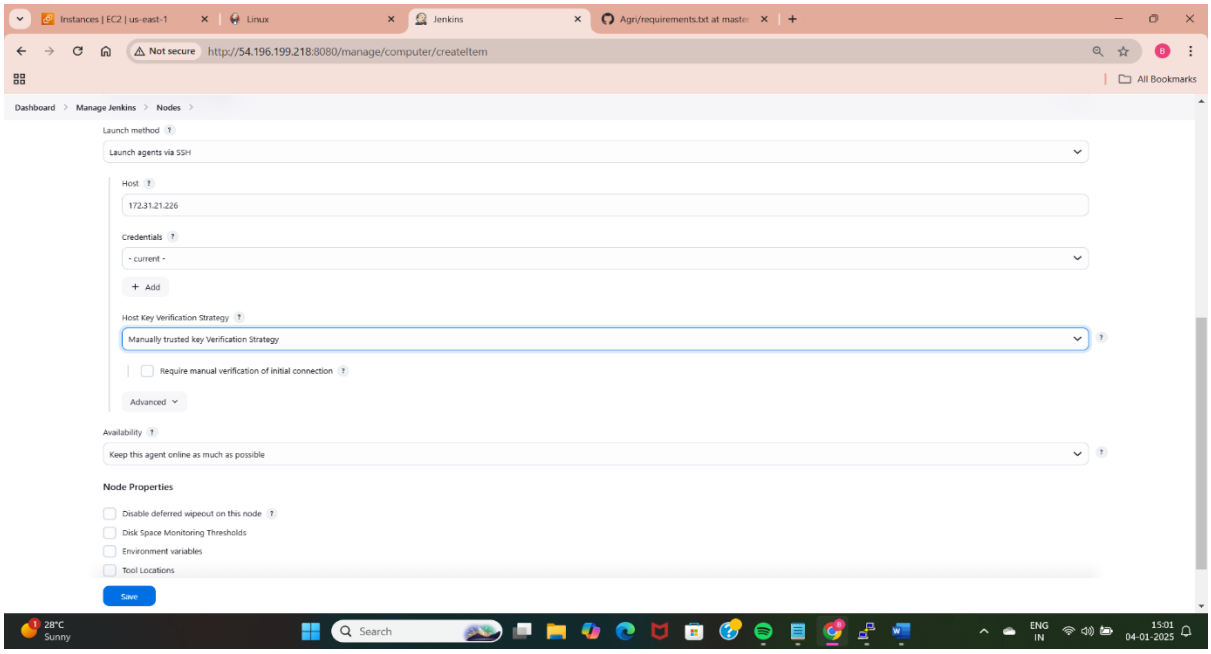
Now launch slave3 server and connect it with the terminal.

- ♣ Here am changing the hostname to slave3 using <sudo hostname slave3>, applied the name to the terminal using <exec bash>.
- ♣ Now connect the slave3 with the master by using the public key of master. For that we need to move into <sudo vi .ssh/>. There we will find the authorized_keys files. Copy the masters public ip in the slaves authorized keys.

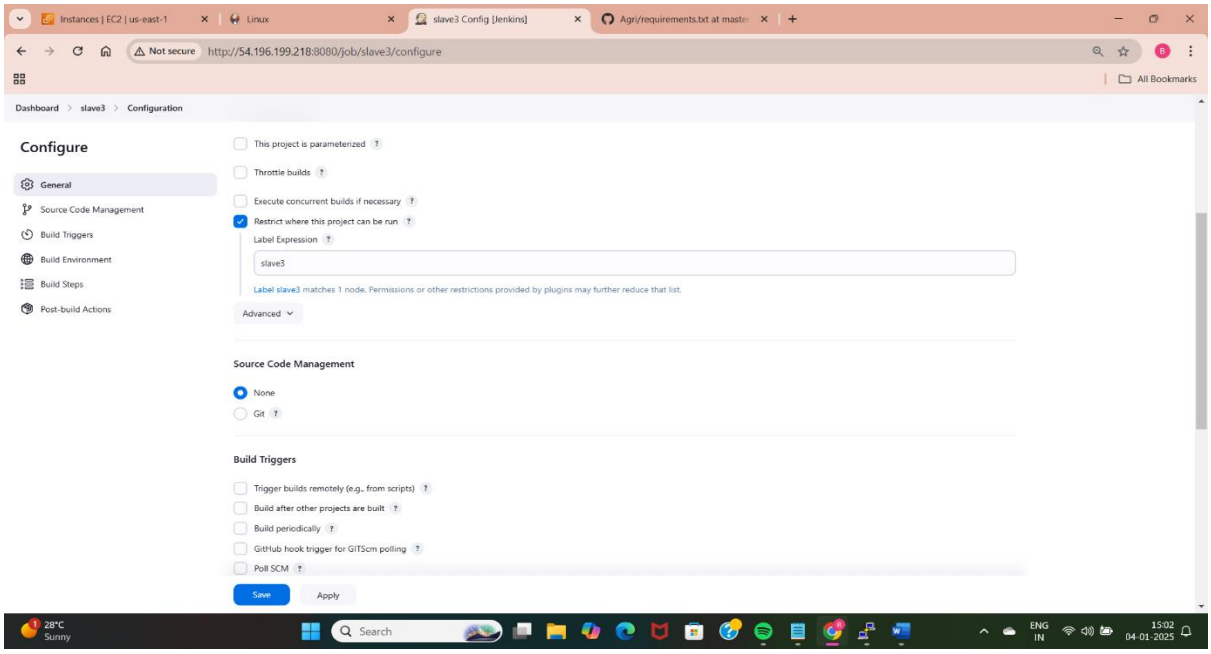


- ✓ Next we need to create a node in Jenkins using the above slave3 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave3”, select the number of executors(no. of builds) according to requirement.

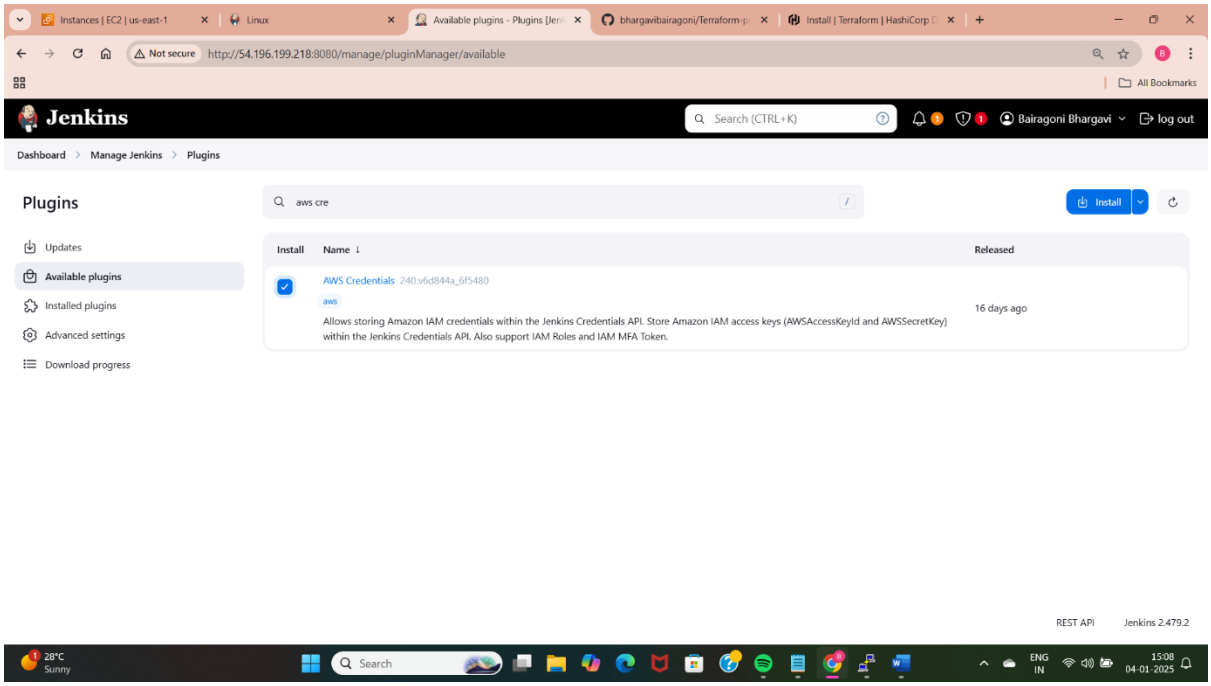
- ✓ Give the remote root directory of the slave3 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.



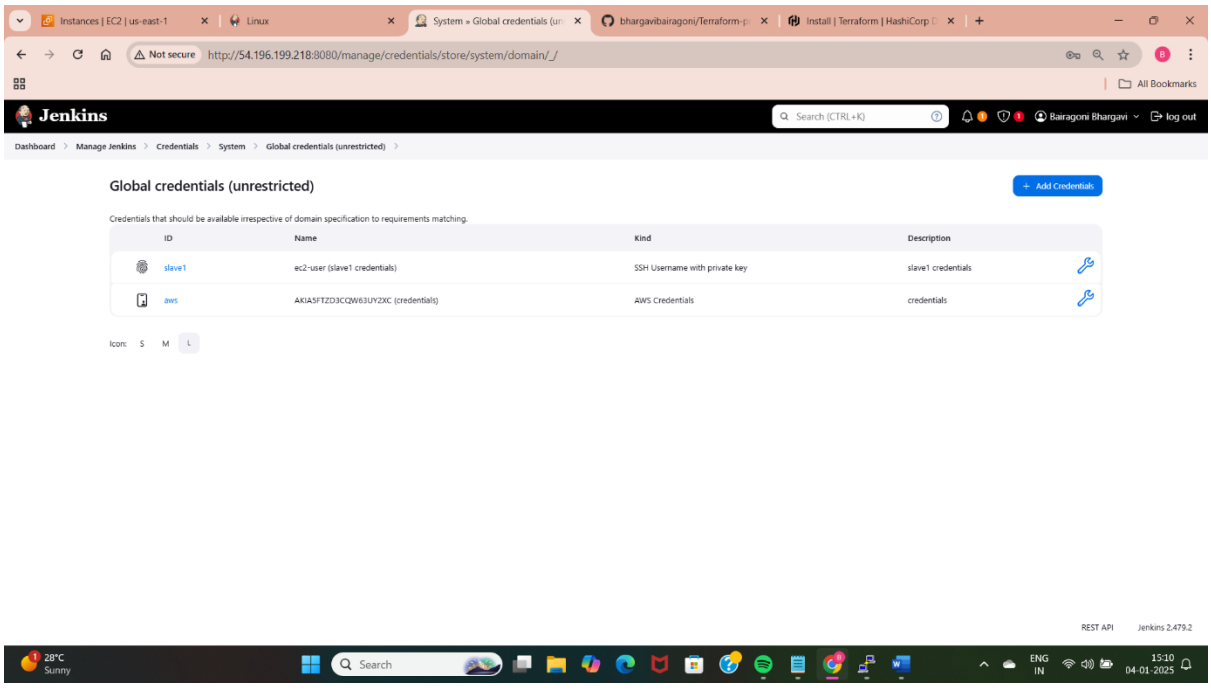
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



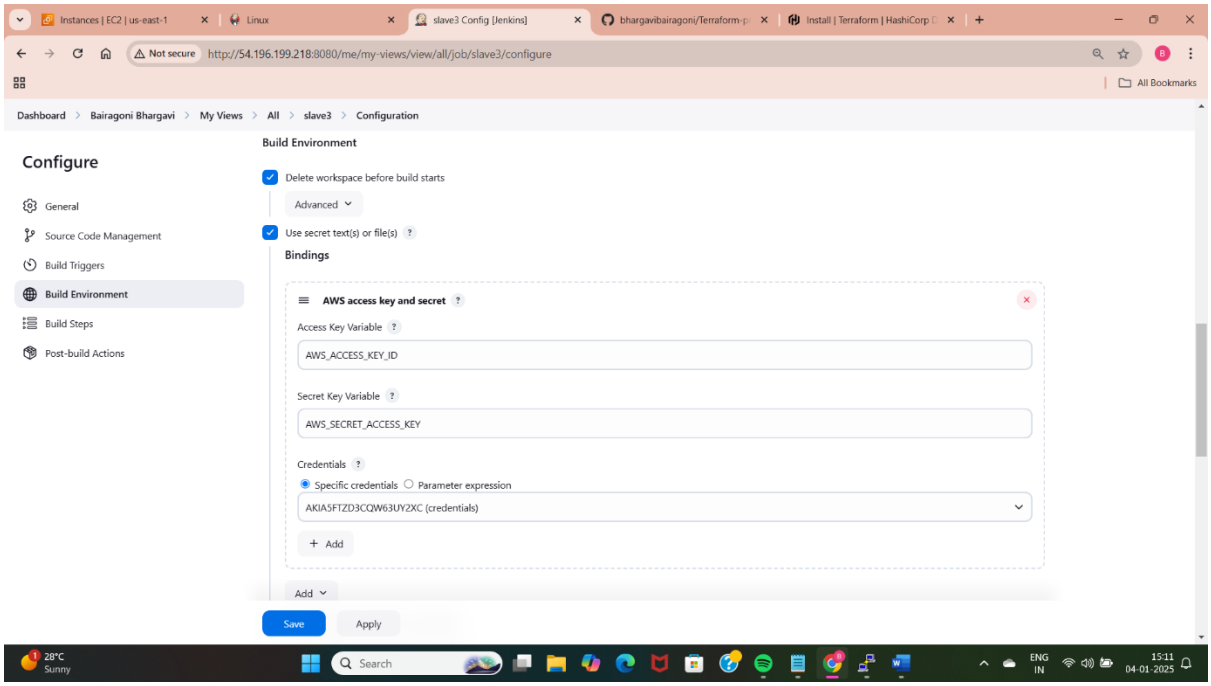
- ✓ Select the above created slave3 by using “Restrict where this project can be run” and select the label name which is created above.



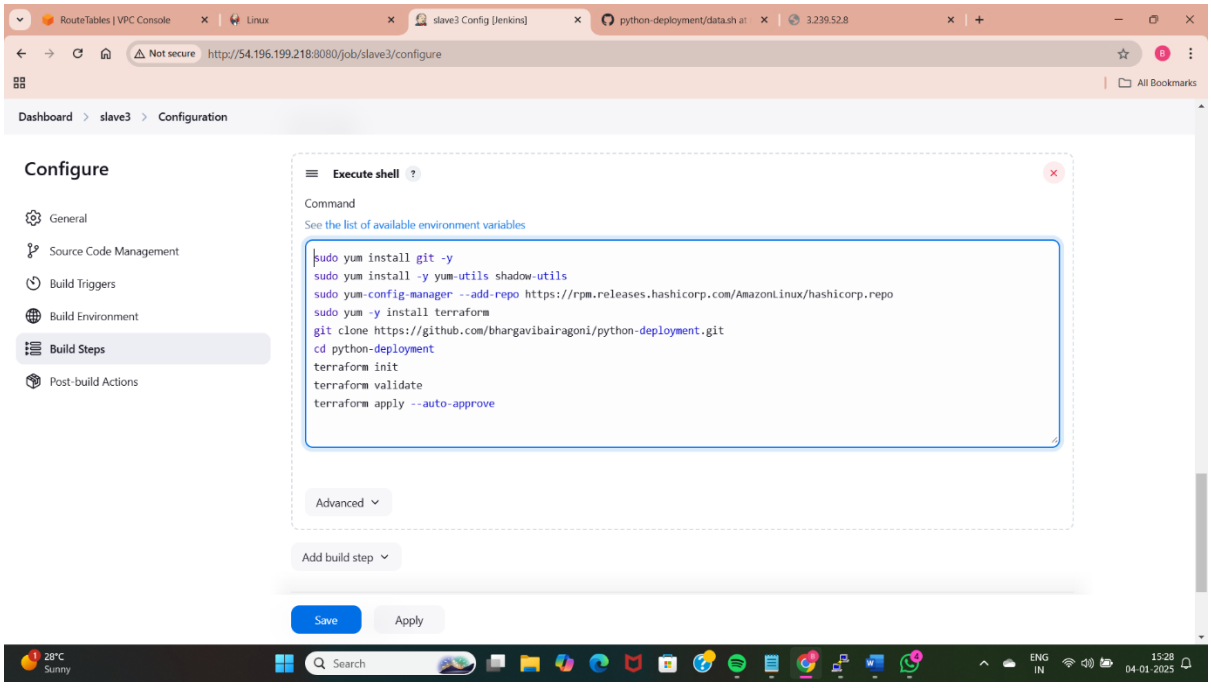
- Go to manage Jenkins select plugins>available plugins install aws credentials plugin to provide aws configurations.



