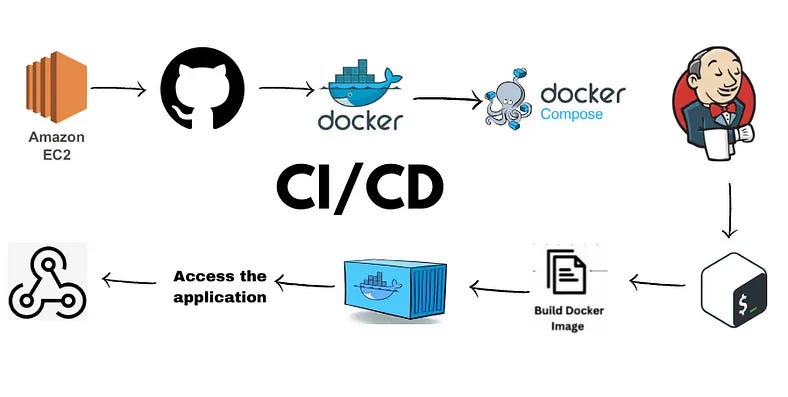
**Project 1: Continuous Integration and Deployment of Node.js Application with Jenkins, AWS EC2 and Docker.**



This outlines the process of deploying a Node.js application on an EC2 instance and setting up a Jenkins-based CICD pipeline.

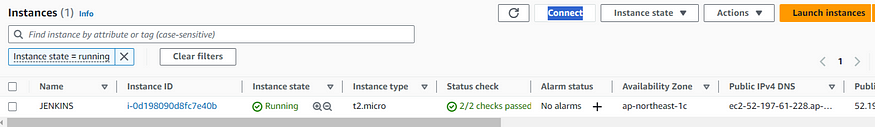
**The tools utilized in this project are as follows:**

1. AWS-EC2
2. GitHub
3. Docker
4. Jenkins

**What is CICD pipeline?**

A CI/CD pipeline is an automated system that integrates, tests, and deploys code changes continuously, allowing for faster and more reliable updates to applications.

## **Step 1: Set up an EC2 instance and establish an SSH connection to it.**

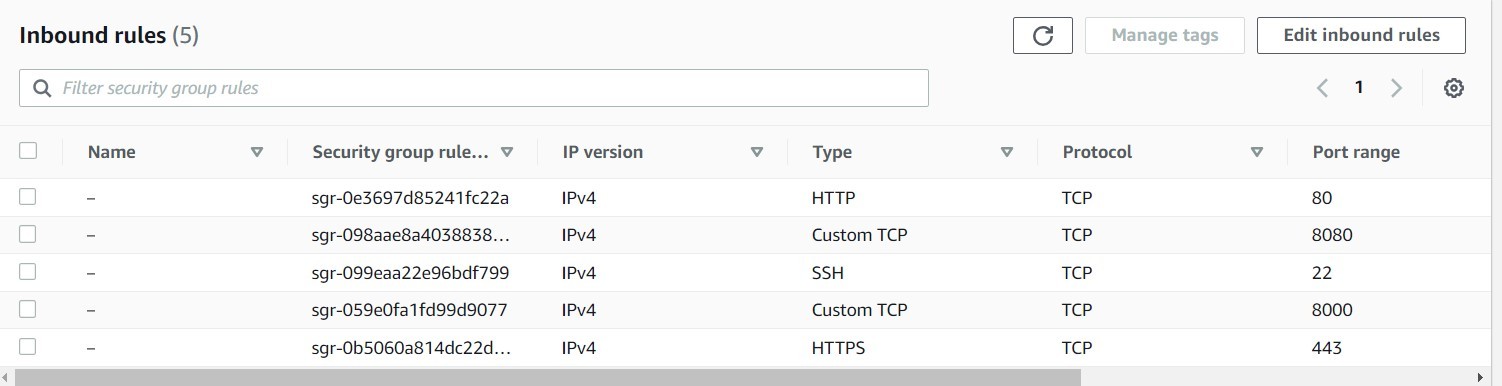


## **Step 2: Install Jenkins on the server**

sudo apt update  
sudo apt install openjdk-11-jre  
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \  
/usr/share/keyrings/jenkins-keyring.asc > /dev/null  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \  
https://pkg.jenkins.io/debian binary/ | sudo tee \  
/etc/apt/sources.list.d/jenkins.list > /dev/null  
sudo apt-get update  
sudo apt-get install jenkinssudo   
systemctl enable jenkins  
sudo systemctl start jenkins  
sudo systemctl status Jenkins

