Guide to Setting Up Git, GitHub, Jenkins with Amazon EC2, and Building a Node.js Application

This comprehensive guide outlines the step-by-step process for setting up and integrating several key technologies crucial for modern software development and deployment. The document will cover the installation, configuration, and use of Git for version control and collaboration, GitHub for remote repository management, and Jenkins for continuous integration and continuous delivery (CI/CD) on an Amazon EC2 instance. Additionally, it will guide you through the process of building and deploying a Node.js application. Each section will provide detailed instructions, from the initial setup of each tool to the final deployment of your application, ensuring that you have a seamless and efficient development workflow. By following this guide, you will be equipped to manage your code repositories, automate your build processes, and deploy applications with ease, leveraging the power and flexibility of these industry-standard tools.

Table of Contents

1. **Introduction** **2**
   - Purpose of the Document 2
   - Scope 2
   - Prerequisites 2
2. **Setting Up Git and GitHub** **2**
   - Installing Git 2
   - Configuring Git 2
   - Creating a GitHub Repository 3
   - Cloning the Repository 5
3. **Setting Up Amazon EC2** **6**
   - Creating an EC2 Instance 7
   - Connecting to the EC2 Instance 10
   - Setting Up Security Groups 11
4. **Installing Jenkins** 13
   - Installing Java 13
   - Installing Jenkins 13
   - Configuring Jenkins 14
5. **Configuring Jenkins for Node.js Application** 14
   - Installing Node.js on EC2 15
   - Setting Up Jenkins Job for Node.js Application 16
   - Configuring GitHub Integration 17
   - Building and Deploying the Node.js Application 18
   - Automate with Webhooks and pm2 Library 19
   - Master – slave concept 21
6. **Conclusion** 25
   - Summary of Steps 25
   - Troubleshooting Tips 25

1. Introduction

Purpose of the Document:

This document aims to provide a comprehensive guide on setting up Git, GitHub, Jenkins with Amazon EC2, and building a Node.js application.

Scope:

The documentation covers installation, configuration, and deployment steps for the mentioned technologies.

Prerequisites:

- Basic knowledge of Git, GitHub, Jenkins, AWS, and Node.js

- AWS account with access to EC2 and SSH key for EC2 access

- GitHub account

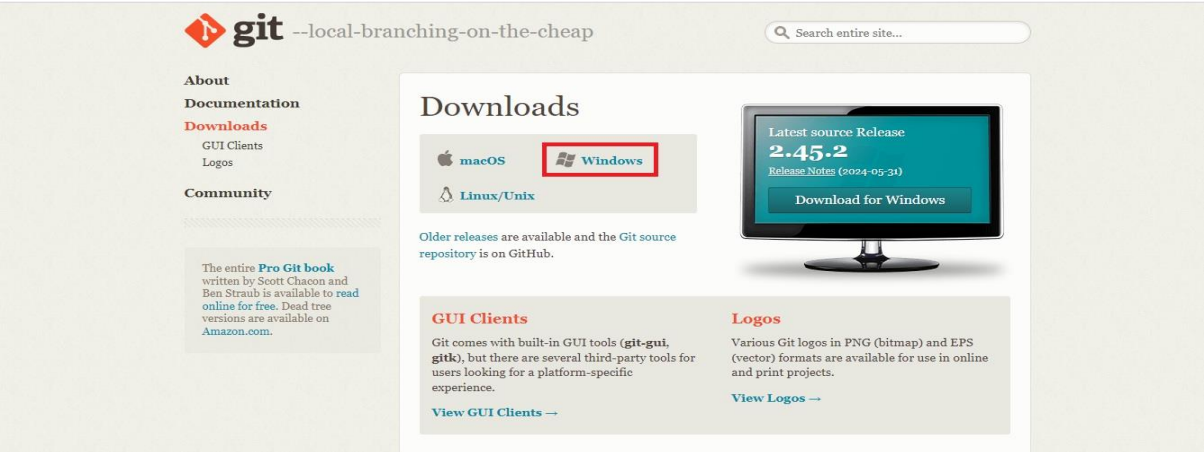
2. Setting Up Git and GitHub

Installing Git:

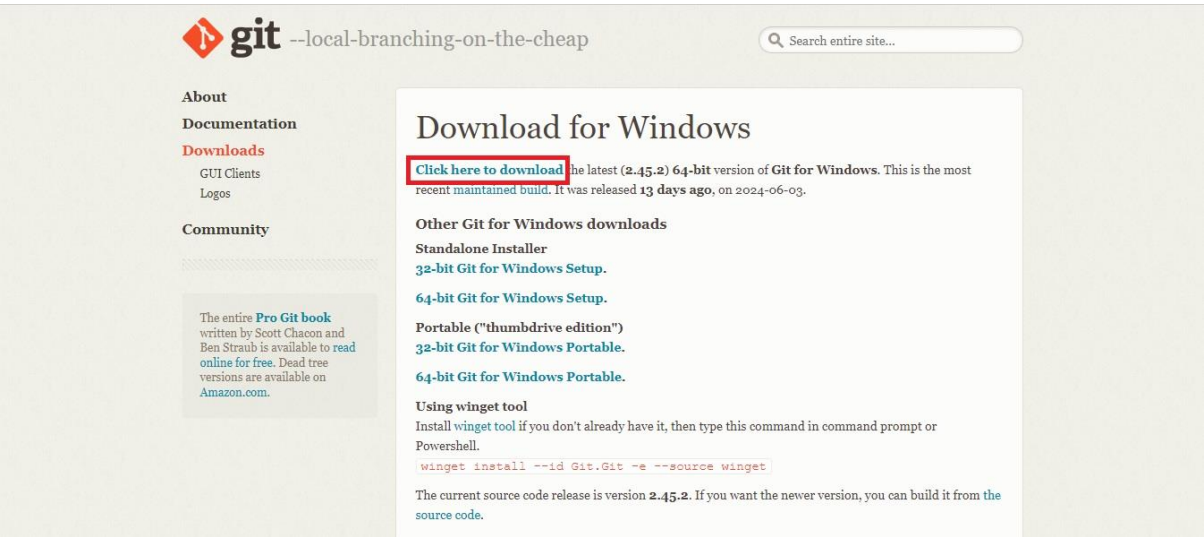
Download Git from [git-scm.com] (https://git-scm.com/downloads).

Follow the installation instructions for your operating system.

For windows:



Make sure that download latest version:



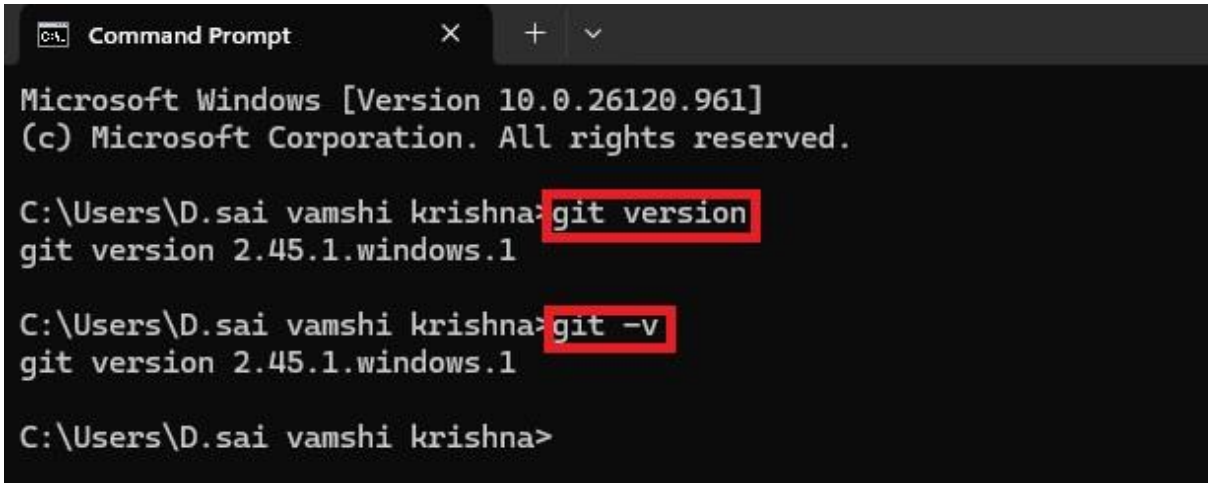
Reference video if needed: https://www.youtube.com/watch?v=8HhEupU4iGU

Configuring Git:

1. Open a terminal or command prompt.

Check the installation status to see whether it is installed or not.

The "**git version**" or "**git -v**" command displays the currently installed version of Git.

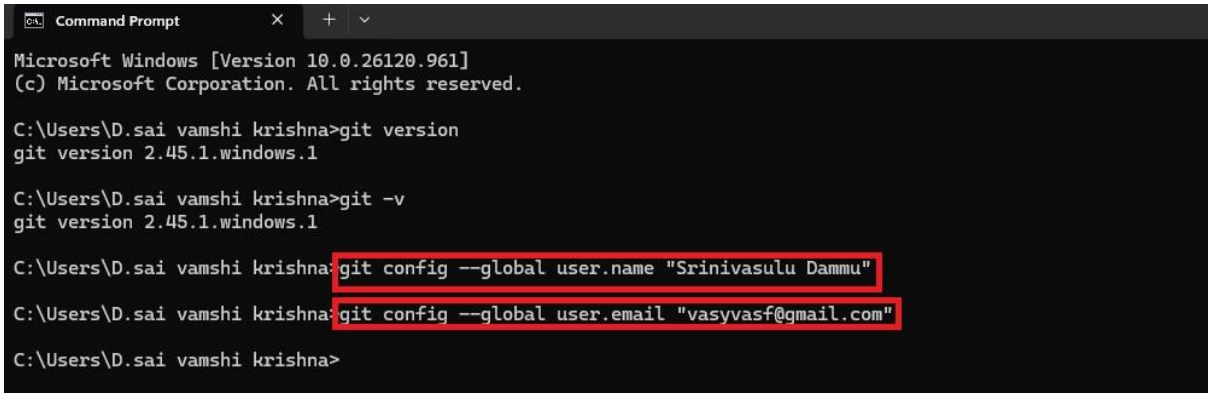


2. Set your Git username and email:

In bash/CMD:

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"



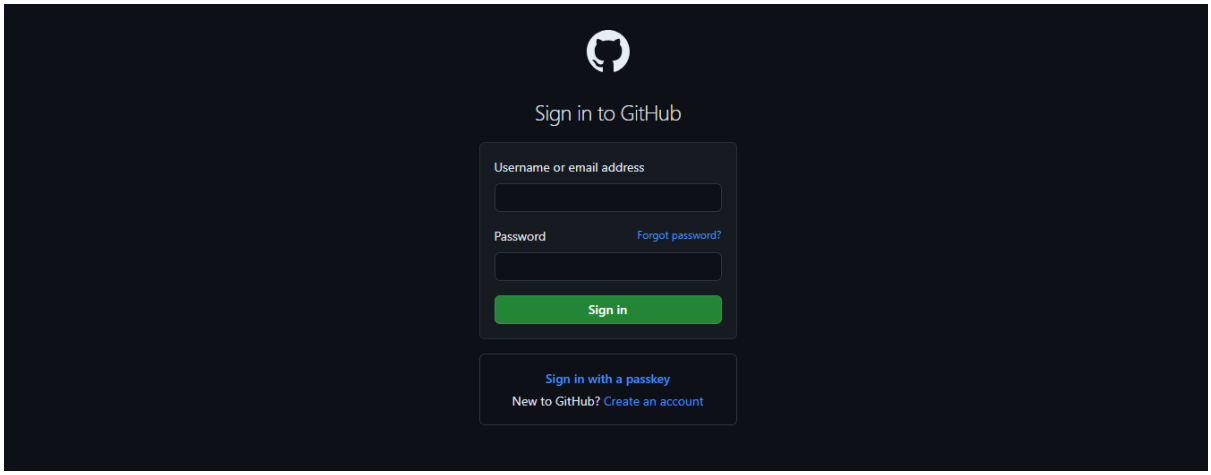
We set up our Git username and email to associate our identity with the commits we make. This information is included in each commit, allowing others to see who made changes to the code and providing accountability and traceability in the version control history.

Creating a GitHub Repository

1. Log in to GitHub.

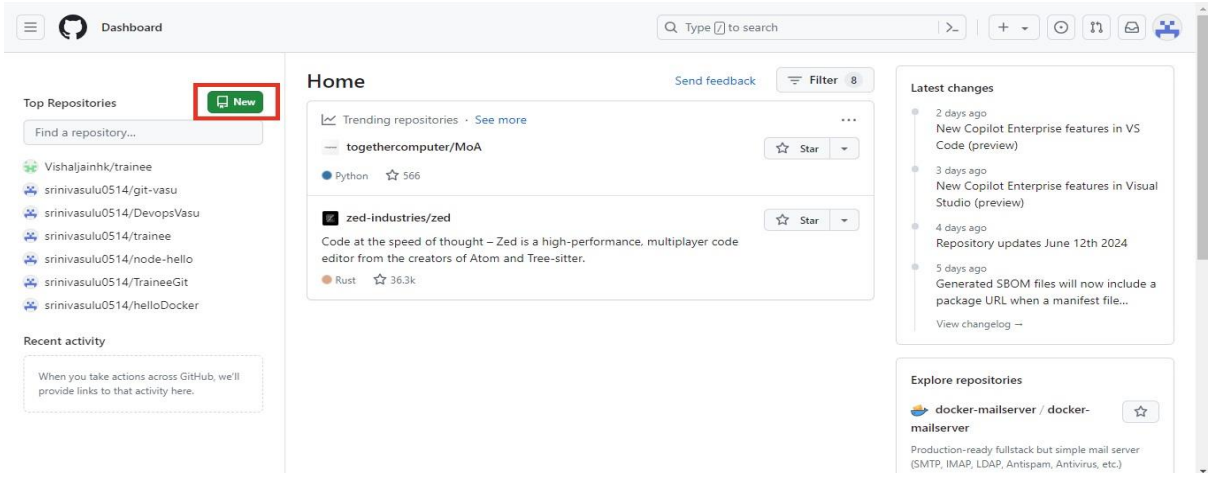
https://github.com/

Note: Create a GitHub account if you don't already have one.

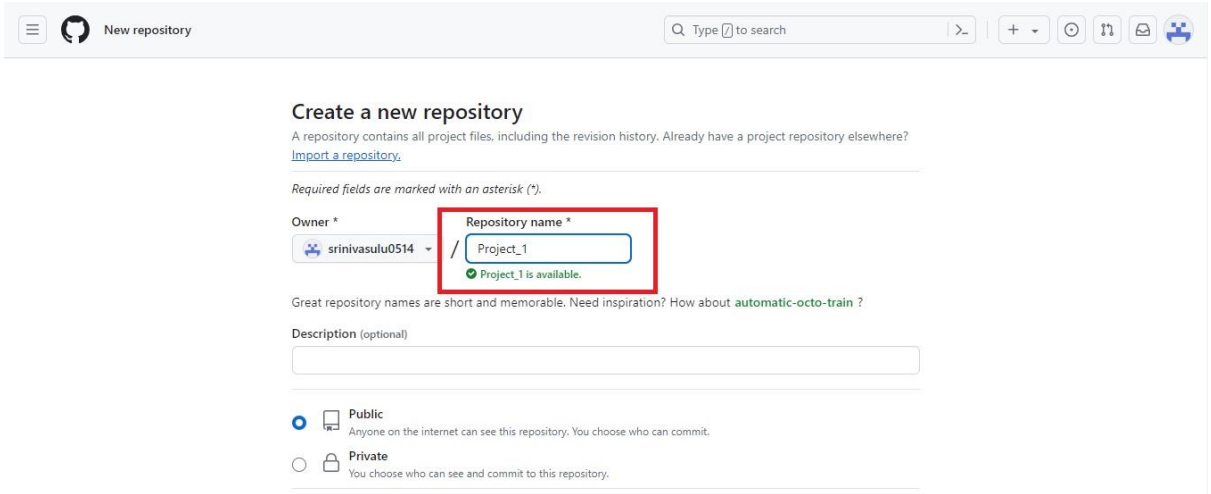


Sign in with our mail id and password.