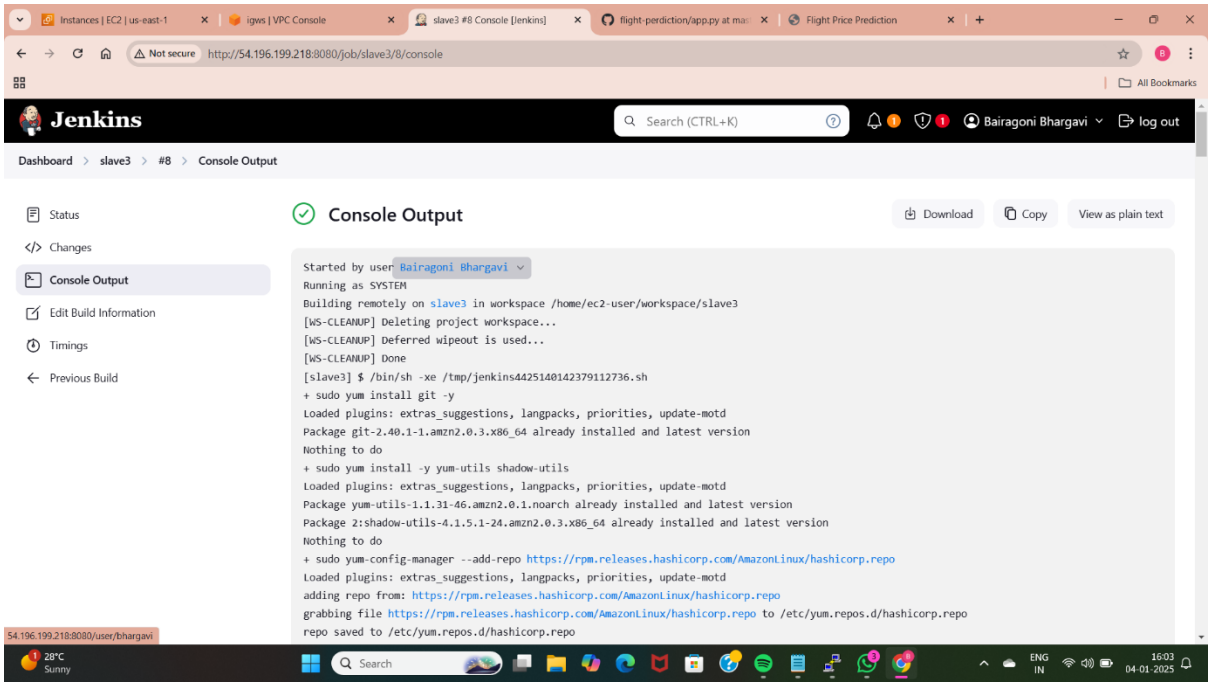
- Go to manage select credentials and add aws configurations using aws credentials and give any id and description. Next give created aws access keys and secret access keys there.



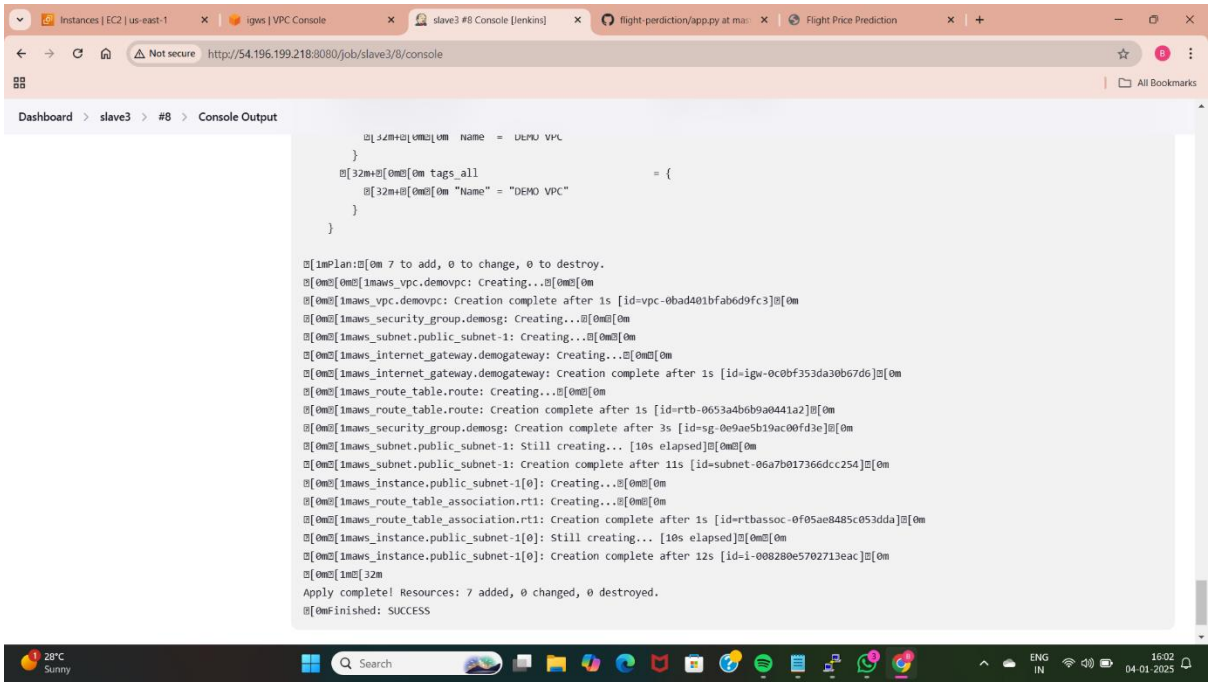
- ♣ Provide aws configurations using the access keys and secret access keys using the aws credentials plugin.



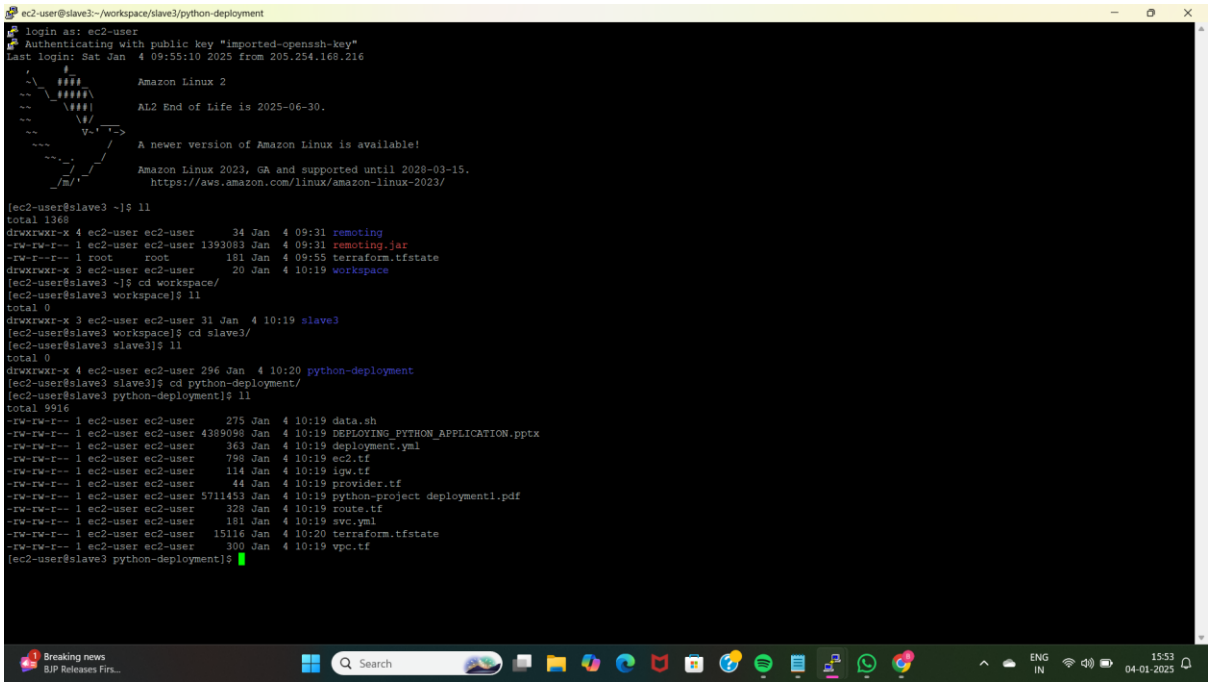
- ♣ Install git to take the code from github which is having files to create aws resources using the terraform.
- ♣ Install terraform in the terminal using the bash script next clone the github repository which is having files related to creation of ec2 instance, vpc.
- ♣ After that we need to initialize the terraform in created files.
- ♣ Check the validation of the code/files using the <terraform validate>.
- ♣ Apply the changes using <terraform apply –auto-approve>.



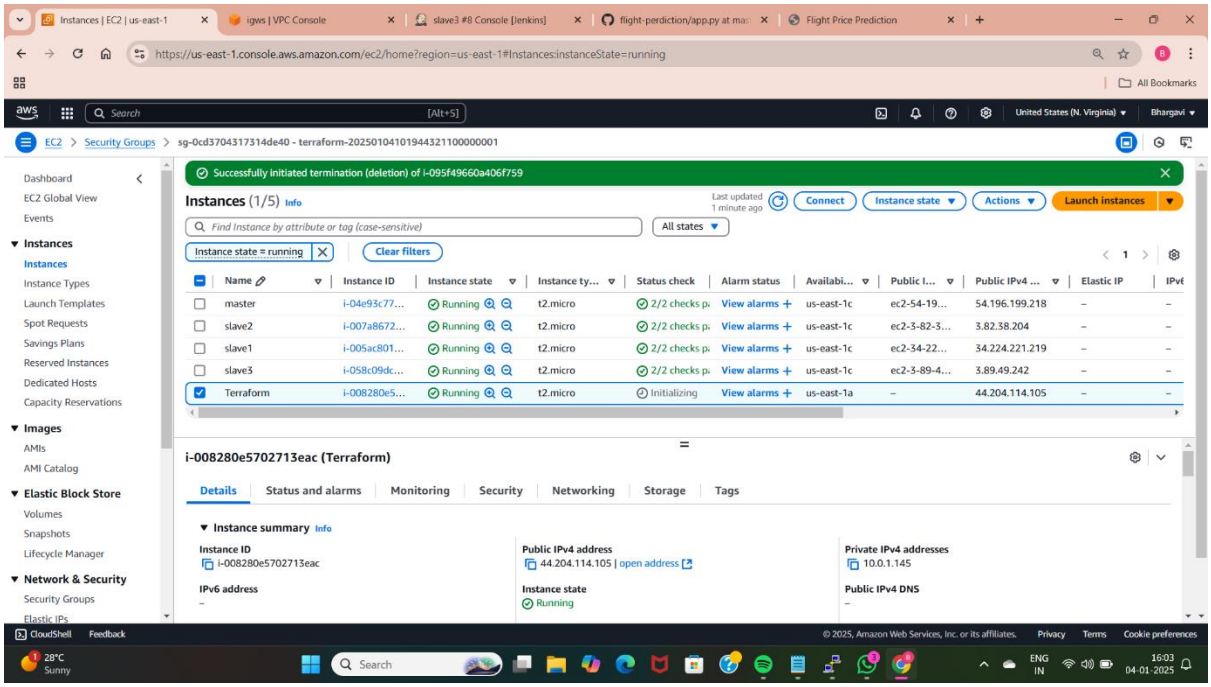
✓ Monitor the job's output to ensure the application is deployed on Slave-3. Here the build is started by the user Bairagoni Bhargavi, build is started by slave3 node.



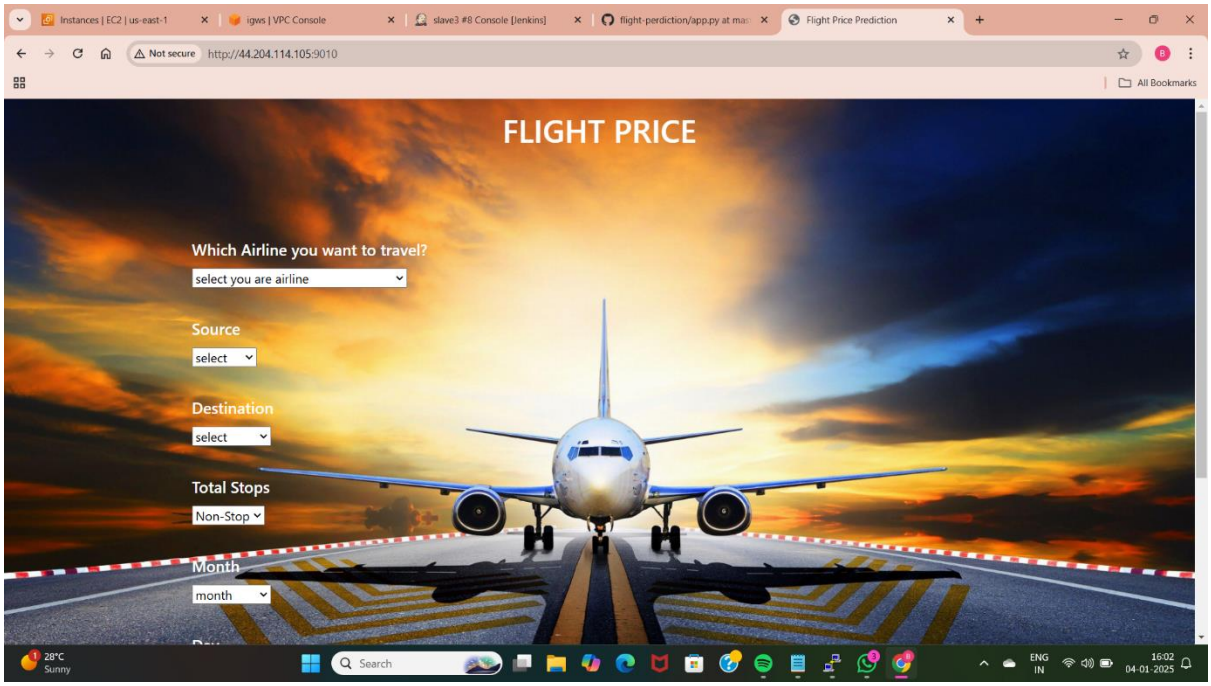
♣ Here we can see all the configurations i.e vpc, security group, instance, internet gateway, route table subnets are created.



