**Deploy React application to remote server using CI/CD pipeline**

**Objective:** To connect to a remote server (EC2 Server) via Jenkins using a SSH Agent and execute docker commands to create a container that runs an application.

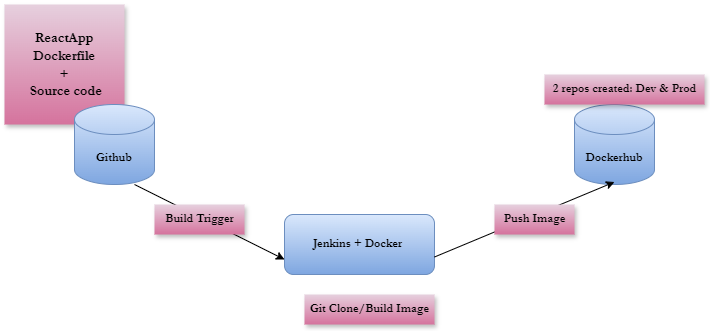
**Tools & services used:**

Jenkins – To automate the CI/CD

GitHub – The application code repository.

Docker - To build image and run containers - Docker file, Docker-compose file)

**High-level Architecture Diagram:**



**Steps Involved:**

* Automating Builds
* Automating Docker image creation
* Automating Docker image upload
* Automating Docker container provisioning

**Pre-requisites:**

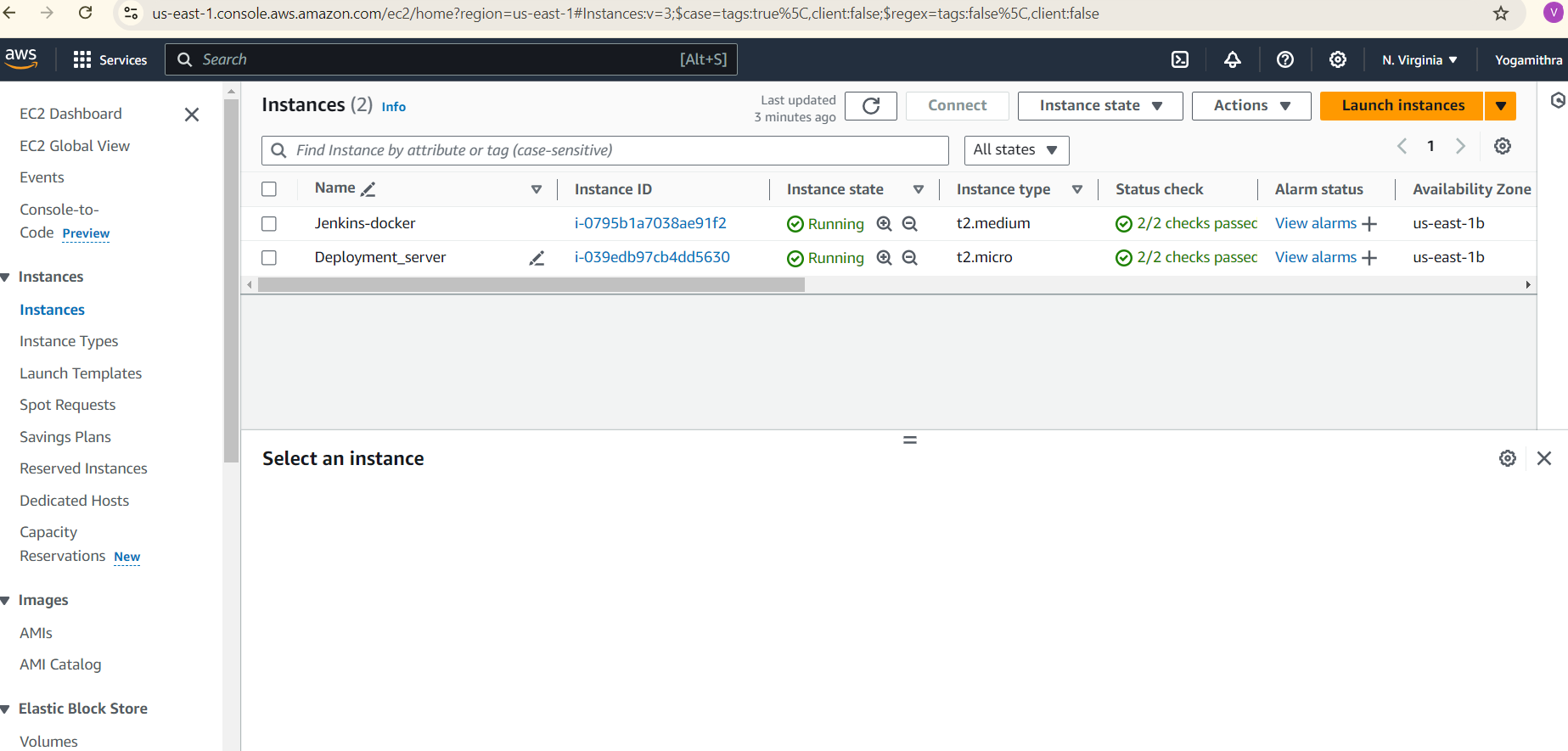
1. Jenkins is up and running
2. Docker & Git installed on Jenkins instance and configured
3. Docker plug-in installed in Jenkins
4. GitHub plug-in installed in Jenkins
5. Docker hub account
6. AWS account with EC2 instance

**Server 1: Jenkins Setup:**

* Launch an EC2 instance for Jenkins (Server 1).
* Name: Jenkins-docker

**Server 2: EC2 Instance Setup:**

* Launch another EC2 instance to deploy your application (Server 2).
* Name: Deployment\_server



**Server 1 – Jenkins, docker, git and other plugins installation and configuration**

Install Docker. In your instance, run the following commands:

sudo yum update -y

sudo yum install -y docker

Start the Docker service:

sudo service docker start

Add the ec2-user to the Docker group so you can execute Docker commands without using sudo:

sudo usermod -a -G docker ec2-user

Log out and log back in again to pick up the new Docker group permissions.

Installing Docker-Compose

To install Docker-Compose, use the following commands:

sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

Next, set the permissions:

sudo chmod +x /usr/local/bin/docker-compose

Verify the installation:

docker-compose –version

Jenkins Installation

yum list | grep java

sudo yum install javapackagename -y

where java-17-amazon-corretto-devel.x86\_64 is a java package

Now install Jenkins

sudo wget -O /etc/yum.repos.d/jenkins.repo <https://pkg.jenkins.io/redhat-stable/jenkins.repo>

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

sudo yum clean all

sudo yum update -y

sudo yum install jenkins -y

jenkins --version

sudo systemctl status jenkins

sudo systemctl start jenkins

Enable 8080 port in SecurityGroup to access Jenkins

Access the public ip of the instance

Run the following command to add the Jenkins user to the Docker group:

sudo usermod -aG docker jenkins

Then, restart the Jenkins service.

This will run Jenkins with root privileges, allowing access to the Docker daemon socket.

**Configure DockerHub credentials in Jenkins:**

1. Create a Docker Hub Account:

- Go to Docker Hub and create an account if you don't already have one.

2. Create a Docker Hub Token:

- Log in to your Docker Hub account.

- Click on your profile picture in the top right corner, then select "Account Settings".

- Scroll down to the "Security" section and click on "New Access Token".

- Enter a description for the token and click "Generate".

- Copy the token.

3. Add Docker Hub Credentials in Jenkins:

- Go to Jenkins Dashboard > Manage Jenkins > Manage Credentials.

- Click "Global Credentials" or "Folder Credentials" depending on your setup.

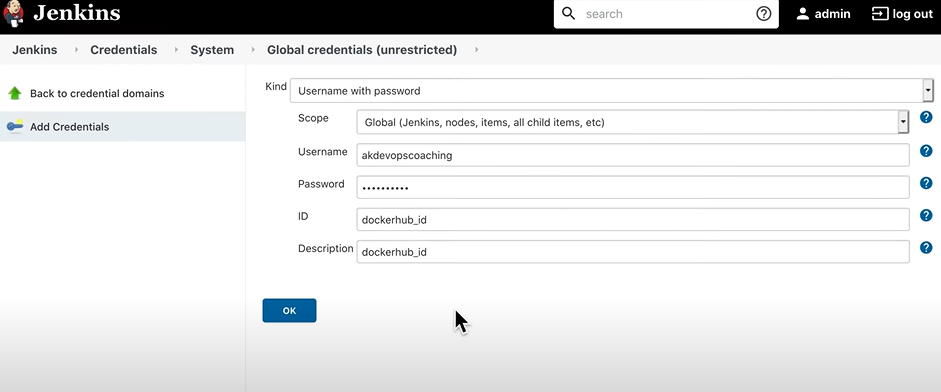
- Click "Add Credentials".

- Select "Username with Password" as the credential type.

- Enter your Docker Hub username as the "Username".

- Enter the Docker Hub token as the "Password".

- Click "OK".****



**Configure GitHub credentials in Jenkins:**

A GitHub Personal Access Token (PAT) is a token that allows you to authenticate with GitHub without using your username and password. It's a more secure way to access GitHub resources, especially when using automated tools like Jenkins.

Here's how to create a GitHub PAT:

1. Go to GitHub Settings > Developer settings > Personal access tokens

2. Click "Generate new token"

3. Choose the scopes (permissions) you want to grant the token (e.g., repo, read:org, etc.)

4. Click "Generate token"

5. Copy the token (you won't be able to see it again)

To use a GitHub PAT in Jenkins:

1. Go to Jenkins Credentials > Global Credentials > Add Credentials

2. Select "Secret text" as the credential type

3. Enter a name and description for the credential

4. Paste the GitHub PAT into the "Secret" field

5. Click "OK"

Then, in your Jenkins job configuration, you can use the PAT as a credential to access GitHub resources.

**Configure Dockerfile:**

In your GitHub repository, create a file named “Dockerfile” in the root directory.

Create the dockerfile using simple commands such as FROM, COPY, EXPOSE, ENTRYPOINT.

**Dockerfile**

FROM nginx:latest

# Copy the built files from the previous stage

COPY build/ /usr/share/nginx/html

# Expose port 80 (the default HTTP port)

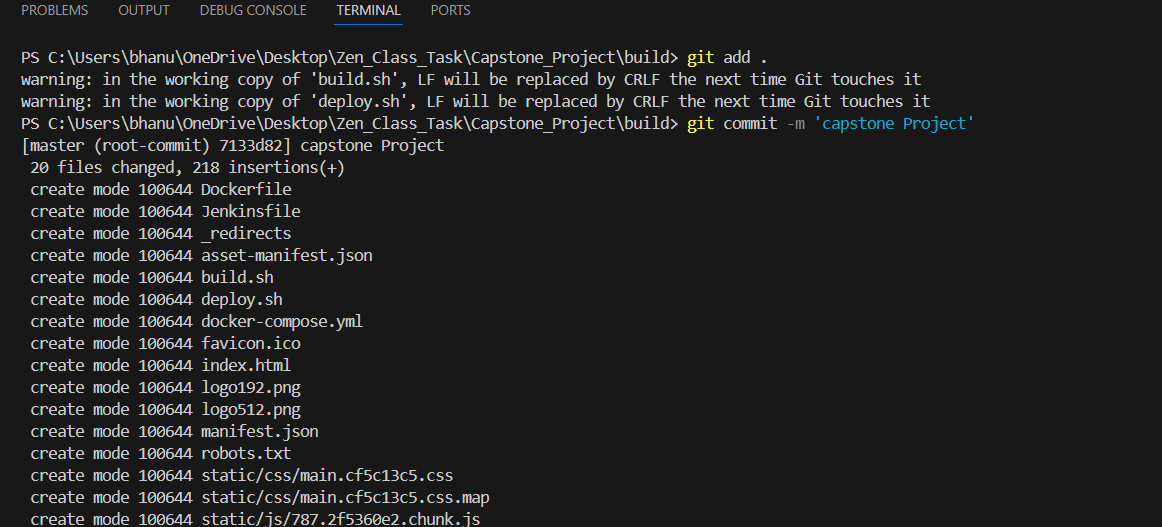
EXPOSE 80

# Start Nginx and keep it running in the foreground

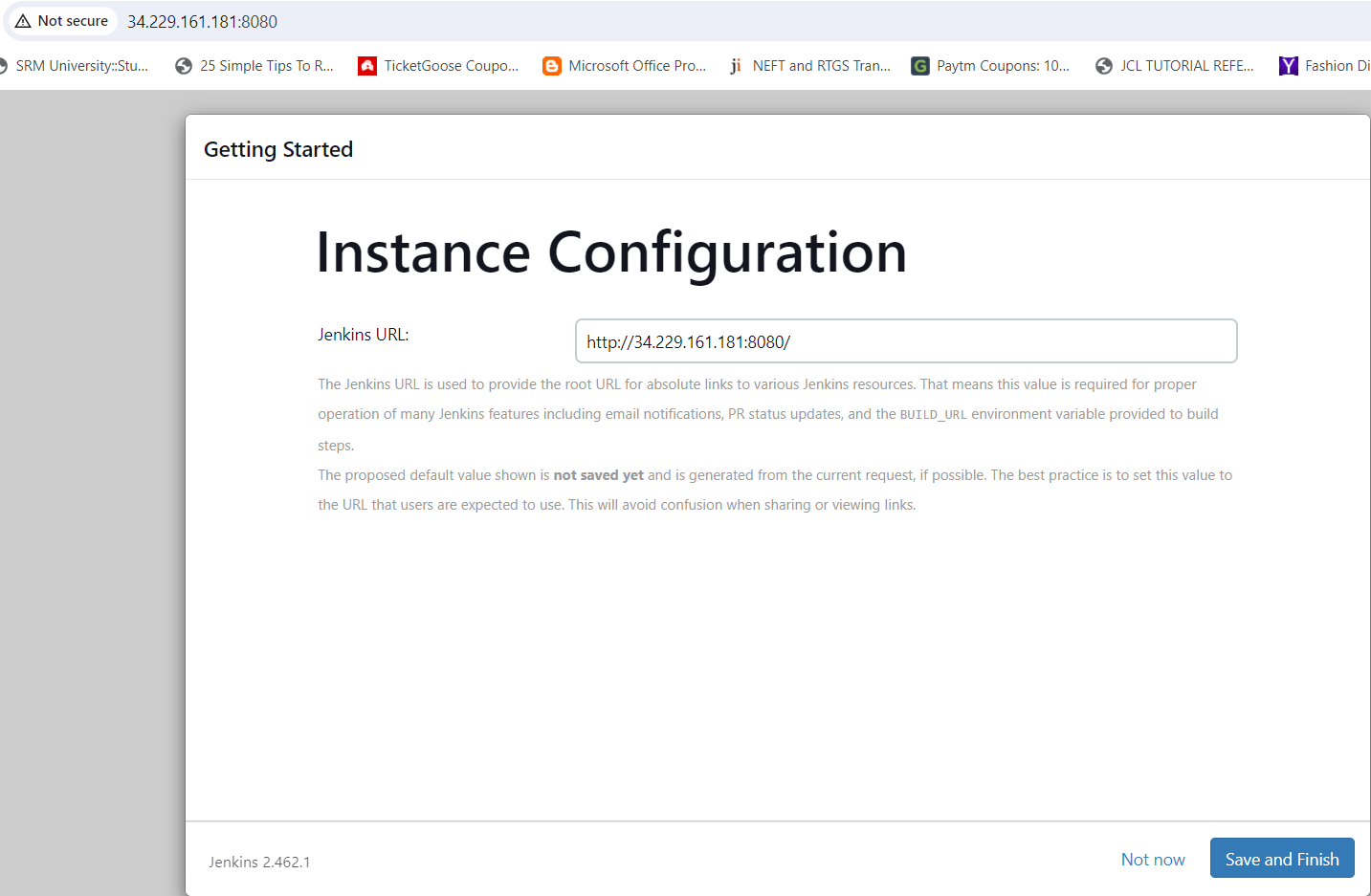
CMD ["nginx", "-g", "daemon off;"]

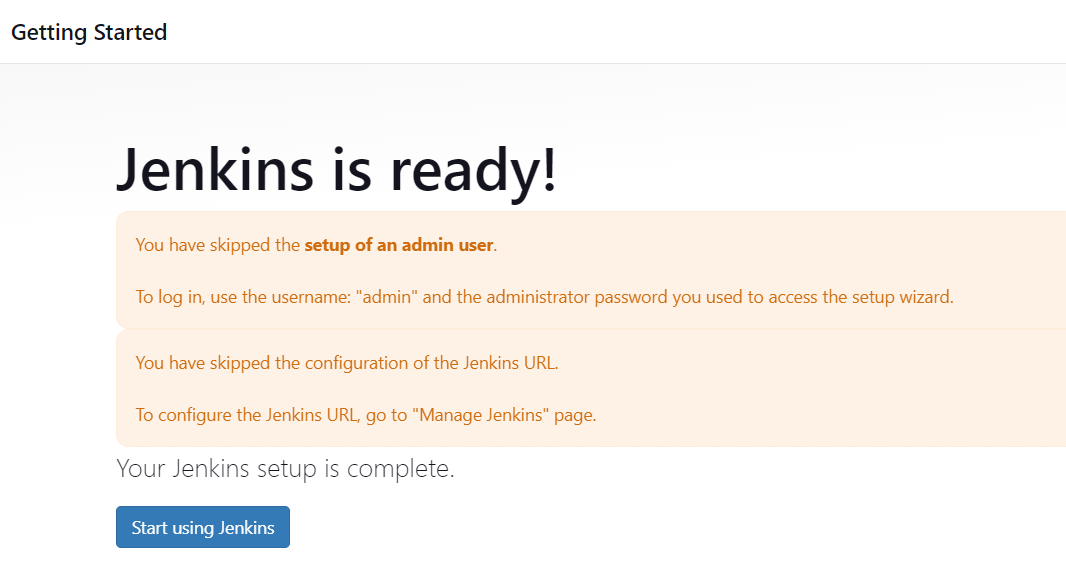
**Version Control:**

Code is pushed to remote repository using GIT CLI commands.