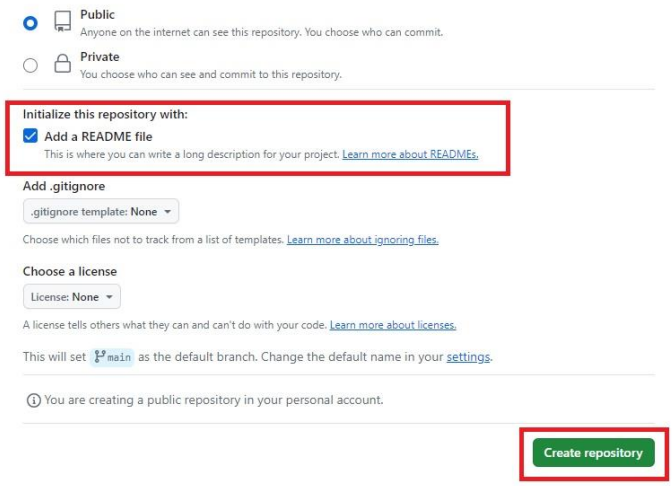
2. Click on "New Repository".



3. Enter a repository name and description.



4. Initialize the repository with a README (optional).



Cloning the Repository

Steps for Forking a Repository on GitHub:

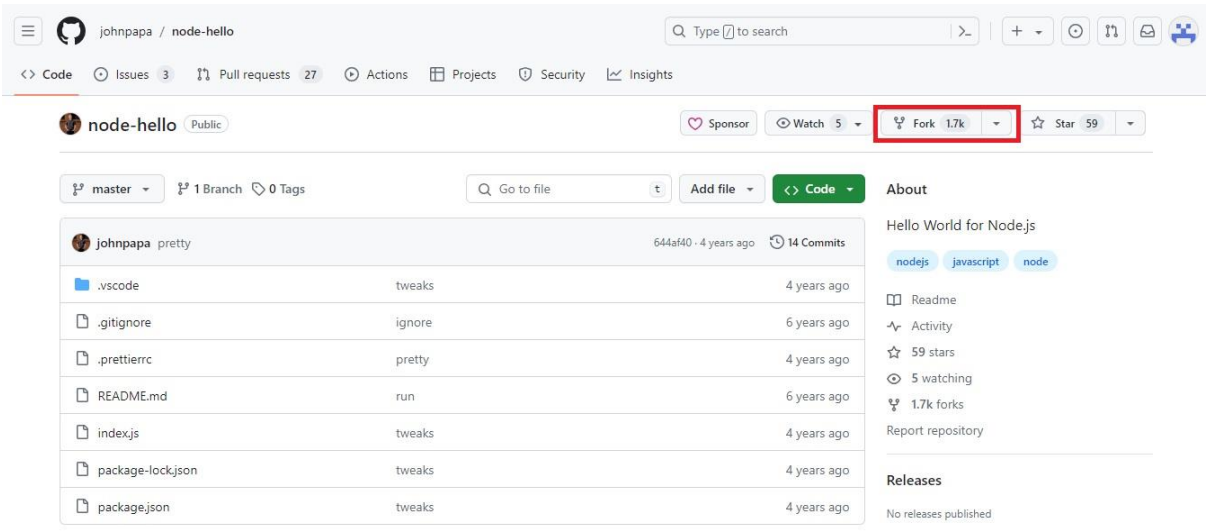
1. Navigate to the Repository:

Go to the repository that you want to fork. You can search for it or navigate directly using the URL (e.g., "https://github.com/username/repository").

URL: https://github.com/johnpapa/node-hello

2. Fork the Repository:

In the upper right corner of the repository page, click the "Fork" button.



GitHub will create a copy of the repository under your account. This may take a few moments.

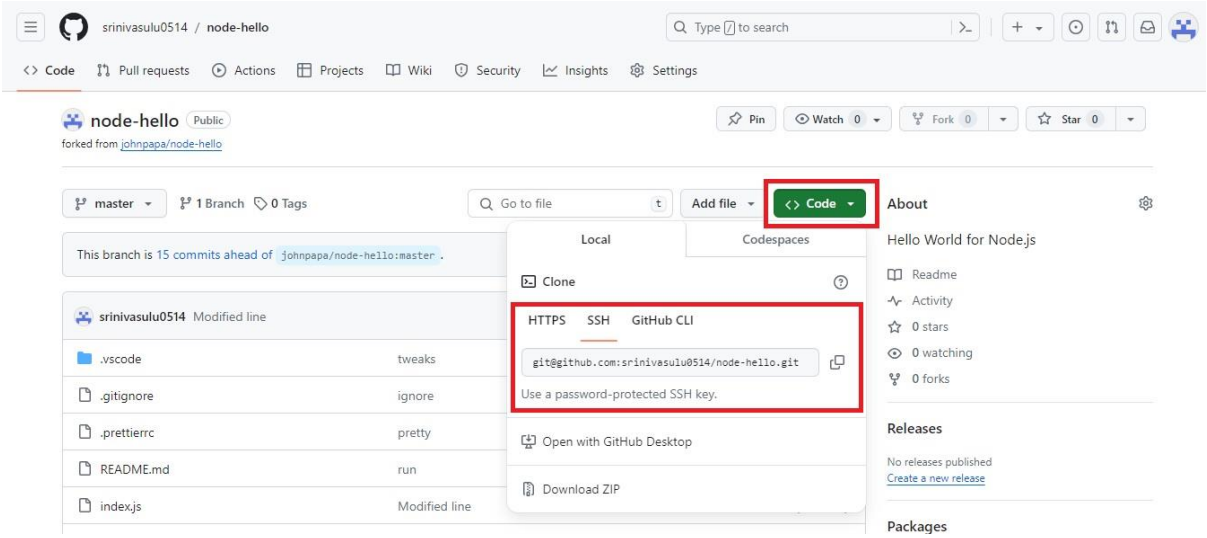
3. Clone Your Forked Repository:

- After forking, navigate to your GitHub profile, and go to the `Repositories` section.

- Click on the newly forked repository.

- Click the “Code” button (usually a green button) to reveal the repository URL.

- Copy the URL (choose either HTTPS or SSH, depending on your preference and setup).



- Open a terminal or command prompt on your local machine.

- Clone the forked repository to your local machine:

git clone https://github.com/your-username/forked-repository.git

(or)

git clone git@github.com:your-username/forked-repository.git

Replace “your-username” with your GitHub username and “forked-repository” with the name of the forked repository.

By following these steps, you will have successfully forked a repository, cloned it to your local machine.

Committing and Pushing Changes

1. Make changes to your files and save it.

2. Add changes to the staging area:

C:\Users\D.sai vamshi krishna\Downloads> git add .

3. Commit the changes:

C:\Users\D.sai vamshi krishna\Downloads> git commit -m "Your commit message"

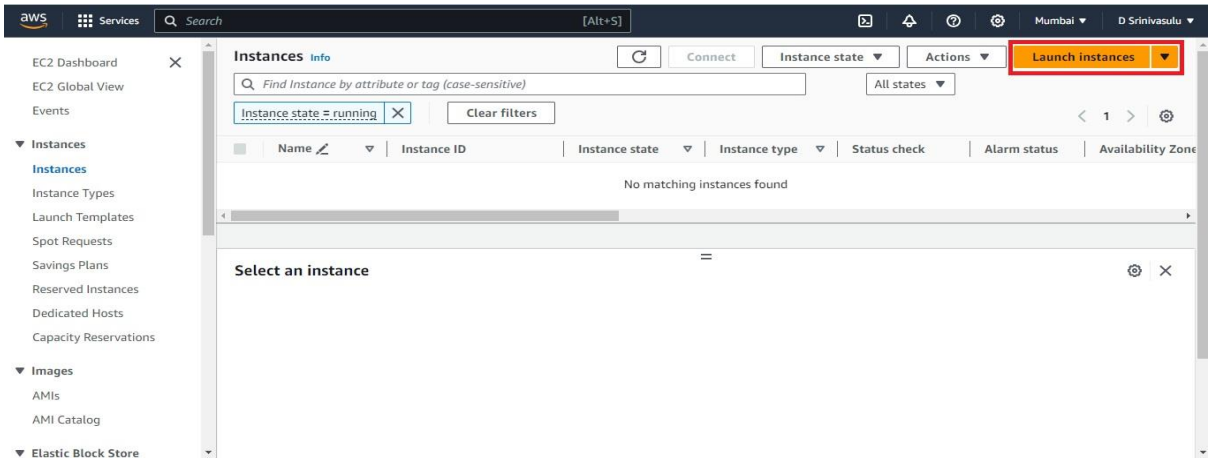
4. Push the changes to GitHub:

C:\Users\D.sai vamshi krishna\Downloads>git push origin main

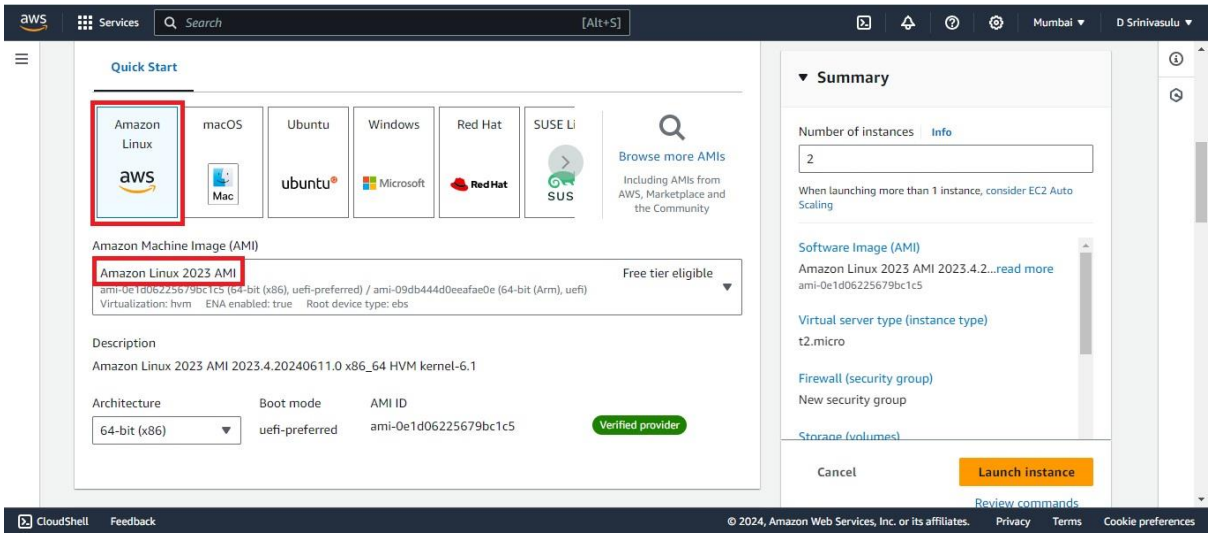
3. Creating an EC2 Instance

1. Log in to AWS Management Console.

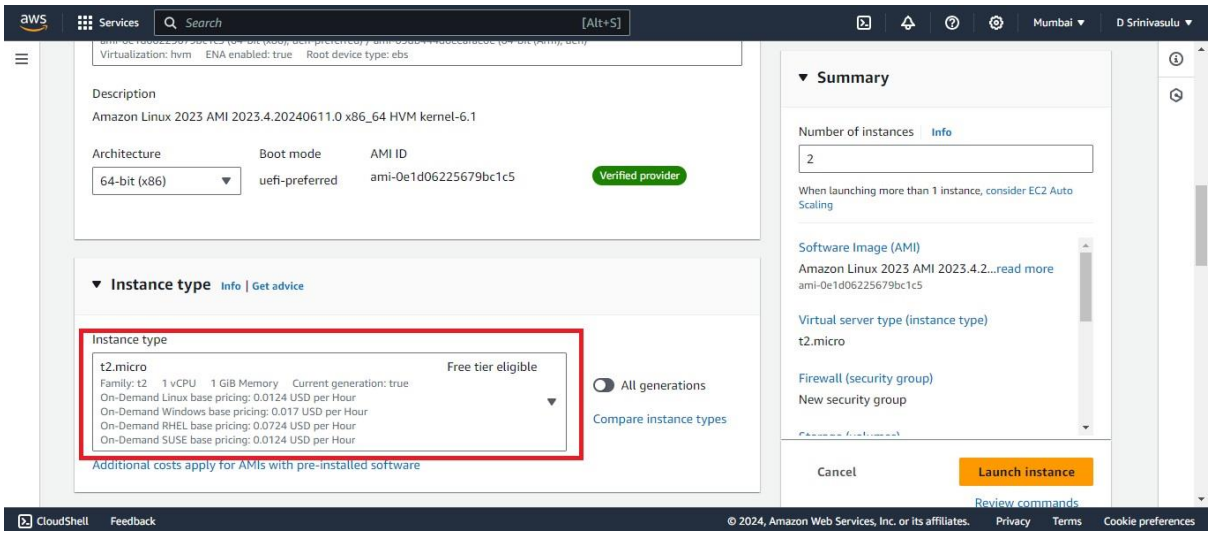
2. Navigate to EC2 Dashboard and click "Launch Instance".



3. Choose an Amazon Machine Image (AMI) (e.g., Amazon Linux 2023 AMI).

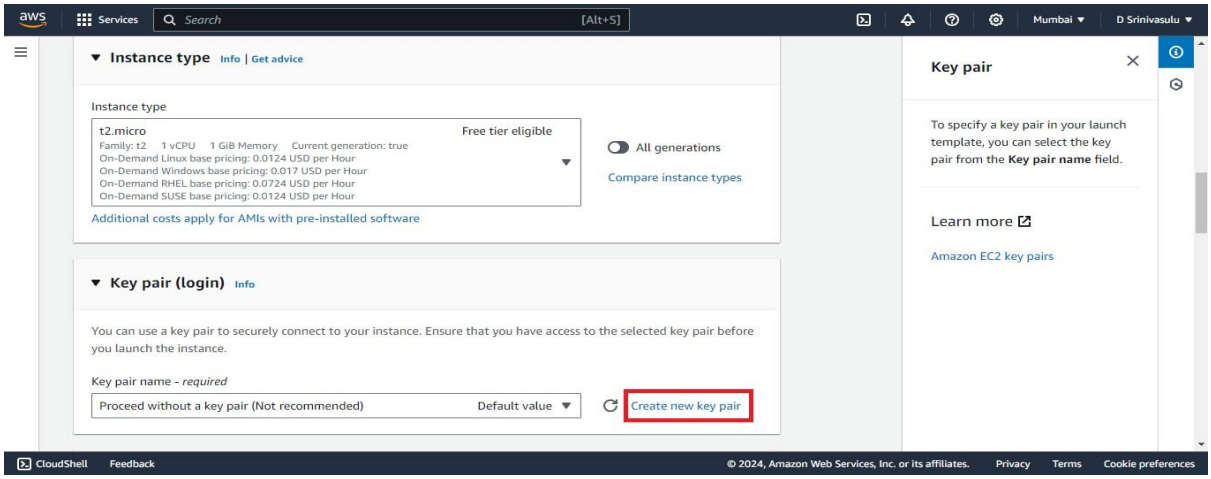


4. Select an instance type (e.g., t2.micro).

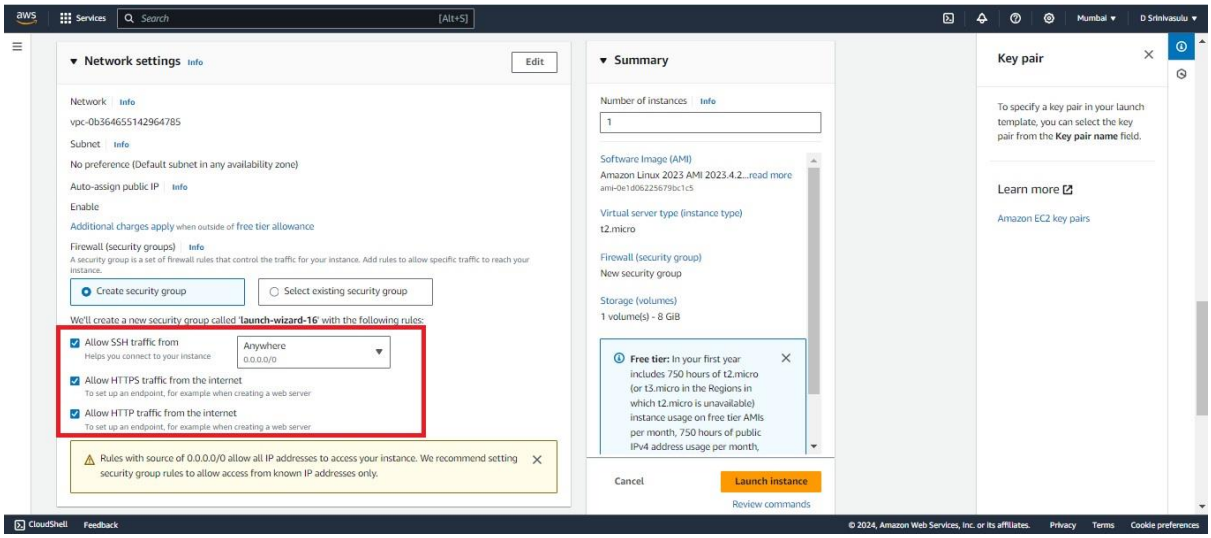


5. Add a key pair.

We need a key pair in Amazon EC2 to securely connect to our instances via SSH. The key pair ensures that only authorized users can access the instance.