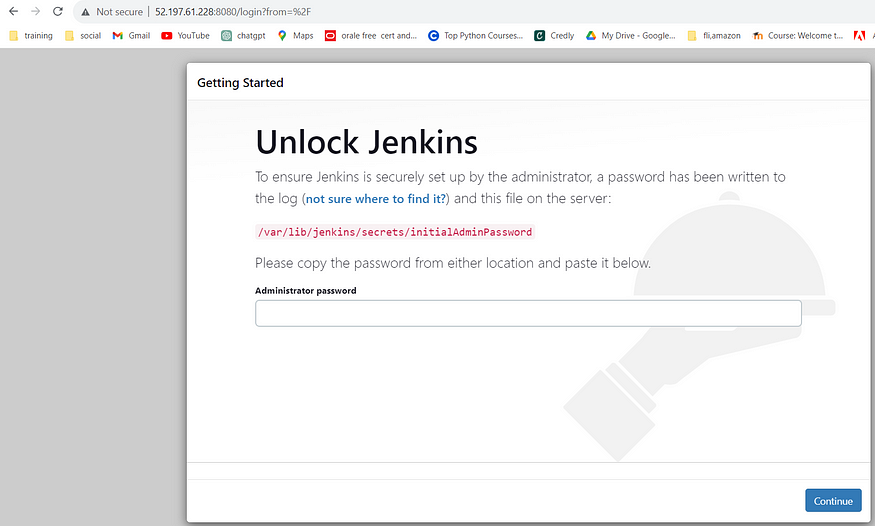
## **To access Jenkins**

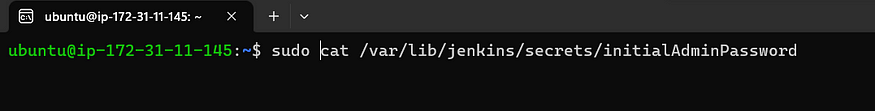
I have set the inbound rules for the security group as below.



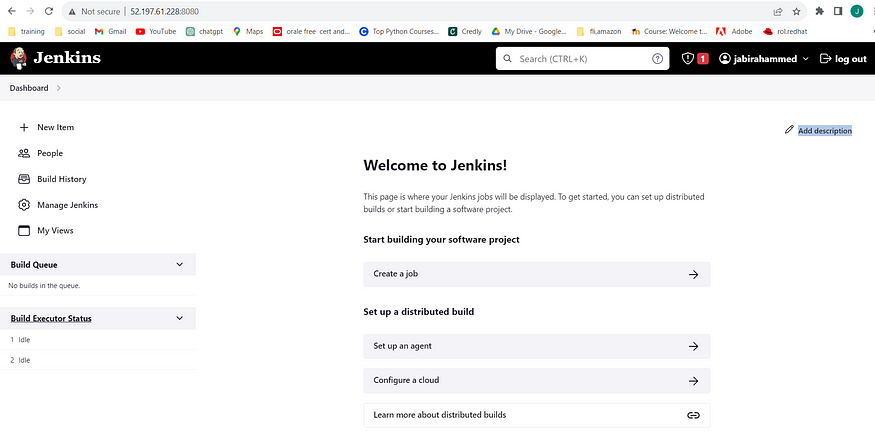
You can access the application using the format “public-ip:8080”, where “public-ip” is the public IP address of your instance.

Upon accessing Jenkins, you will be directed to a page where you can obtain the password. Use the “cat” command with the provided path on that page to retrieve the password.