- We can also check in the terminal when build is success whether the files are saved in Jenkins path or not. For that move to terminal and check files and find workspace. If workspace is found move inside the workspace and see the created job. Move inside the job and check whether files are deployed or not.
- Here we can see all the files related to terraform.

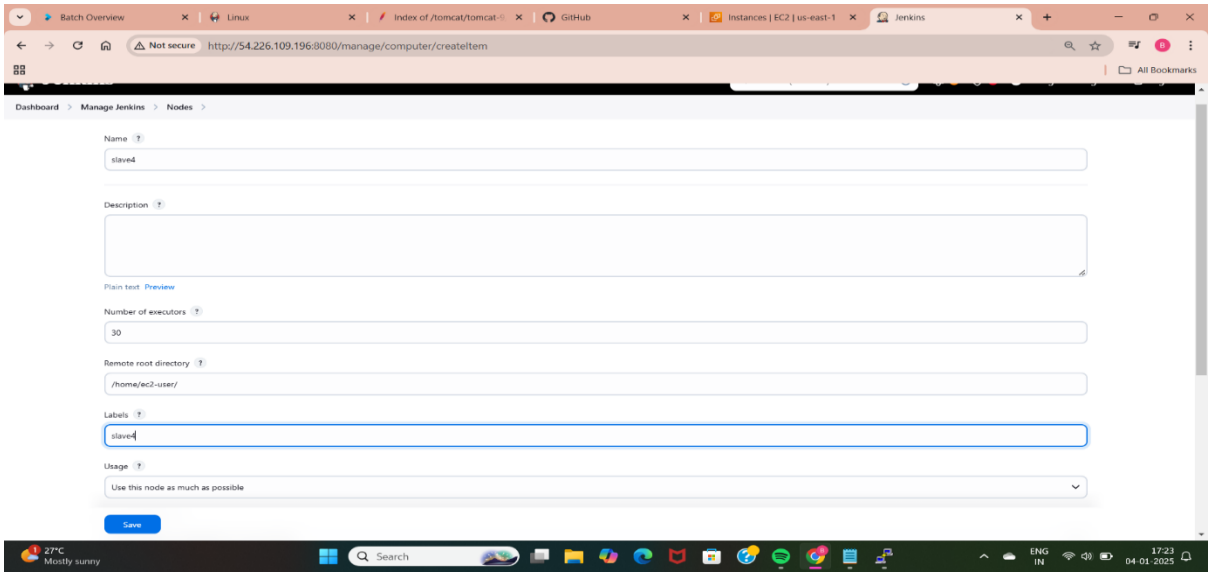


- After applying the changes it is going to create the aws instance with t2.micro, associating the public ip address to the created instance. Similarly it is creating all the configurations which are mentioned in the terraform files.

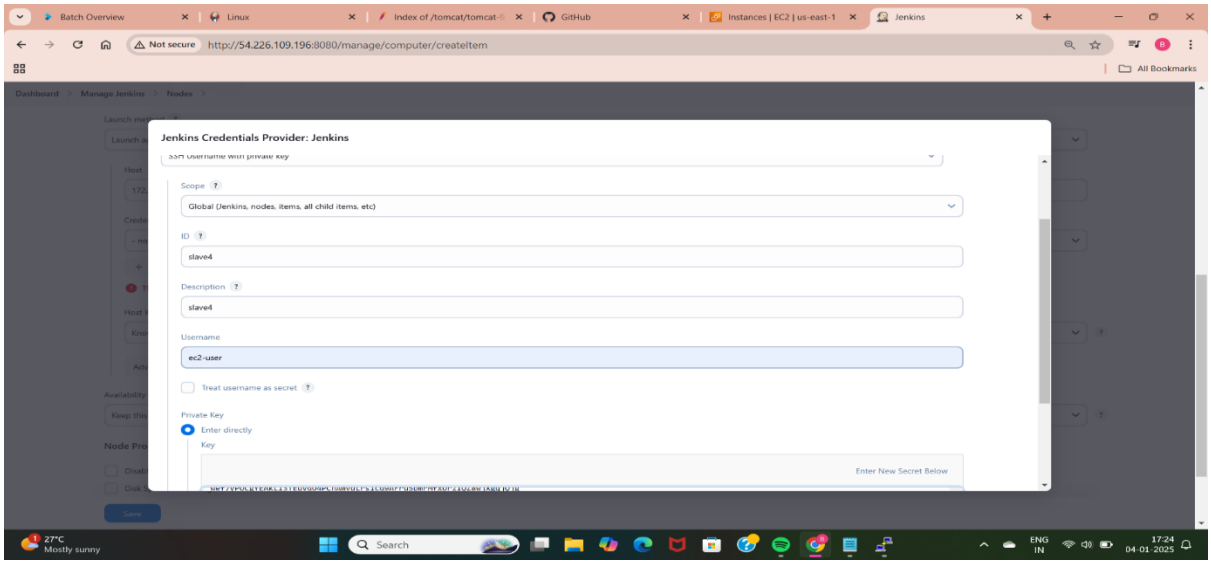


- ♣ Now to get the application we need to copy the public ip address of the slave server and paste in browser to access the application.

Slave-4: Deploy Java-Based Applications (Manual Deployment)

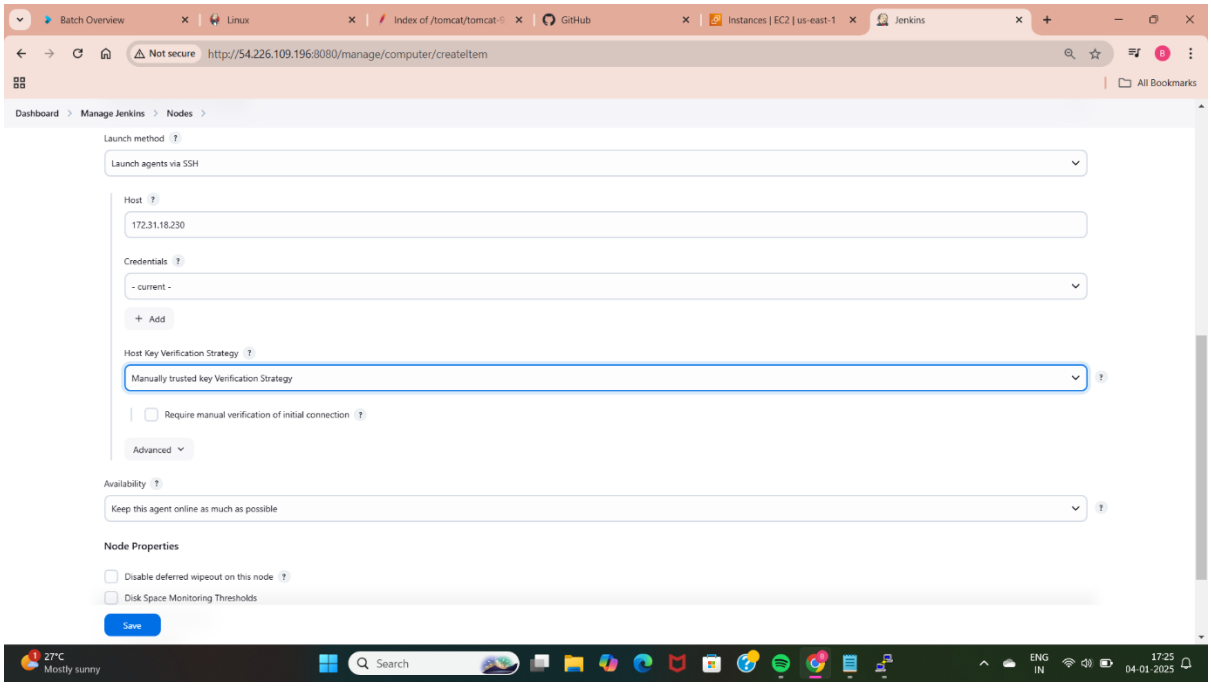


- ✓ Next we need to create a node in Jenkins using the above slave4 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave4”, select the number of executors(no. of builds) according to requirement.
- ✓ Give the remote root directory of the slave4 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.

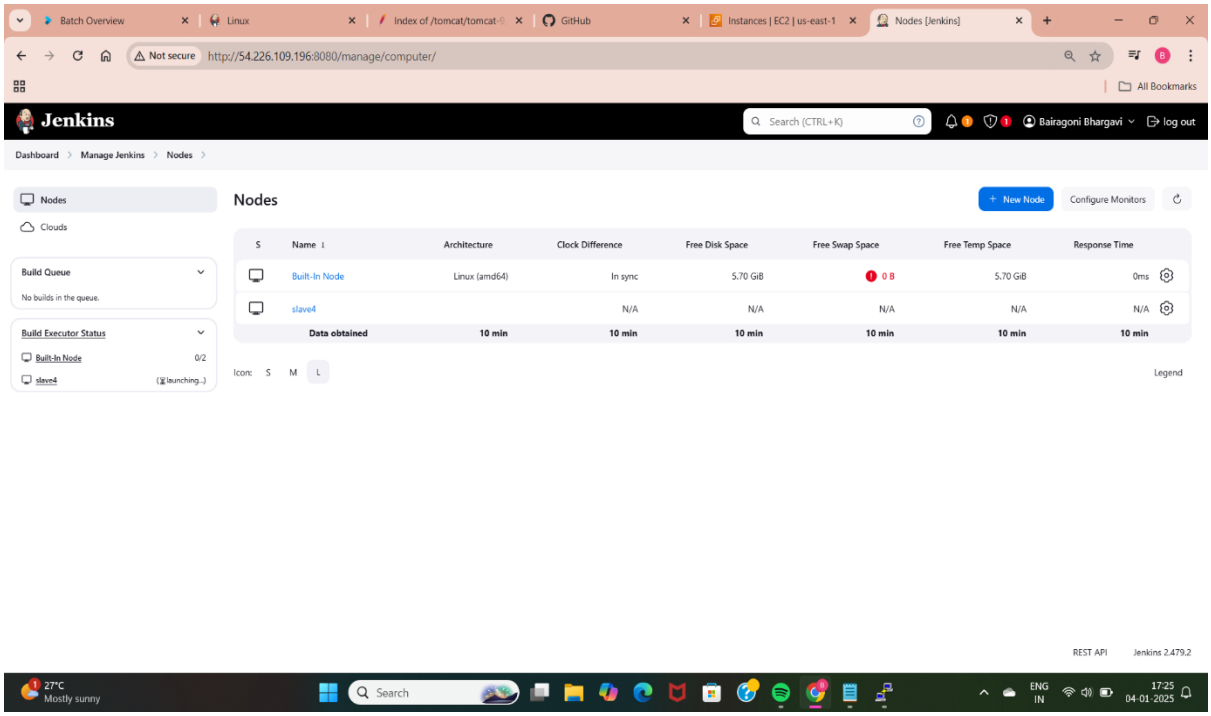


Add SSH credentials:

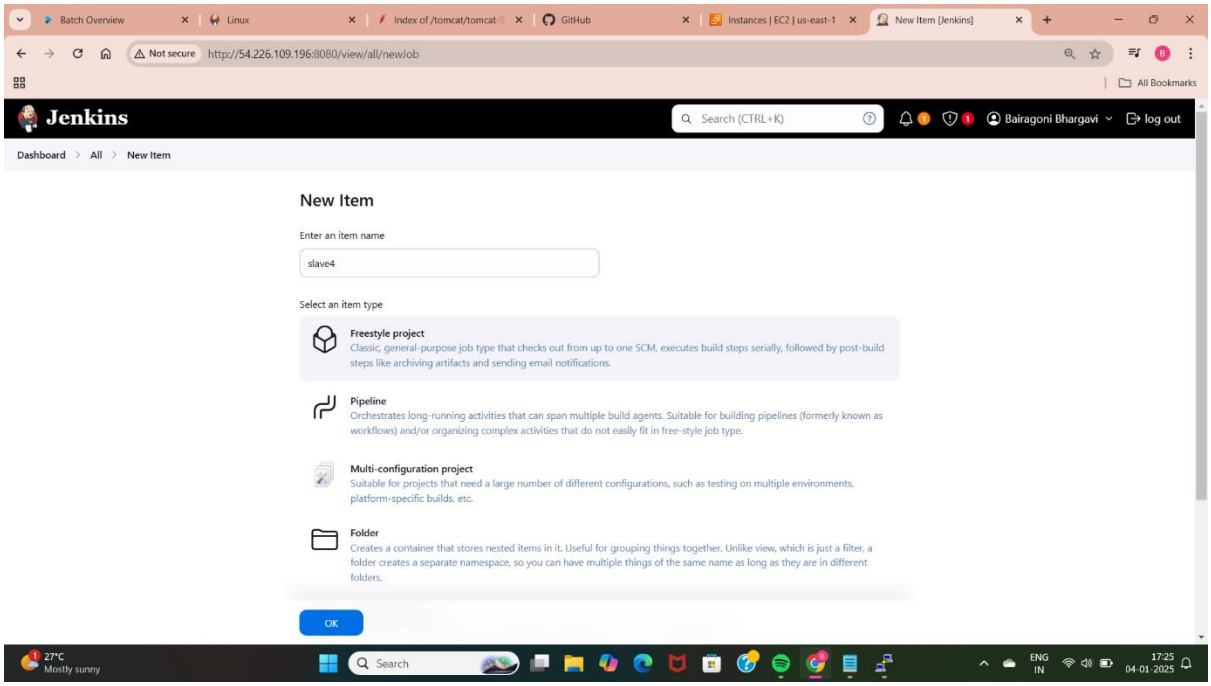
- ID: Assign a name for the credential.
- Username: Enter the slave's instance username (ubuntu, ec2-user, etc.).
- Private Key: Paste the private key generated on the master server.