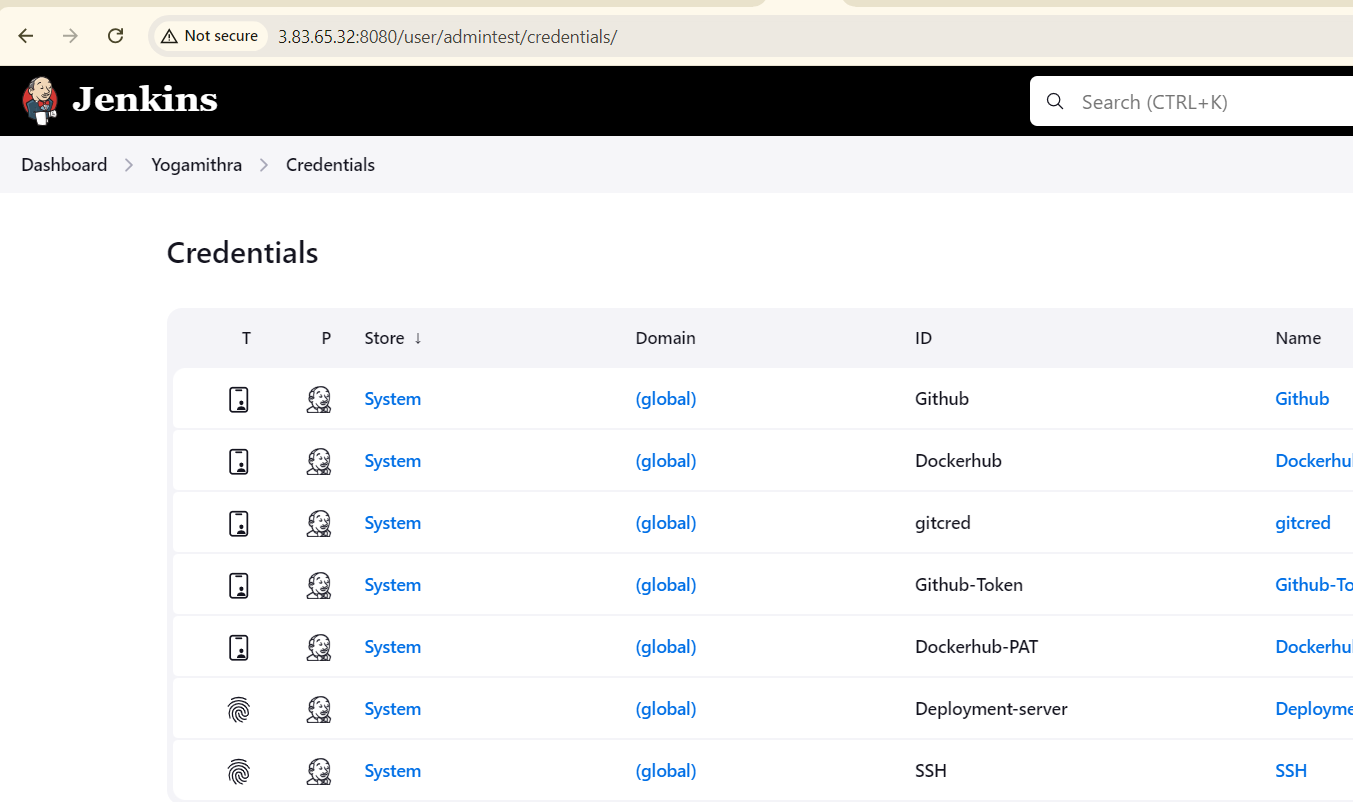


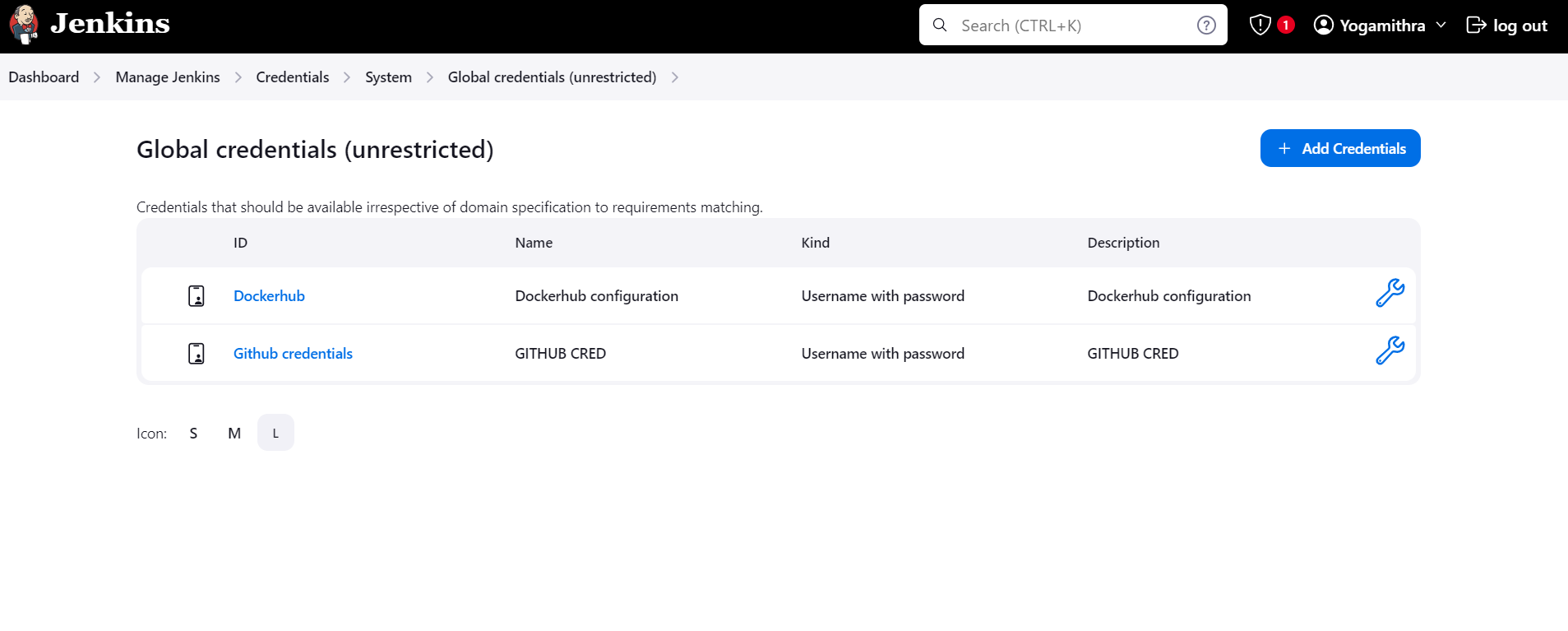
**Jenkins login page, configuration settings:**





Jenkins cred: admintest

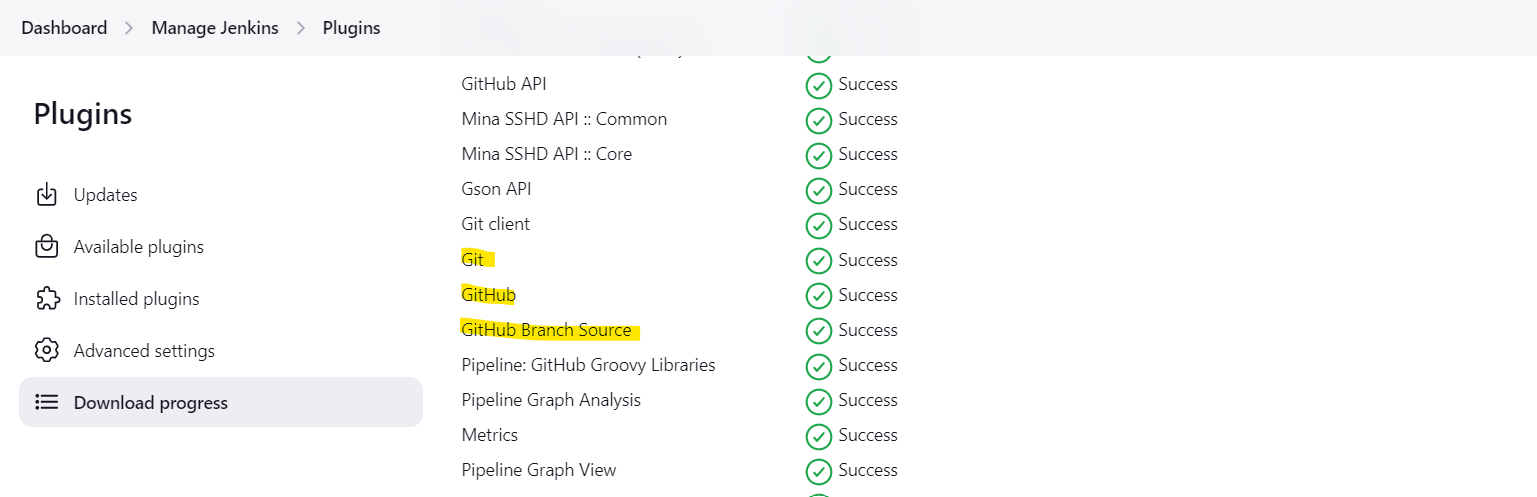


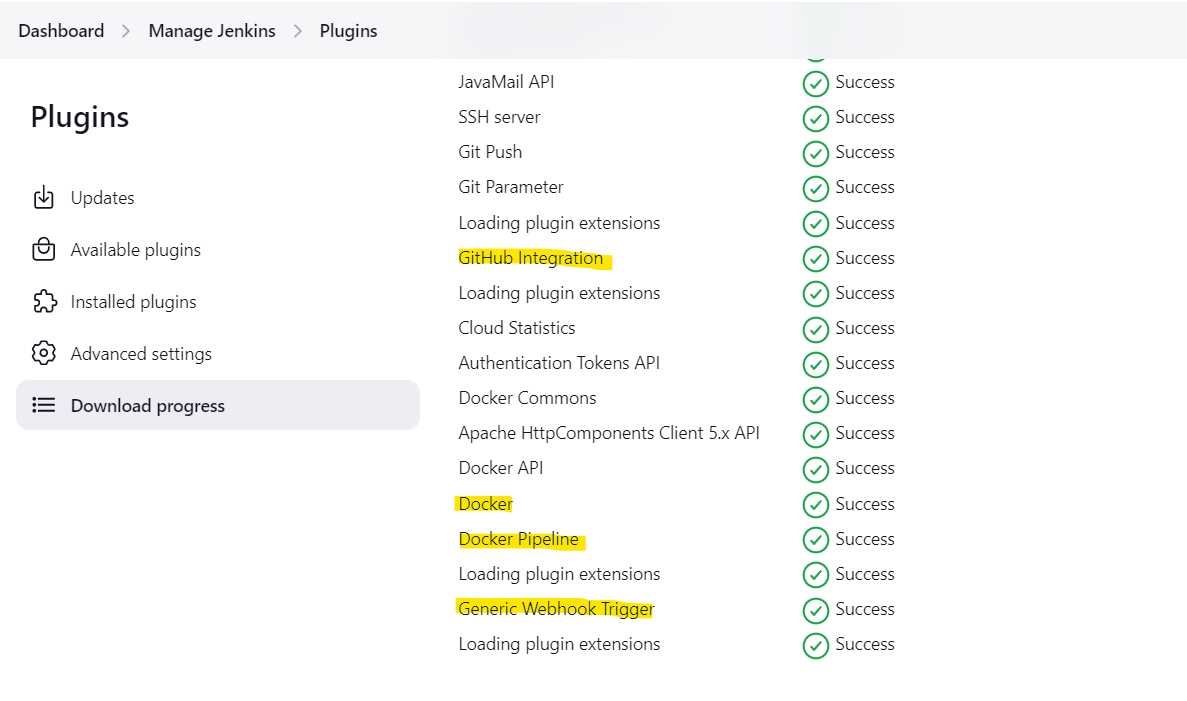


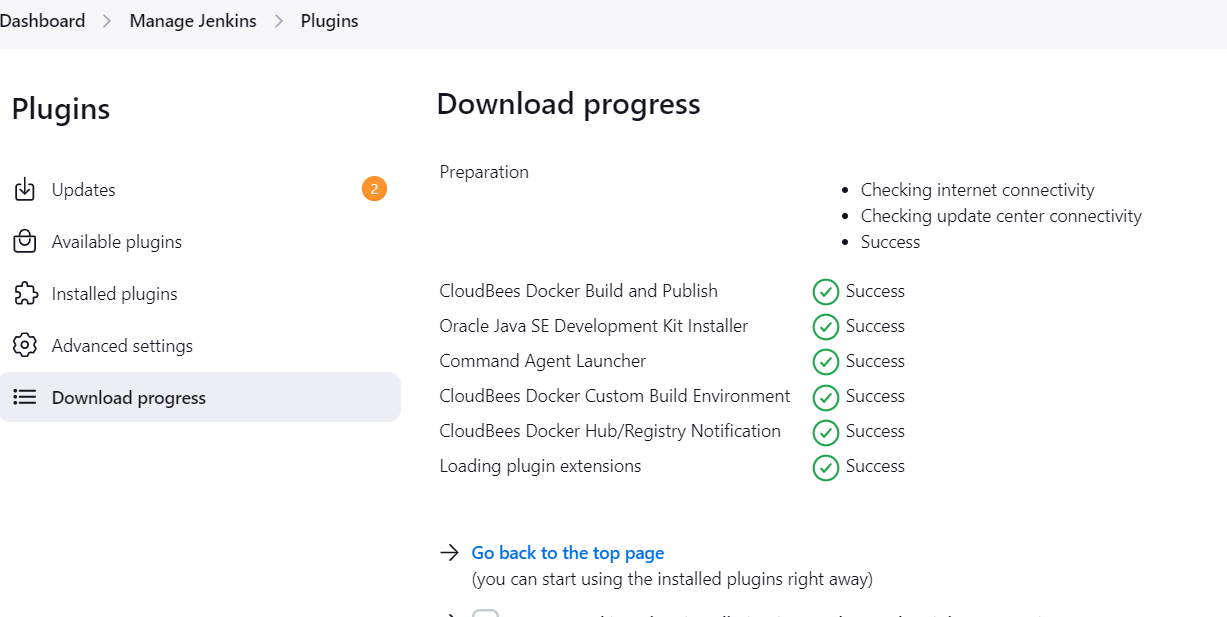
**Managing Plugins:**

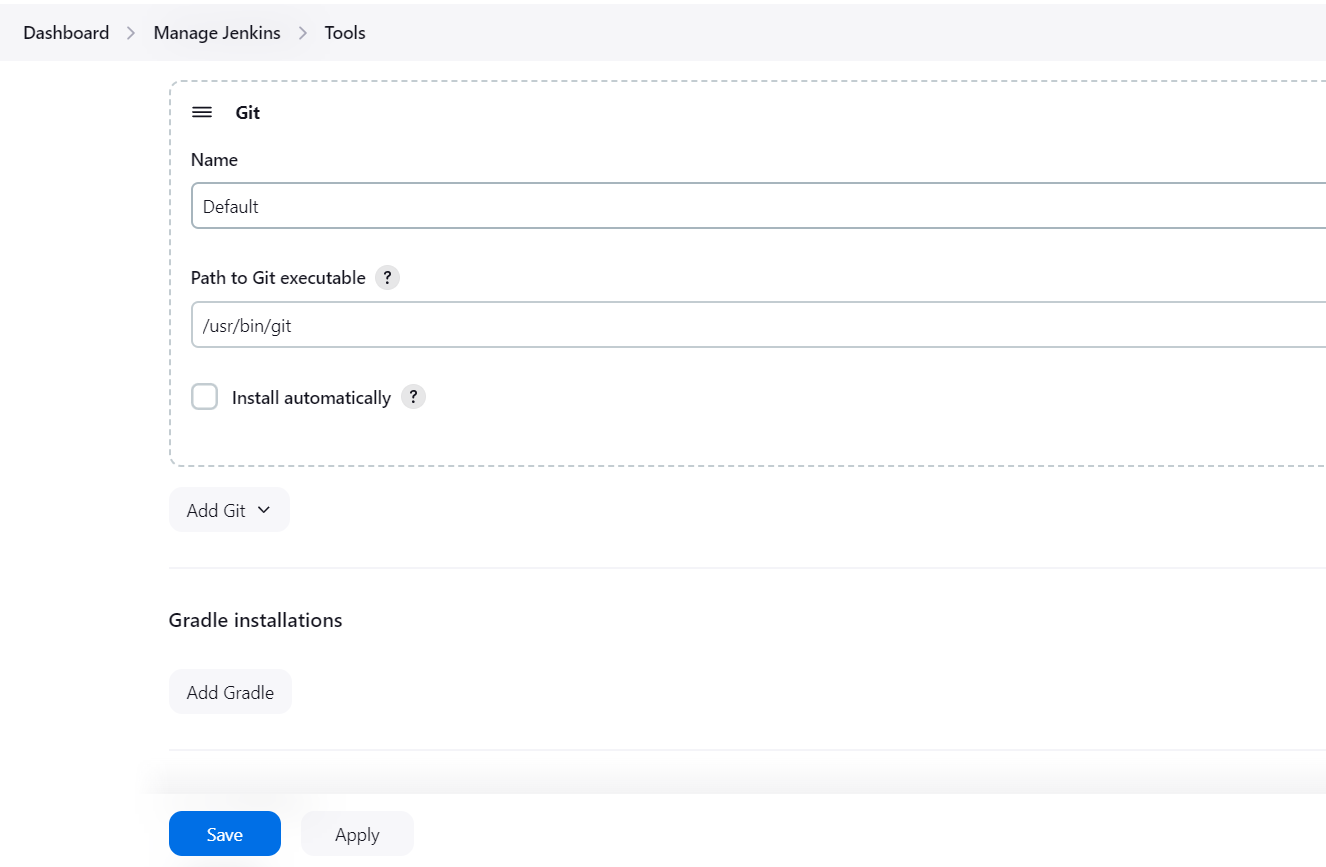
Install the below plugins as per requirement.