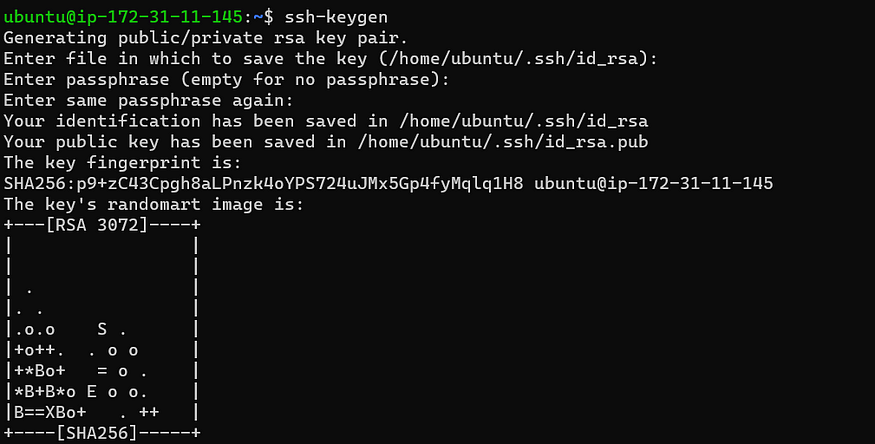


* Follow the on-screen instructions to complete the Jenkins setup, including installing suggested plugins.

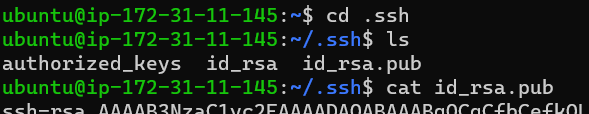


## To connect your Jenkins job with a GitHub repository, follow these steps:

Before configuring Jenkins, you must add the public key to facilitate communication between Jenkins and GitHub. This allows Jenkins to access the source code from the GitHub repository.



The “ssh-keygen” command will generate both a public key and a private key.