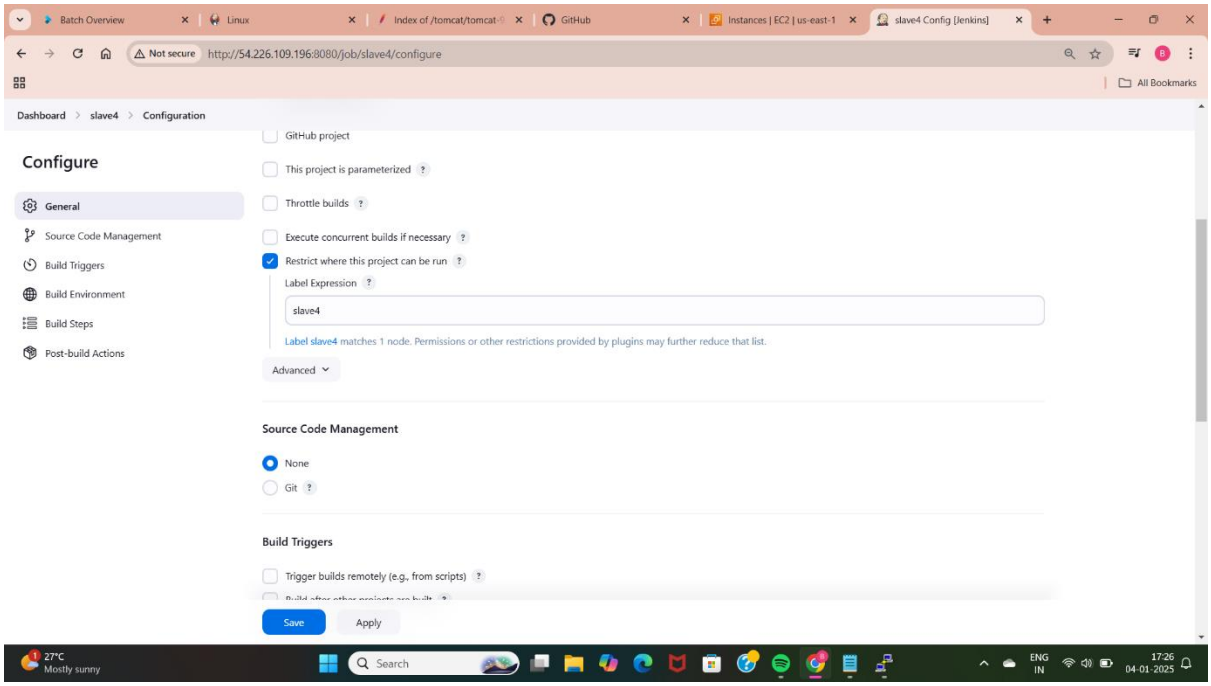
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



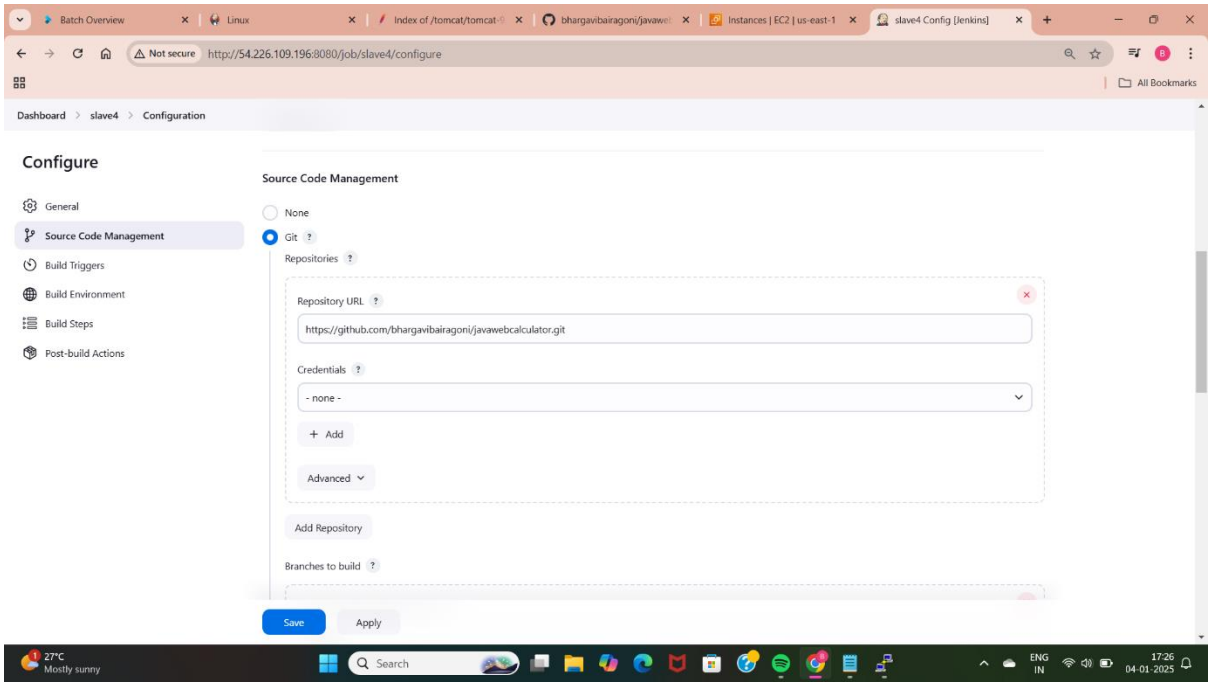
- Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful.



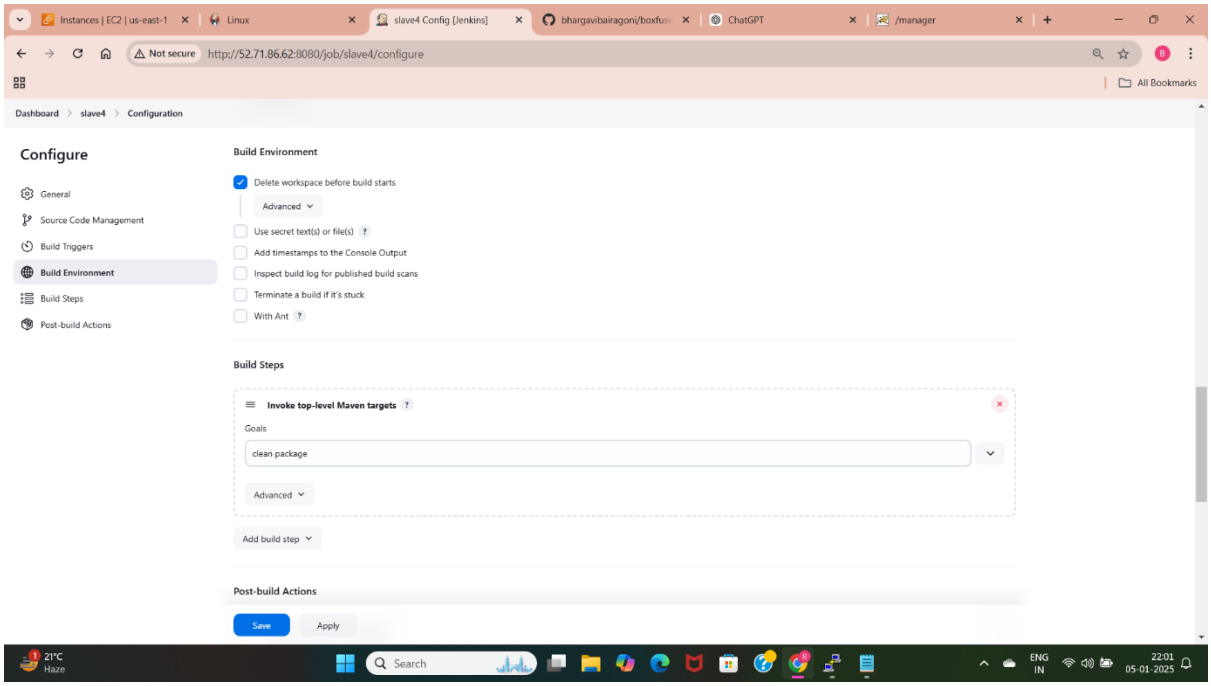
- Now am creating a new job by selecting new item at dashboard, give the name as slave4 for the new job.



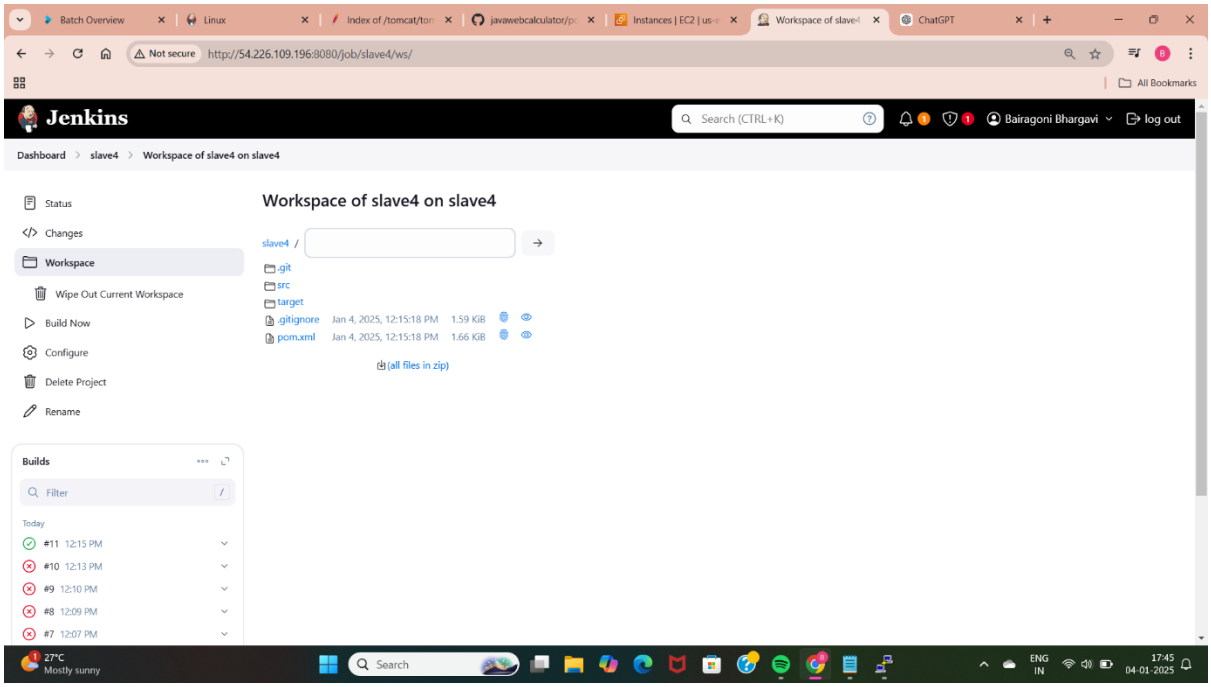
- Select the above created slave 4 by using “Restrict where this project can be run” and select the label name which is created above.



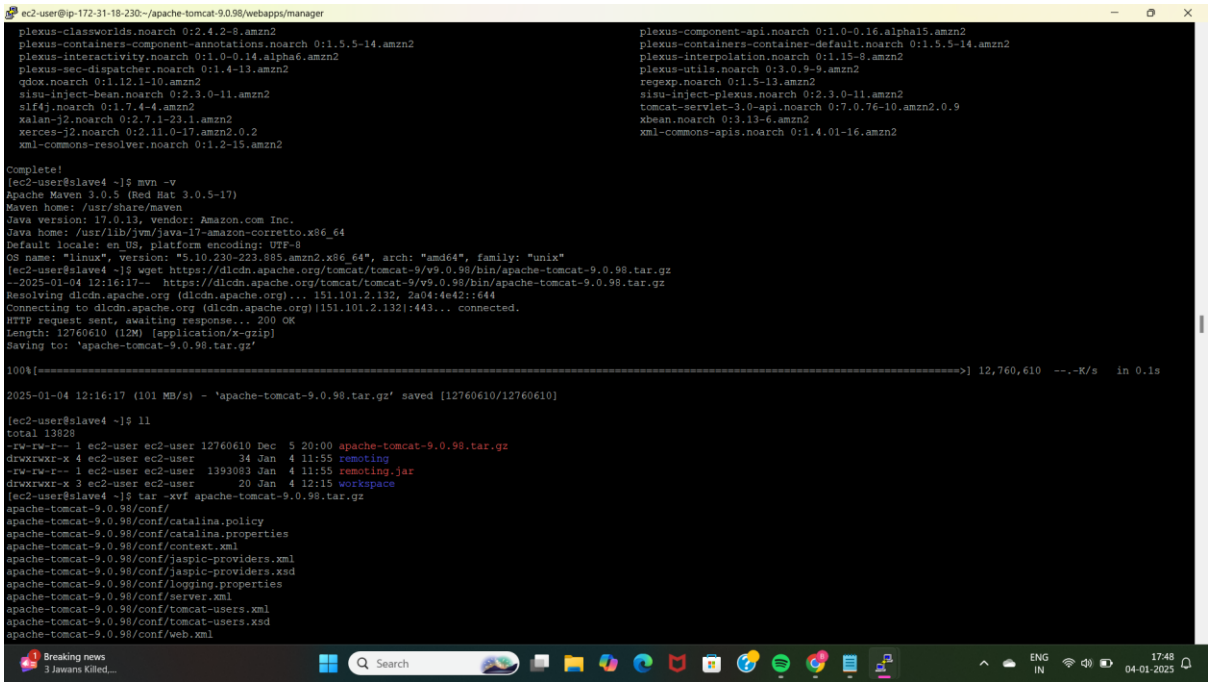
- Go to source code management and select git and give git url which is having the code related to java application and select the branch where we are having the code.



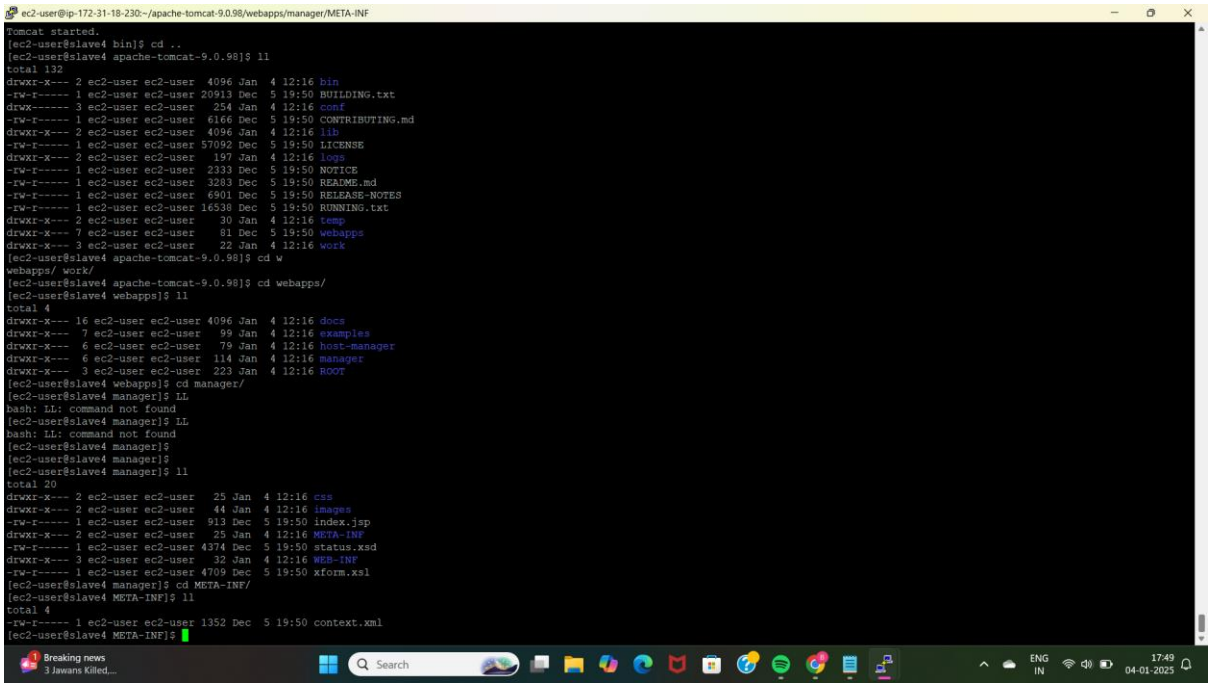
✓ Go to build steps and select “Invoke top level maven targets” and give goal as “clean package” to generate the target file. Save the job configuration and click Build Now. Monitor the job's output to ensure the application is deployed on Slave-1.