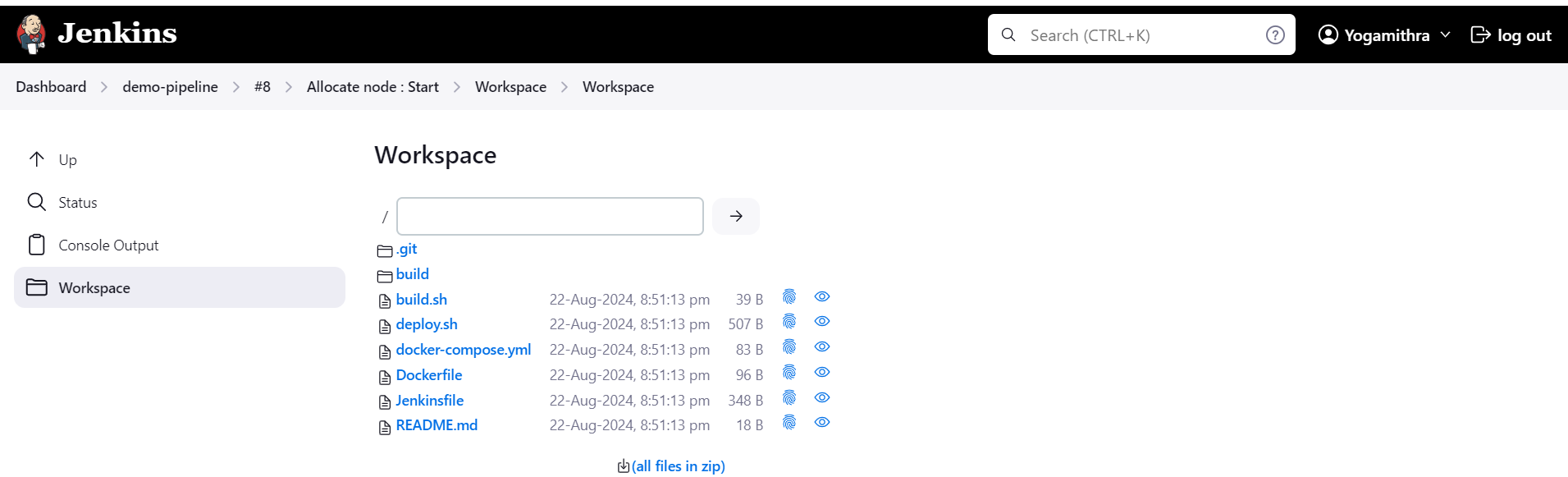
* Git
* GitHub
* GitHub Branch Source
* GitHub Integration
* Docker
* Docker Pipeline

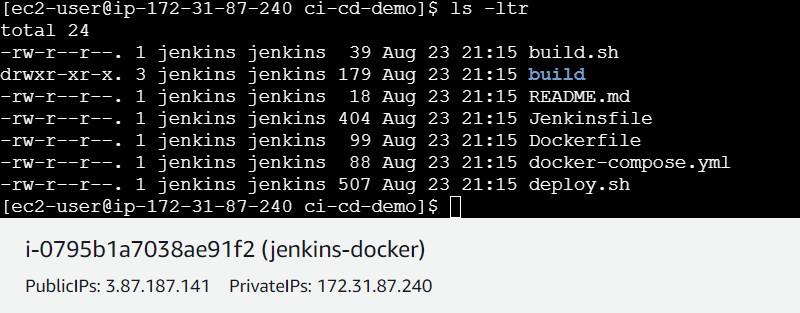








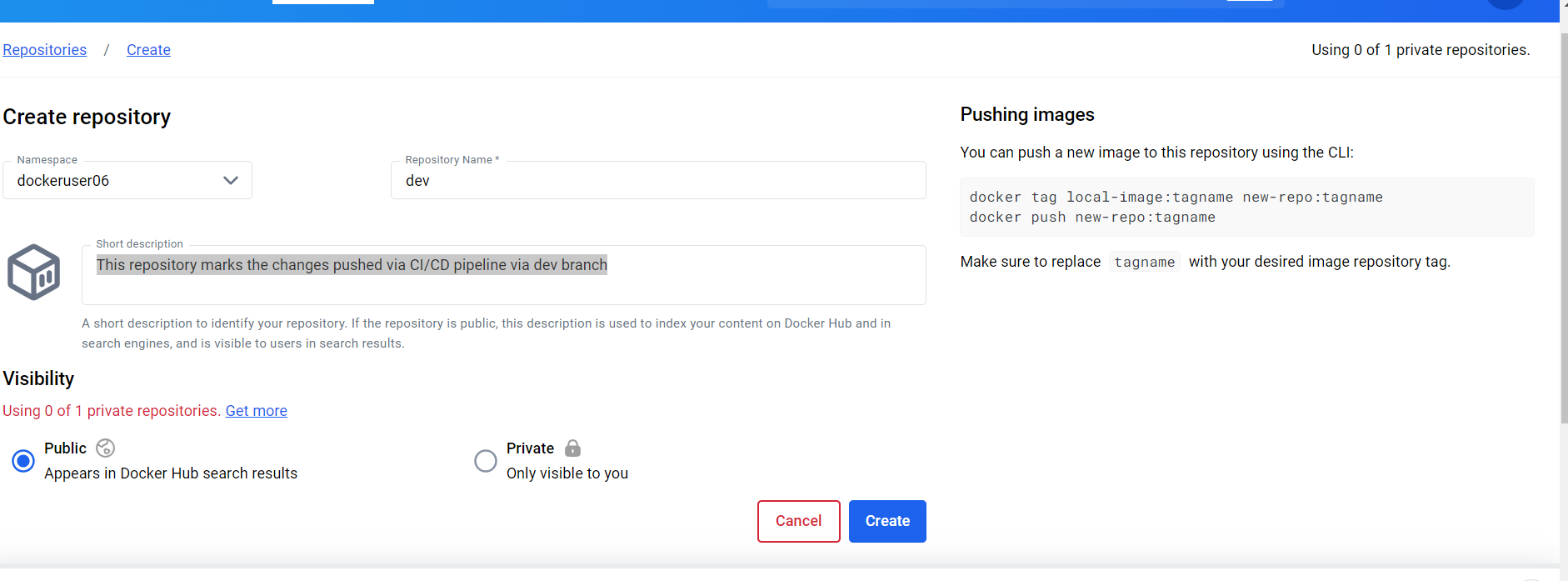


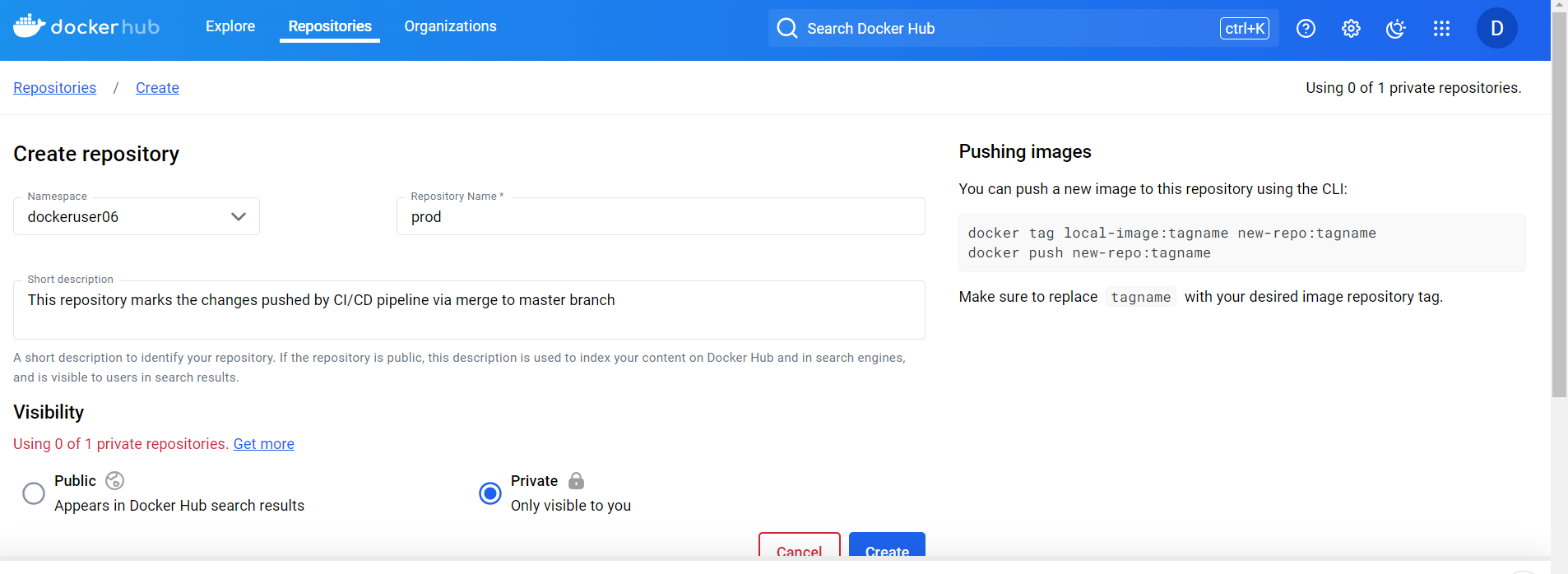


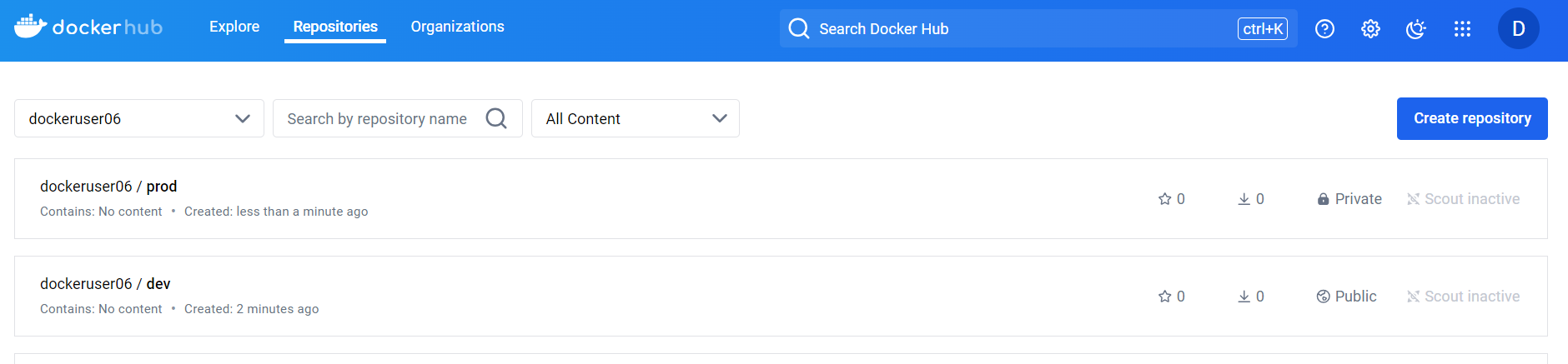
**Create Docker Hub repository:**

* Create 2 repos “dev” and “prod” to push images.

“prod” repo must be private and “dev” repo can be public.







**Build Step:**

As part of build step – first checkout the code from repository.

1. Checkout the react project from GitHub
2. Build docker image using docker build
3. Login to docker hub
4. Push the image to docker hub
5. Deploy the application to EC2 server

**Configure Jenkinsfile:**

In your GitHub repository, create a file named "Jenkinsfile" (without quotes) in the root directory. This file defines your pipeline.

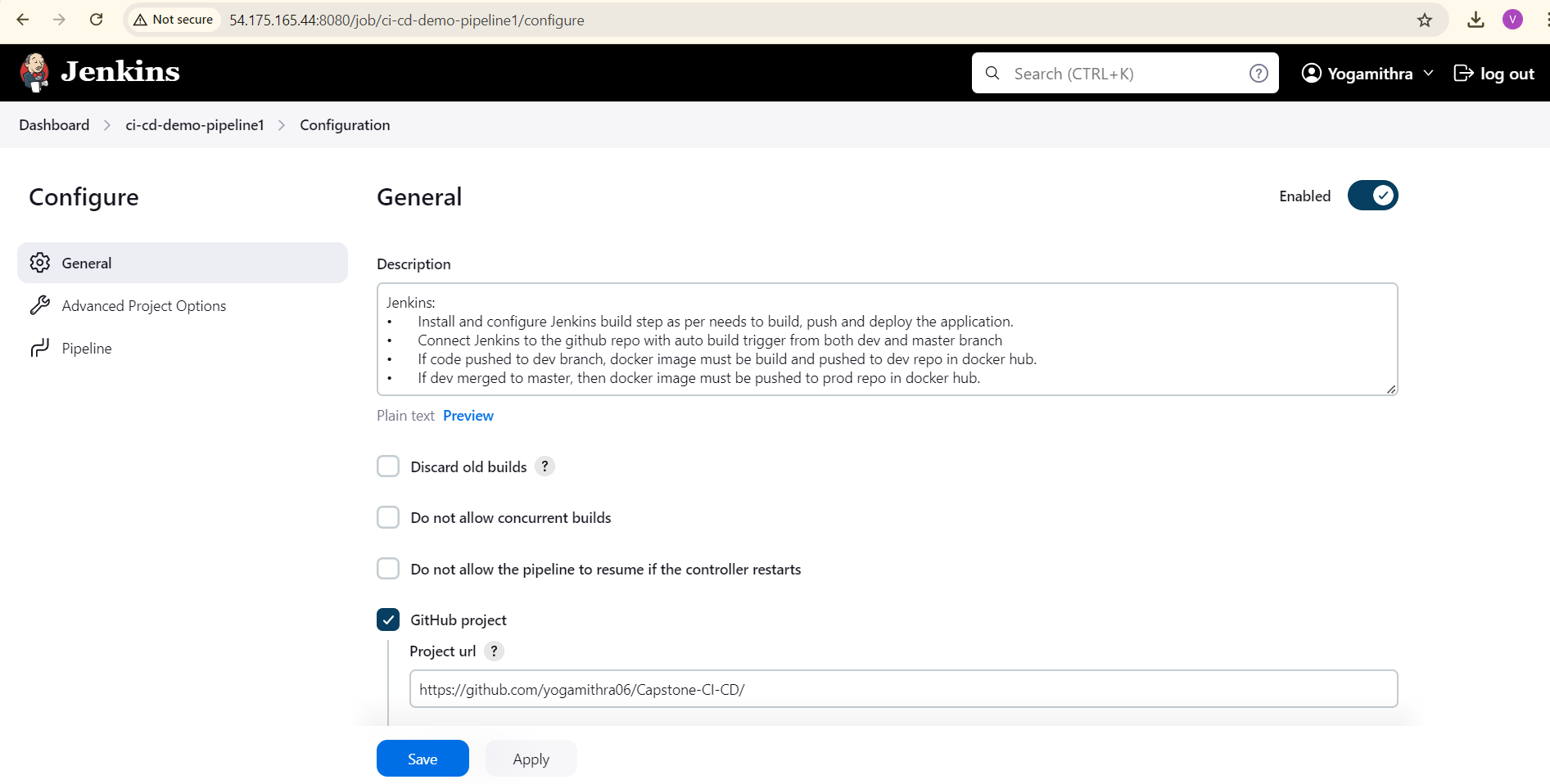
This Jenkinsfile defines 4 stages:

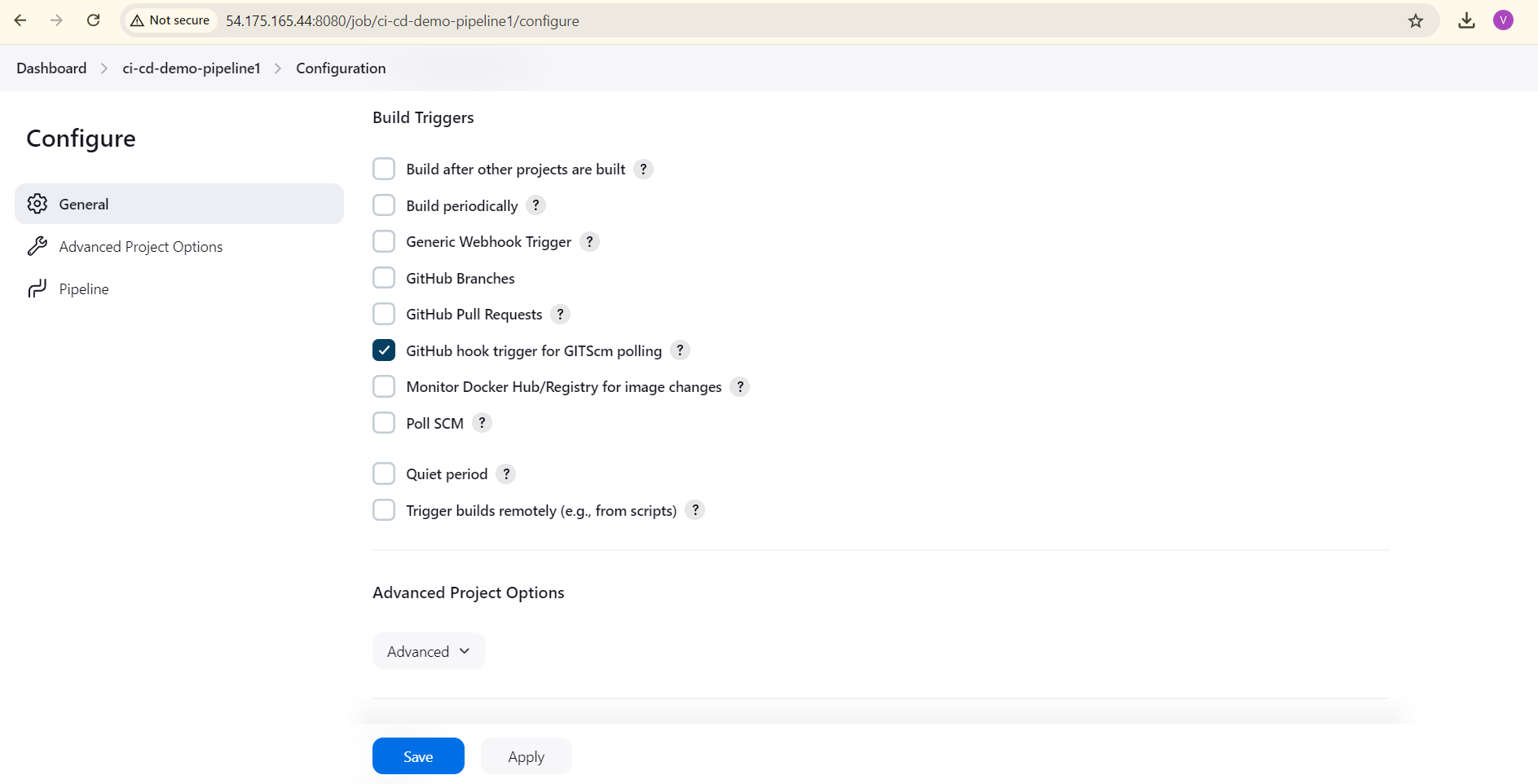
1. Checkout Code,
2. Build Docker image,
3. Push Docker Image and
4. Deploy to EC2.