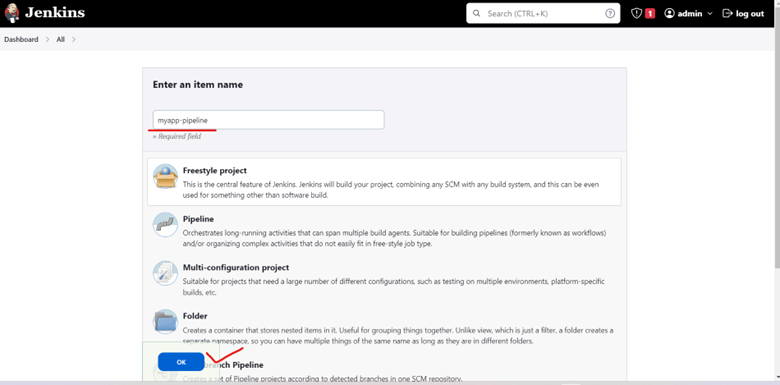


## **To enable Jenkins integration with GitHub, follow these guidelines:**

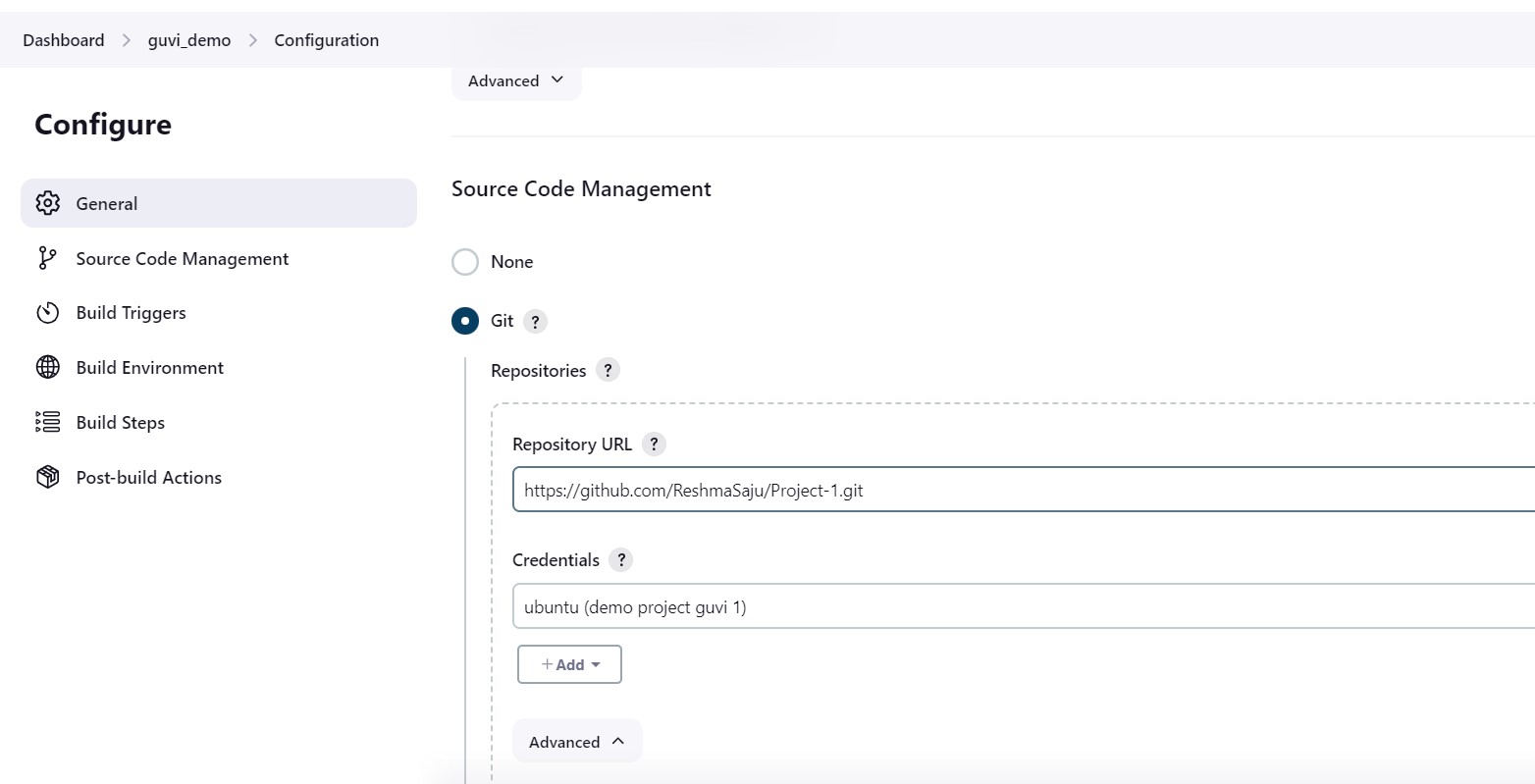
1. Open your GitHub account settings.
2. Find and access the “SSH and GPG keys” section in settings
3. Insert the public key generated using the “ssh-keygen” command. Be sure to choose the correct key-type, like “Authentication key.”

## **This is the step-by-step guide to configuring Jenkins for your Node.js application:**

Create a New Freestyle Project for your Node.js app in Jenkins.

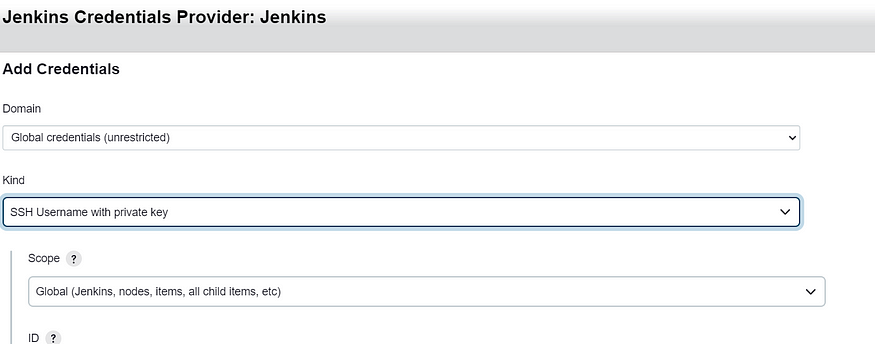


In the project configuration, provide the GitHub project URL to link it with your repository.



Choose “Git” as the version control system and set up the necessary credentials.

To allow Jenkins access to the GitHub code, add the credentials in the “Source Code Management” section.