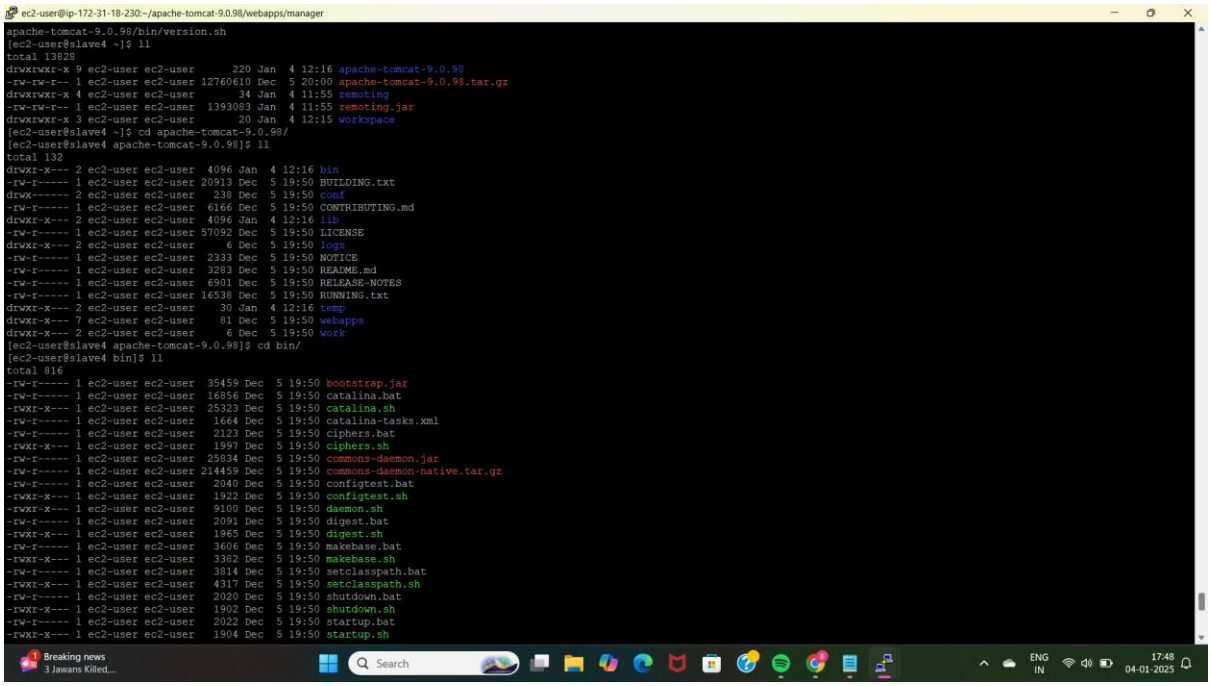
✓ Now we can build the job by clicking on “build now” option. Here build is successful so the files are deployed in Jenkins workspace.



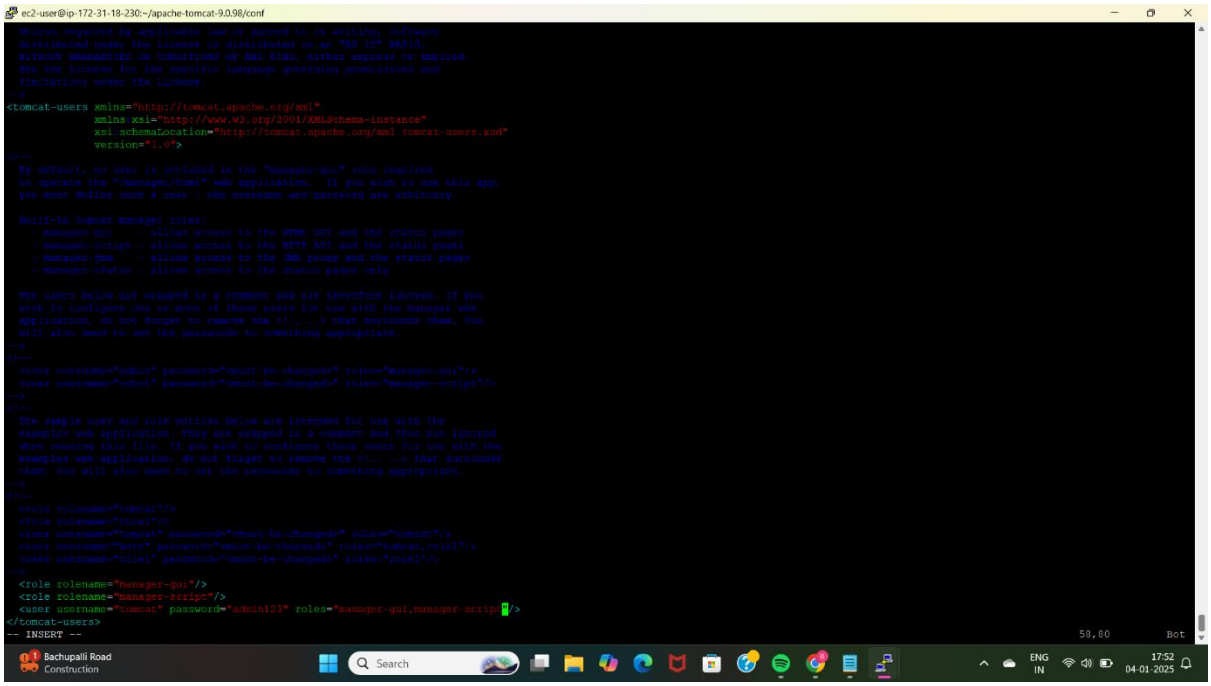
- Install tomcat on slave4 server. For that go to dlcdn.apache.org.in select tomcat and copy the tomcat url and download it using <wget tomcat-url>.
- Extract the file using <tar -xvf apache-tomcat-9.0.98.tar.gz>.



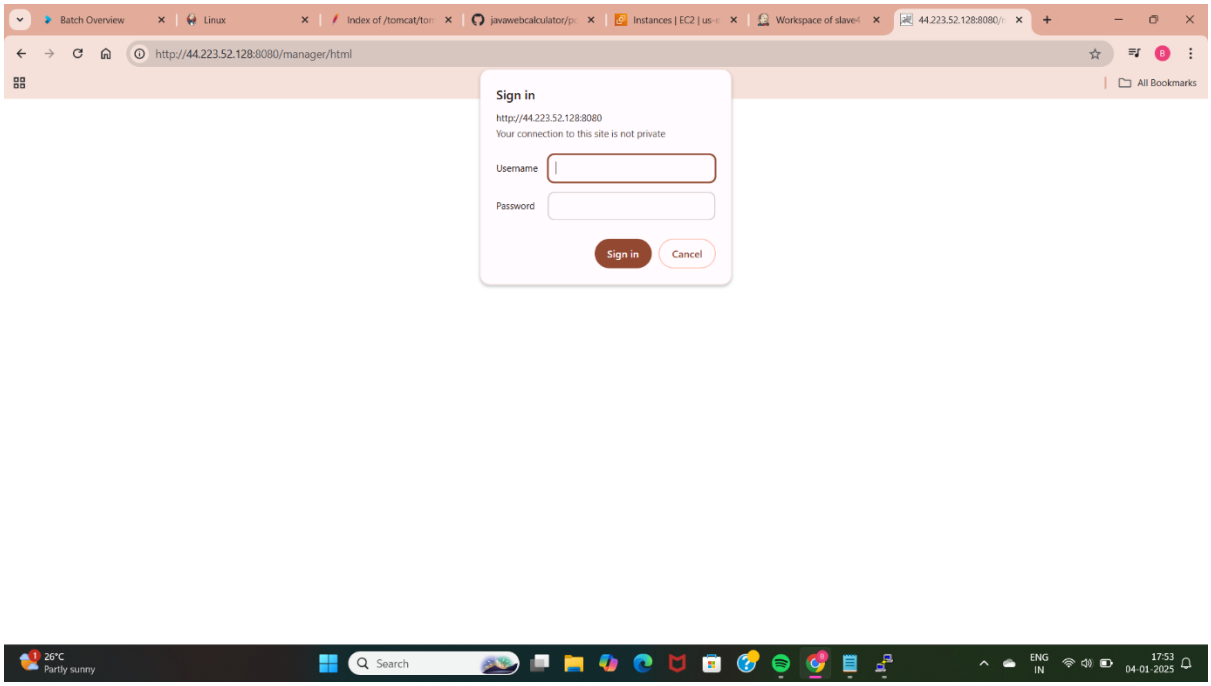
- Go to apache-tomcat.9.0.98/webapps/manager/META-INF/context.xml and delete two lines. The two lines typically removed are related to restrictions on access to the Manager App.



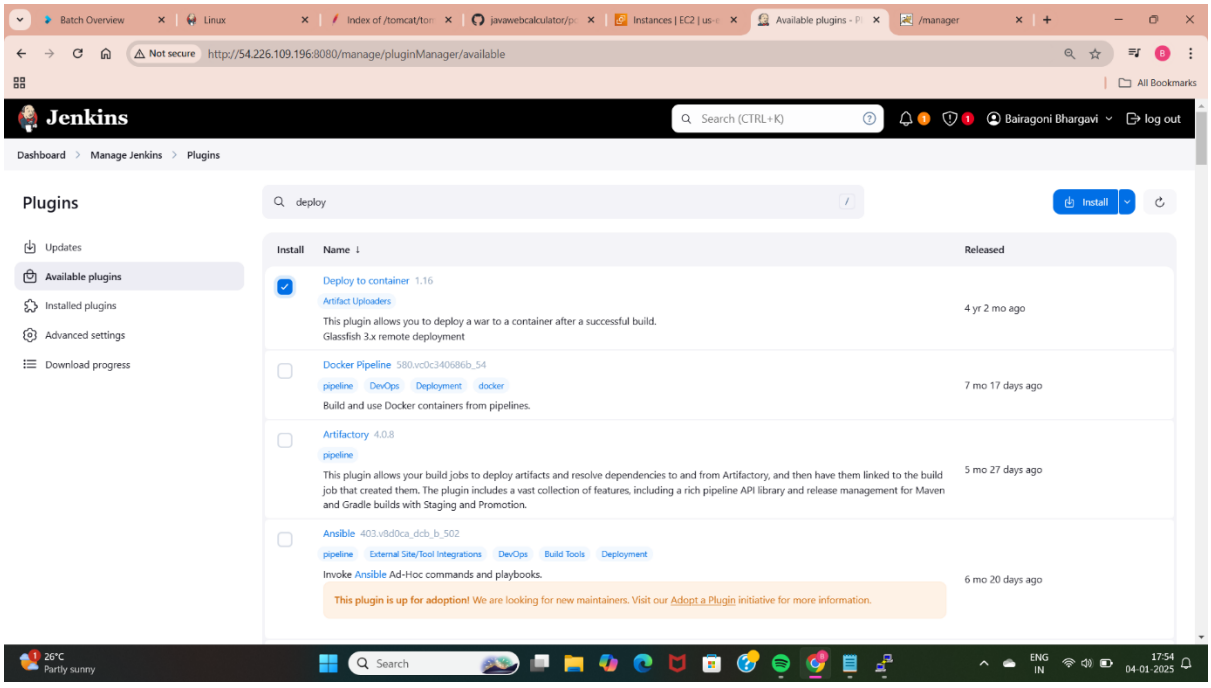
- ✓ After making changes, restart Tomcat by moving into apache-tomcat.9.0.98/bin to apply changes.
- ✓ First shutdown the tomcat using sh shutdown.sh and then start using sh startup.sh.
- ✓ Now copy the public ip address where we installed tomcat paste it in browser as <slavepublicip:8080>.
- ✓ Select the manager and access it. To access it we need to create the user and add roles.



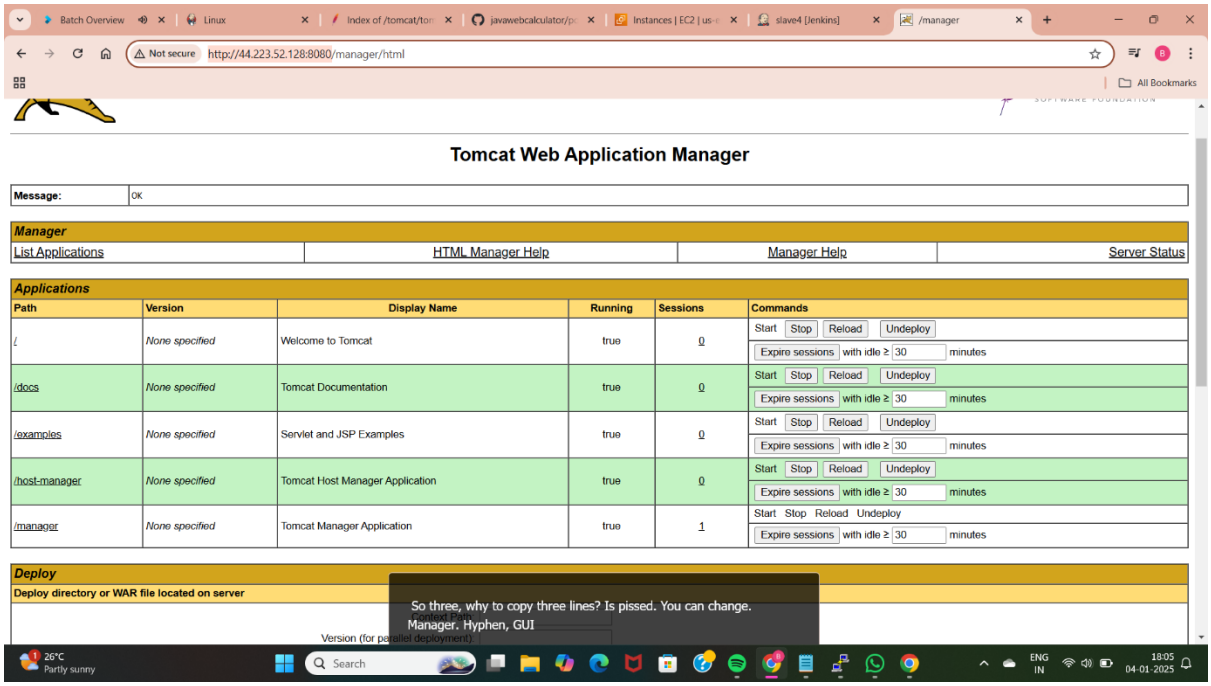
- ✓ Now go to apache-tomcat.9.0.98/conf and add a user with manager-gui and/or manager-script roles to enable access.



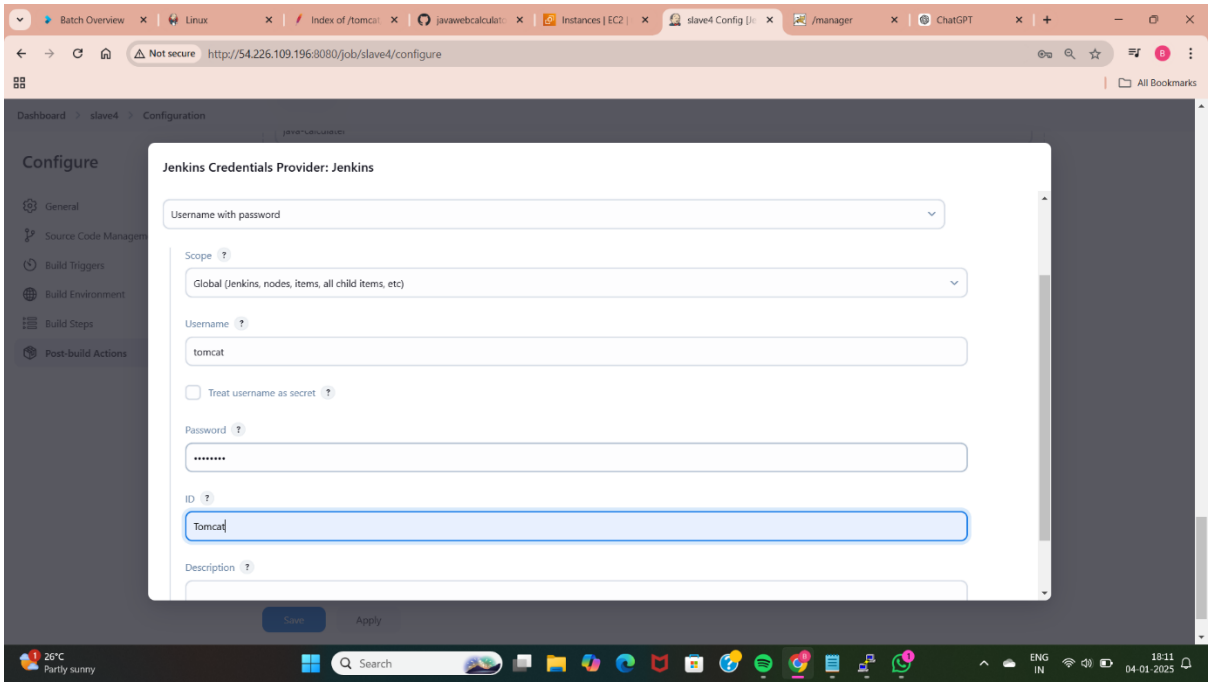
- ✓ Sign into tomcat manager using the above created username and password.