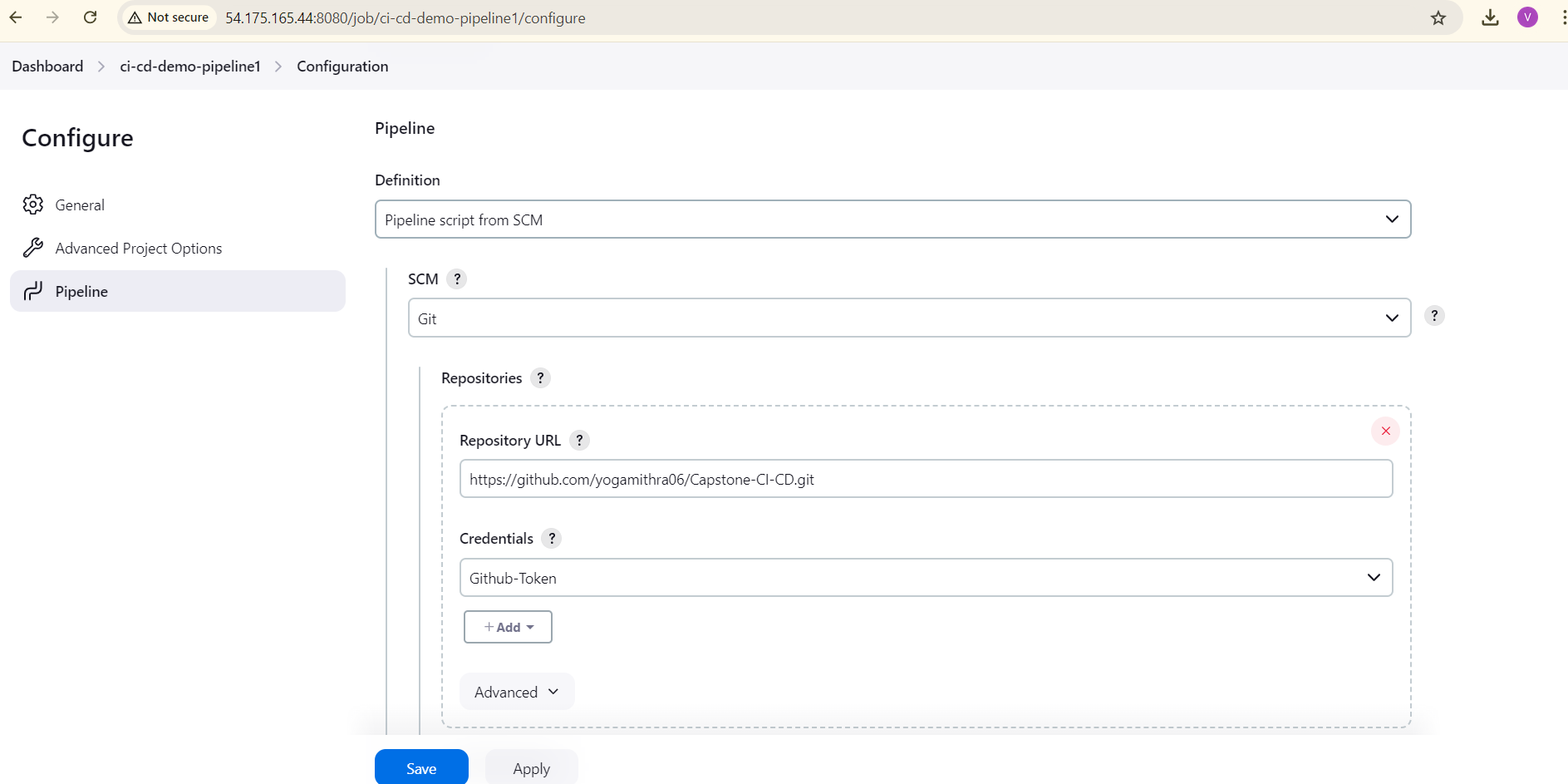
It checks out the code from the repository, builds the dockerized image, pushes the image to the docker registry and finally deploys the container application to EC2 server.

****

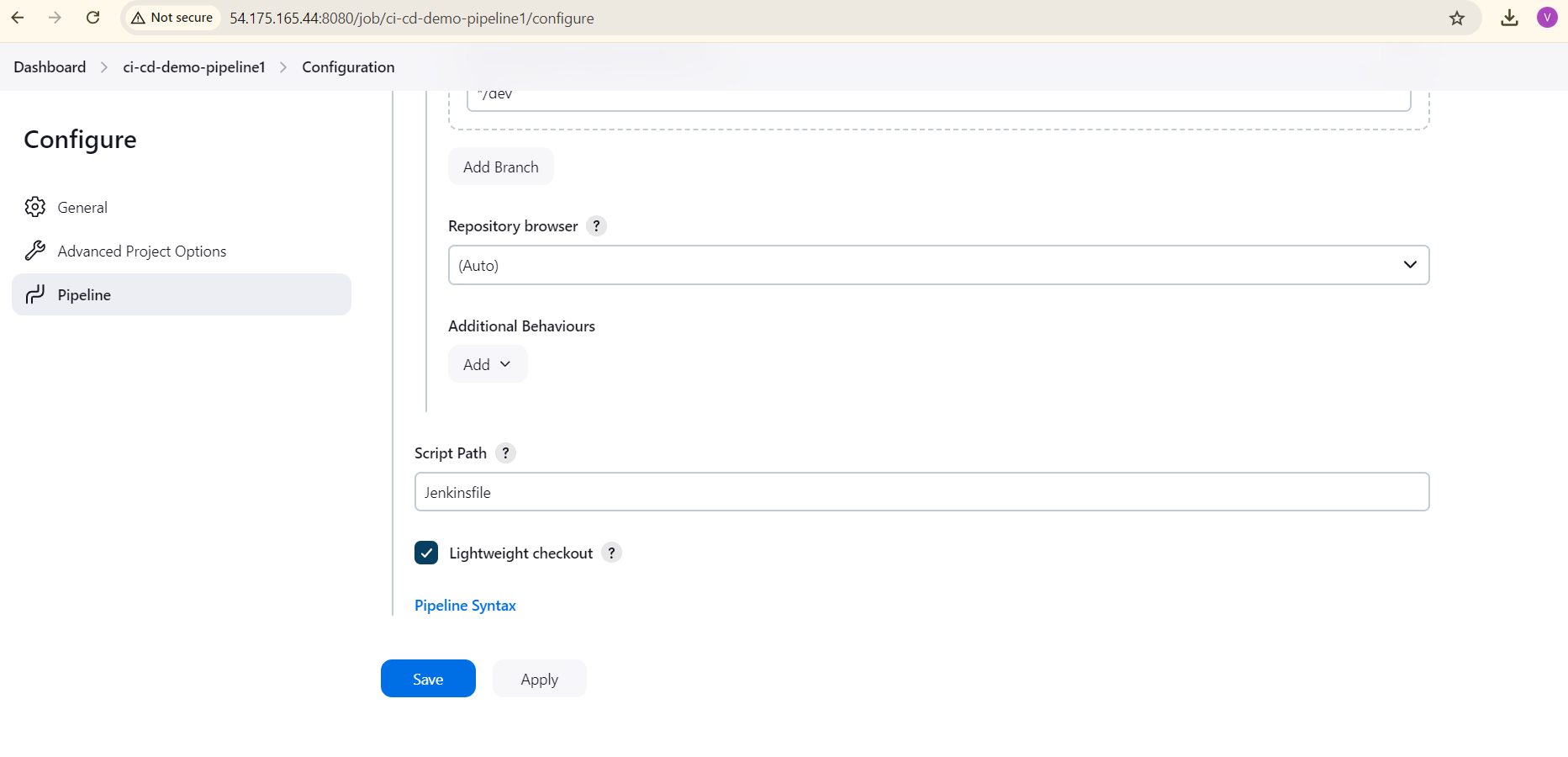
**Jenkins job configuration:**

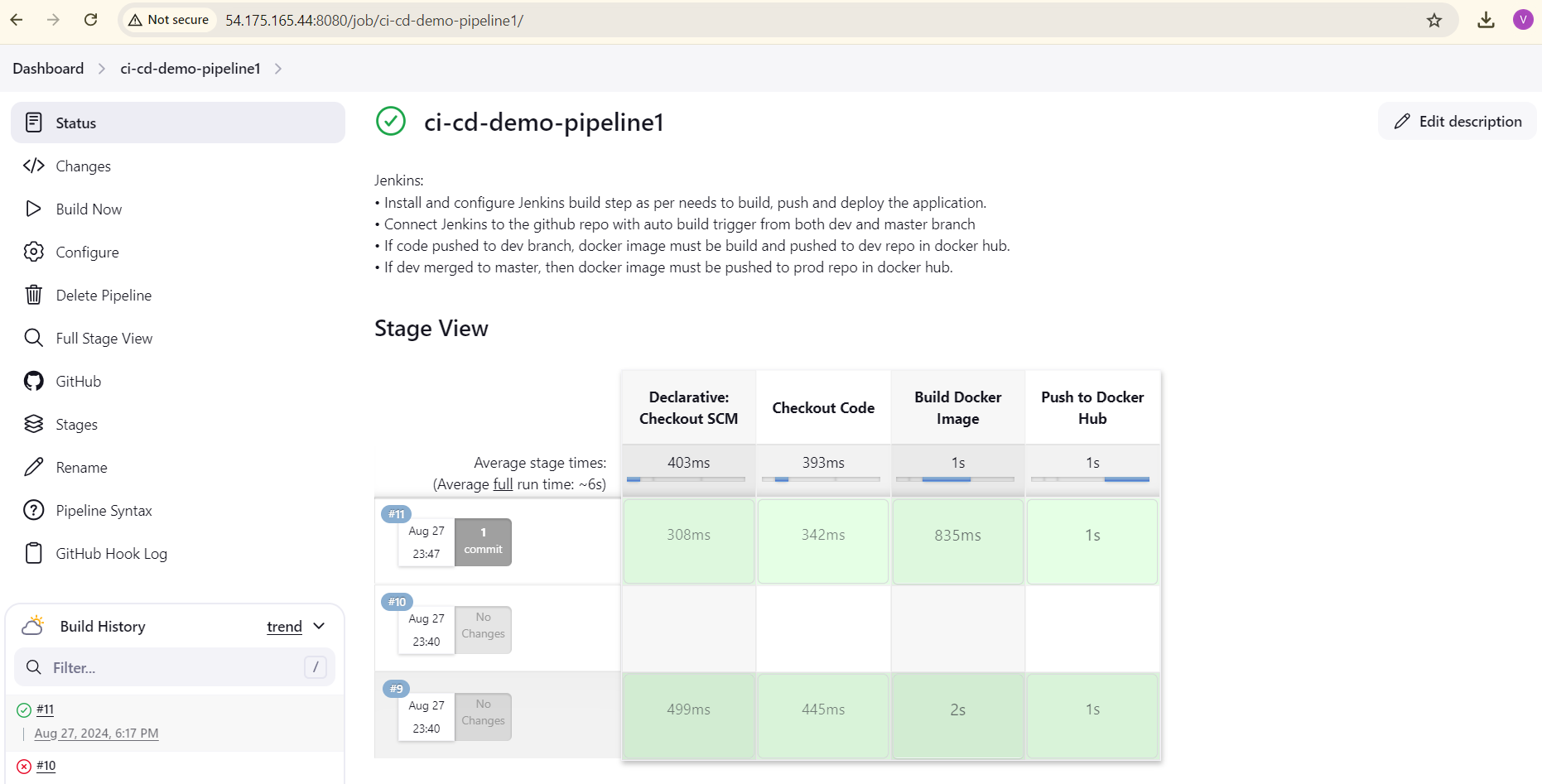


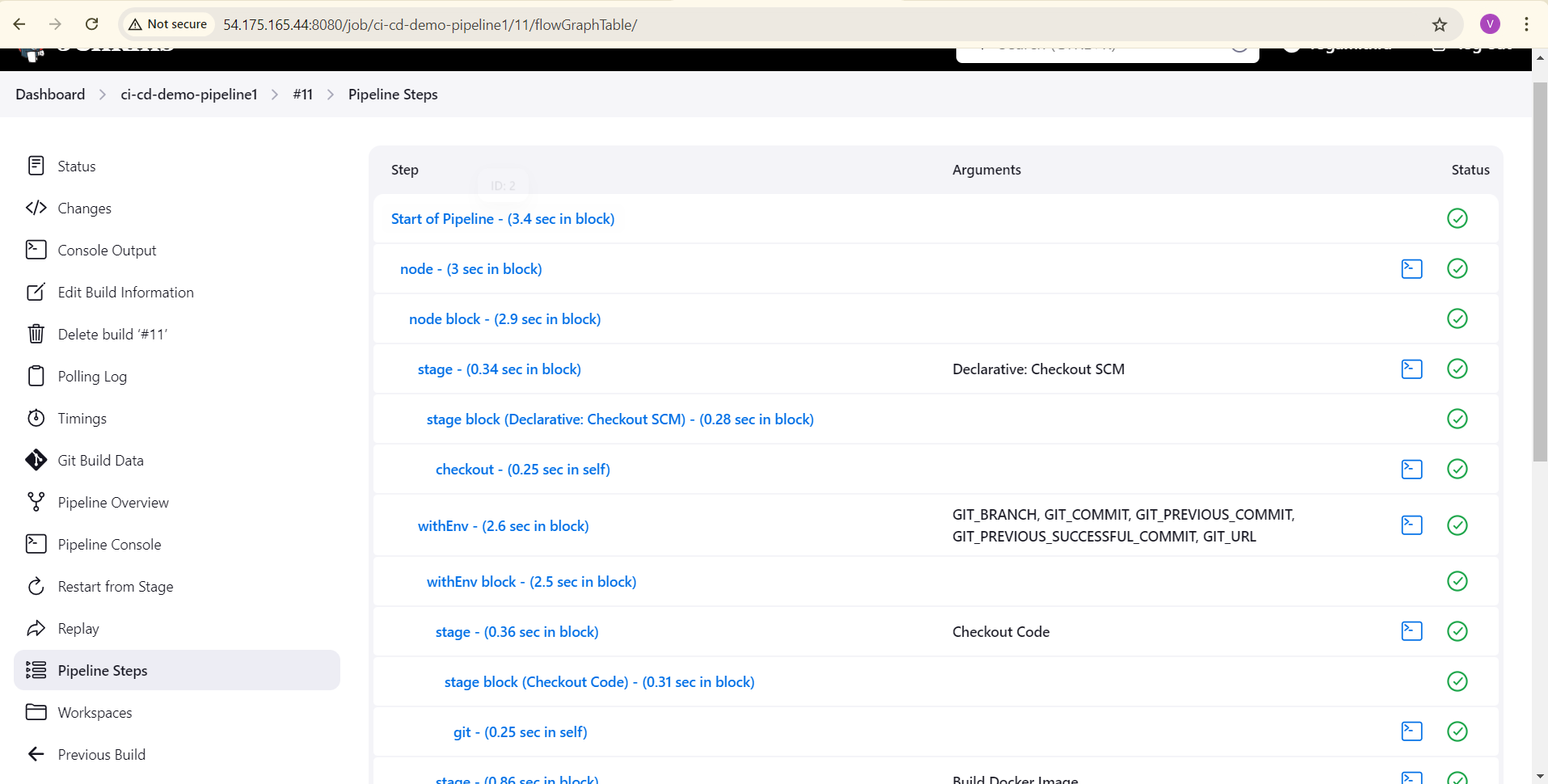


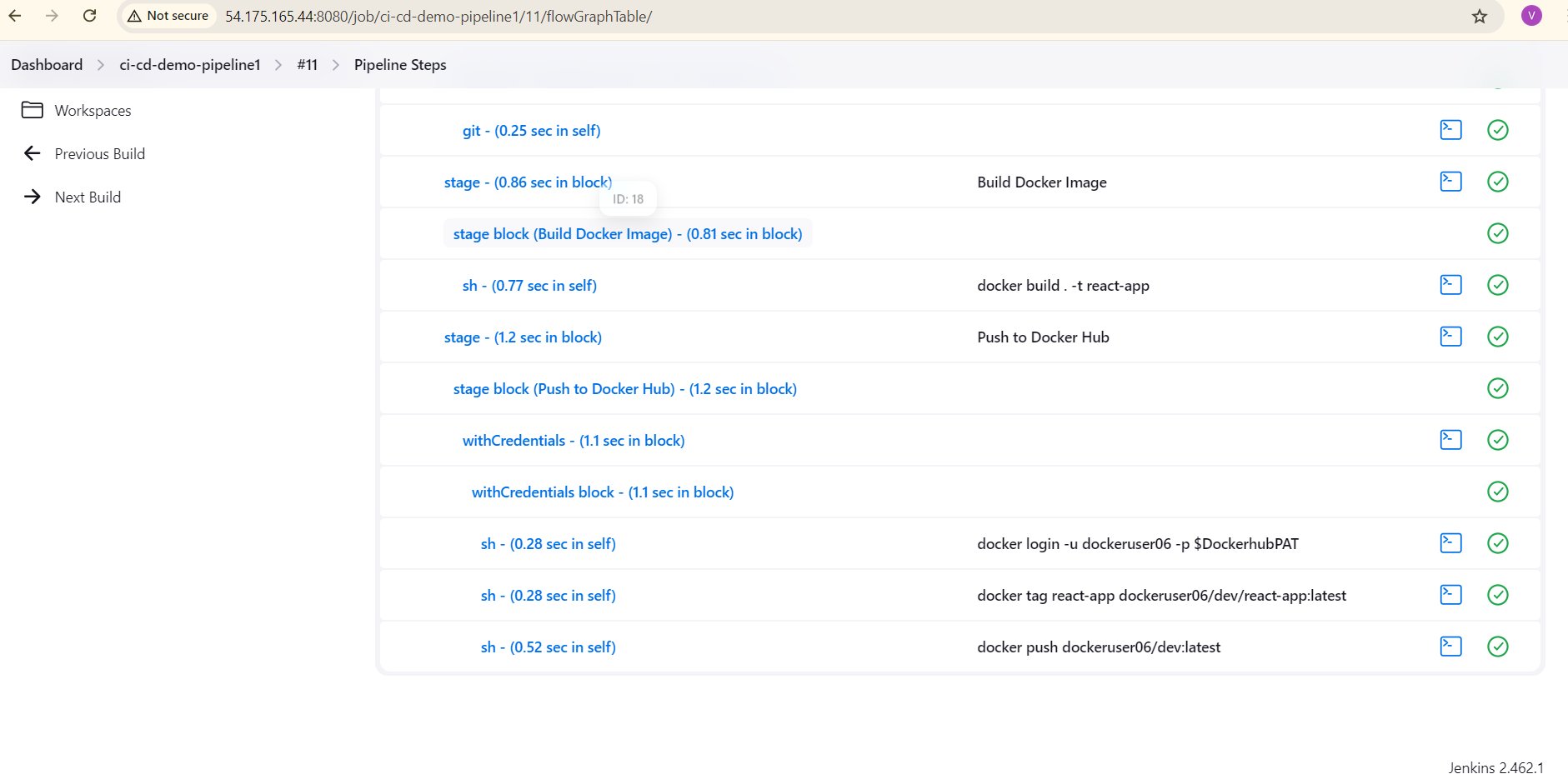












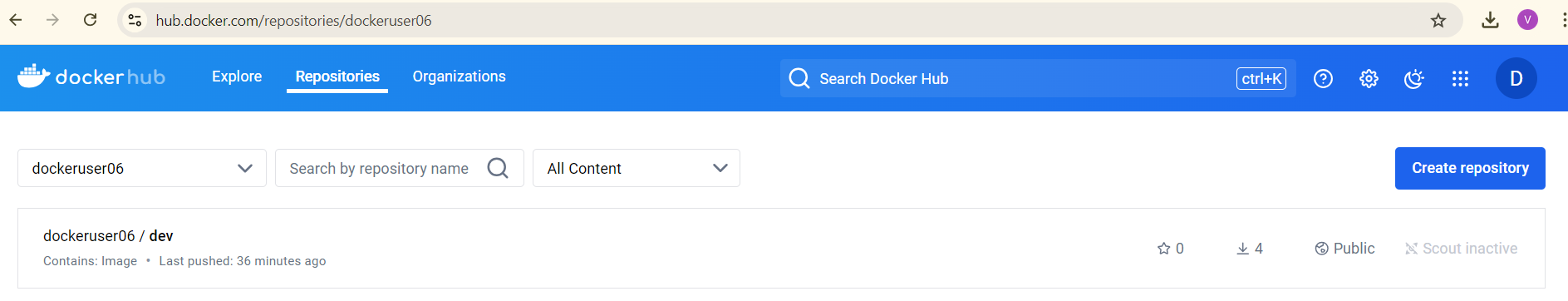
Console Output:

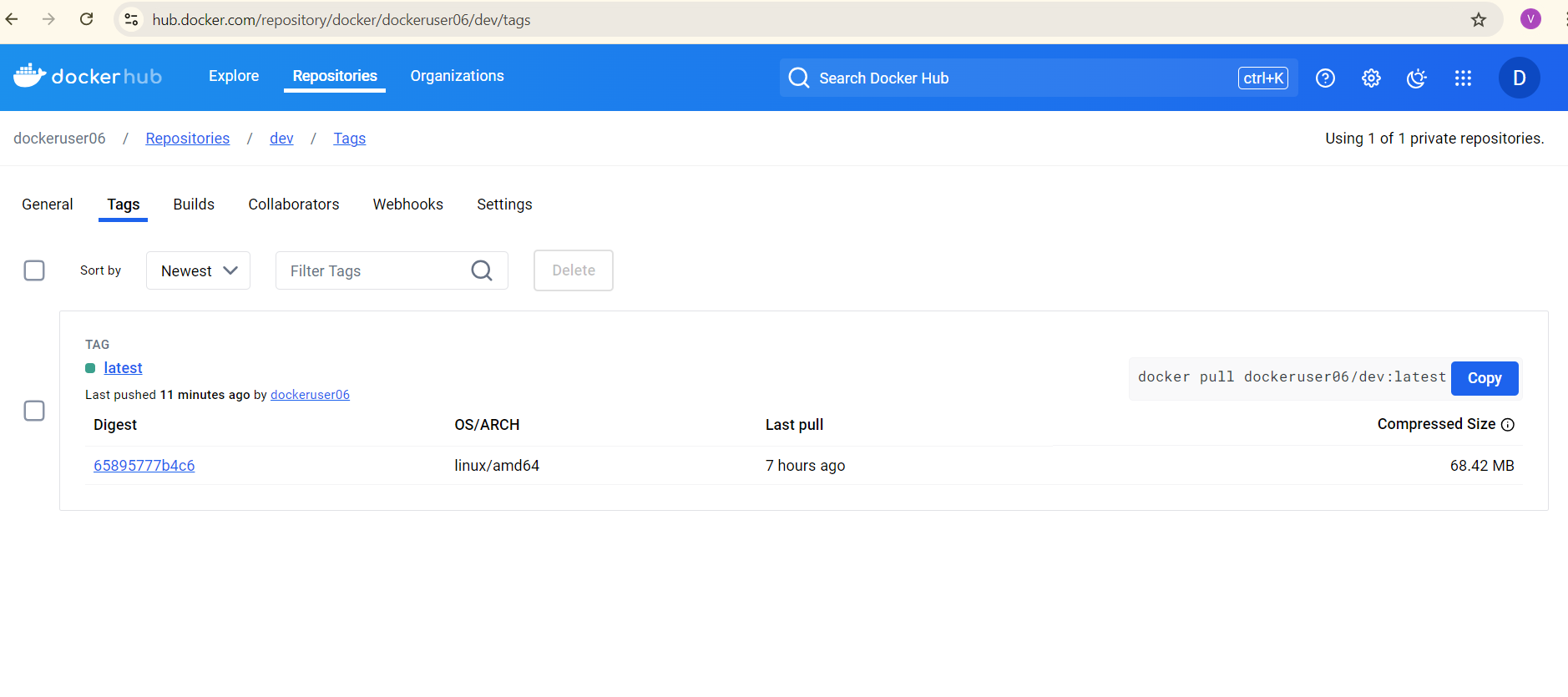


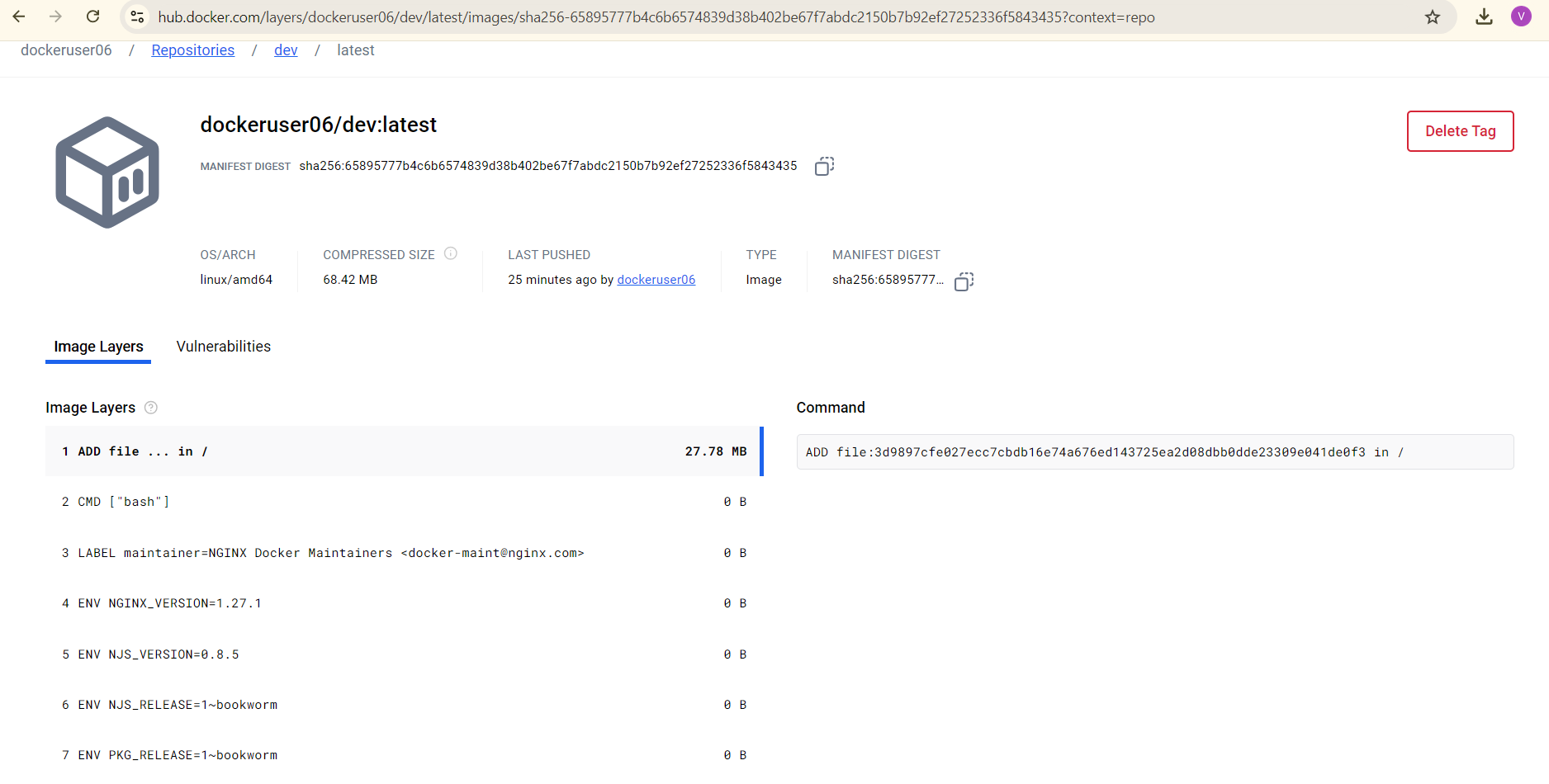
**Setting Up a Webhook in GitHub:**

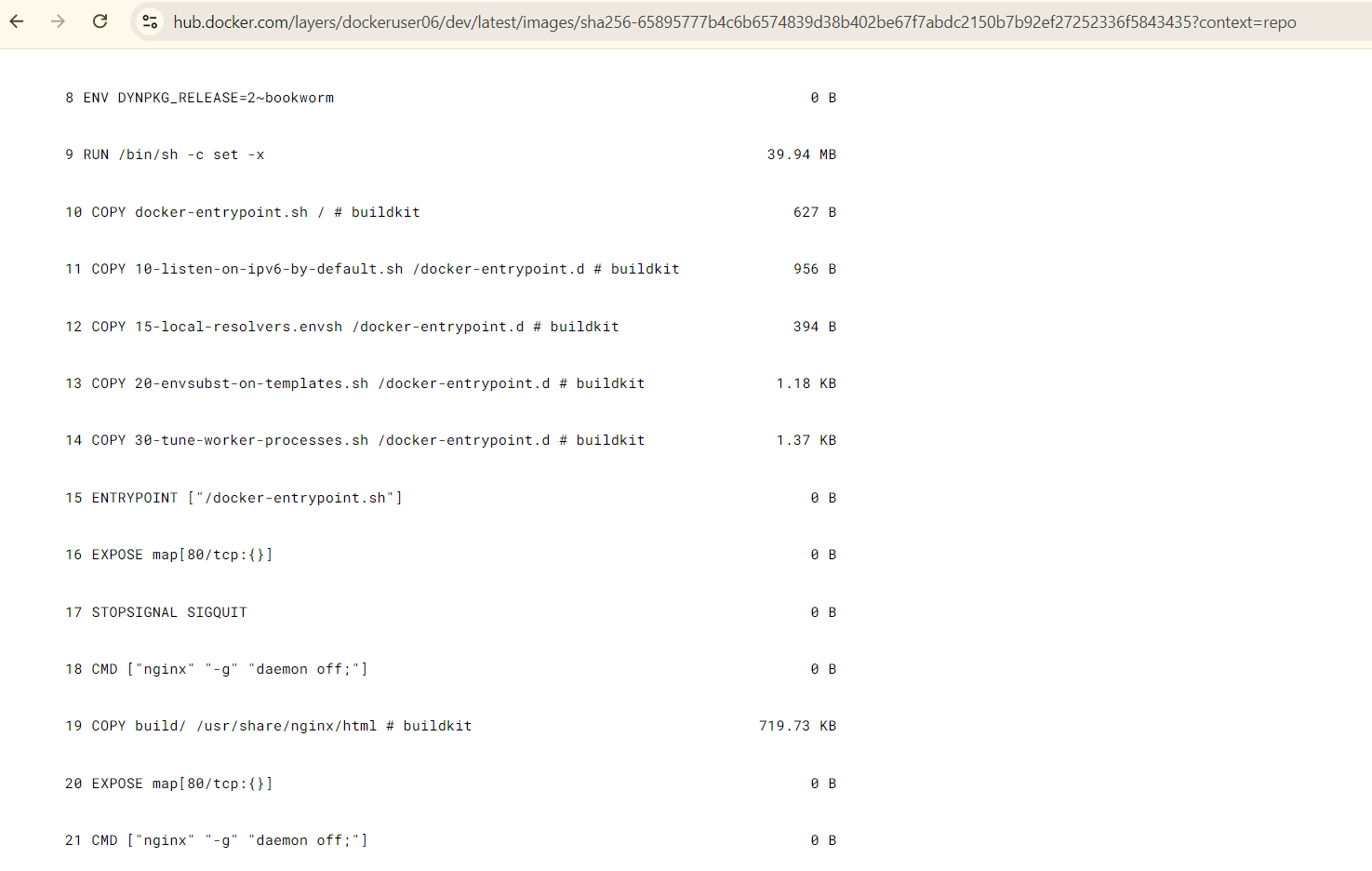
By setting up a webhook in GitHub, you can trigger automated actions in your Jenkins instance or other services when specific events occur in your repository.