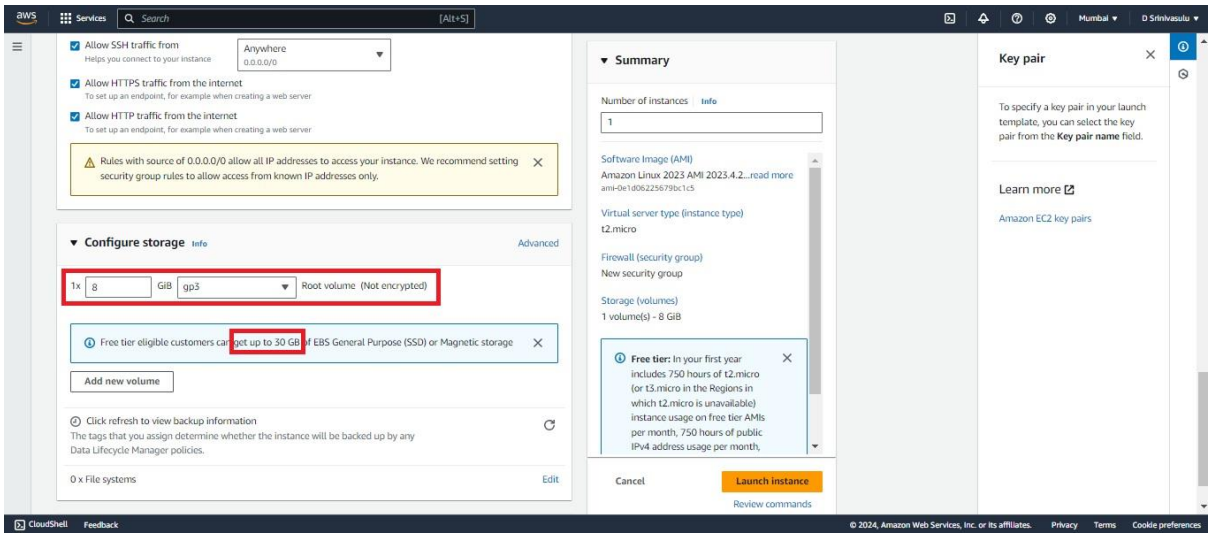


6. Configure security groups (allow SSH, HTTP/HTTPS).



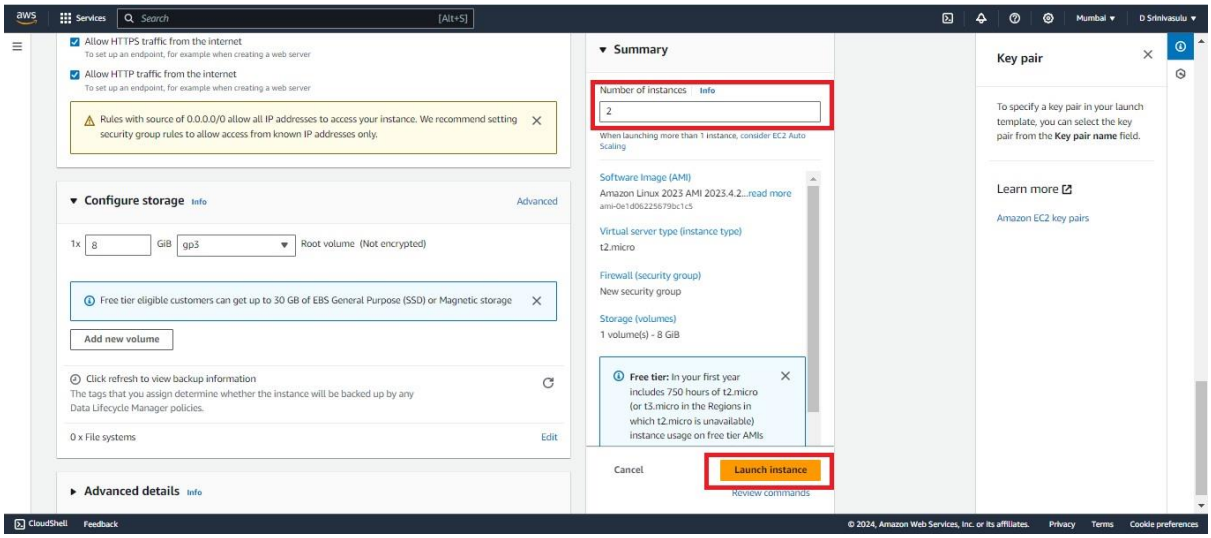
If we already created a security groups, choose one of the groups accordingly.

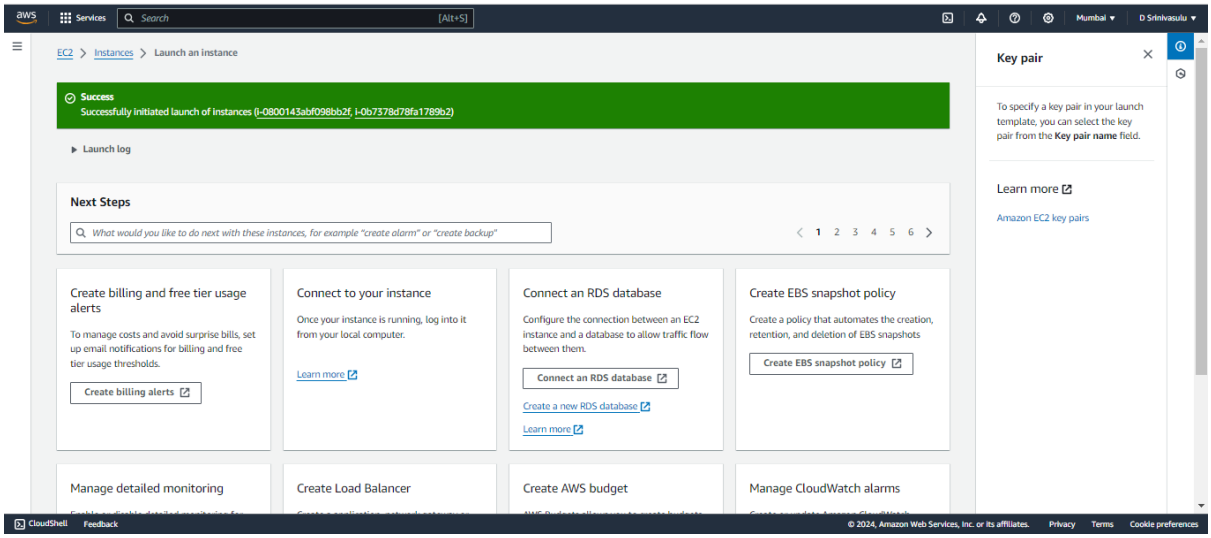
7. Configure instance details and add storage.



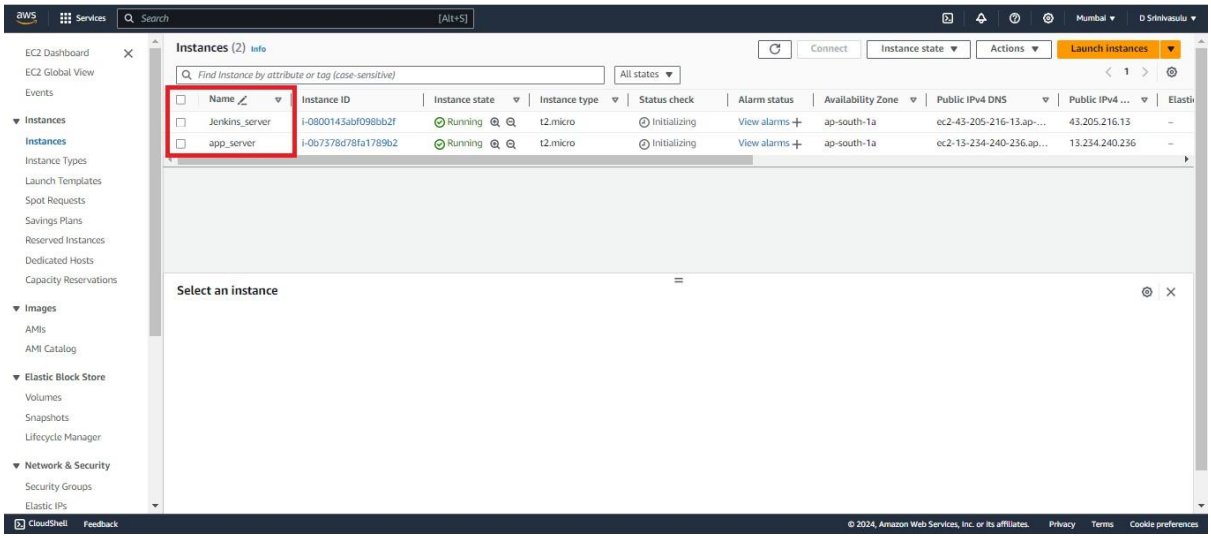
8. Review and launch the instance, after, download the SSH key pair.

Here, we are creating two instances “Jenkins server (Master)” and “application server”.





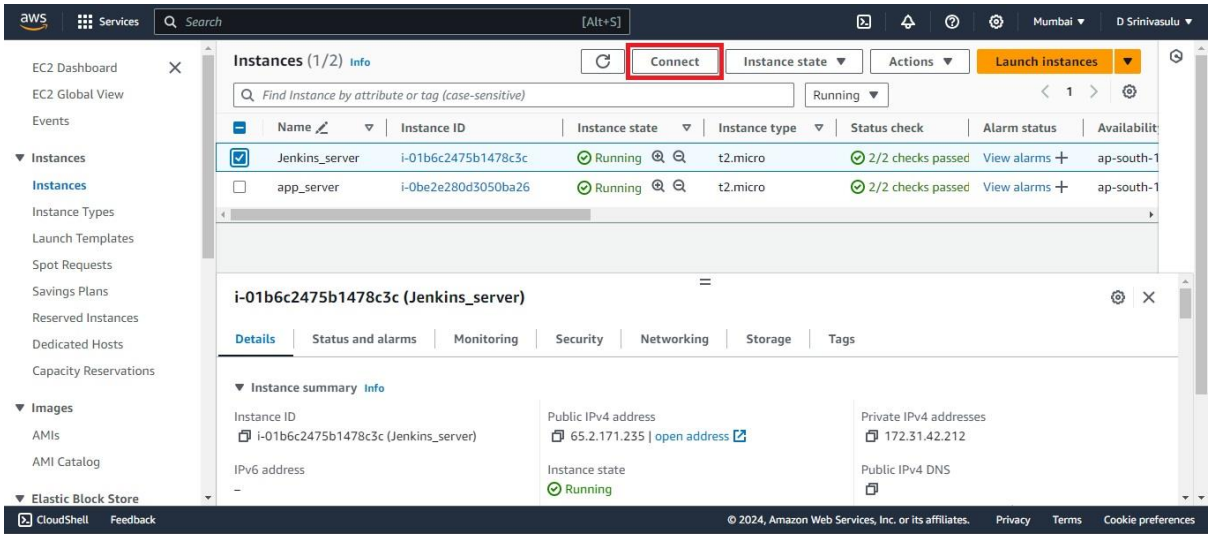
Open the instances and name the one of the instances as “Jenkins_server” and another instance as “app_server”.



Connecting to the EC2 Instance:

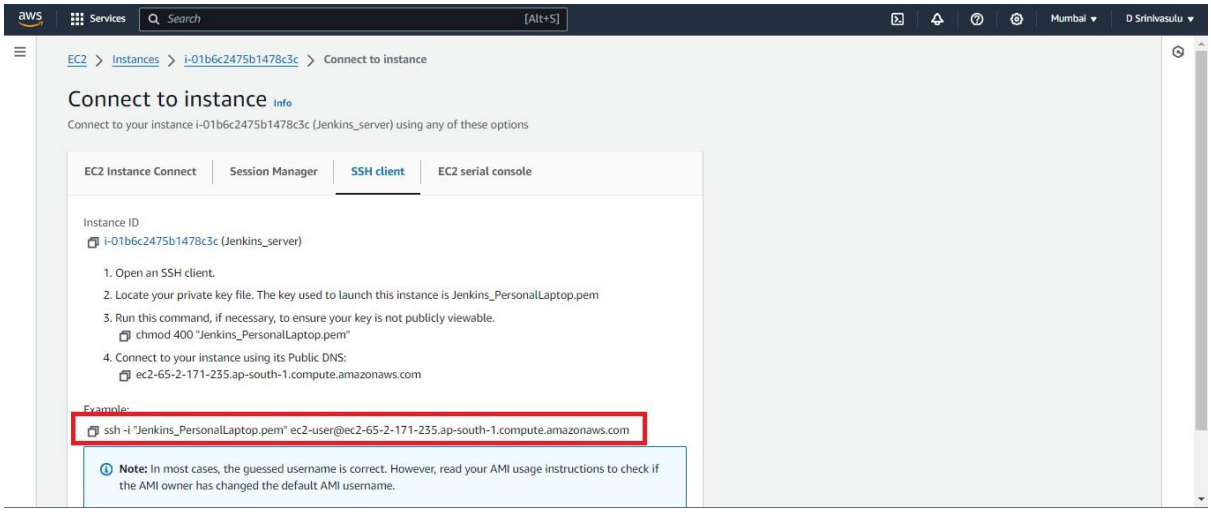
1. Open a terminal.

2. Go to the directory, where the private key pair file is stored and connect the Jenkins_server instance.



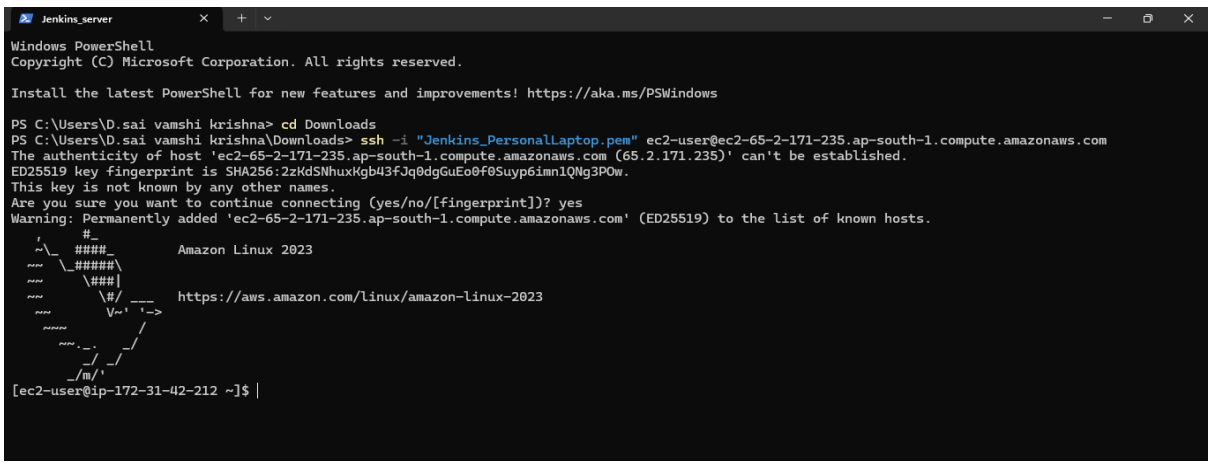
3. Connect to the instance:

Connect the instance in the power shell/command prompt with SSH client, follow the instructions in the picture below.