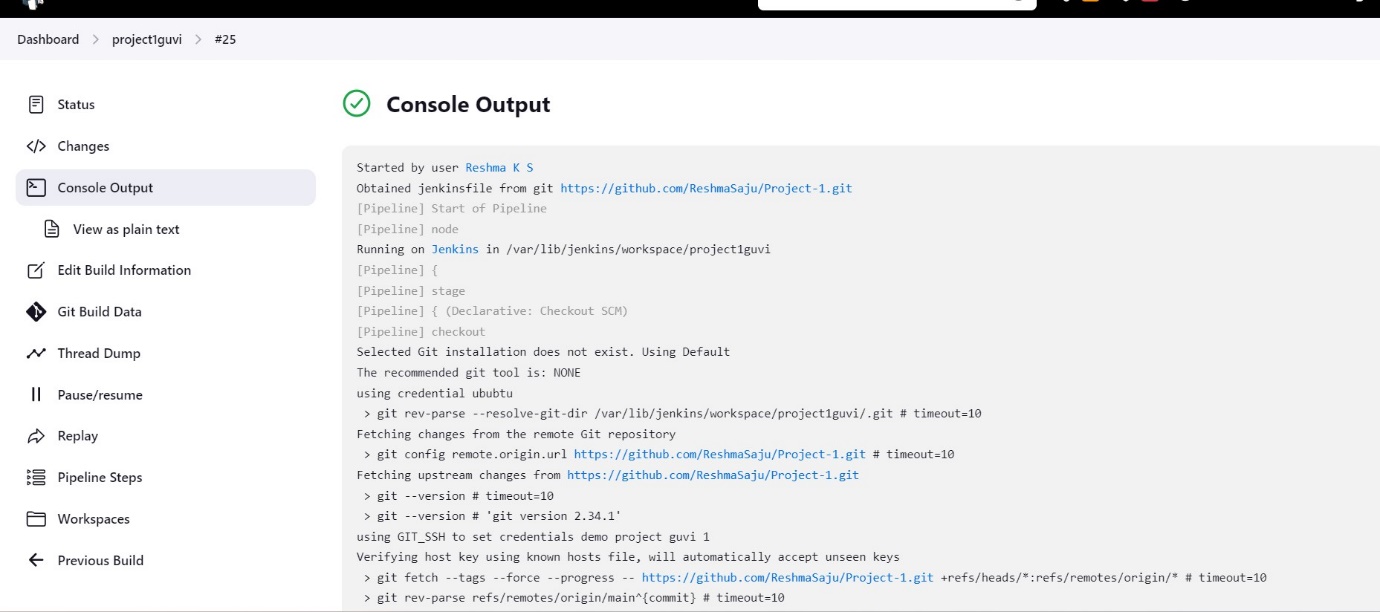


Click “Add” and input the private key generated with “ssh-keygen” as the authentication method.

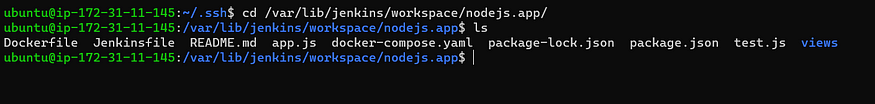
Provide a unique ID and description for the credentials. Paste the private key in the “Private Key” field and add the passphrase if applicable. Click “Add” to save the credentials.

Save the project configuration.

Now, you’re all set to build the job. Click “Build Now” to initiate the build process.



Check the instance to ensure the repository is successfully cloned.



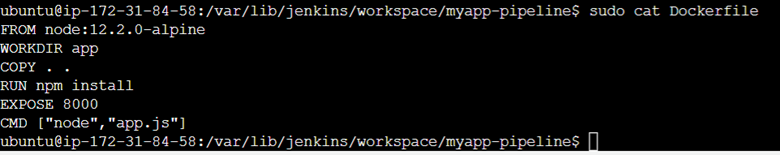
By following these steps, Jenkins will be configured to build your Node.js application and clone the repository onto the instance.