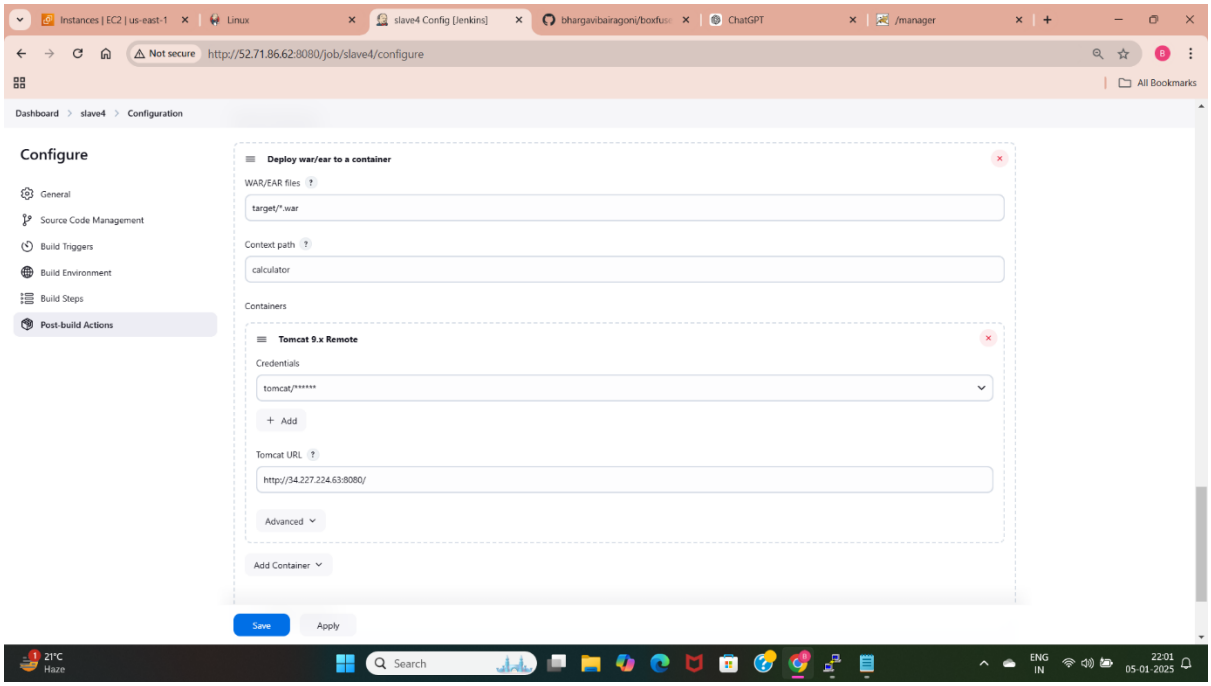
- ✓ To deploy java application into tomcat we need a plugin i.e “deploy to container” install it by moving to manage Jenkins>plugins>available plugins.



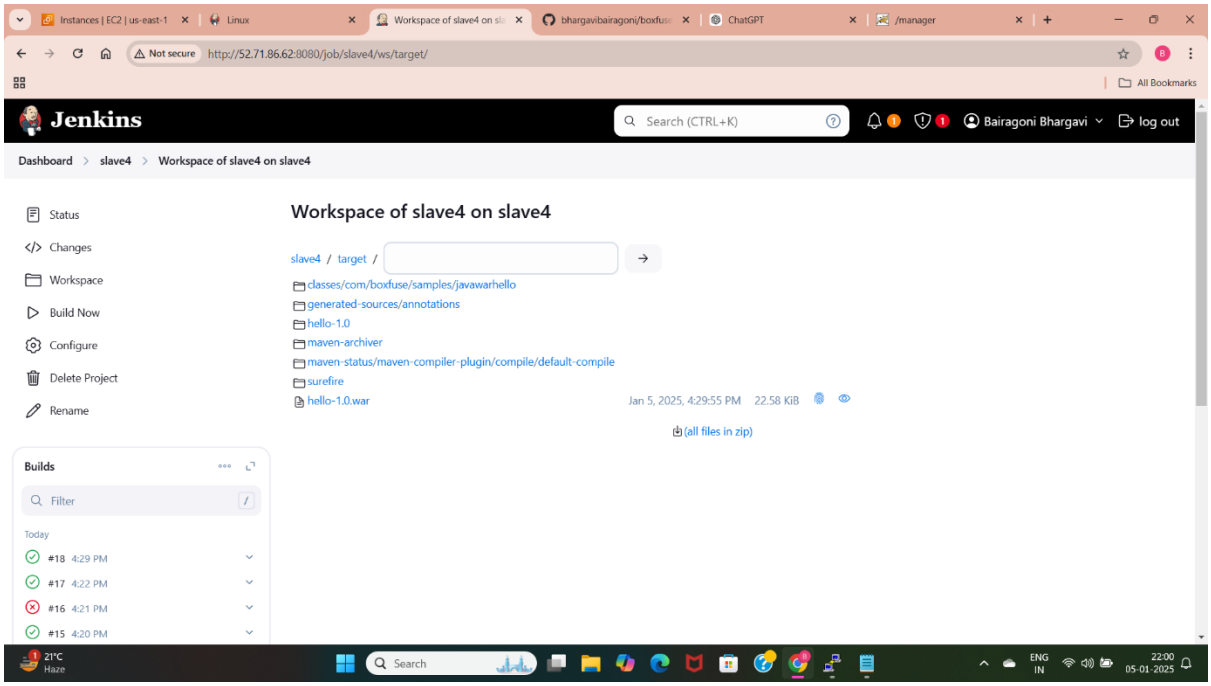
✓ This is the manager page we have logged into it using username and password.



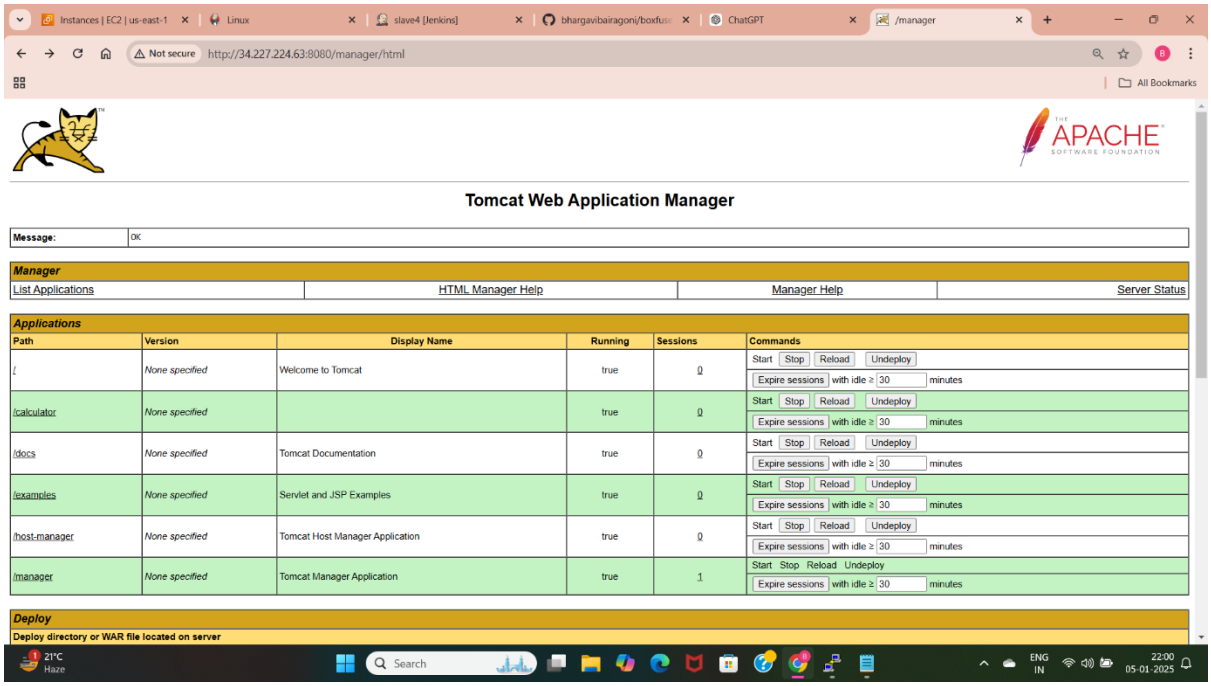
✓ We need to create credentials for tomcat. Go to credentials in manage Jenkins and select “username with password” and give username as tomcat which is created above and give above created password, id and save.



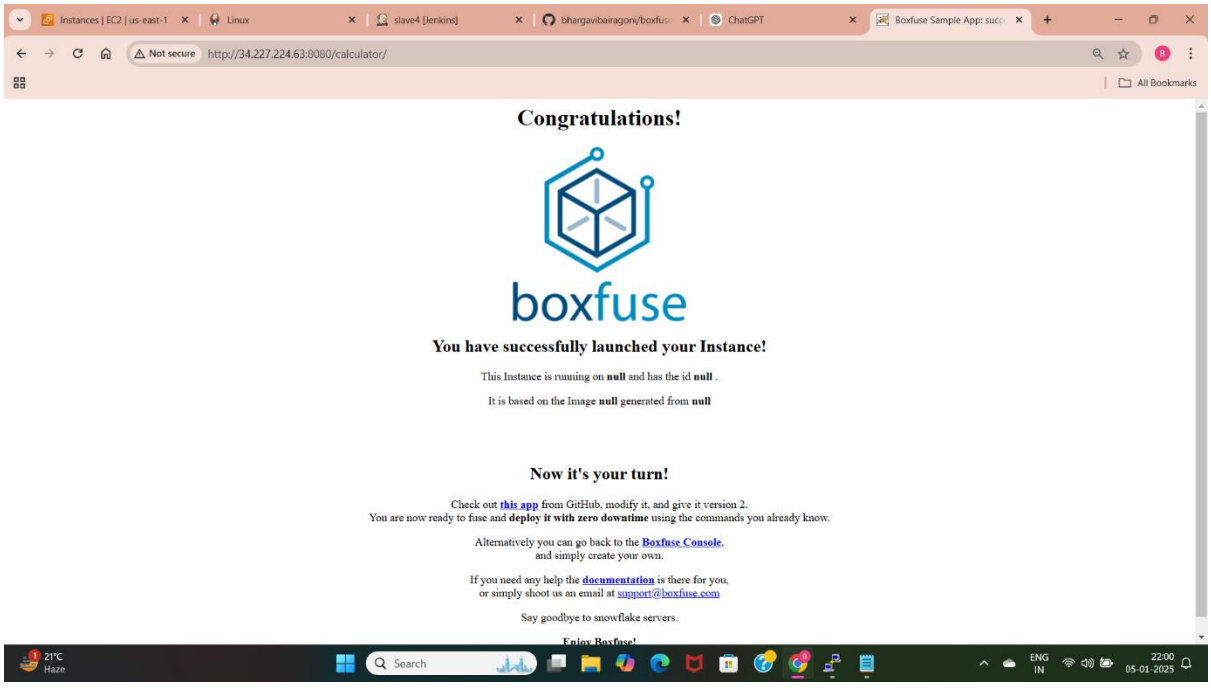
- ✓ Now go to “Post-build actions” and select deploy war/ear to a container and give the war file which is generated using maven. War file is under target folder so give name as target/*.war.
- ✓ Give any context path which is displayed on tomcat page to access the java application. I have given the context path as “calculator”.
- ✓ Select add containers “tomcat 9x remote” give created credentials and provide the tomcat url so we can deploy the java application there.



- ➢ After a successful build, Jenkins places the output files (e.g., compiled code, artifacts, logs) into the workspace directory of the job.

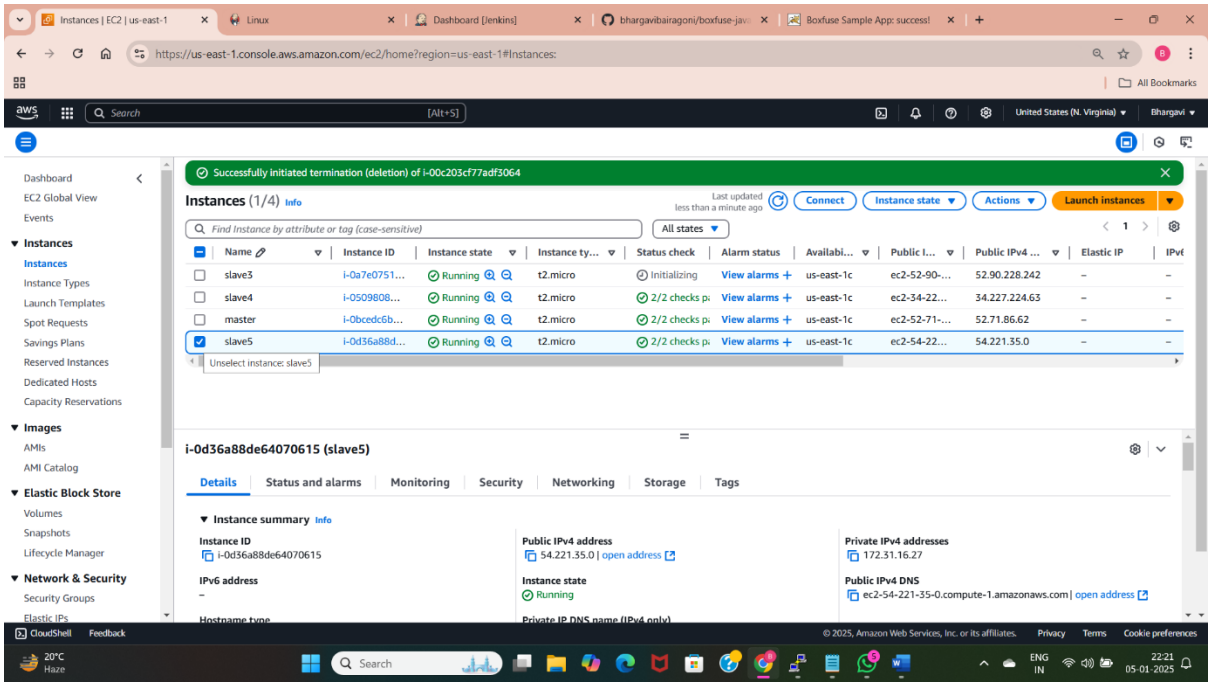


- In tomcat page we can find the calculator application deployed using Jenkins.