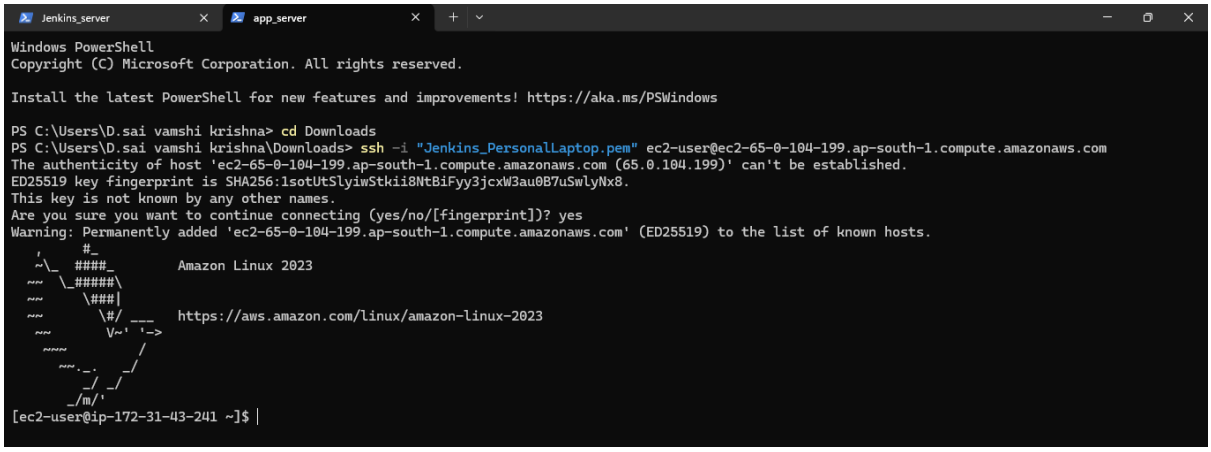


Follow these same steps in power shell for the “app server” too

Jenkins server:



App server:



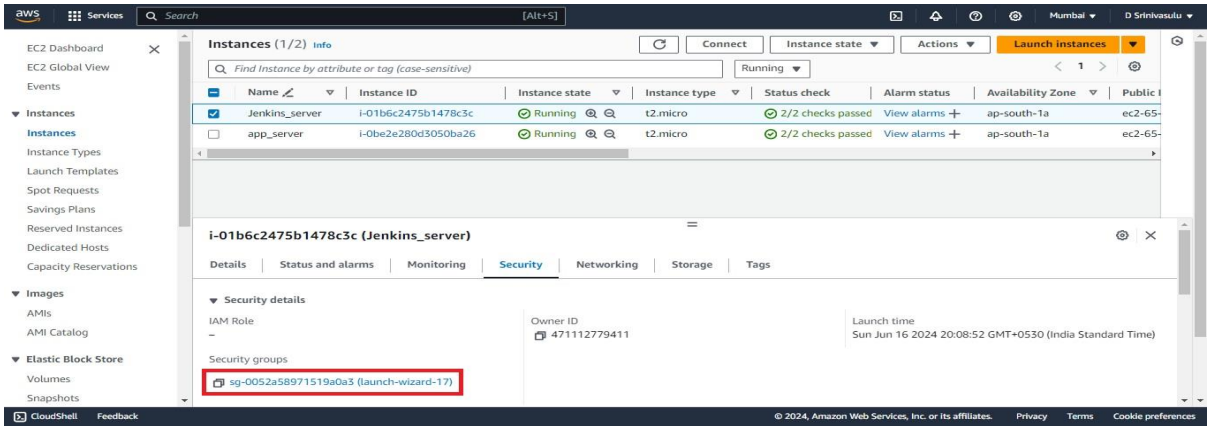
Setting Up Security Groups

1. In the EC2 dashboard, select "Security Groups".

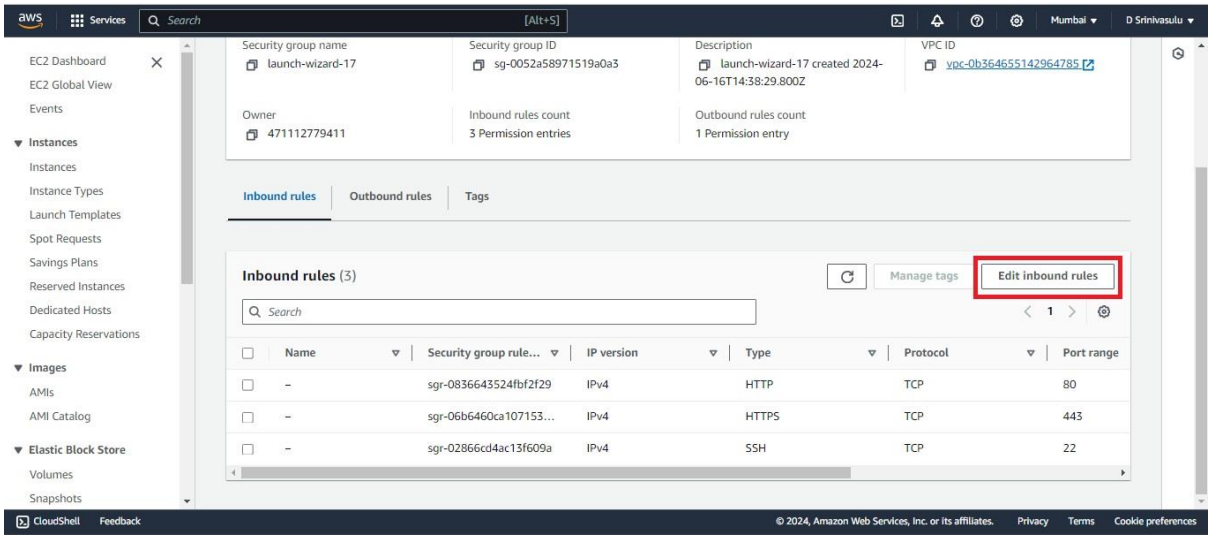
2. Edit the inbound rules to allow necessary ports (e.g., 8080 for Jenkins, 3000 for Node.js).

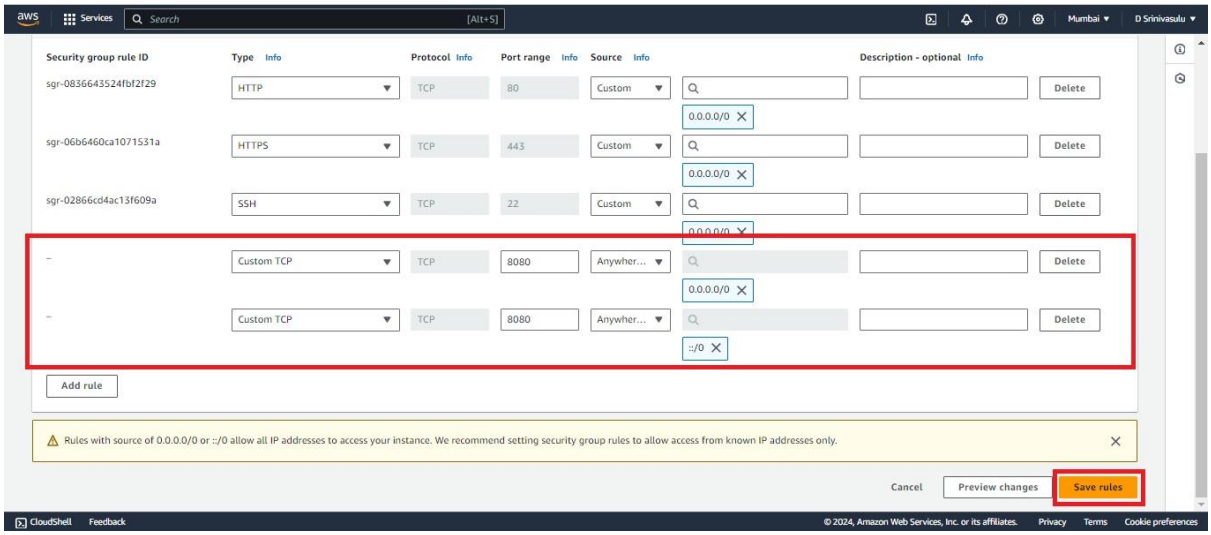
Navigate to the security group section within the Jenkins_server Instance and proceed to select the security group.

For Jenkins server:

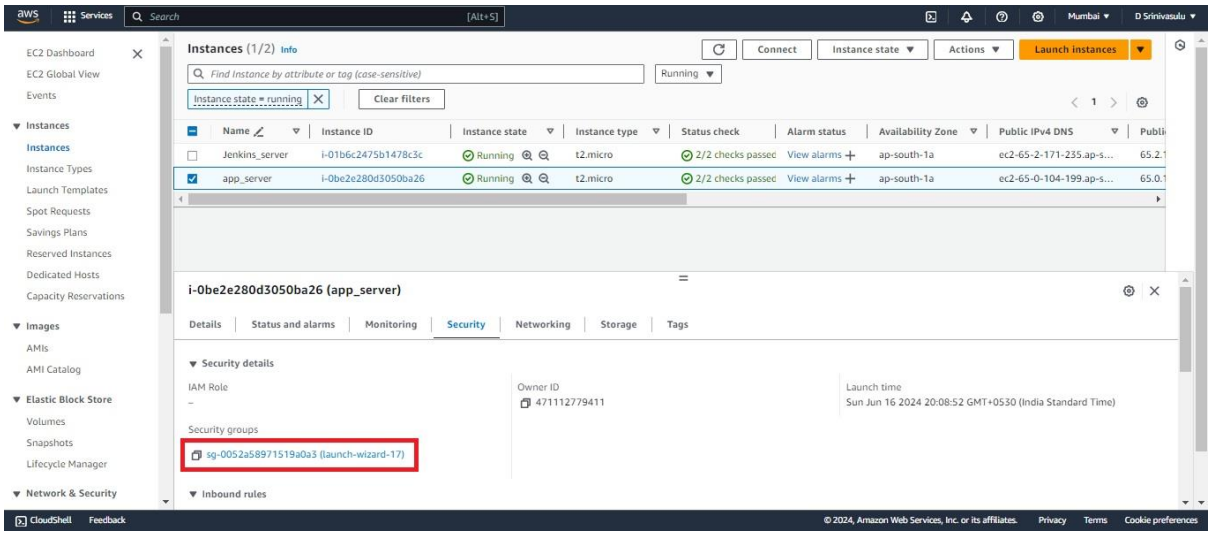


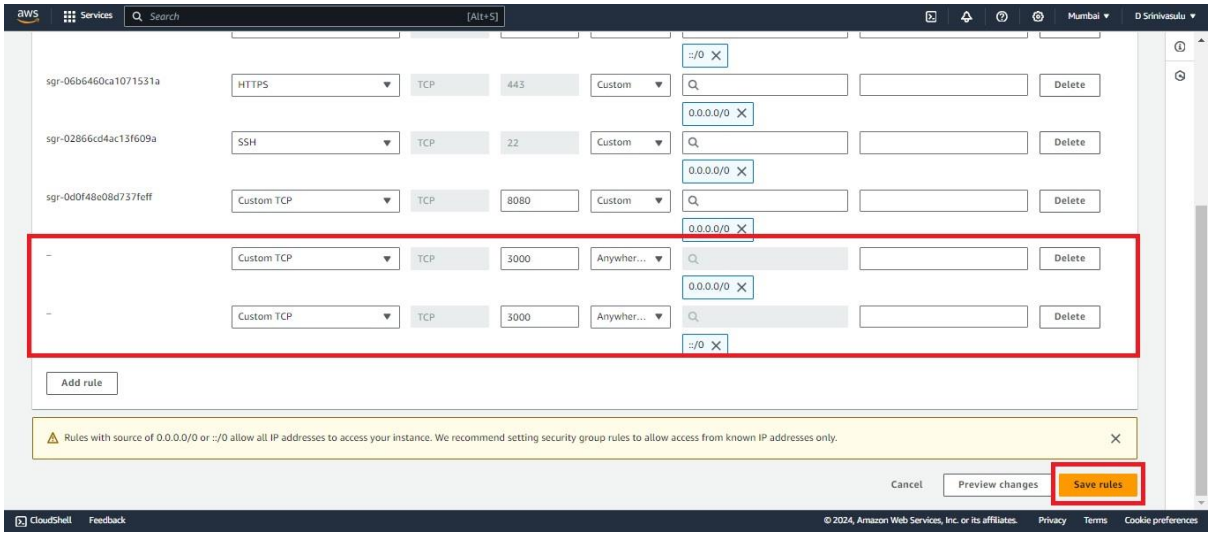
Click on the “Edit Inbound rules” and Add 8080 ports to Anywhere-IP4 and IP6.





For App server:





4. Installing Jenkins in the Jenkins server:

Installing Java

1. Update the package index:

[ec2-user ~]$ sudo yum update –y

2. Add the Jenkins repository and import the key:

[ec2-user ~]$ sudo wget -O /etc/yum.repos.d/jenkins.repo \

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

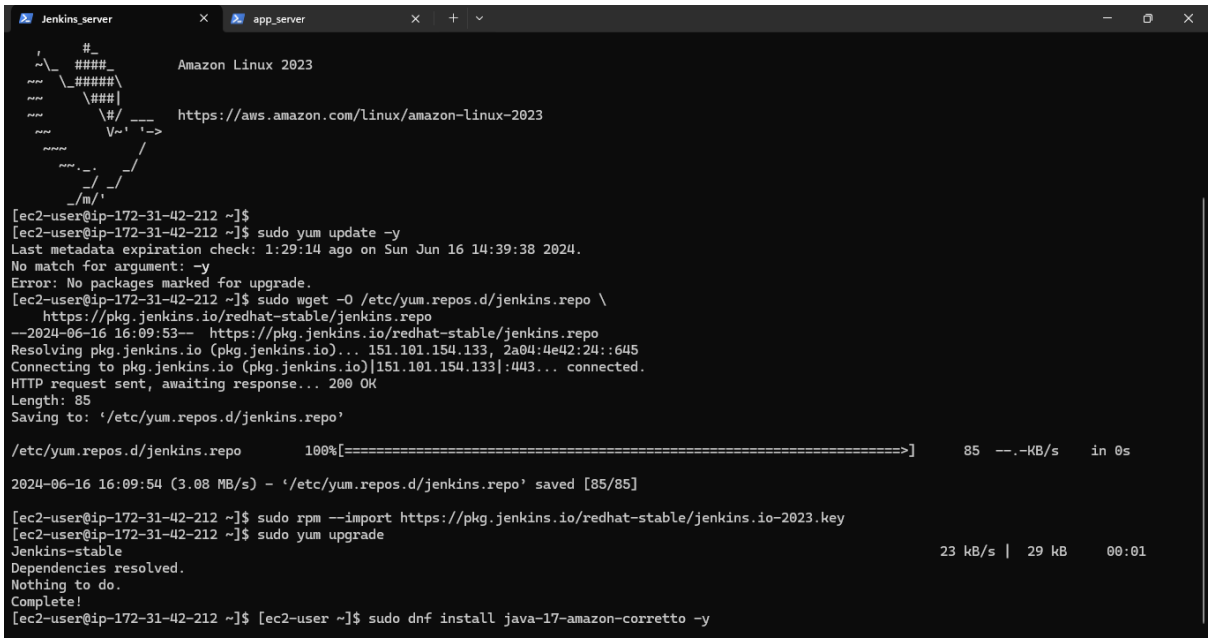
3. Import a key file from Jenkins-CI to enable installation from the package:

[ec2-user ~]$ sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io- 2023.key

[ec2-user ~]$ sudo yum upgrade

4. Install Java (Amazon Linux 2023):

[ec2-user ~]$ sudo dnf install java-17-amazon-corretto -y



Installing Jenkins

1. Install Jenkins:

[ec2-user ~]$ sudo yum install jenkins -y

Configuring Jenkins

1. Start Jenkins:

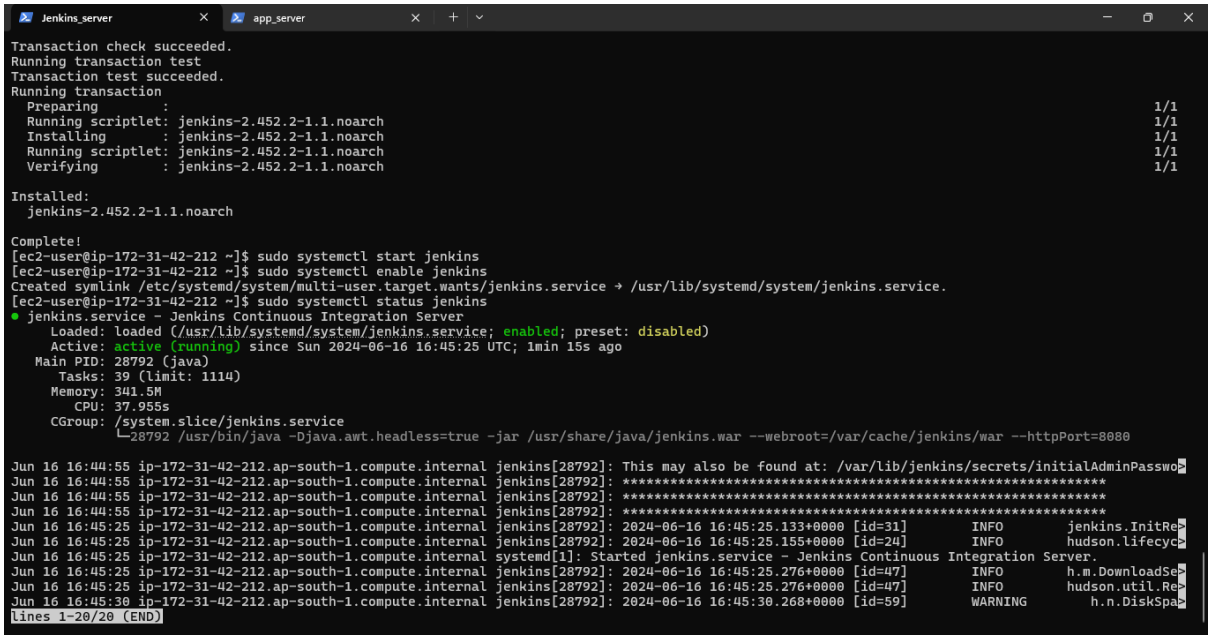
[ec2-user ~]$ sudo systemctl start jenkins

2. Enable Jenkins to start on boot:

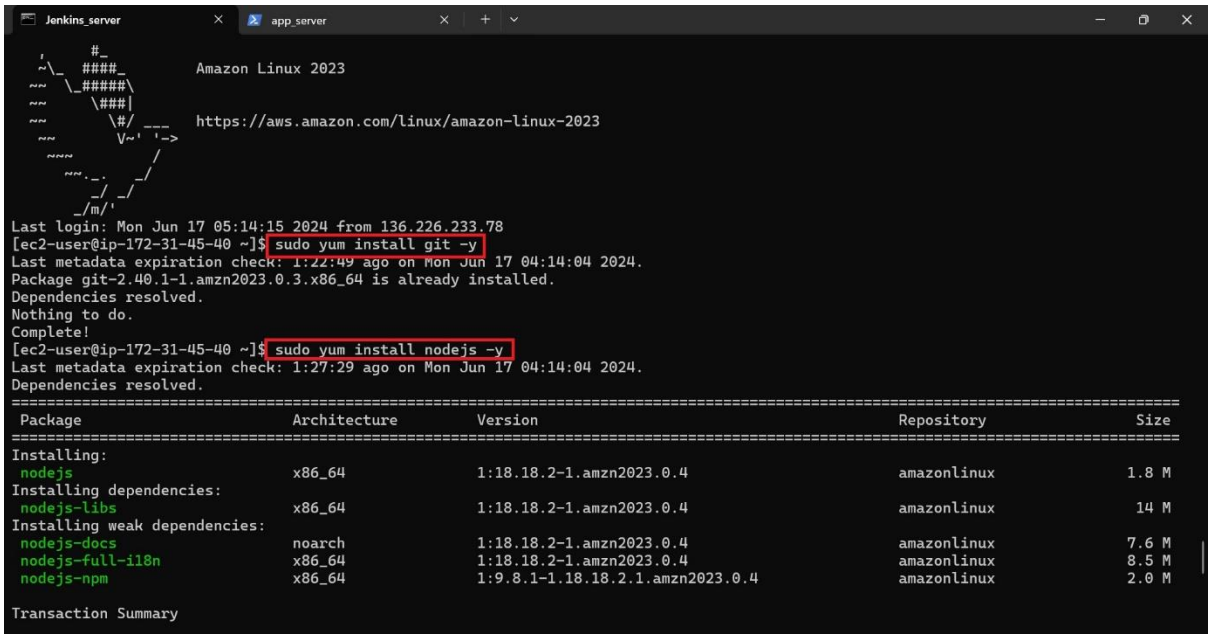
[ec2-user ~]$ sudo systemctl enable Jenkins

3. Check the status of the Jenkins service

[ec2-user ~]$ sudo systemctl status Jenkins



Install git and NodeJS in Jenkins server to build the application:



3. Open Jenkins in your browser:

http://your-ec2-ip:8080

4. Follow the setup wizard instructions.

Unlock the Jenkins (sudo cat /var/lib/jenkins/secrets/initialAdminPassword) → Create account → Use the Jenkins

5. Install git in Jenkins server (sudo yum install git -y) → to check the git version in Jenkins_server (git -v (or) git -version)

5. Installing Node.js on app server EC2:

1. Open the app server which is already created in the above steps.

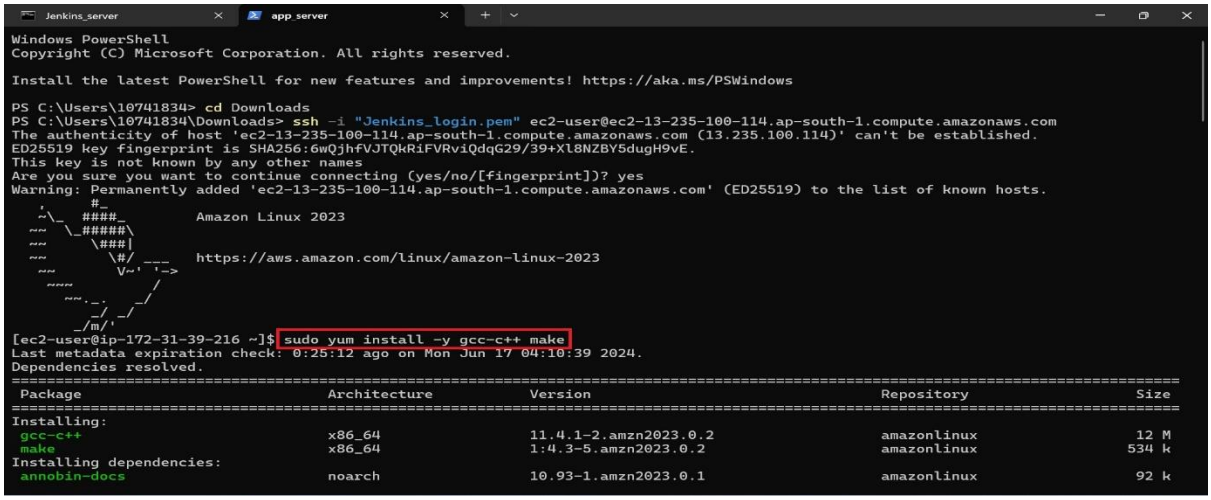
2. Install Node.js:

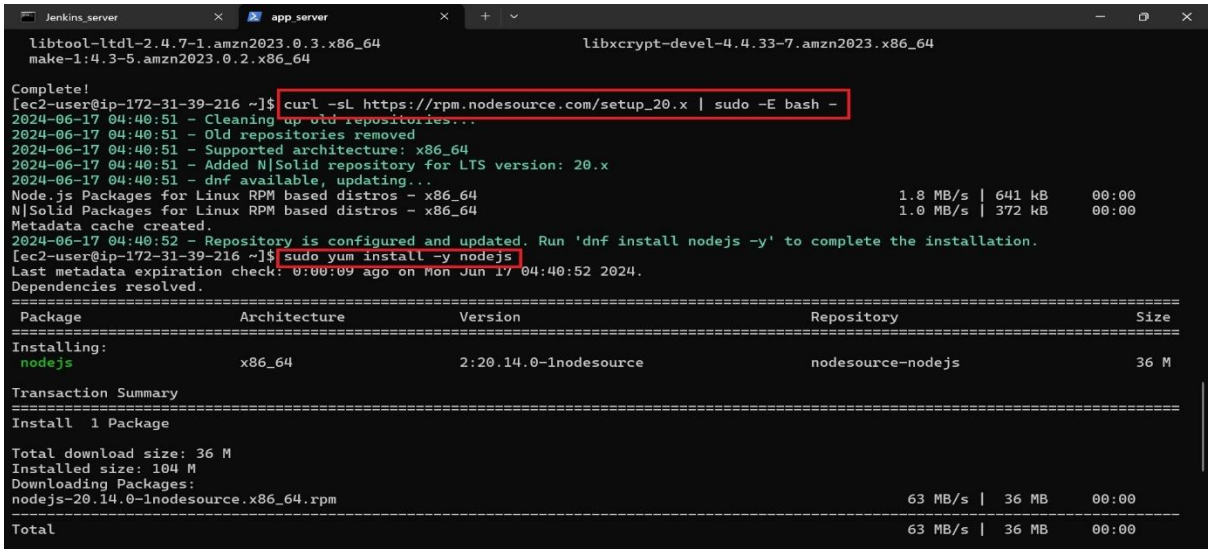
Configure yum repository:

sudo yum install -y gcc-c++ make → curl -sL https://rpm.nodesource.com/setup_20.x | sudo -E bash -

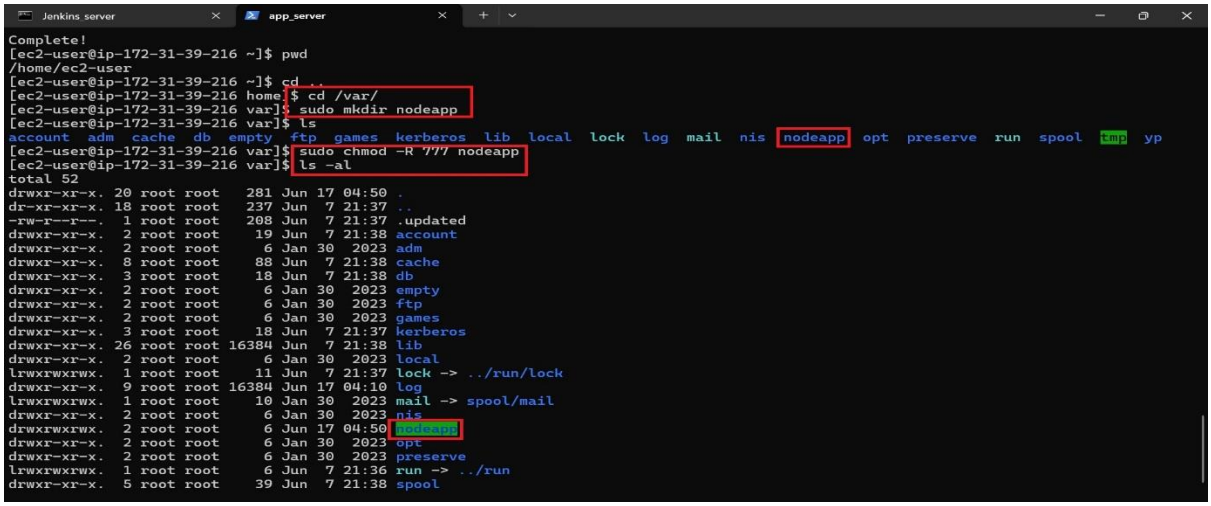
Installing Node.js on app server

sudo yum install -y nodejs





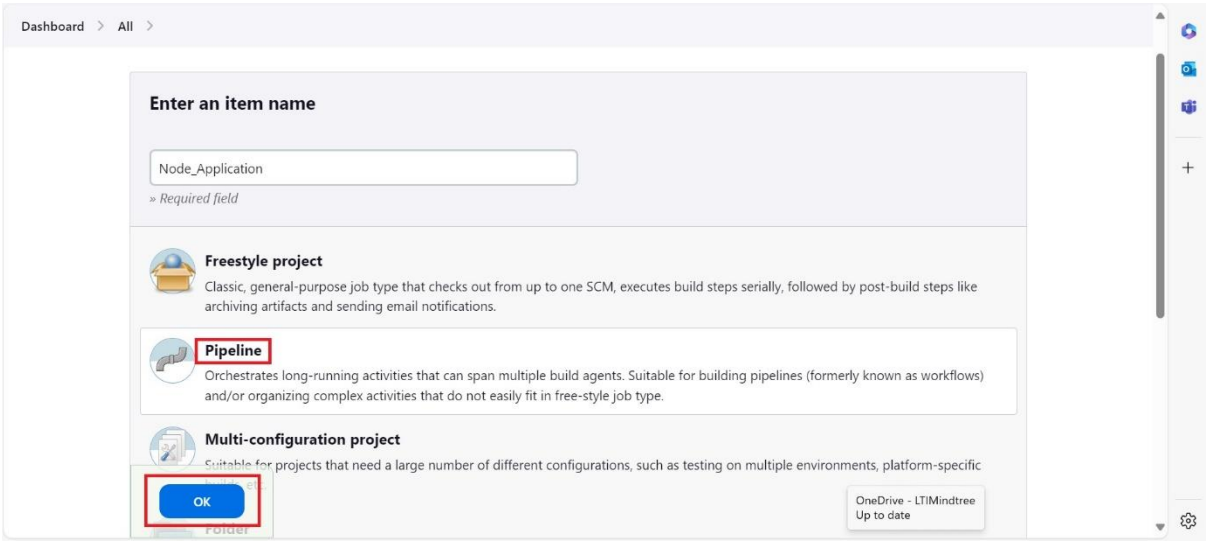
Create a directory “nodeapp” in the “var” directory, then give permissions to the “nodeapp” which we are created.



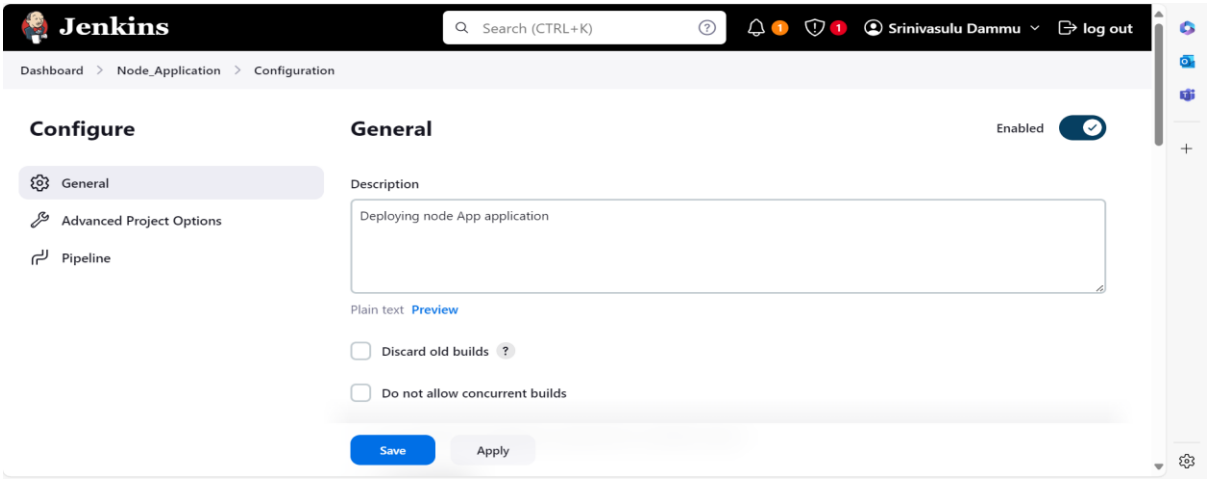
Setting Up Jenkins Job for Node.js Application:

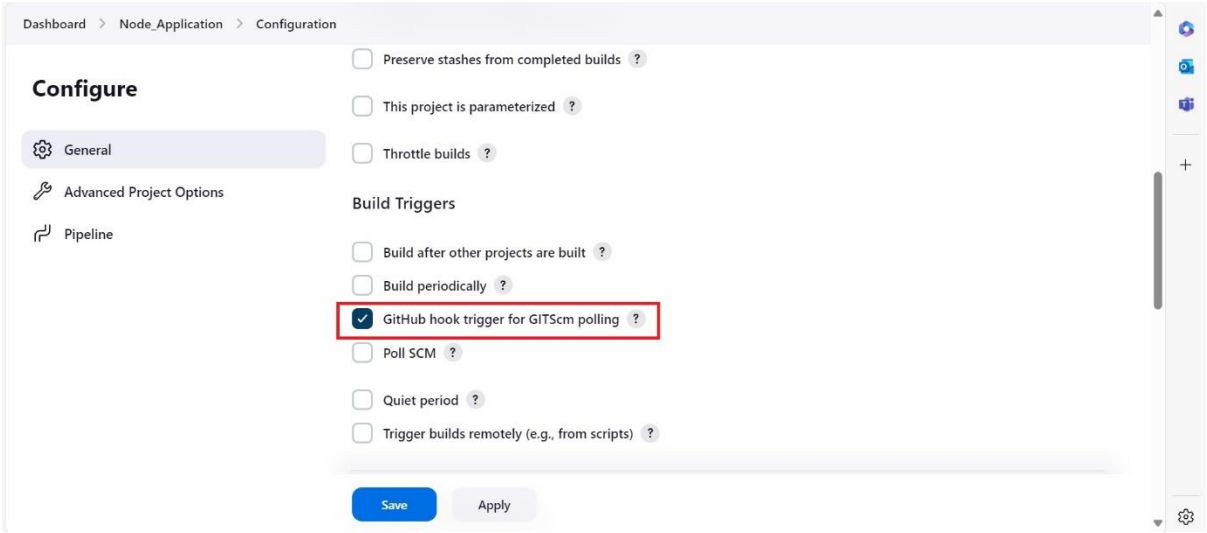
1. In Jenkins, create a new job (Pipeline project).