sudo apt install nodejs  
sudo apt install npm  
node app.js

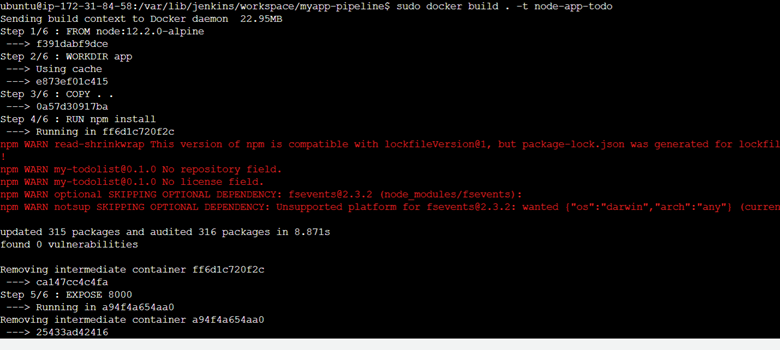
Dockerize the application so it can run in any environment.

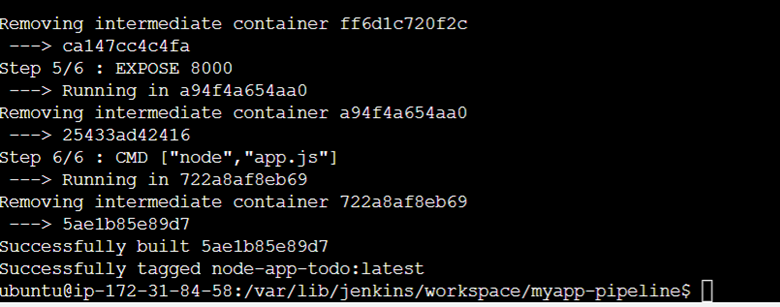
sudo apt install docker.io

vi Dockerfile



Before we setup this in Jenkins, lets check manually if we can containerize the application.





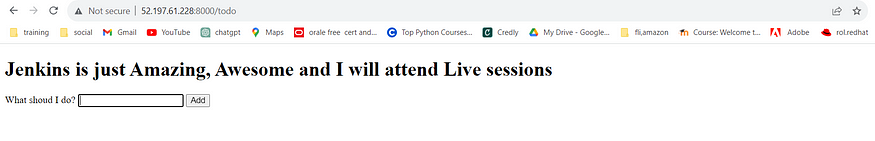
Docker Image has been built. Now lets containerize.

No alt text provided for this image

A container will be created

sudo docker ps

Access the application using public ip through port 8000

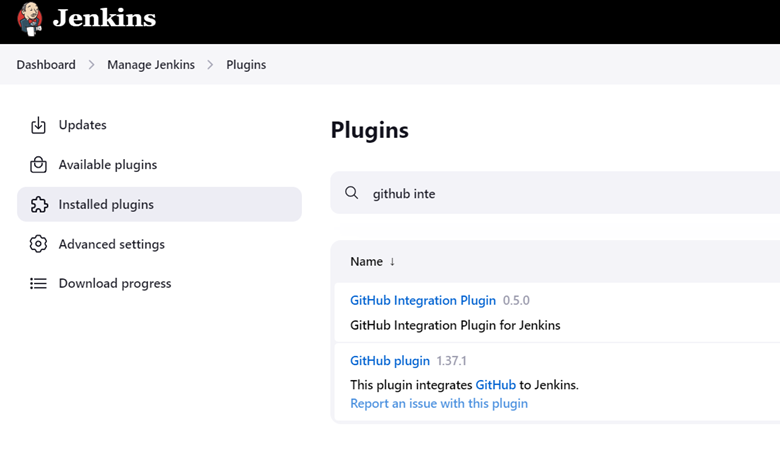


Application is perfectly containerized and the output is absolutely fine. Lets setup webhooks and integrate with Jenkins.