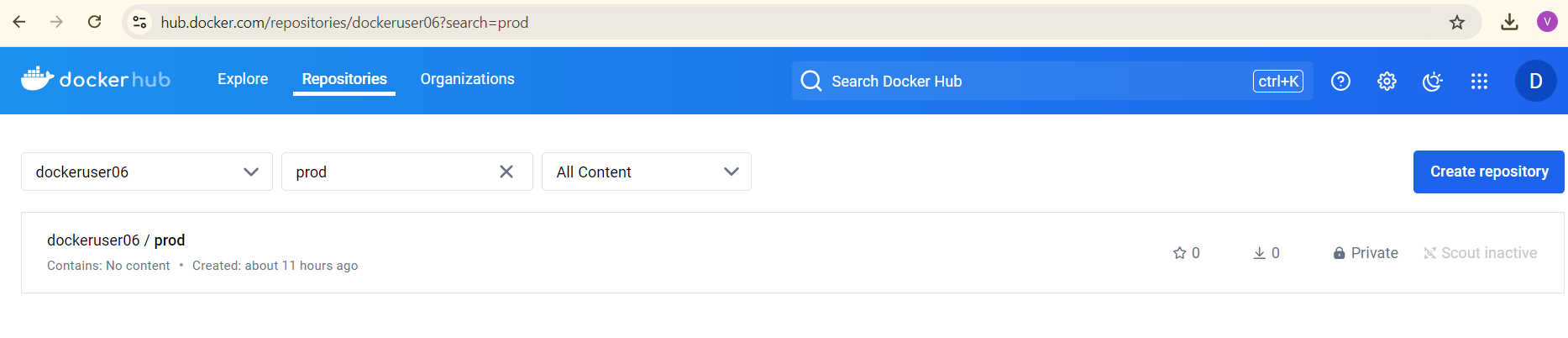




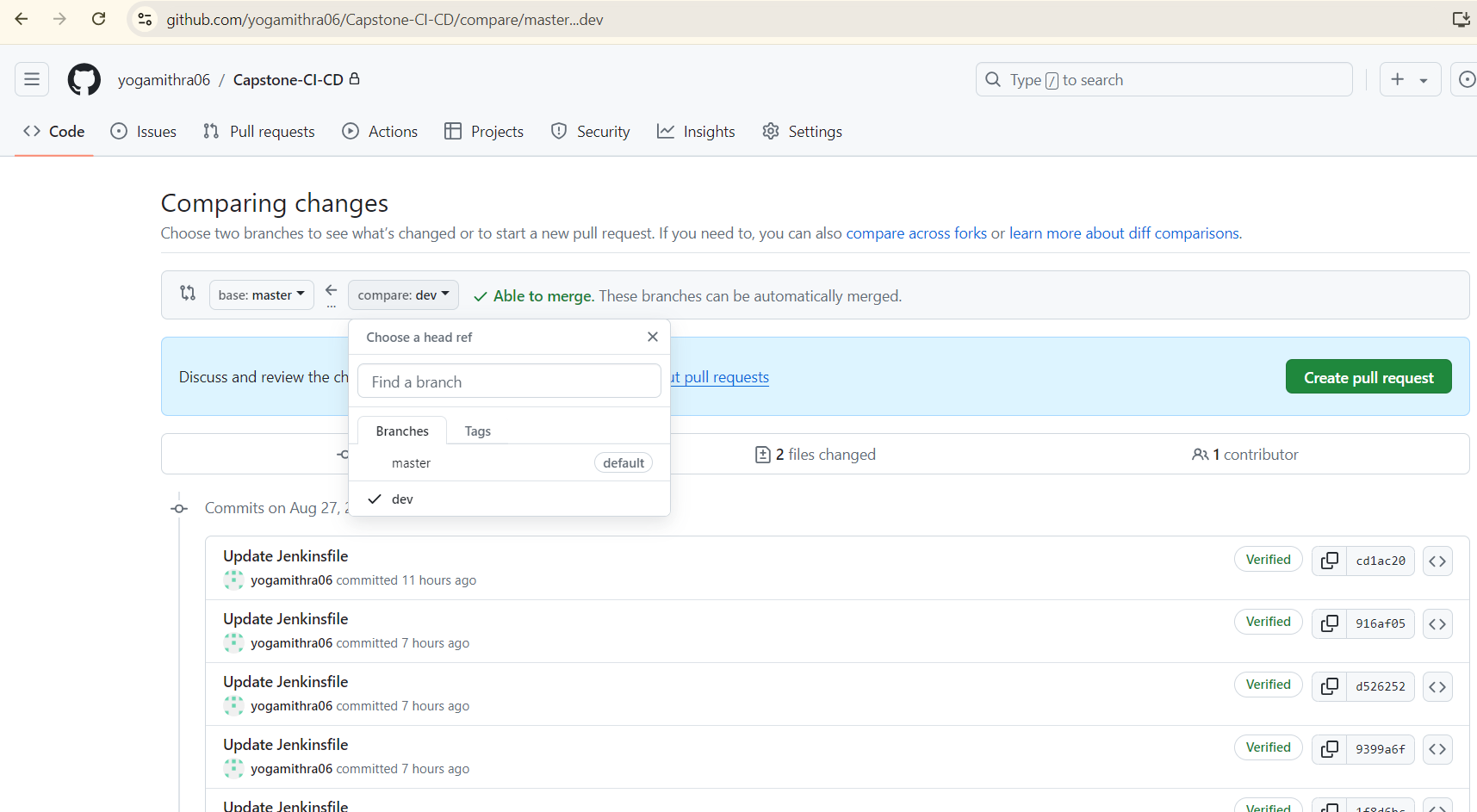


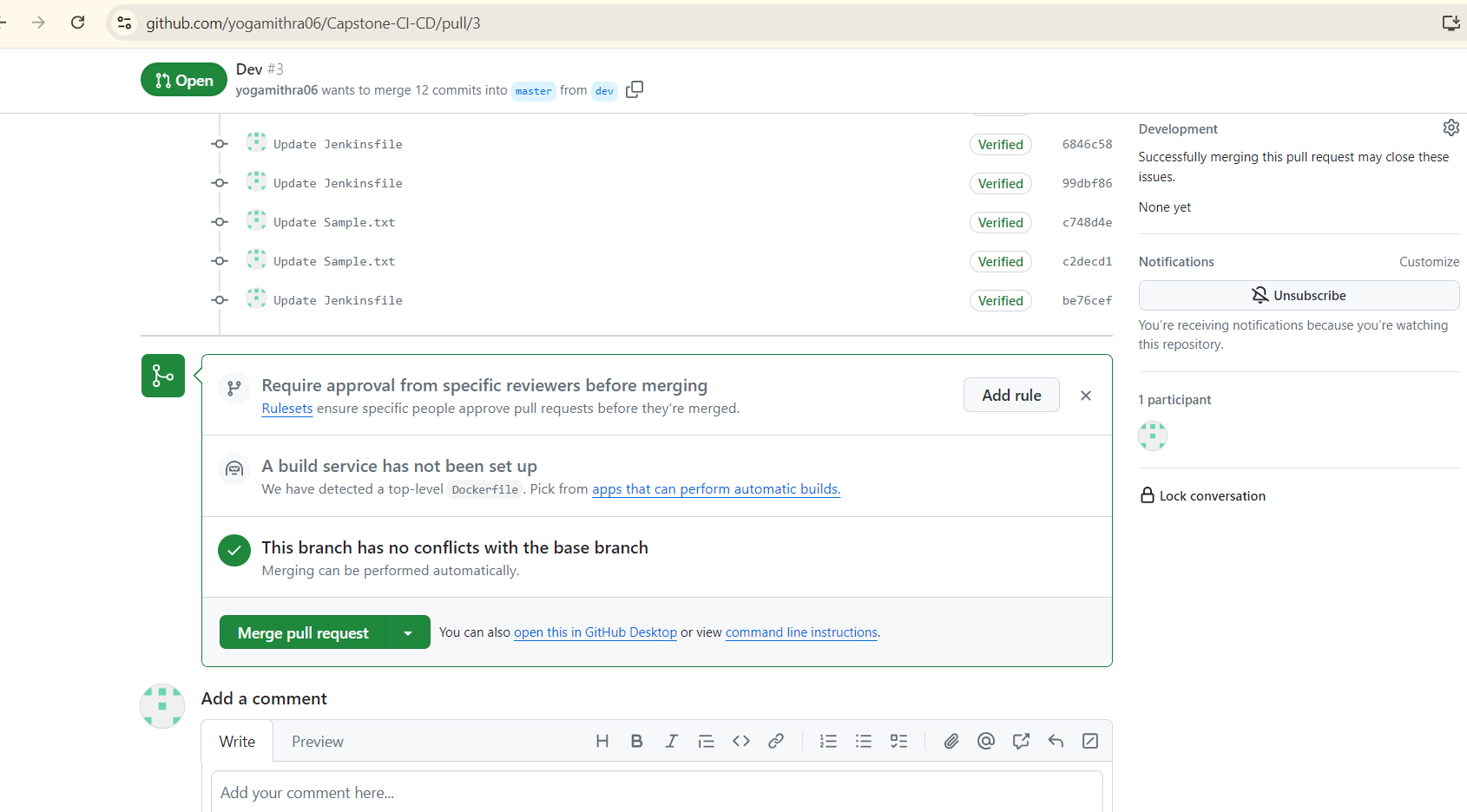


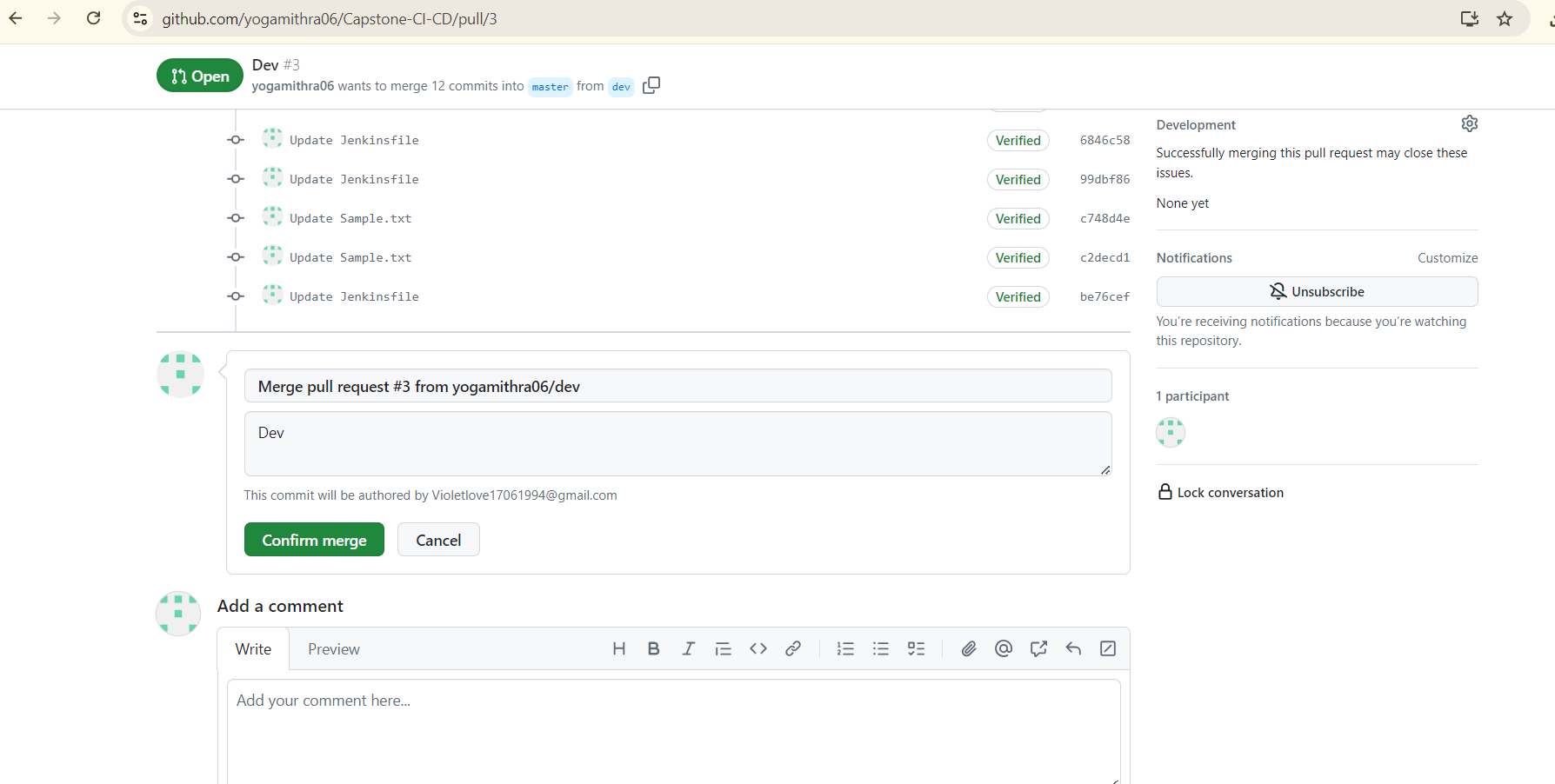


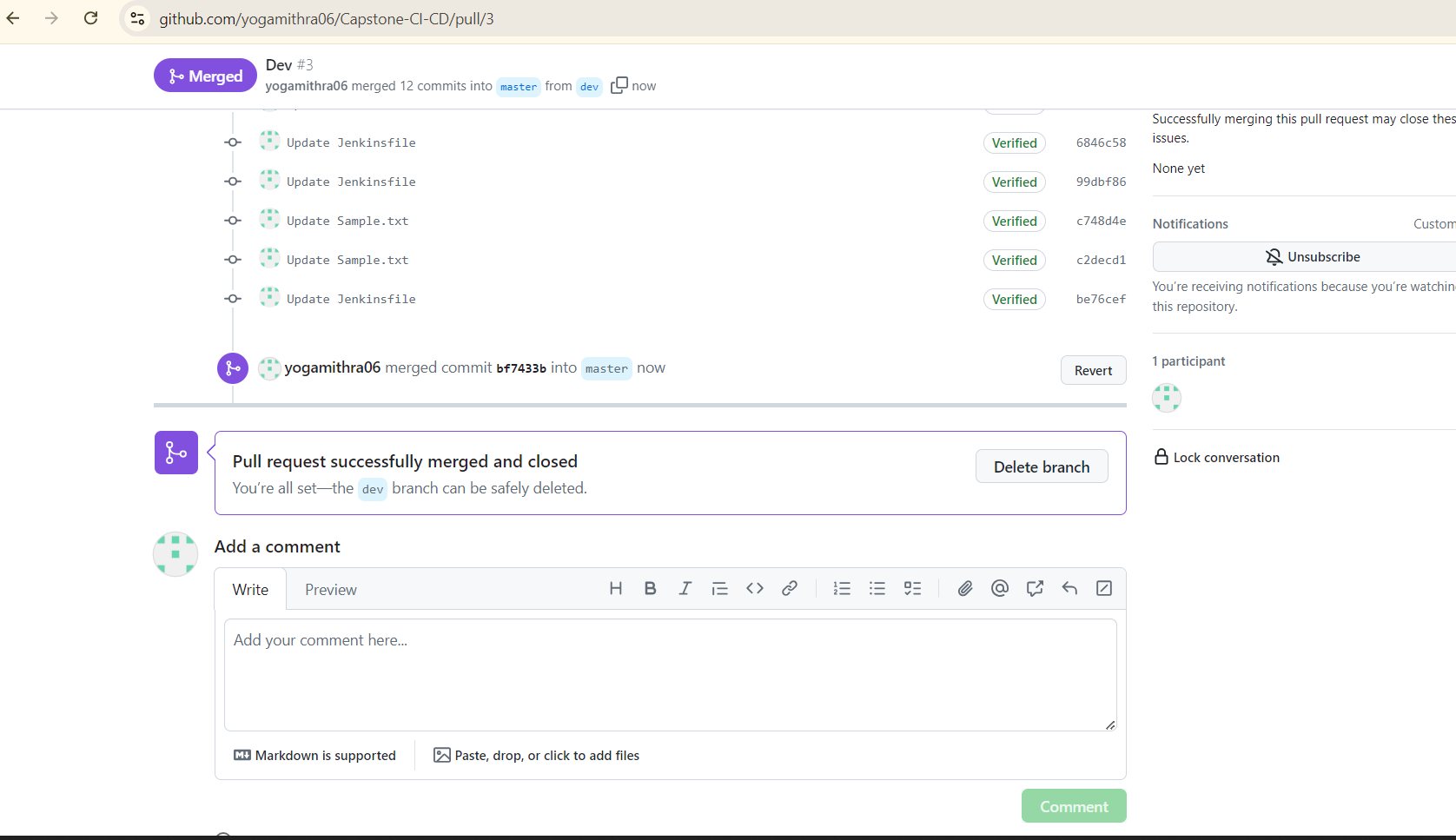






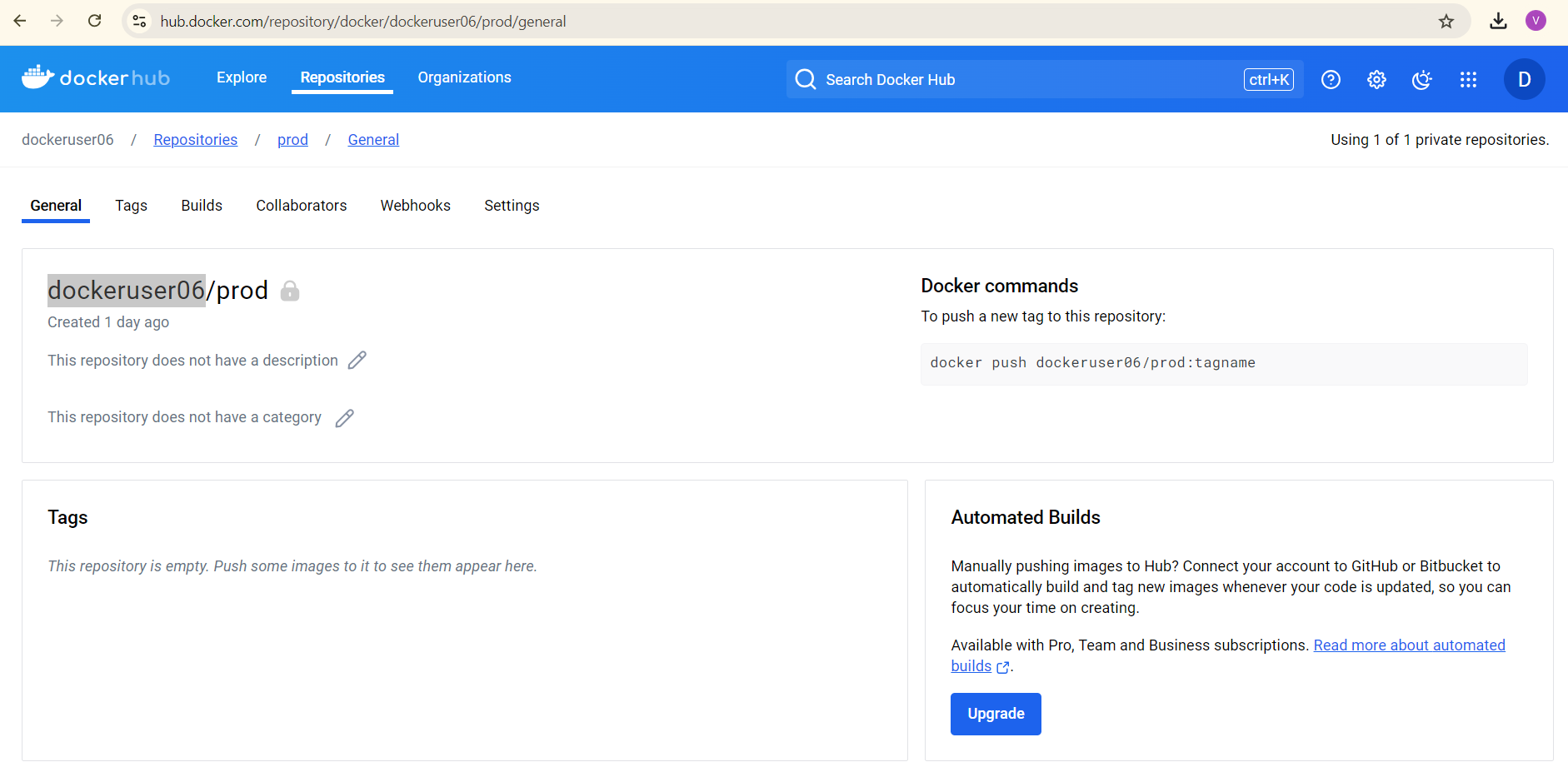


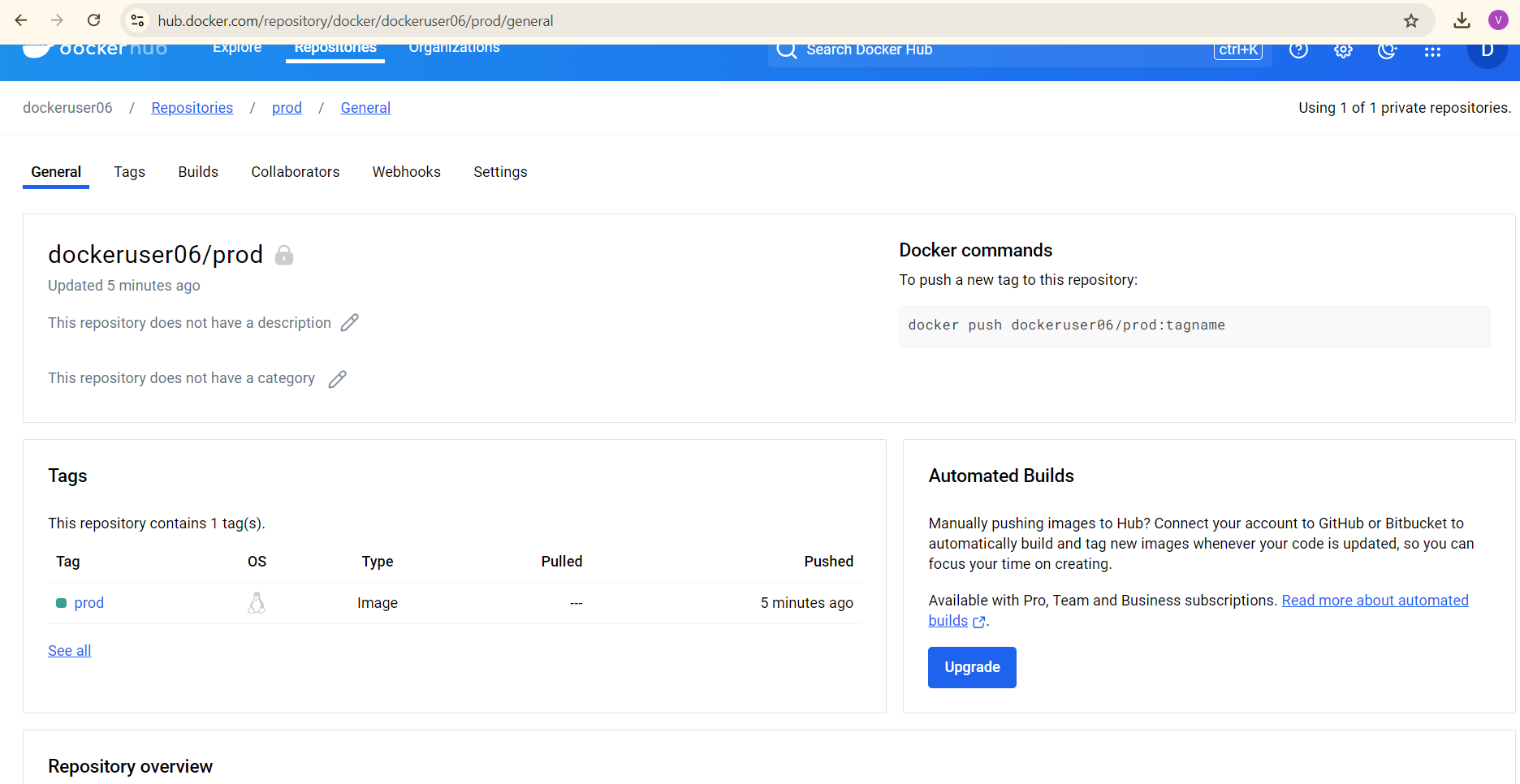


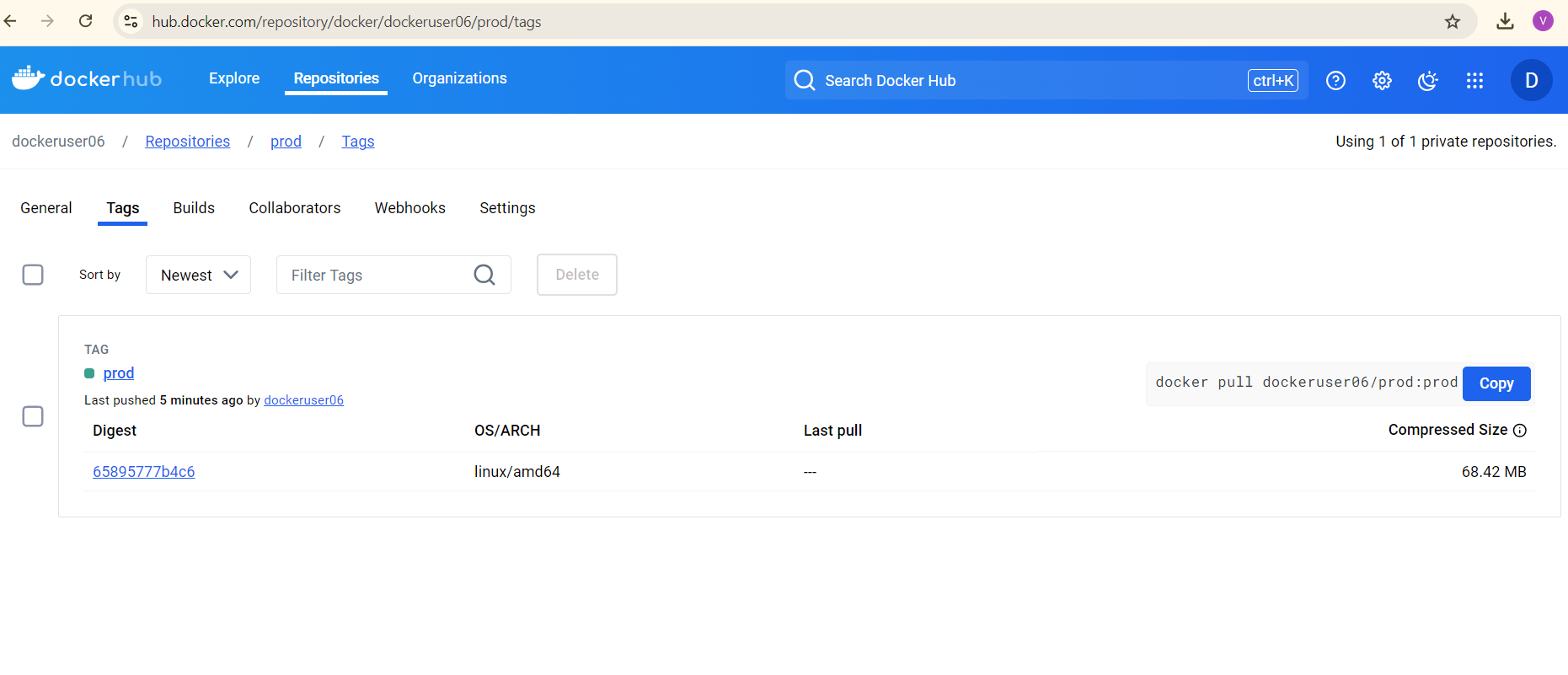


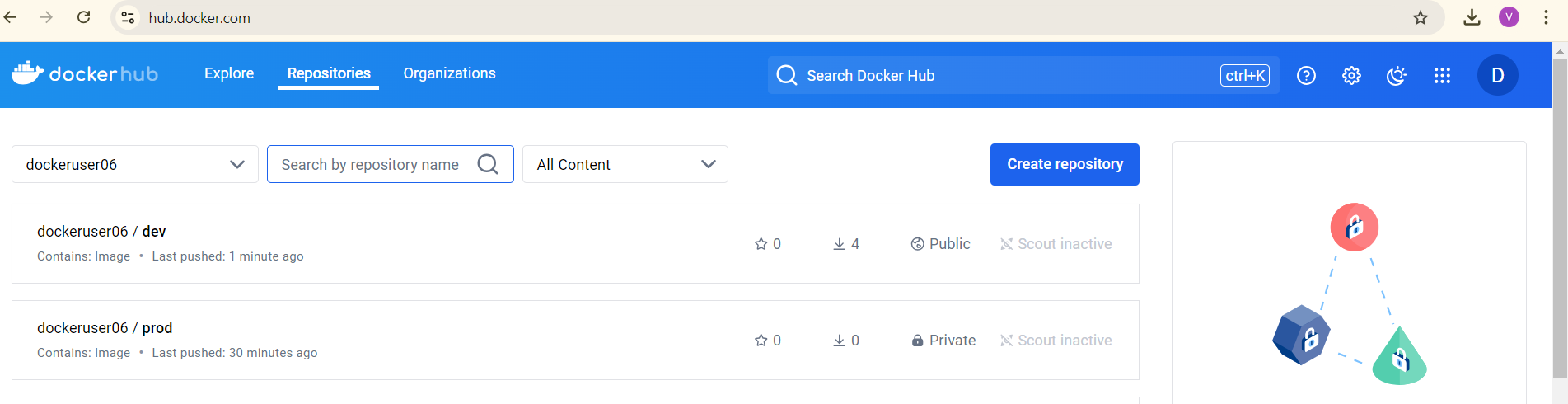










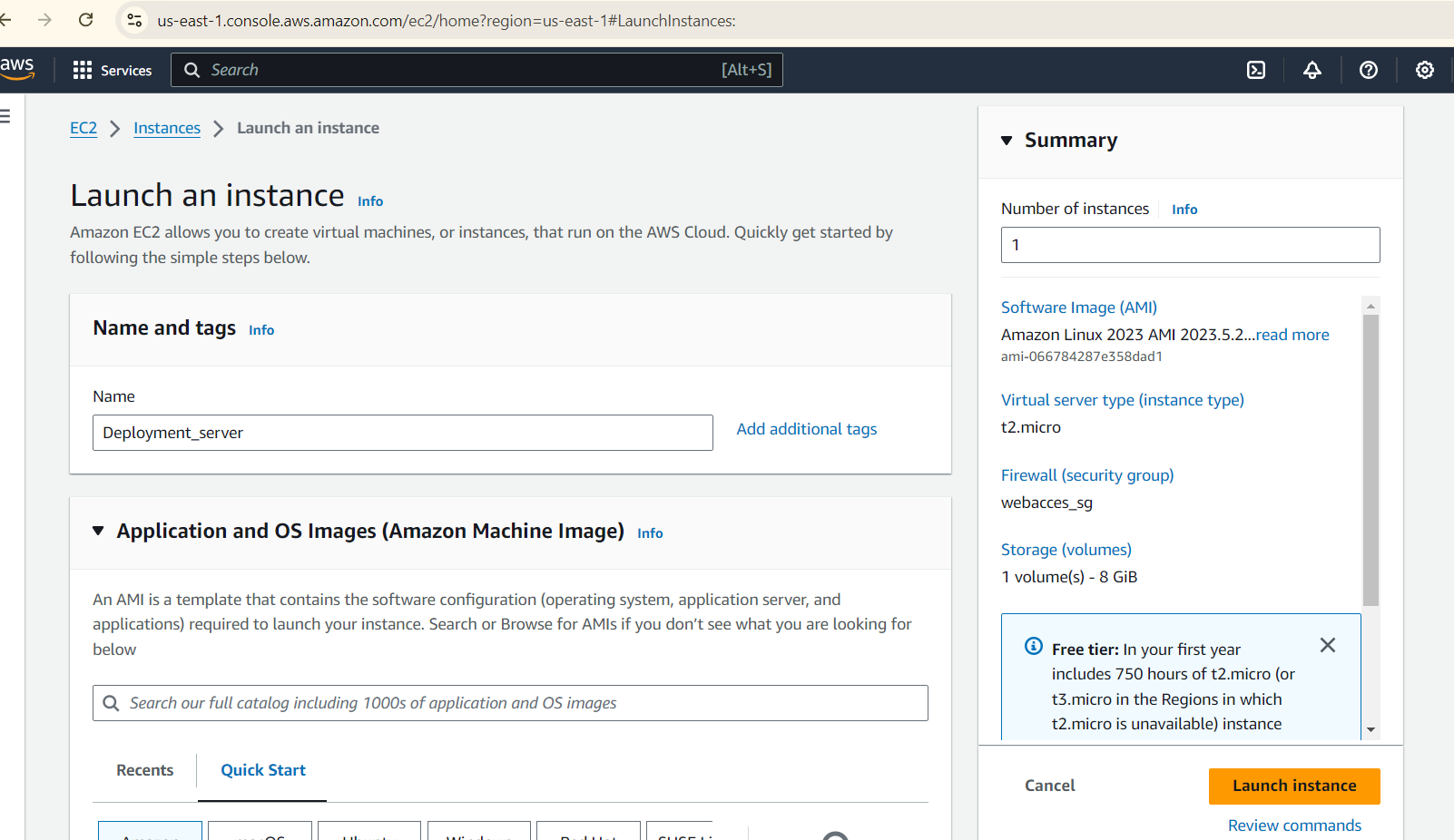




**Server 2 - Deploying the containerized application on AWS using Jenkins**

Launch another EC2 instance for your