Click on “New item” → choose pipeline project

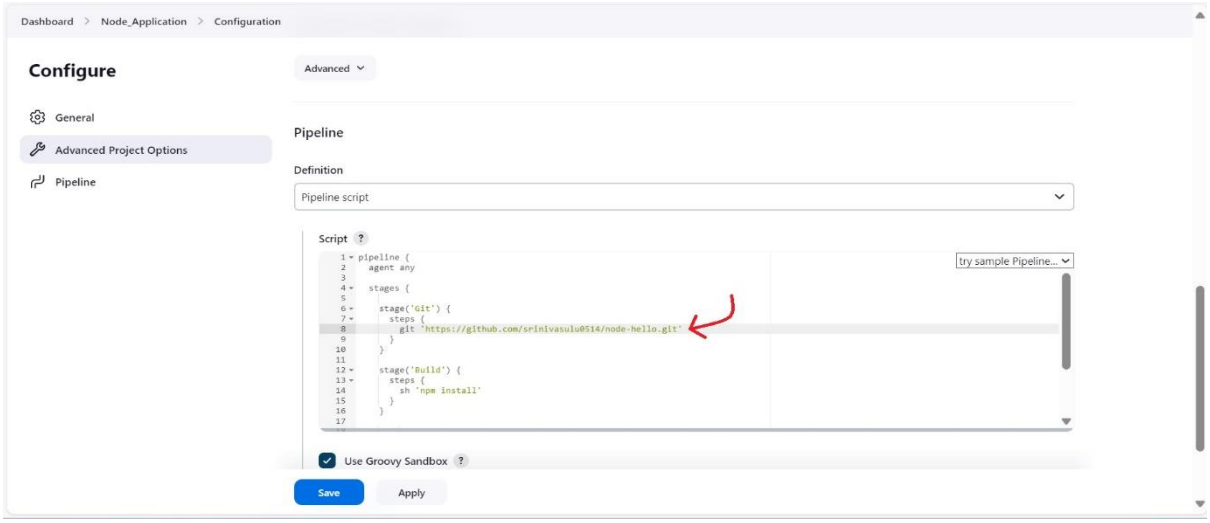


After creating pipeline → Configure the required steps for the pipeline → Choose GitHub hook trigger under “Build Triggers”

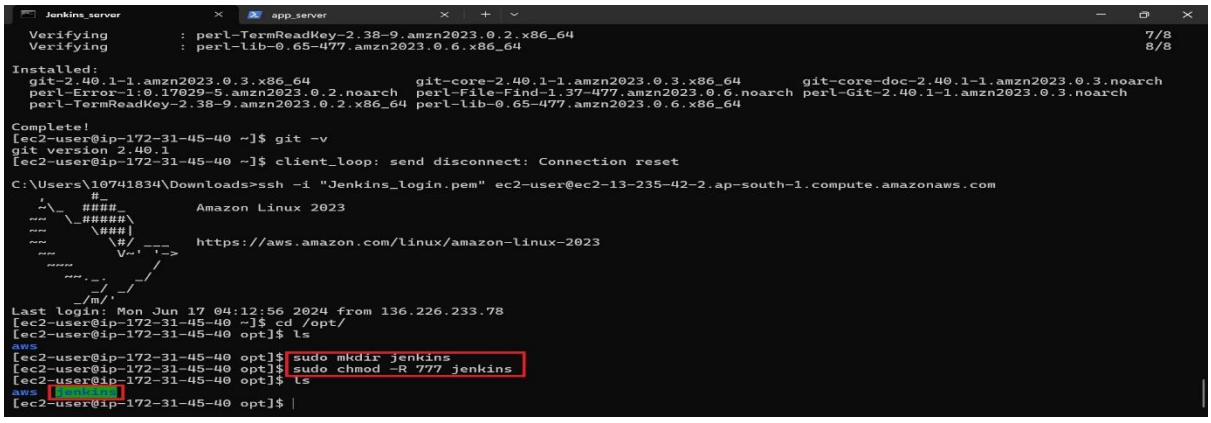




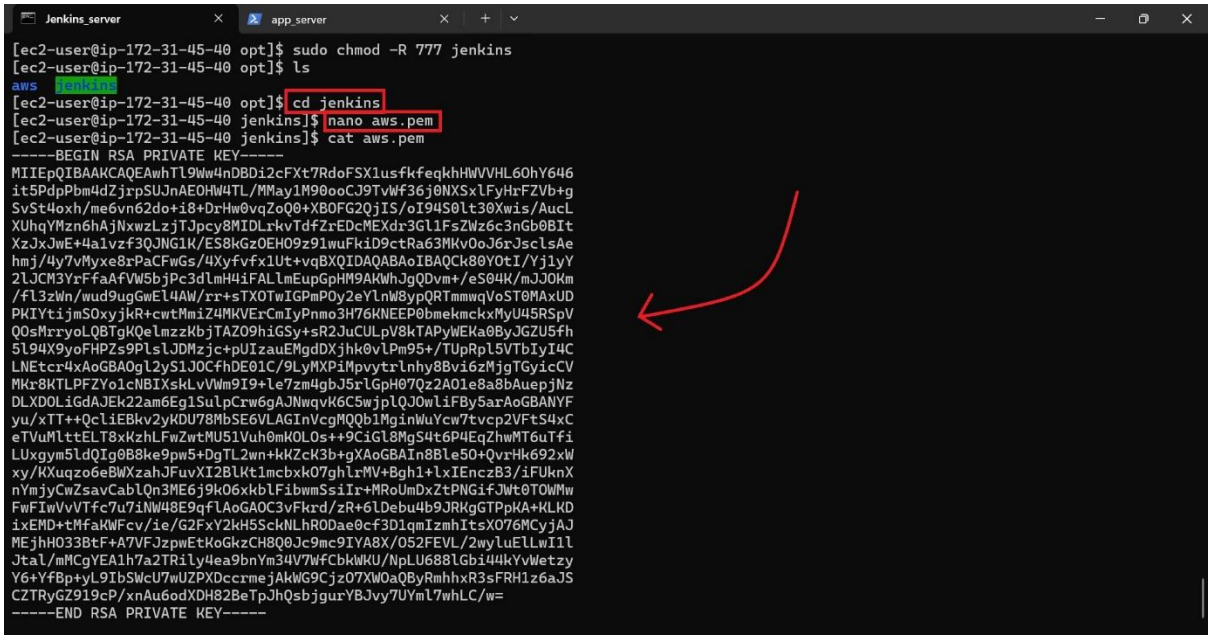
Make sure that paste the Git repository URL on which we are performing.



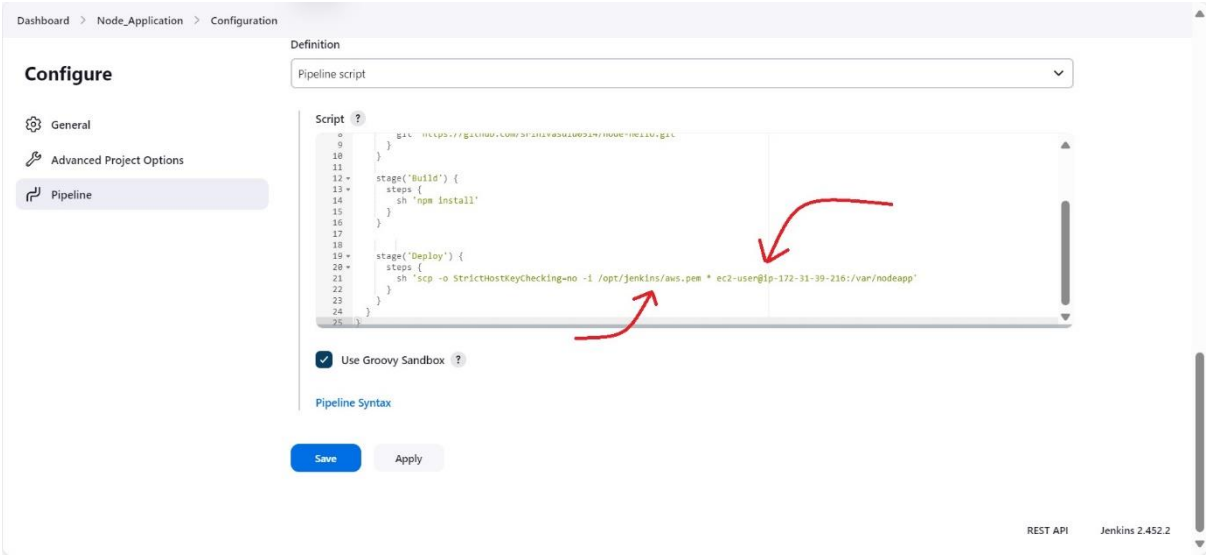
Before going to Deploy stage, we need to create the “jenkins” Directory (Directory name can be written as per user wish) in “opt” directory, after creating “jenkins” directory give permission to it. (Jenkins server)



Now copy the key pair data (as we created Jenkins_login.pem/Jenkins_PersonalLaptop.pem) , and now move to the “jenkins” directory with “cd jenkins”, create a file aws.pem with help of “nano aws.pem” and save it.

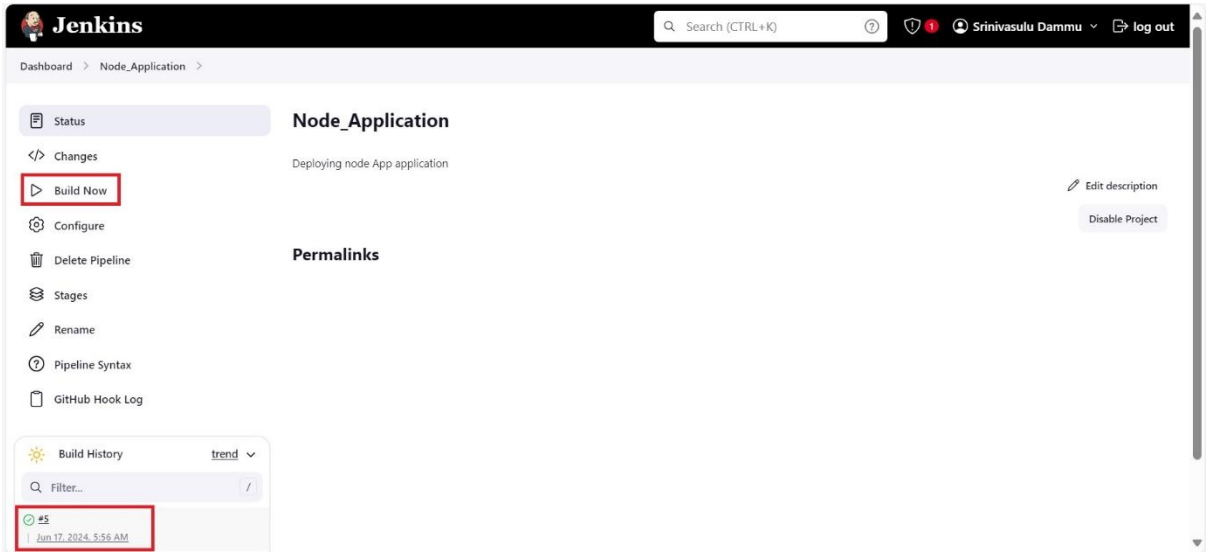


Now, write the final step (Deploy stage) of the groovy script:



i) First arrow indicates the path in Jenkins server. ii) Second arrow indicates the ec2 server and path in app server

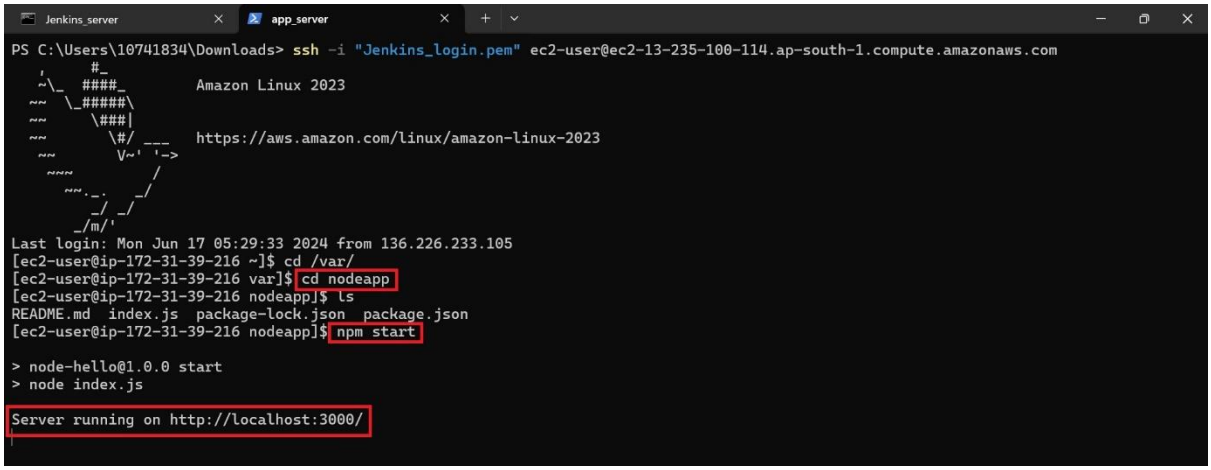
Now, save the configuration and build the app.

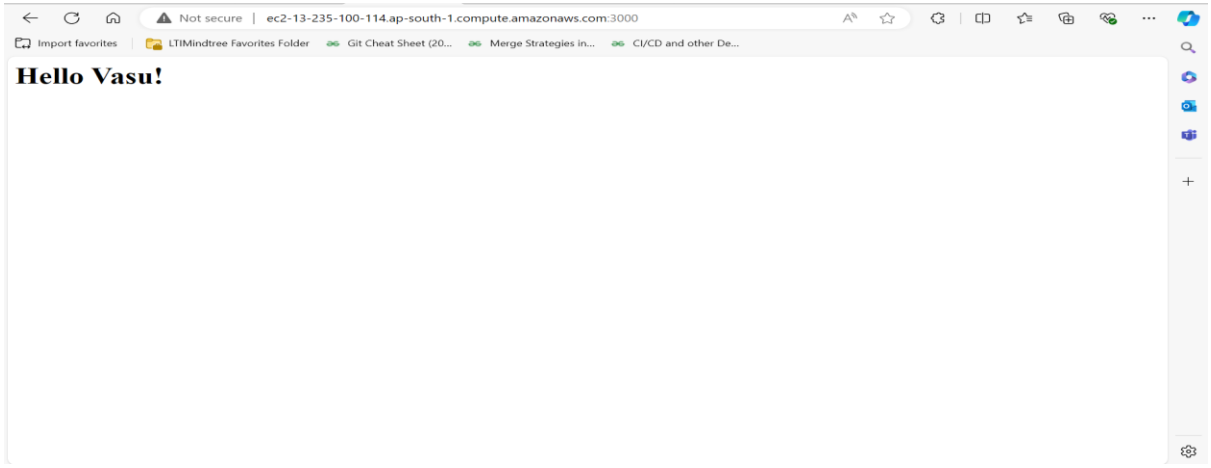


To check our application, go to app server and make sure that be in “nodeapp” directory, use command “npm start”.

After open the Public IP4 address/ Public IP4 DNS → make https to http and add ‘:3000’ at the end of the URL.