Kill the existing container first.

docker kill <container\_id>

**Install the GitHub Integration plugin in Jenkins:**

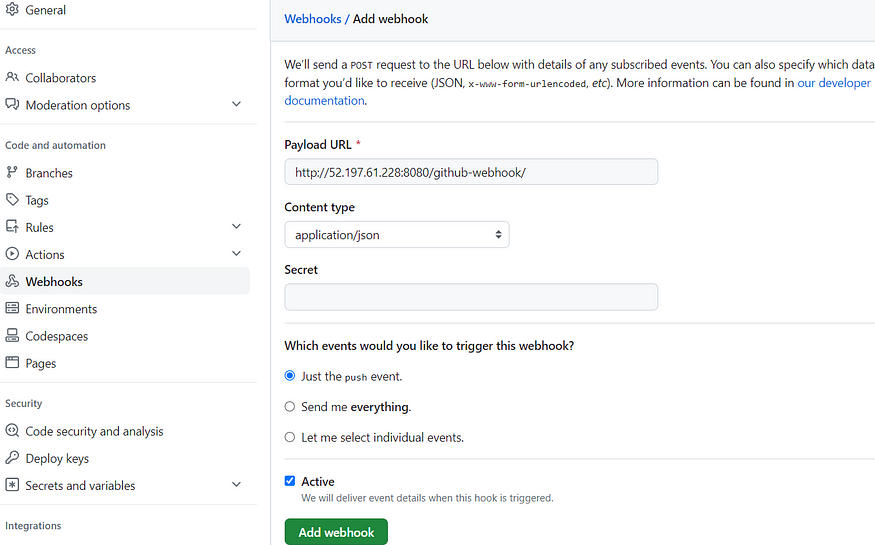


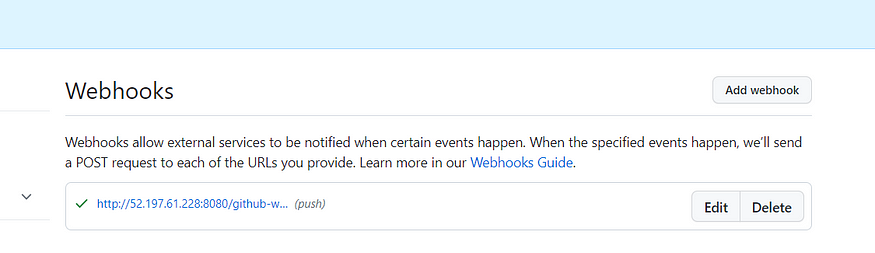
**Configure webhook in the GitHub repository settings:**

Go to repository settings -> webhook -> Add webhook -> Update Payload URL with the Jenkinslink/github-webhook/

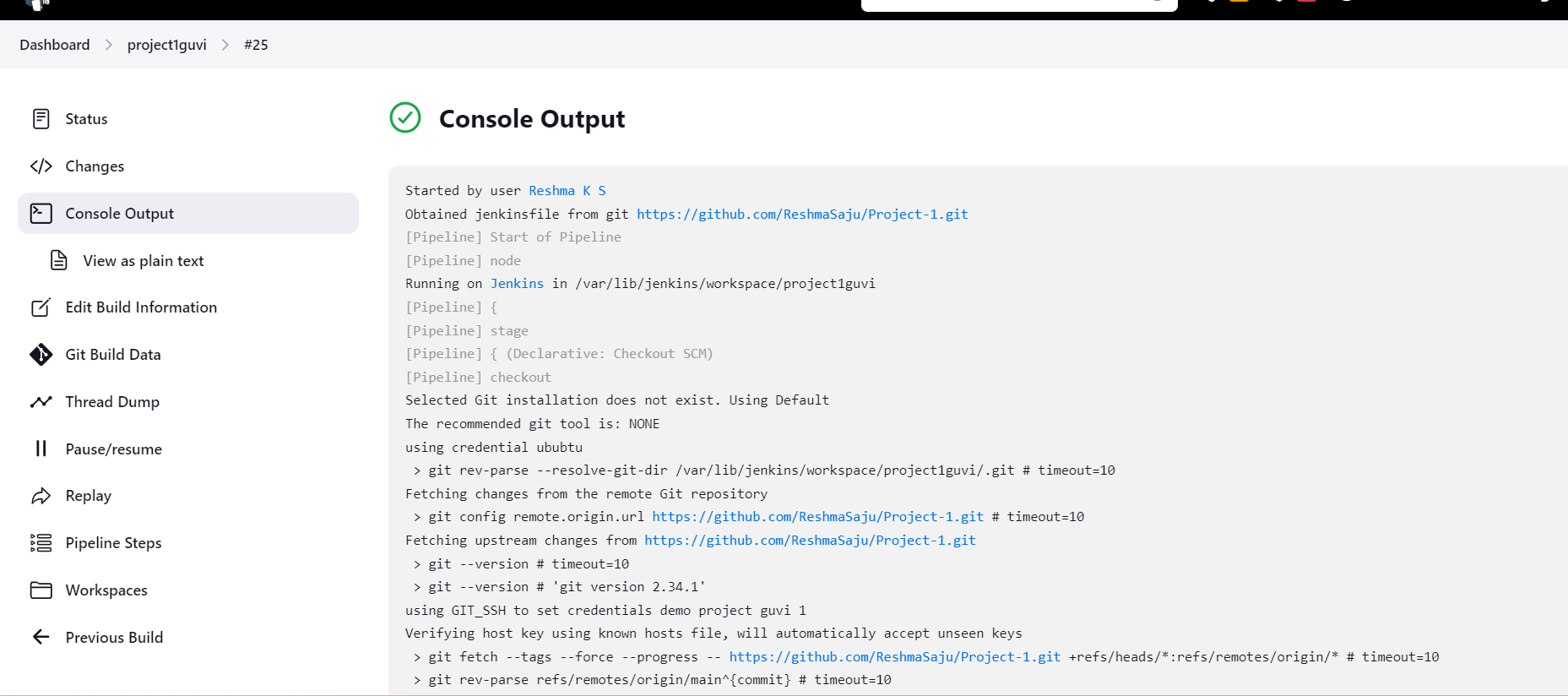
Check the screenshot below :