application (Server 2) Create a new key pair and store it in a secure and accessible location on your computer.

Name: Deployment\_server



**Install Docker. In your instance, run the following commands:**

sudo yum update -y

sudo yum install -y docker

**Start the Docker service:**

sudo service docker start

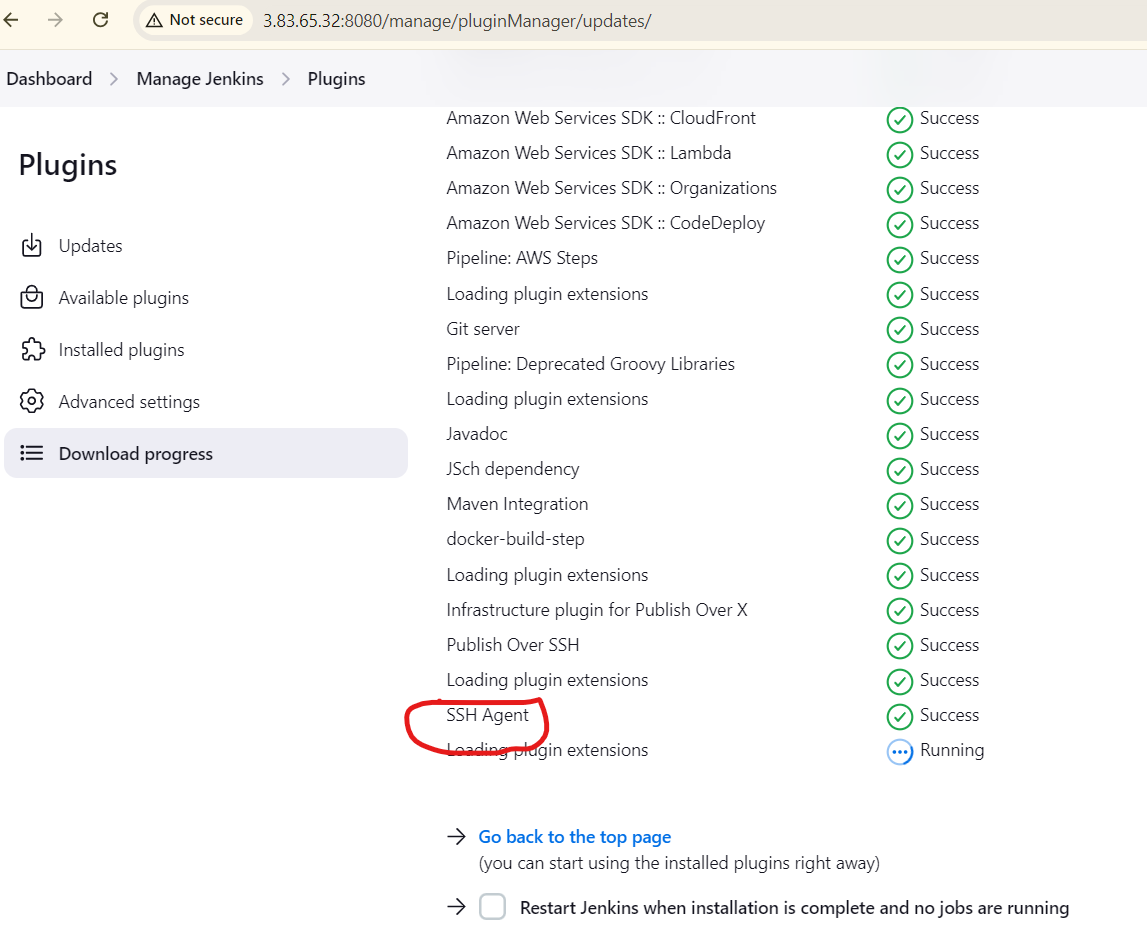
Add the ec2-user to the Docker group so you can execute Docker commands without using sudo:

sudo usermod -a -G docker ec2-user

Log out and log back in again to pick up the new Docker group permissions.