Ex: http://ec2-13-235-100-114.ap-south-1.compute.amazonaws.com:3000/

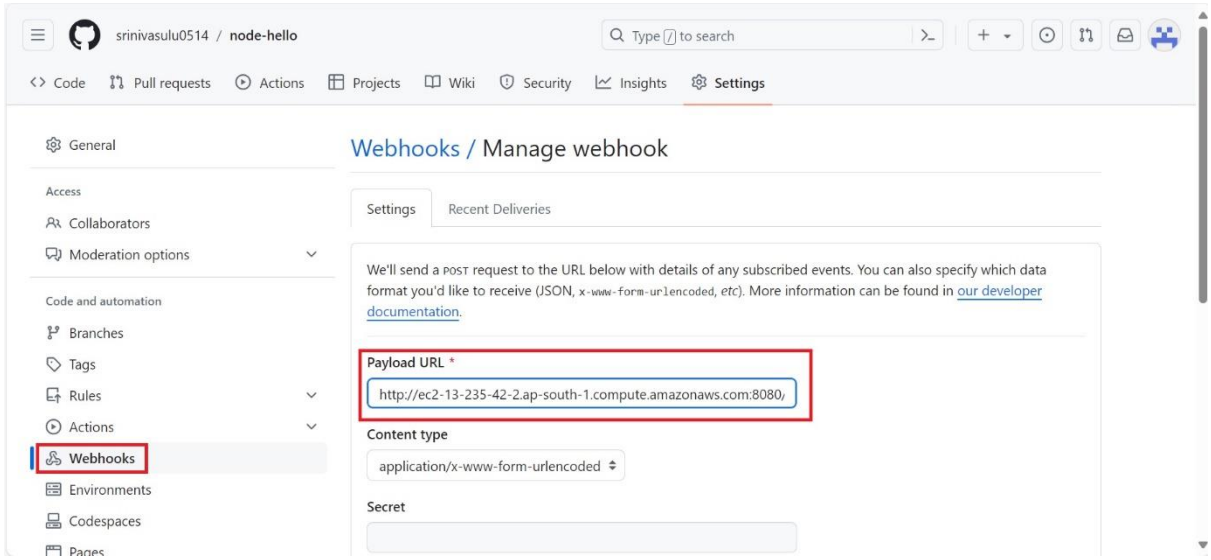




To Automate the **Build** process, we need to step the GitHub webhooks in GitHub and for **refreshing** the website install pm2 library in app server (add watcher to the pm2).

Steps:

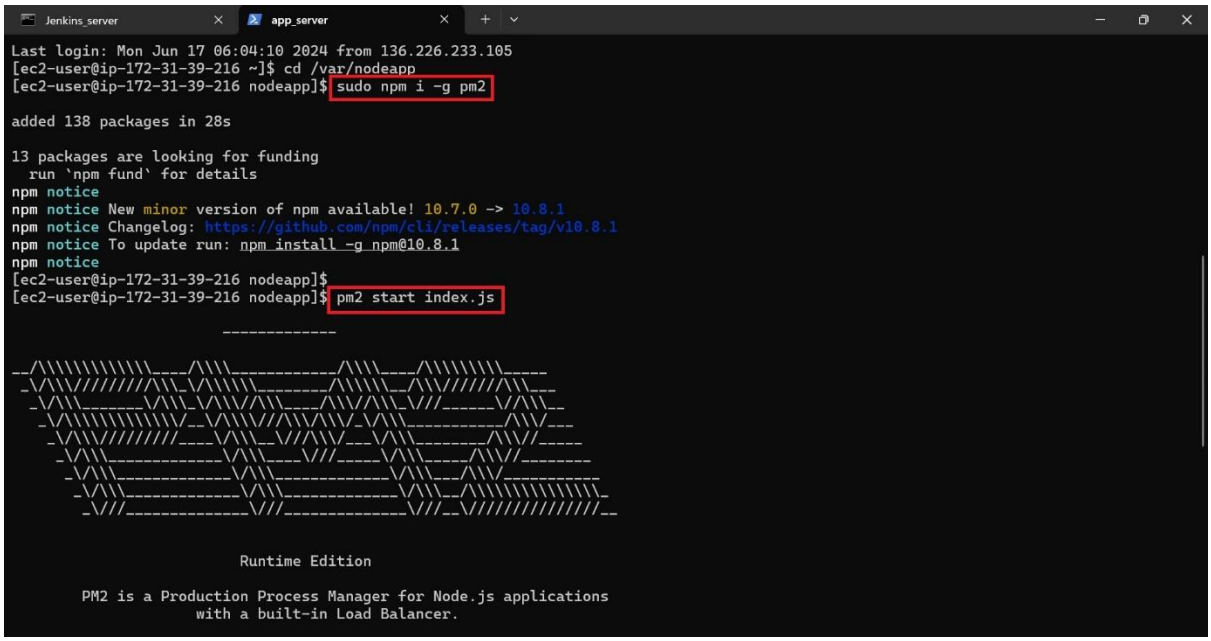
Go to the node-hello repository in GitHub → Click on settings which is beside of insights → choose Webhooks → add the Jenkins URL under Payload URL → save.

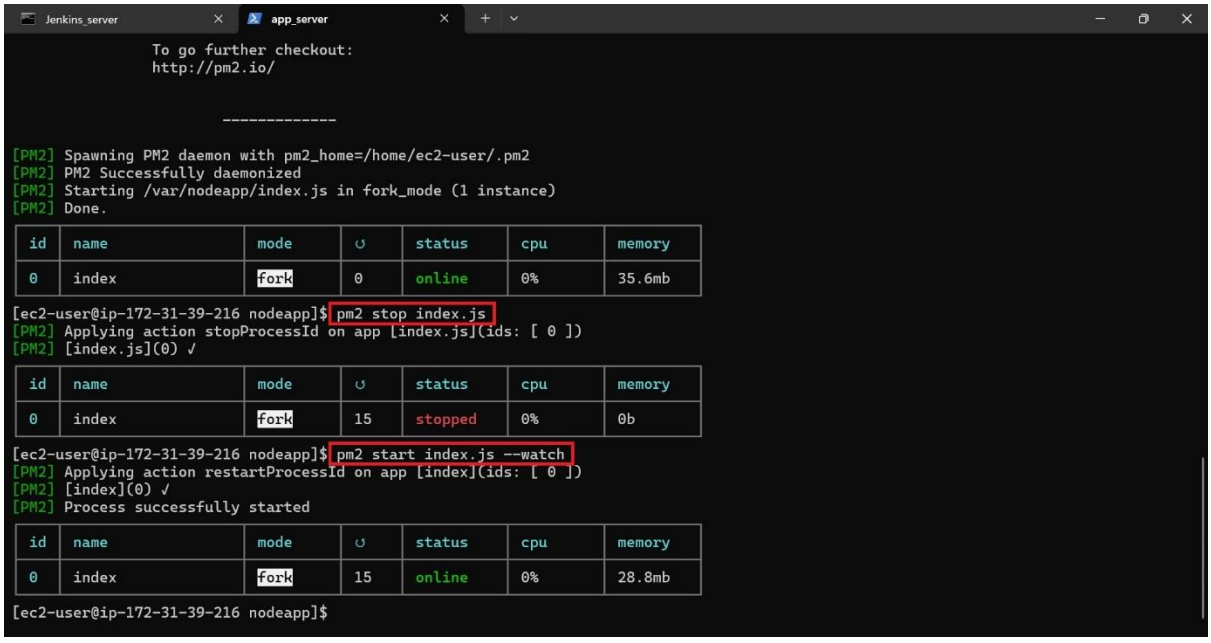


URL should be: Jenkins server//github-webhook/

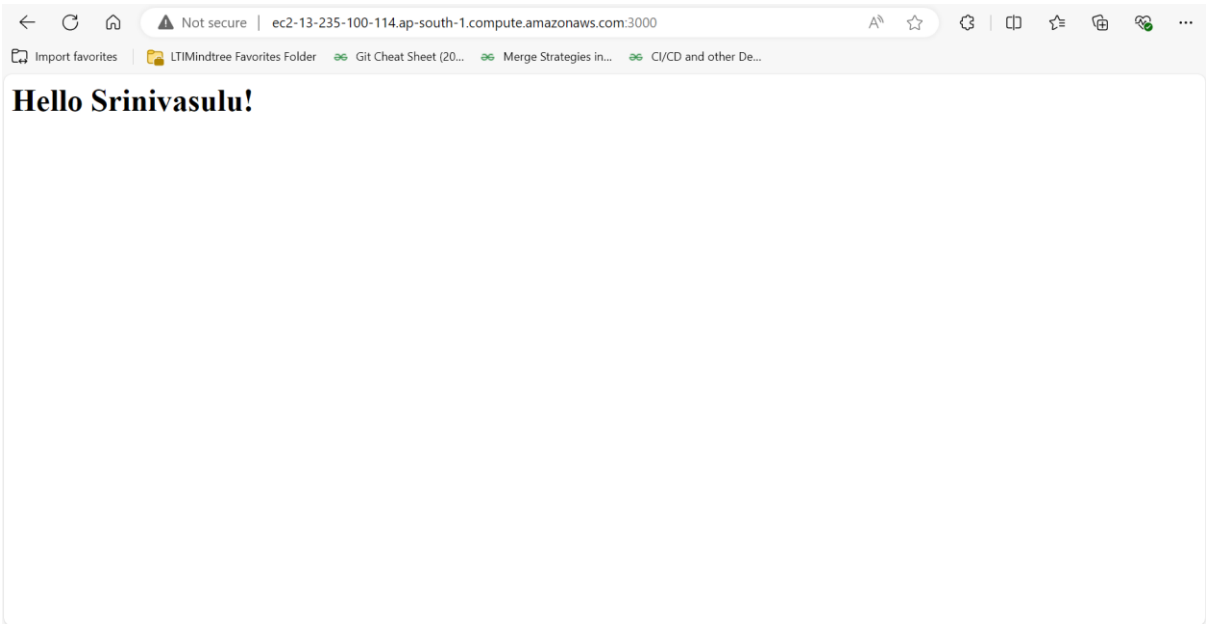
Ex: http://ec2-13-235-42-2.ap-south-1.compute.amazonaws.com:8080//github-webhook/

Now, install pm2 library → make our index.js file to watch





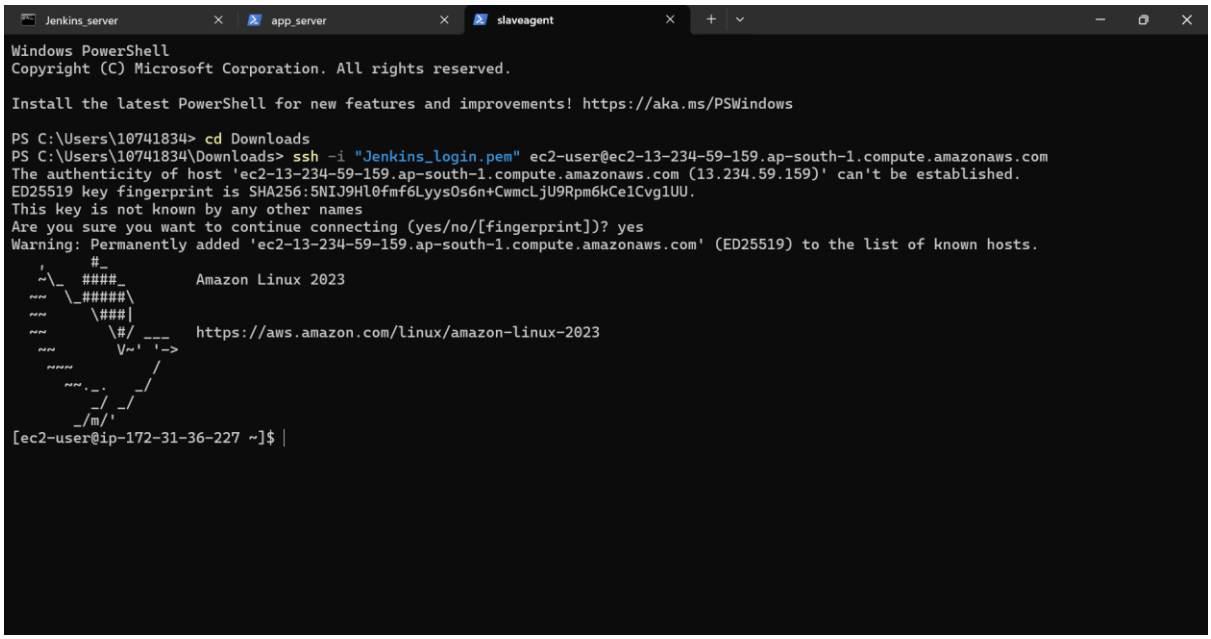
Now commit changes in the git through VS code/ other Editor tools → push to the GitHub → see the changes in Jenkins and application.



Master – slave:

In Jenkins, the **Master** manages build jobs and dispatches them to **Slave** nodes for execution. Slaves handle the actual build tasks.

Create a new EC2 instance for slave agent and connect the instance in command prompt.



Now install git, java, node.js in the slave agent server.

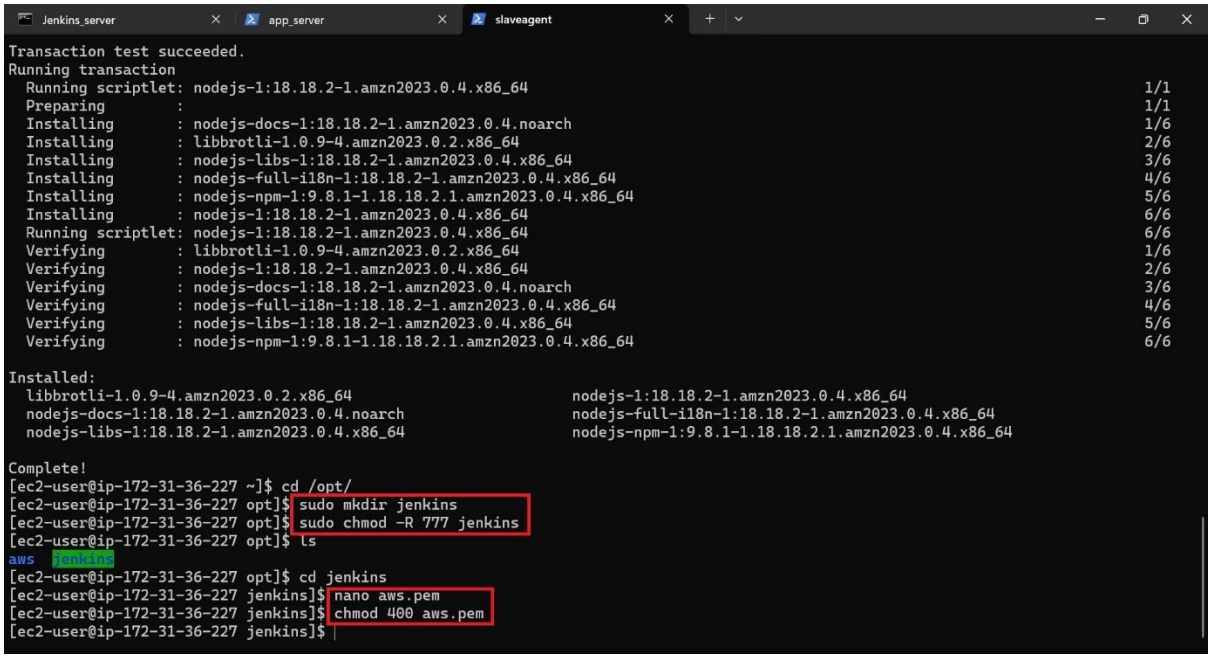
For git: sudo yum install git -y

For node.js: sudo yum install nodejs -y

For Java: sudo yum install java -y

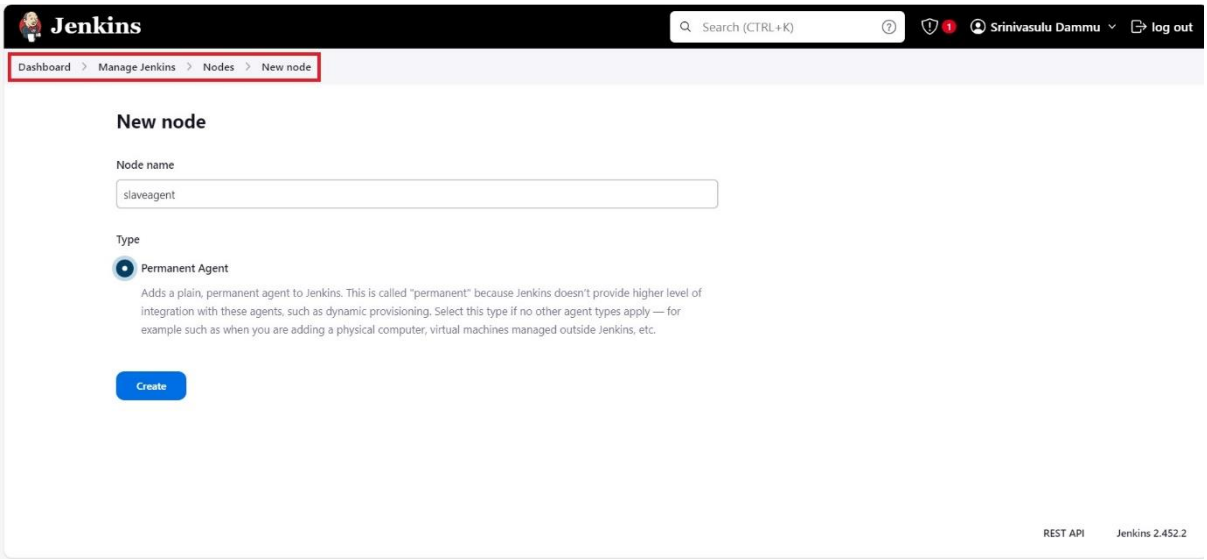
Create “jenkins” directory in the “opt” directory and change the permissions. Create “aws.pem” key pair in the “jenkins” directory.