Content type -> application.json



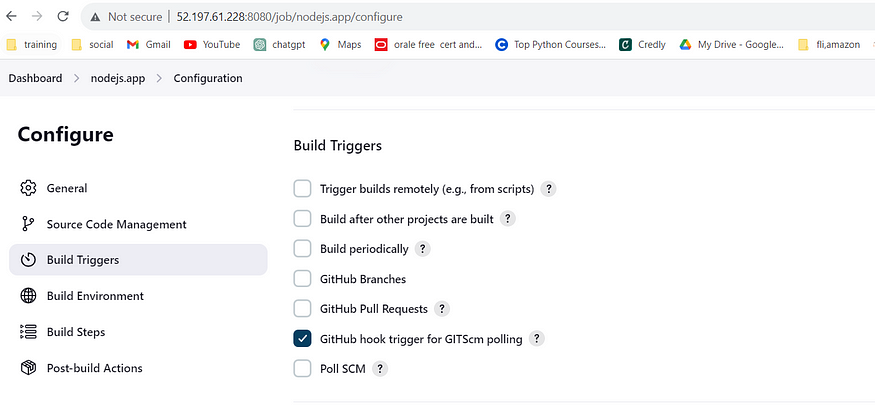


Build Success



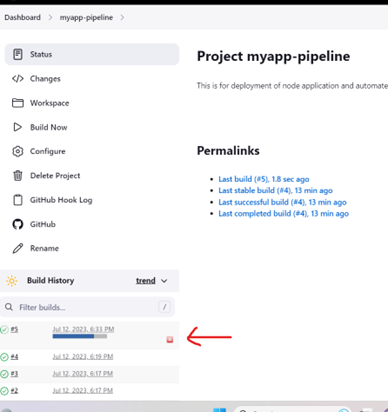
To trigger the Jenkins job automatically.

Go to jenkins -> configure



Change the code and commit to git repo. As soon as the changes updated in the git the Jenkins will initiate build.

Now we can see the job triggered automatically.



In summary, this project successfully established a Jenkins-based CI/CD pipeline to deploy a Node.js application on an EC2 instance. By integrating AWS, GitHub, Docker, and Jenkins, we achieved automated code integration, testing, and deployment, resulting in faster updates and improved application reliability. The use of Docker ensured application portability, and setting up GitHub webhooks enabled seamless automation. This project showcases the power of CI/CD in streamlining development and enhancing overall software deployment processes.

Github: https://github.com/ReshmaSaju/Project-1.git