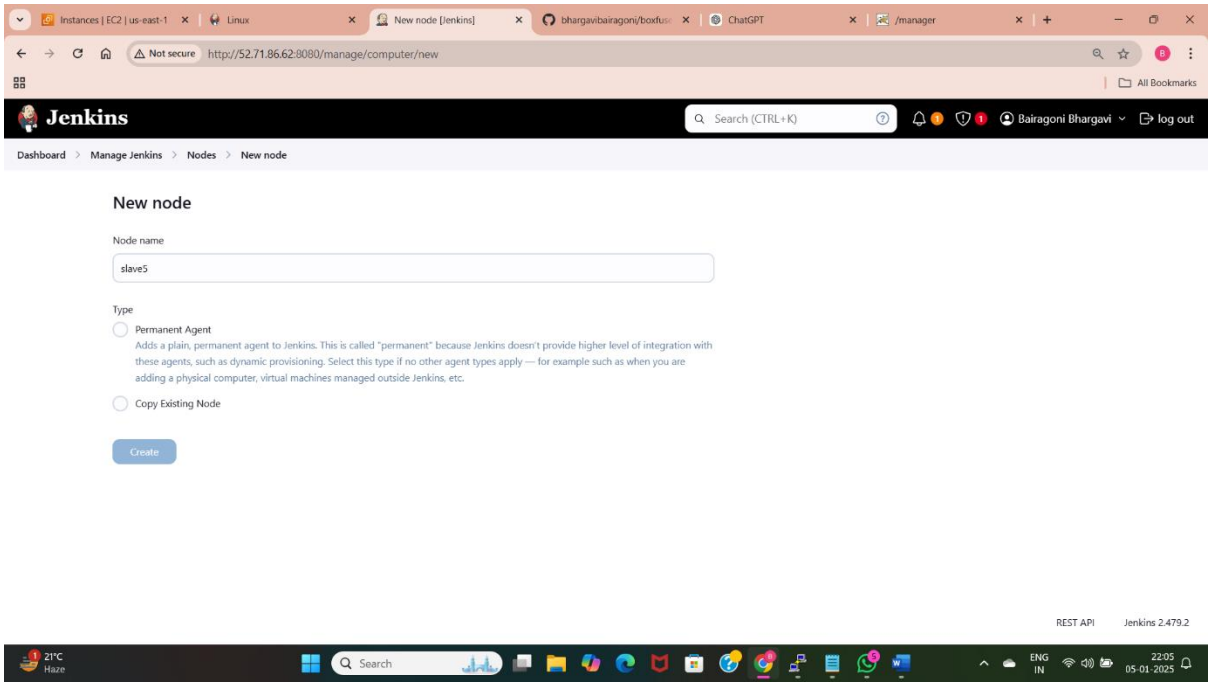


- This is the java application which is deployed using Jenkins and tomcat.

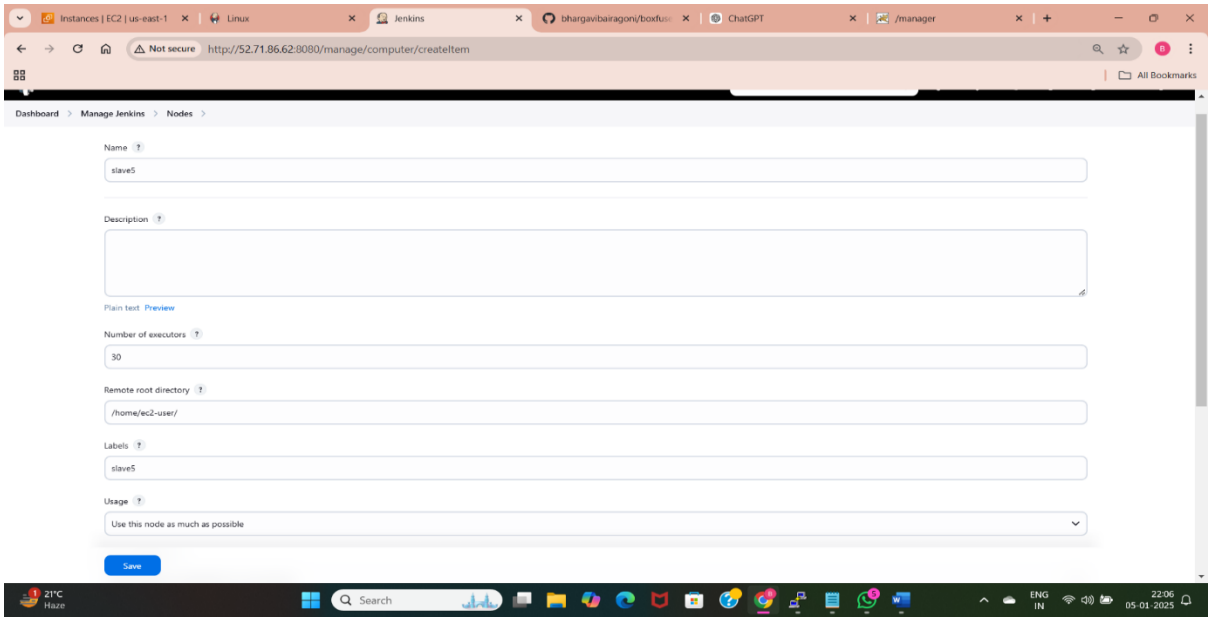
Slave-5: Deployment Using Bash Script



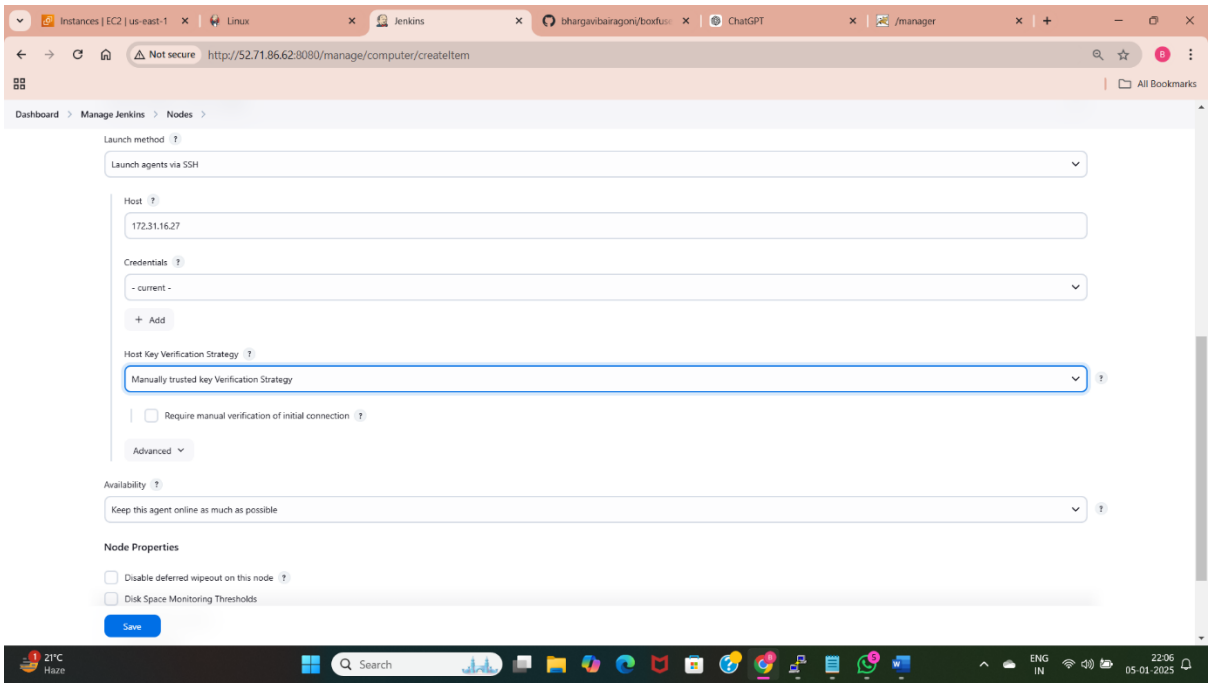
✓ I have launched a slave5 server to deploy java application using bash script.



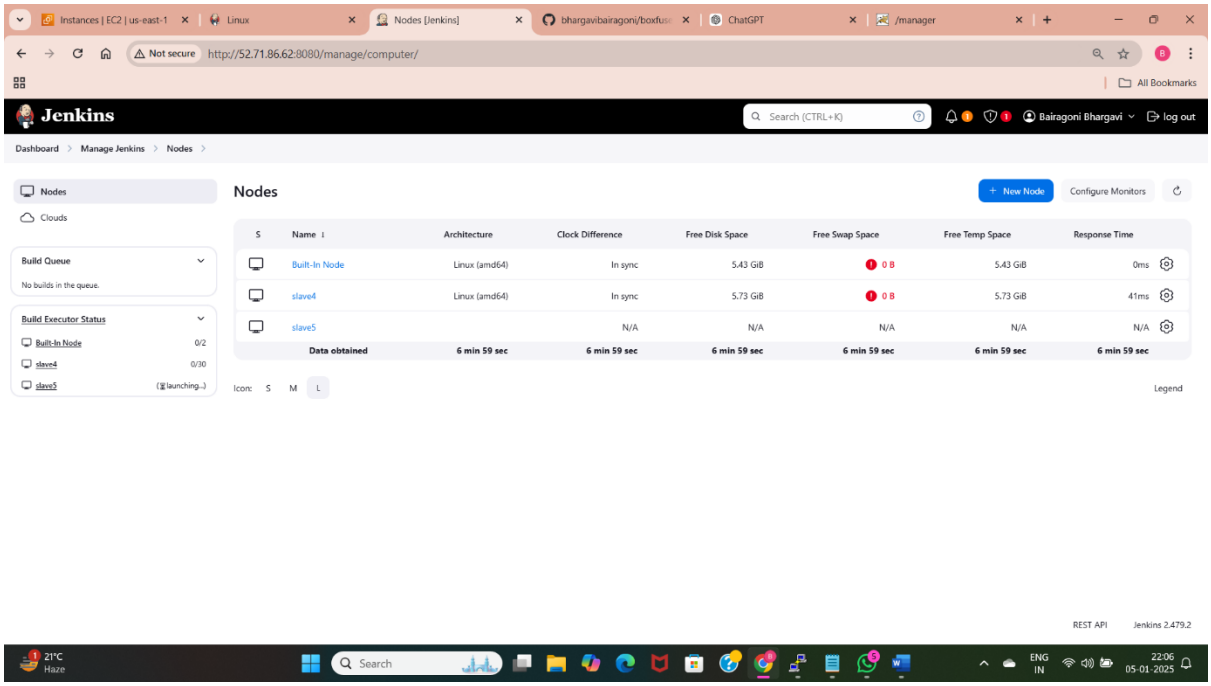
✓ Go to Manage Jenkins > Manage Nodes > New Node, give name for the node and select permanent agent and click “ok”.



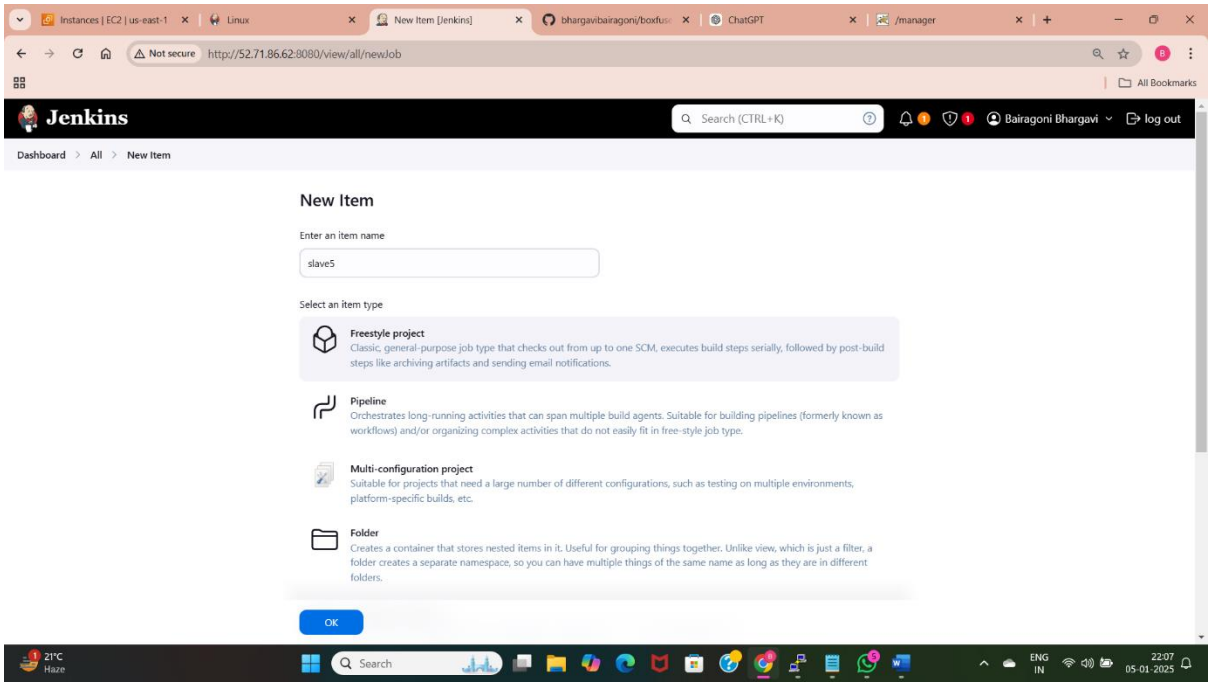
- ✓ Next we need to create a node in Jenkins using the above slave5 server.
- ✓ Here first we need to go to manage Jenkins>Nodes.
- ✓ Now I have given the name as “slave5”, select the number of executors(no. of builds) according to requirement.
- ✓ Give the remote root directory of the slave5 server i.e I have taken amazon-linux so root directory is “/home/ec2-user/”
- ✓ Give the name for label. We will use this label while running builds.
- ✓ Select usage as “use this node as much as possible” to execute builds.



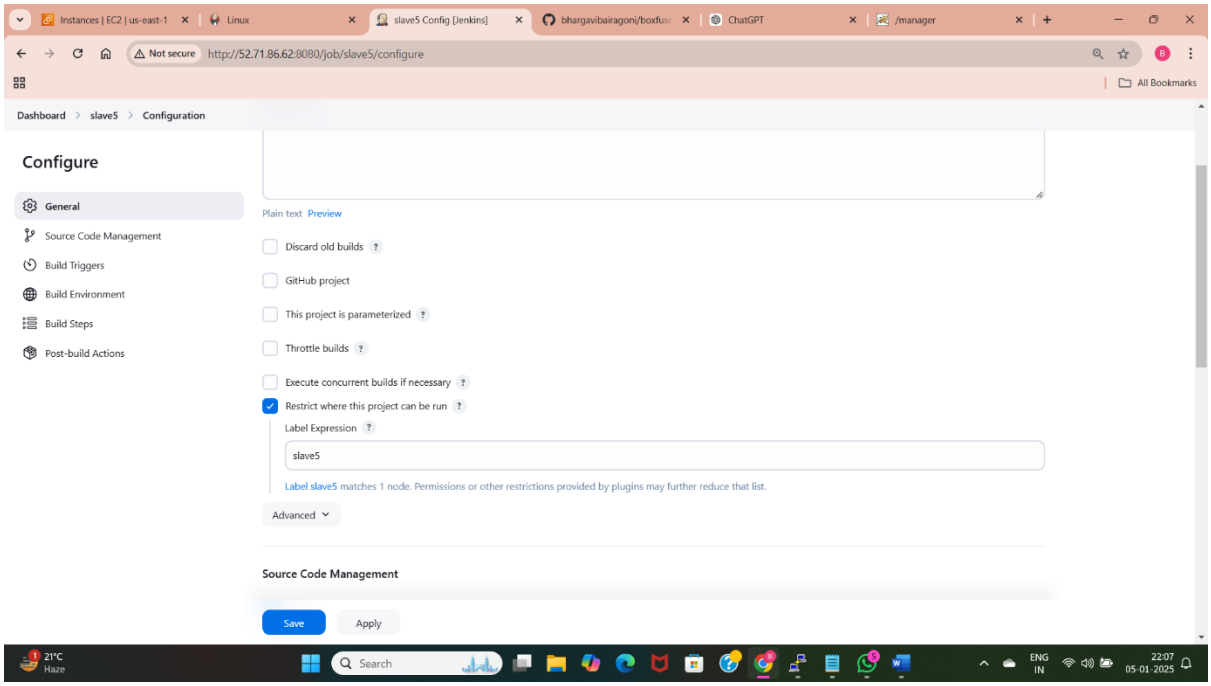
- ✓ I have selected the launch method as “launch agent via SSH” and provide the private keys there for authentication purpose.