Note: **chmod 400 <filename>**, it sets the permissions of the file so that only the owner has read permissions. This is commonly used to secure sensitive files, such as SSH private keys.

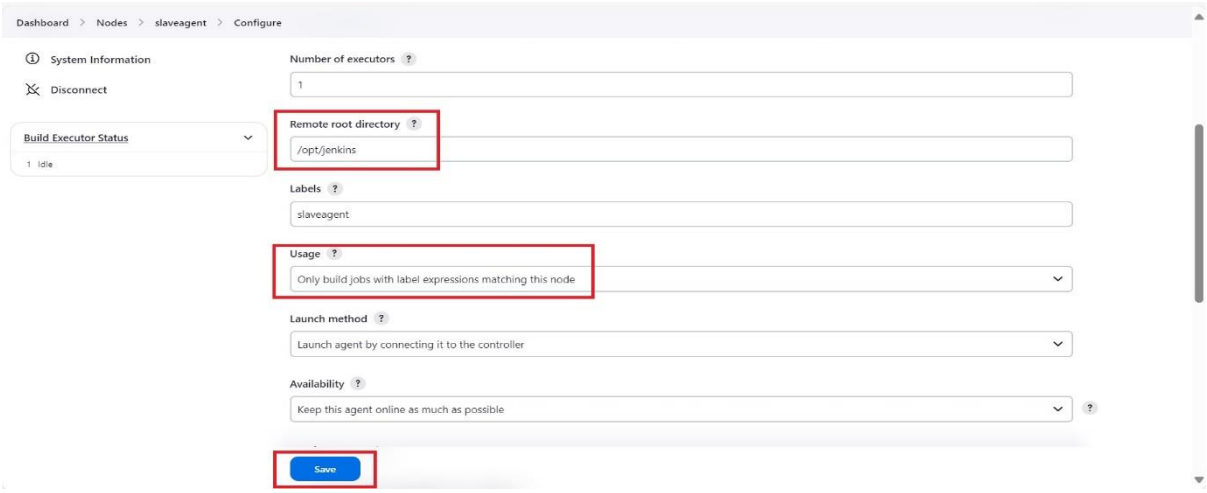


Now create a slave agent node in Jenkins:

Dashboard → Manage Jenkins → Nodes → new Node

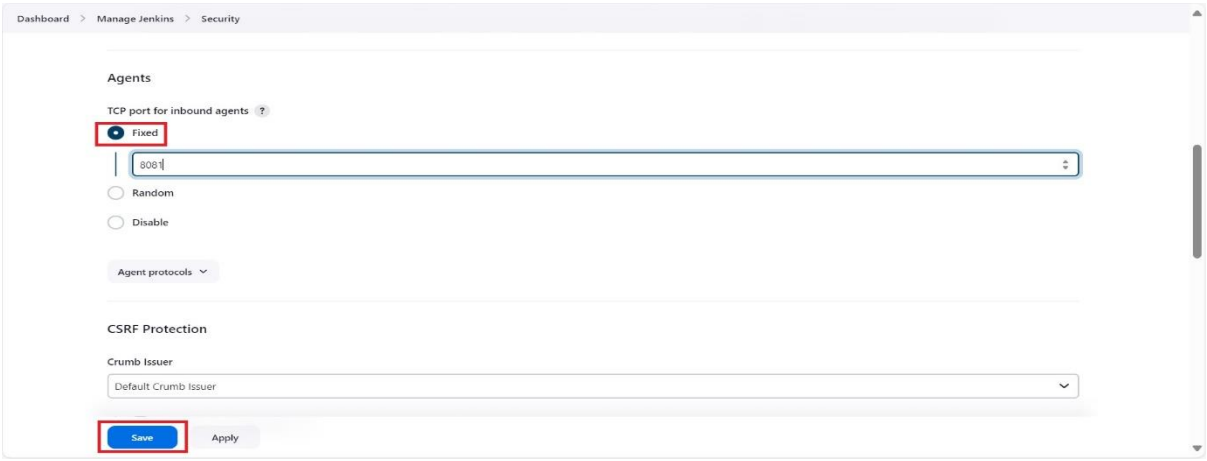


Make sure that, the remote directory should be matches the path of application in the app server.



Now add the security port in the Jenkins: (Dashboard → Manage Jenkins → Security)

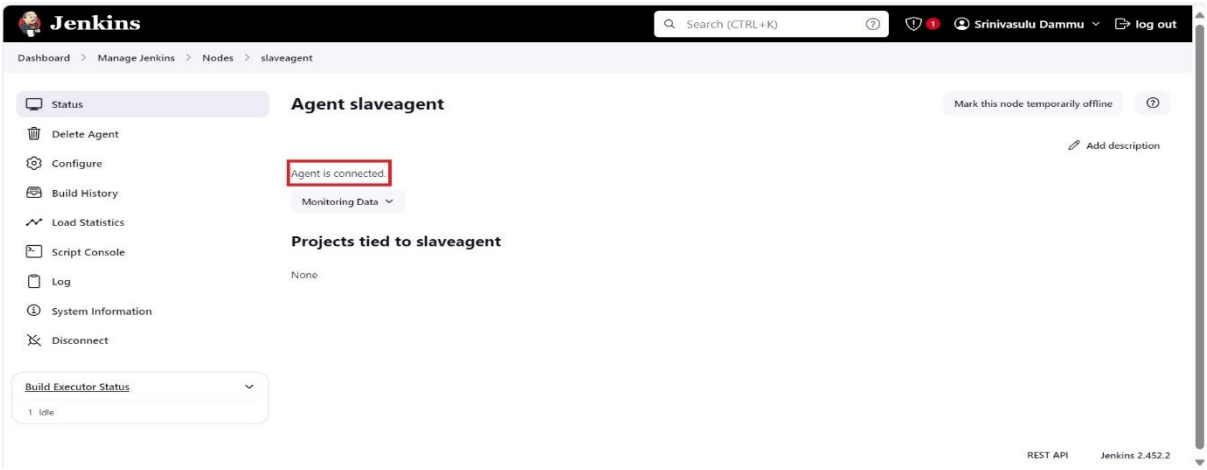
Adding a security port for the Jenkins slave agent ensures secure communication between the master and agents.



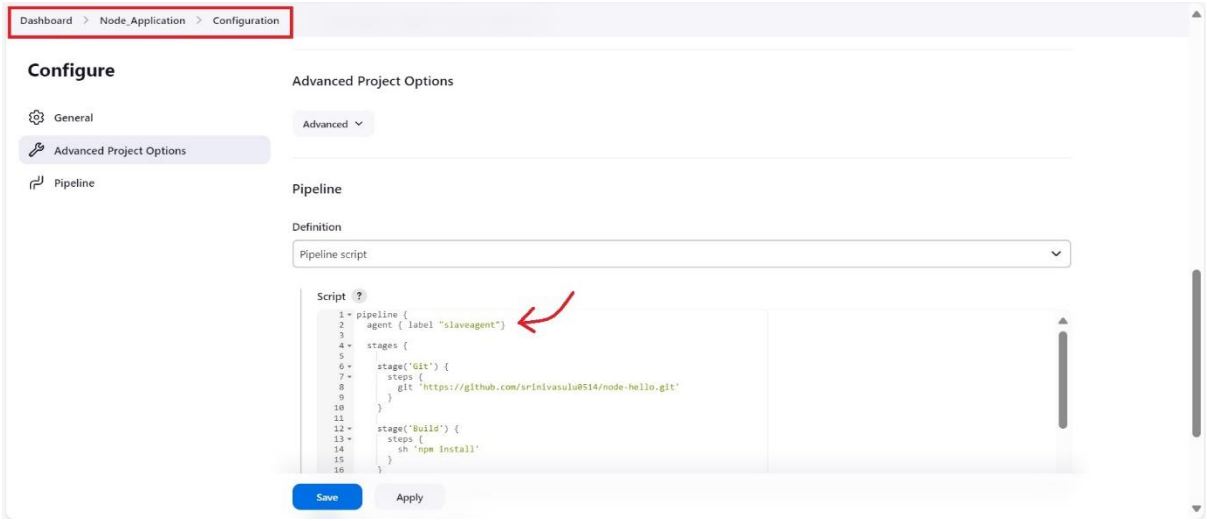
Add “8081” in security group (slave_agent EC2 instance).

In Jenkins (Dashboard →Manage Jenkins → Nodes → “slaveagent”)

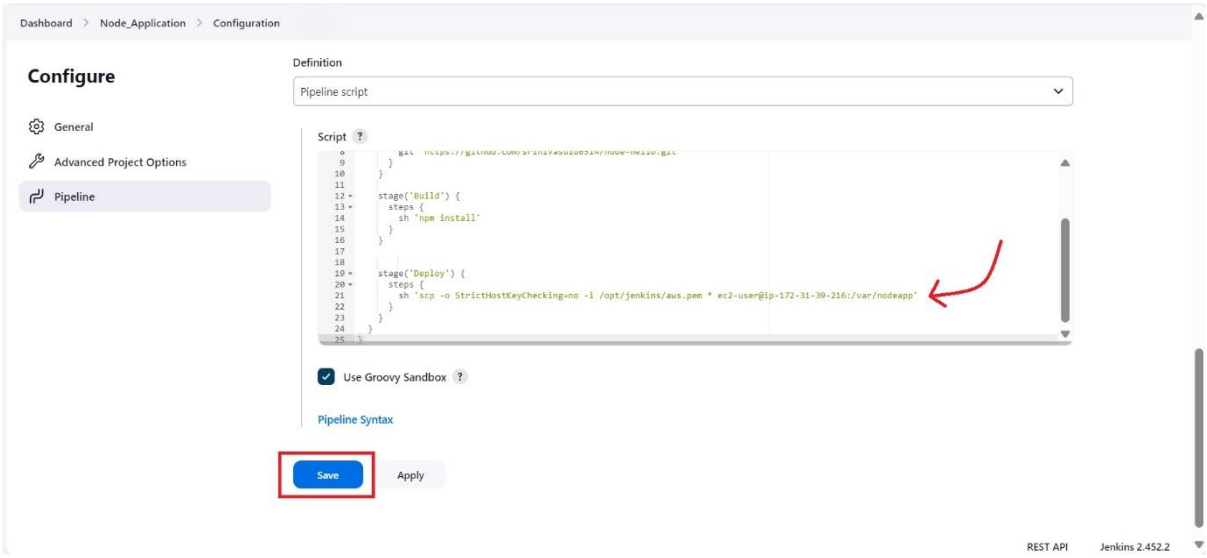
Copy the Unix commands and connect the slaveagent.



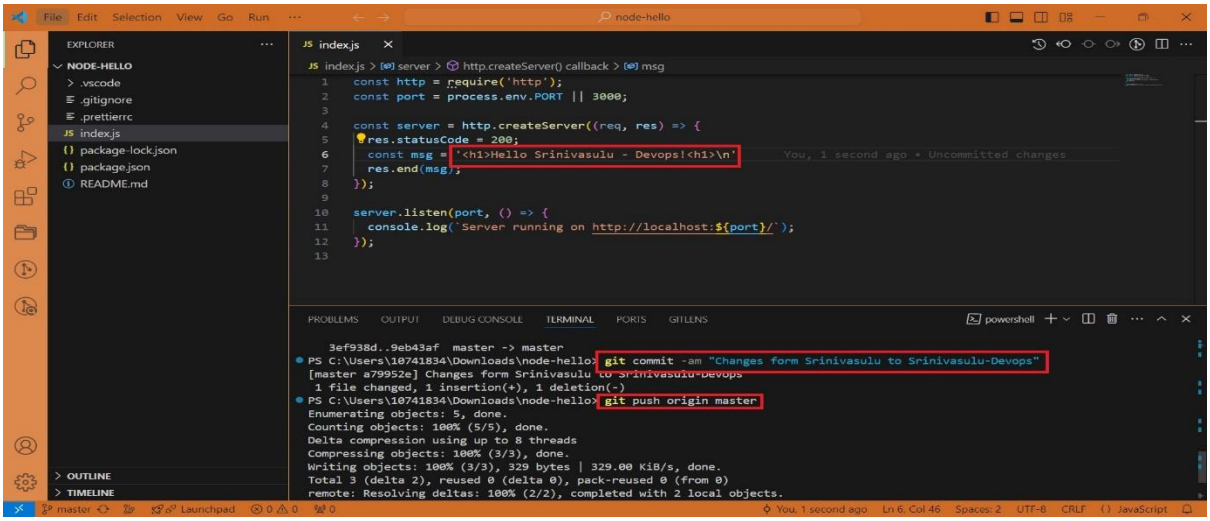
Now, go to the (Dashboard → Node_Application (pipeline project which we have created) → configure)



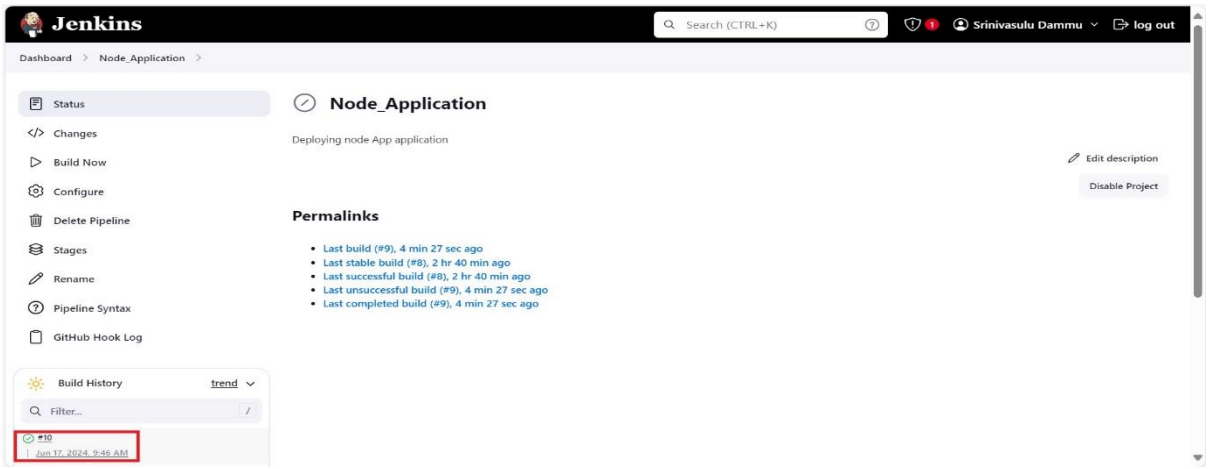
Note: Name the label which we were created (ex: agentslave).



We have already created automate process above, refer to it. Now commit changes in the git through VS code/ other Editor tools → push to the GitHub → see the changes in Jenkins and application.



Auto Build is completed.



Finally, the updated application is done with help of slave agent.

6. Conclusion

Summary of Steps

- Set up Git and GitHub for version control.
- Launch and configure an Amazon EC2 instance.
- Install and configure Jenkins on EC2.
- Configure Jenkins to build and deploy a Node.js application.
- Creating slave agent to perform the task.
- Performing task using Master-slave concept.

Troubleshooting Tips

- Ensure security groups are correctly configured.
- Check Jenkins logs for build errors.
- Verify GitHub webhook configuration.