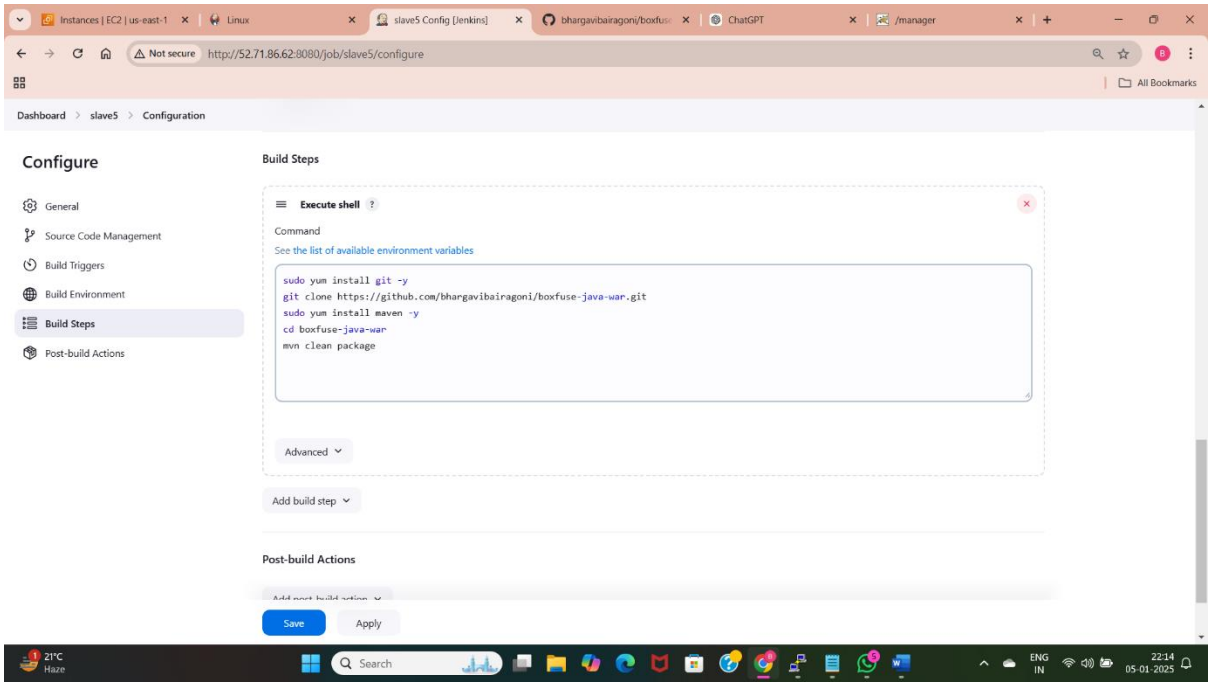
✓ Save the configuration and Jenkins will attempt to connect to the slave. Ensure the connection is successful and here slave5 is connected to master successfully.



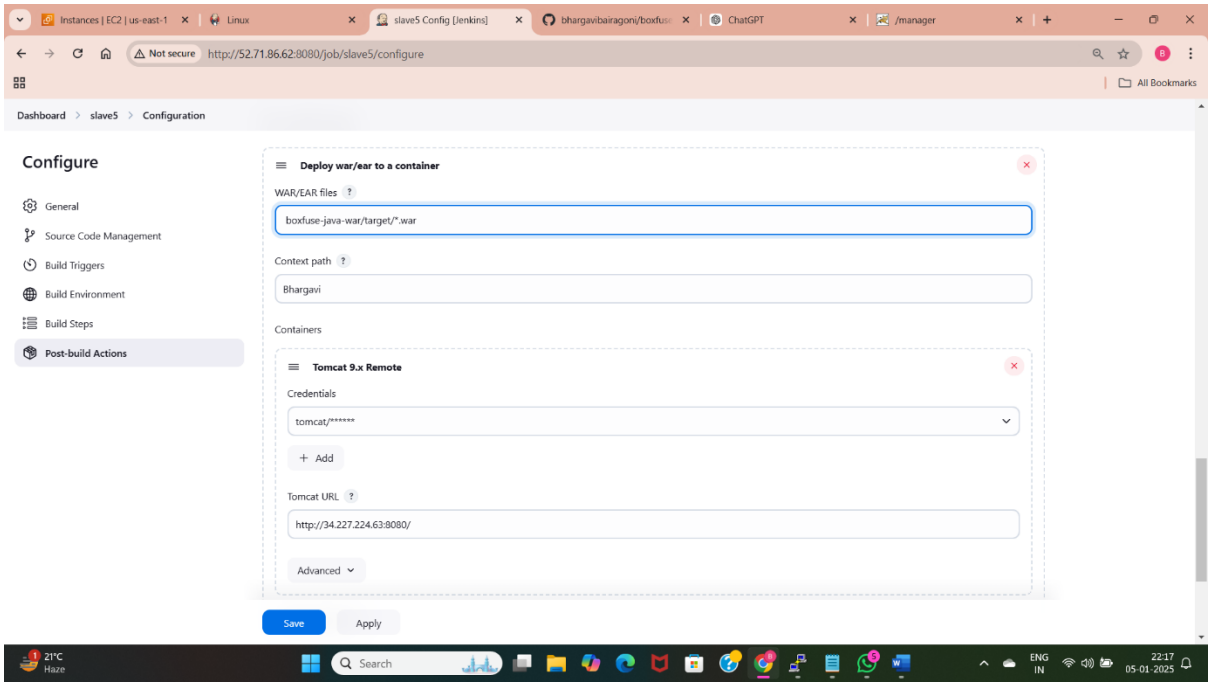
✓ Now am creating a new job by selecting new item at dashboard, give the name as slave5 for the new job.



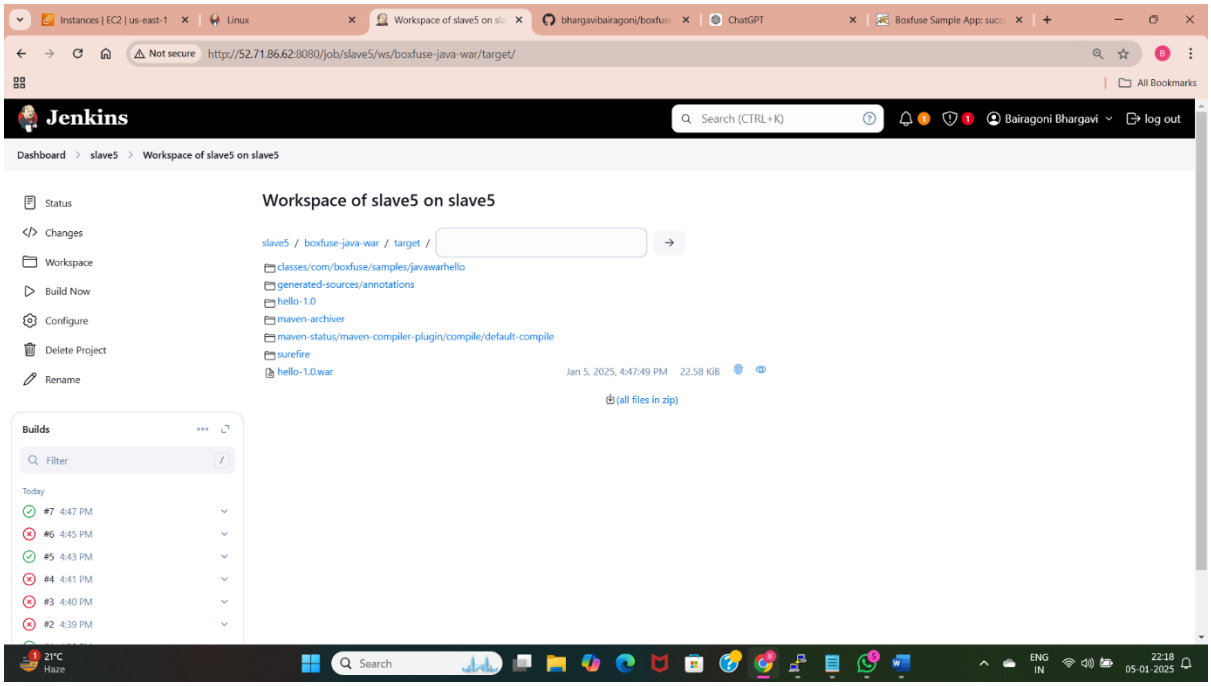
- ✓ Select the above created slave5 by using “Restrict where this project can be run” and select the label name which is created above.



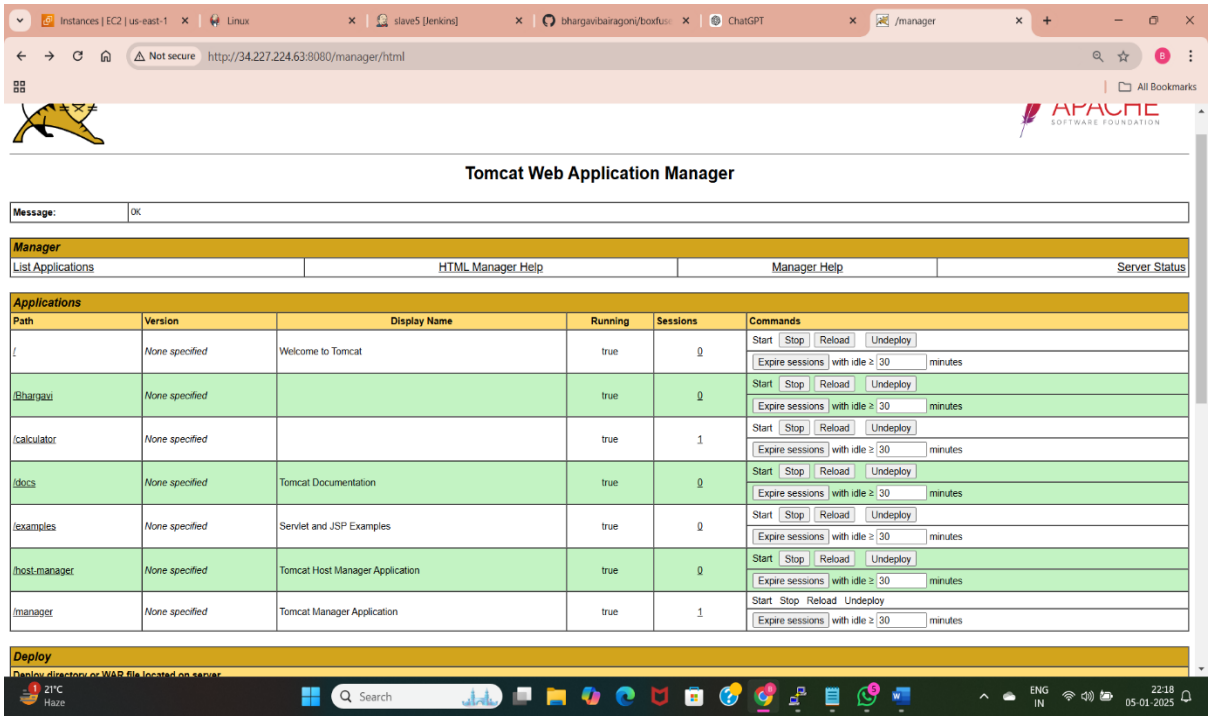
- ✓ Now go to build steps and select execute shell write commands to install git, and clone the github url which is having java application.
- ✓ Next install maven in that directory and move to the repository where we have java files.
- ✓ Now generate war file from the application using <mvn clean package>.



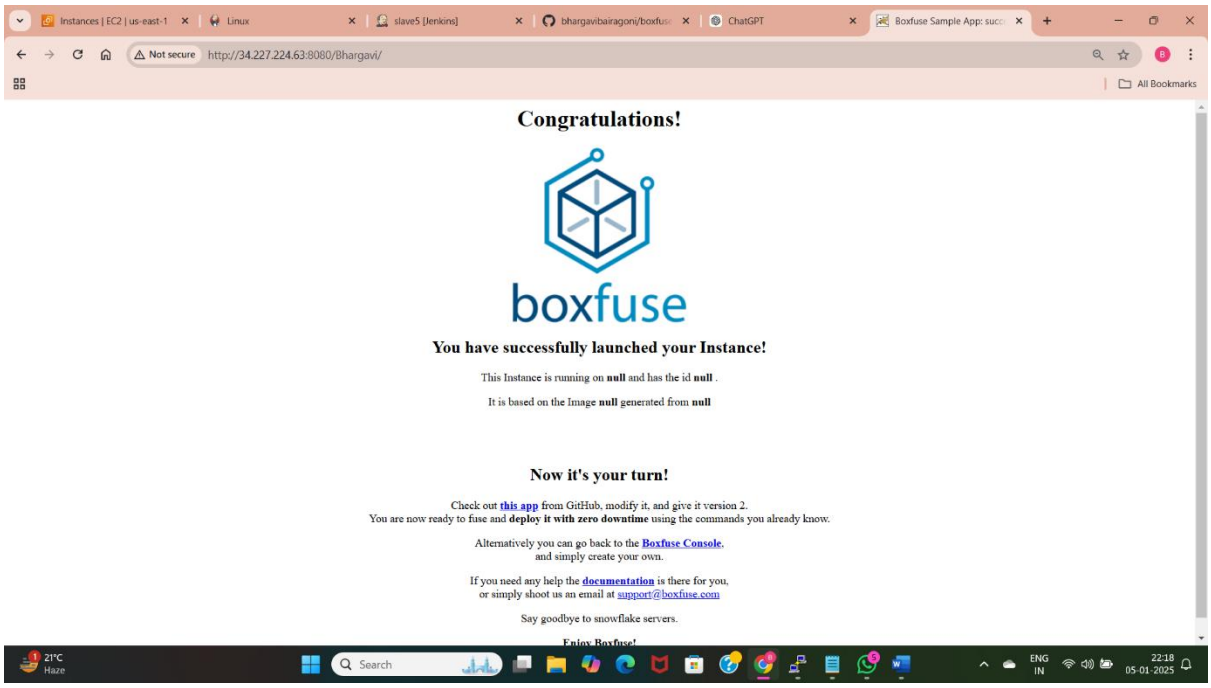
- ✓ Now go to “Post-build actions” and select deploy war/ear to a container and give the war file which is generated using maven. War file is under target folder so give name as target/*.war.
- ✓ Give any context path which is displayed on tomcat page to access the java application. I have given the context path as “Bhargavi”.
- ✓ Select add containers “tomcat 9x remote” give created credentials and provide the tomcat url so we can deploy the java application there.



- ✓ Now click on build now option so that we can deploy the application into tomcat and it can also performs the tasks whatever mentioned in the execute shell.



- ✓ In tomcat web page we can see the application path “Bhargavi” which is having the java application.



- ✓ We can access the application by clicking on the path in tomcat and this is the application I have deployed.