**Jenkins Configuration**

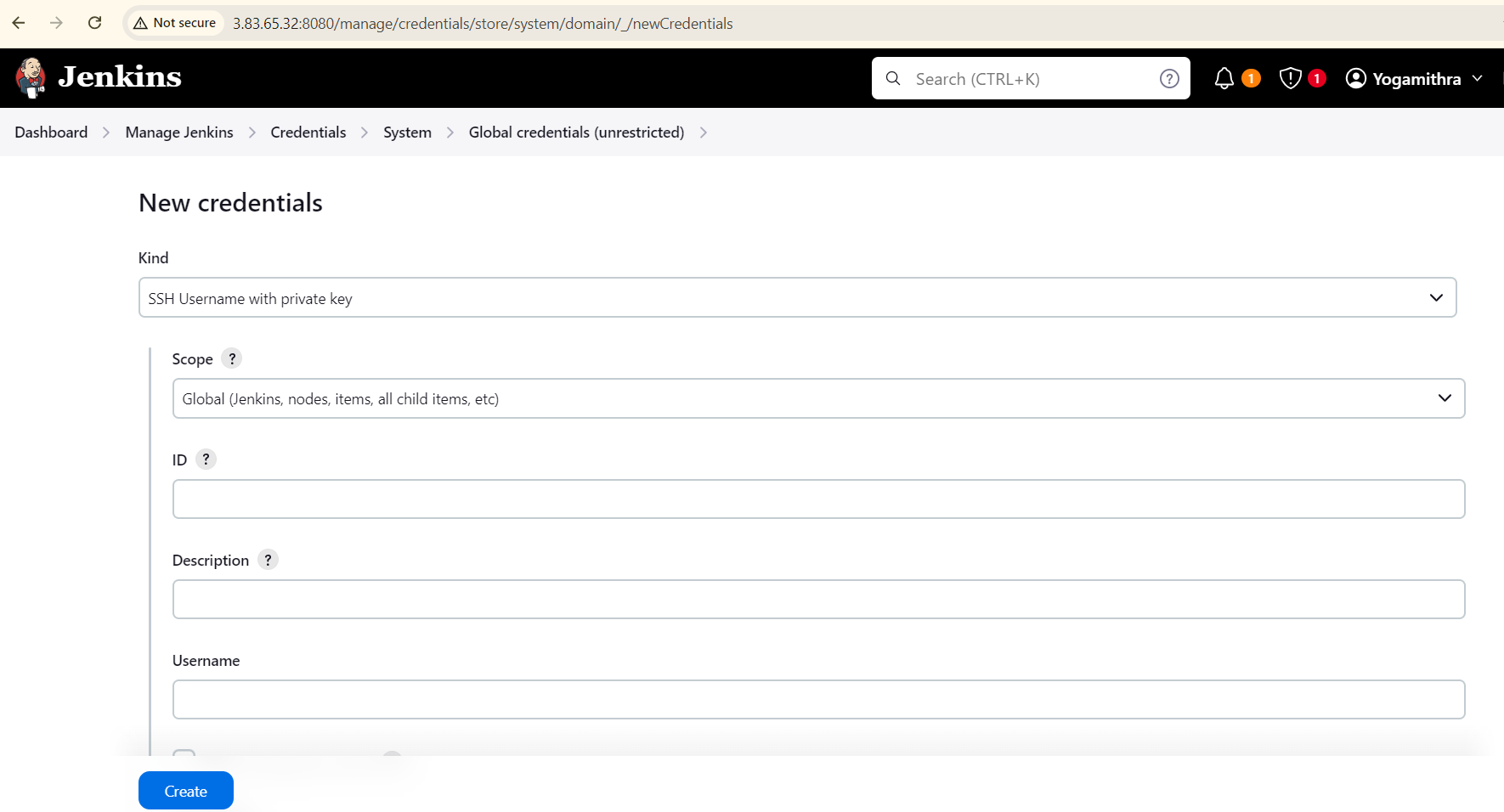
**Install SSH Agent plugin**



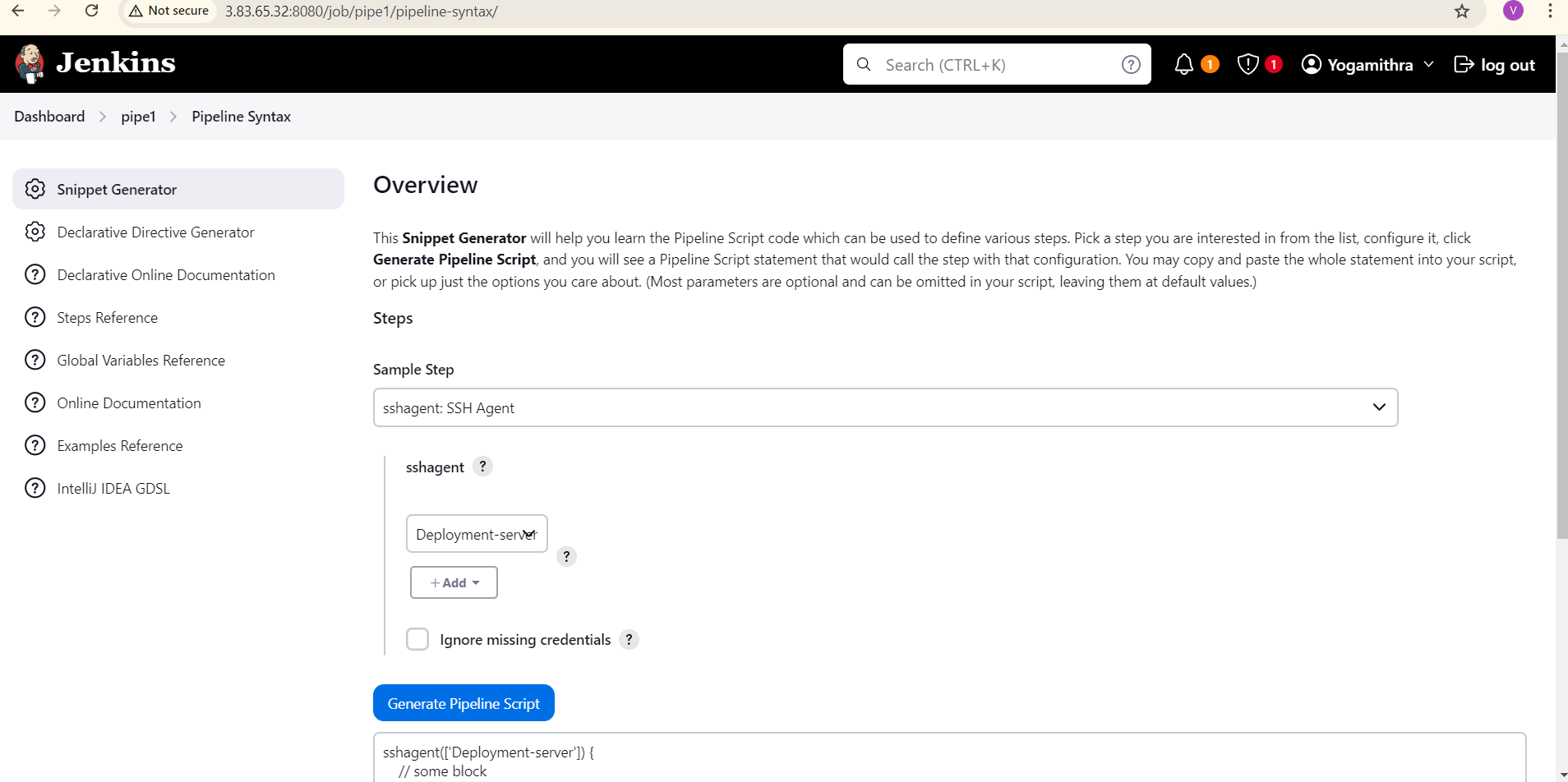
**Configure SSH agent in Jenkins:**

Inside credentials for multi-branch

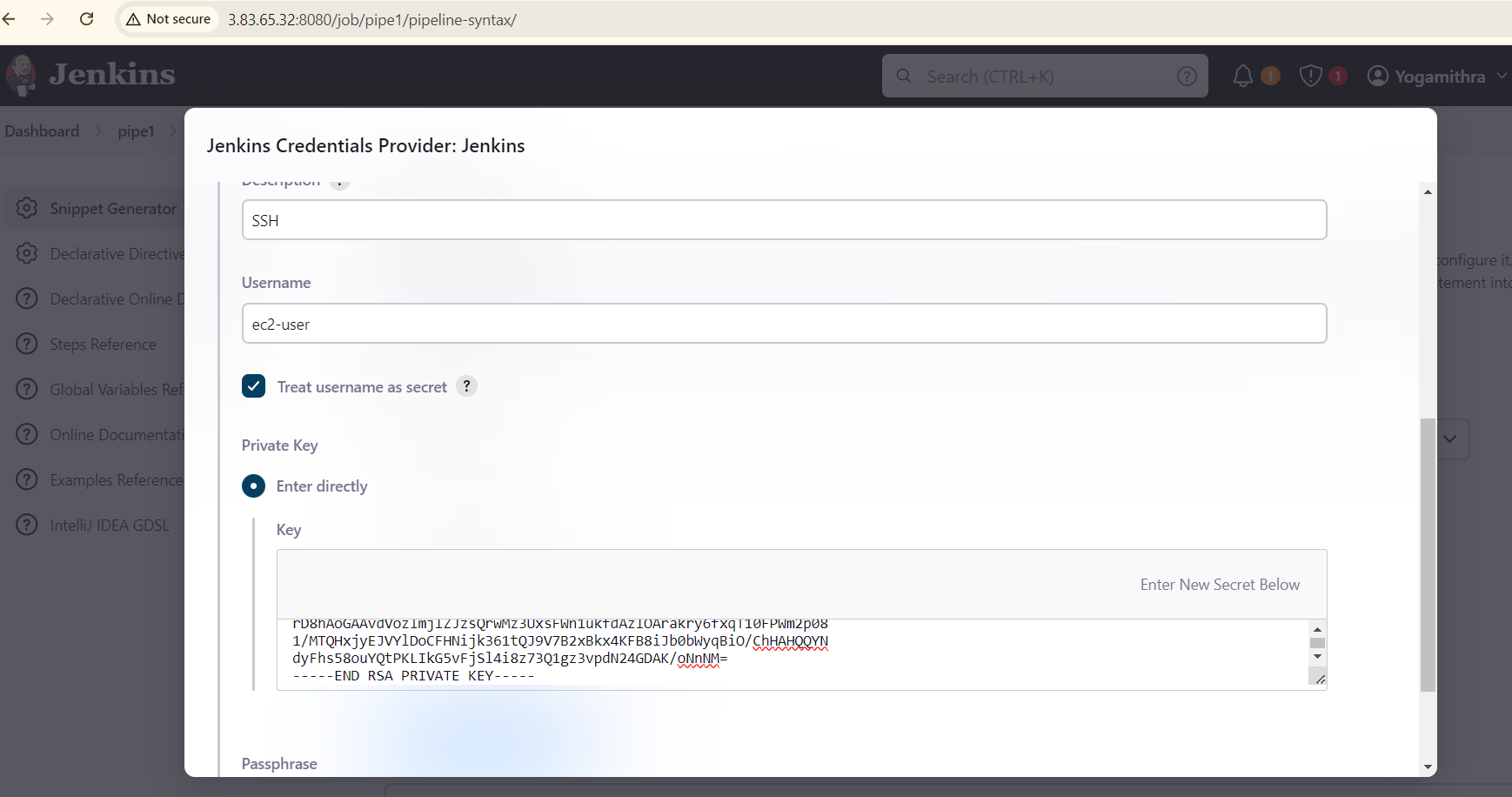
* Go to "Manage Jenkins" > "Manage Credentials" > "Global credentials" > "Add Credentials".
* Choose "SSH Username with private key".



* Give it an ID, enter the username (ec2-user), paste the private key downloaded earlier (for server 2).
* Add SSH Agent credentials to the Jenkins file:



Choose the 'sshagent: SSH Agent', choose 'ec2-user' and generate.



**Configure the Firewall on Server 2 (Deployment\_server)**  
Add an inbound rule for server 2 to allow SSH access from the Jenkins server.

1. "Whoever has the IP address can access the application":

- This means that anyone with the IP address of the EC2 instance can access the application running on it.

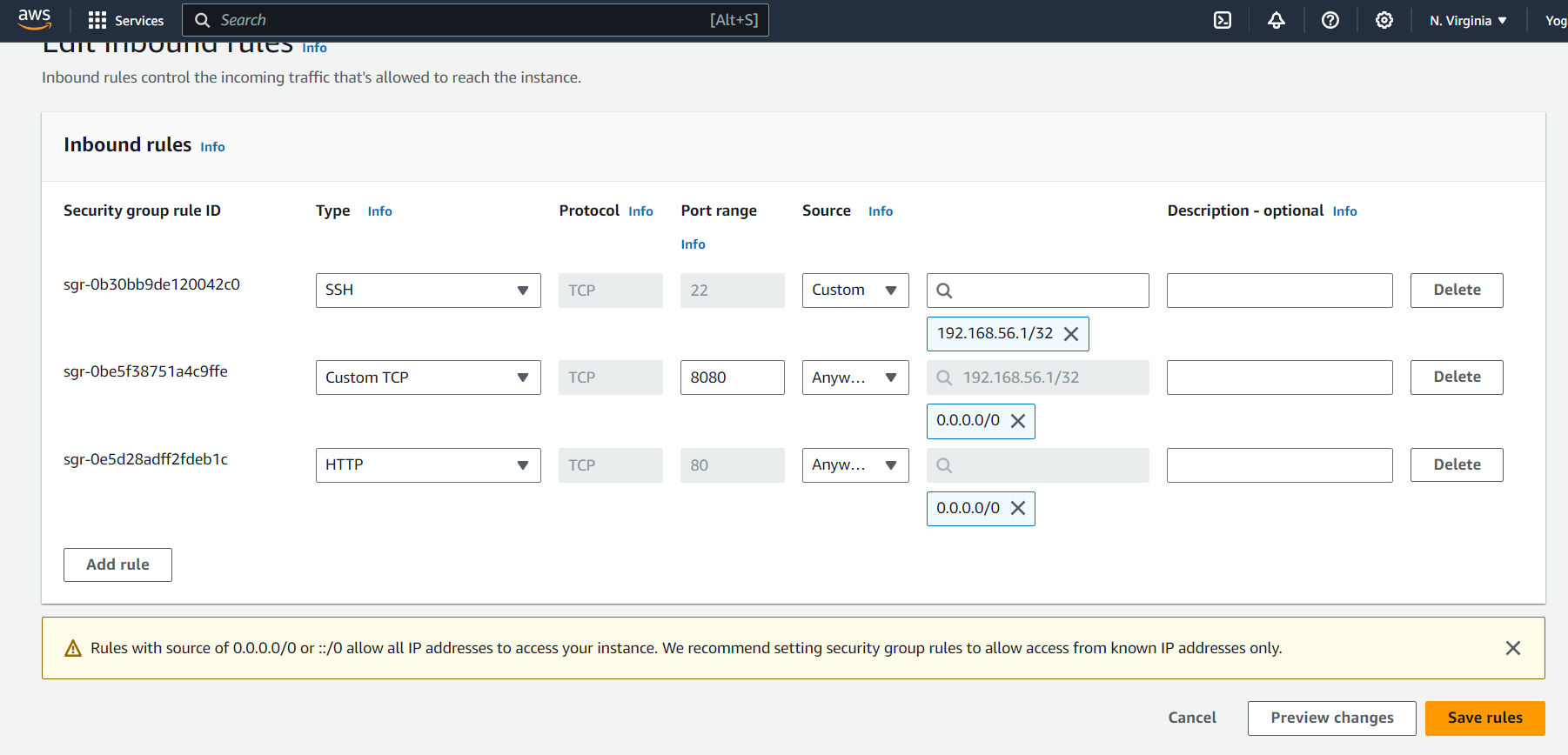
- In the Security Group, this would be configured as an inbound rule allowing traffic on the application's port (e.g., port 80 for HTTP) from any IP address (0.0.0.0/0).

2. "Login to server should be made only from your IP address":

- This means that only you, with your specific IP address, should be able to SSH into the EC2 instance.

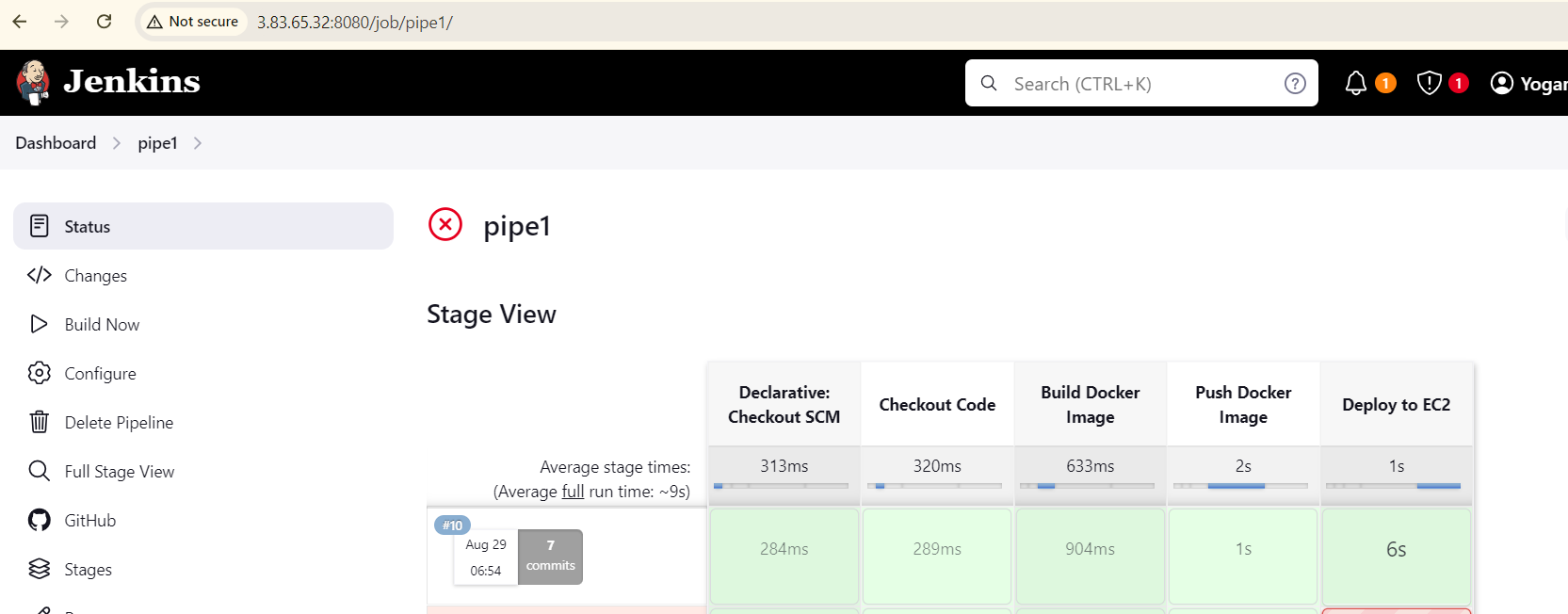
- In the Security Group, this would be configured as an inbound rule allowing SSH traffic (port 22) only from your specific IP address.

In summary, the first pointer allows anyone to access the application, while the second pointer restricts SSH access to only your IP address, adding an extra layer of security to the server.

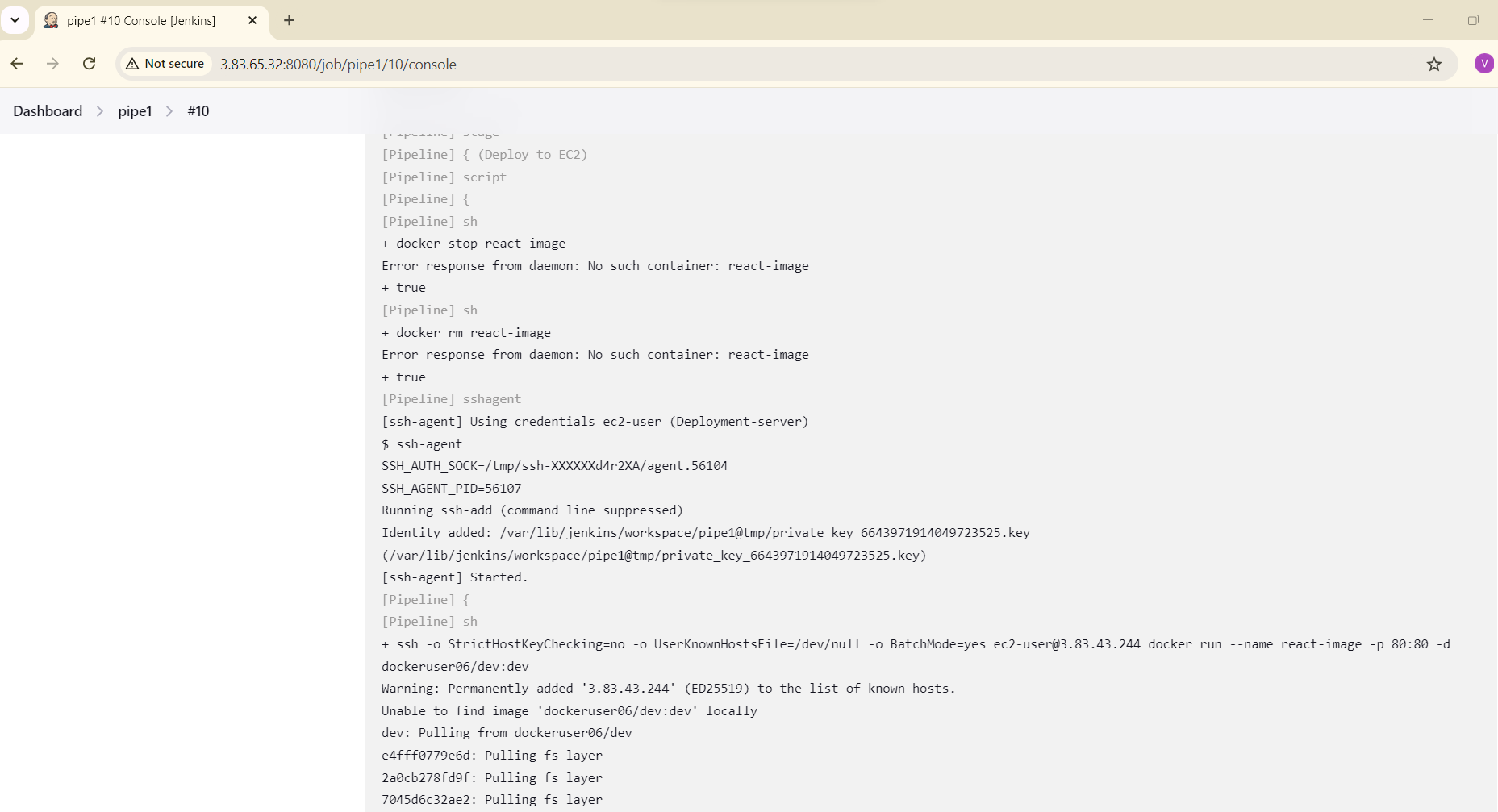


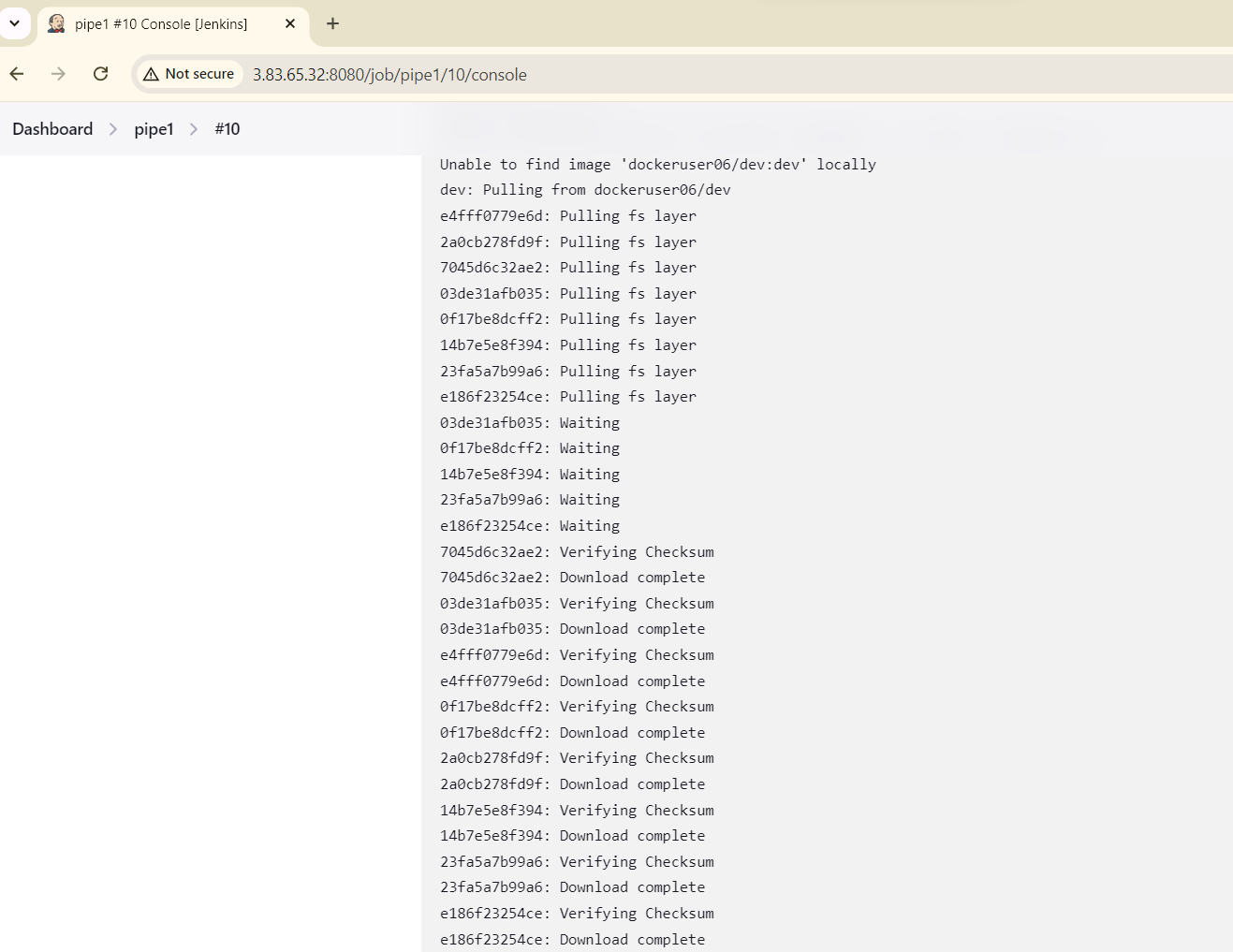
1. Commit your code changes to the Git repository.
2. Auto trigger the Jenkins pipeline
3. Jenkins will build your Docker image, push it to Docker Hub.
4. Then for the deploy stage, Jenkins will SSH in to the EC2 Server, pull the image from Docker Hub and start the container.

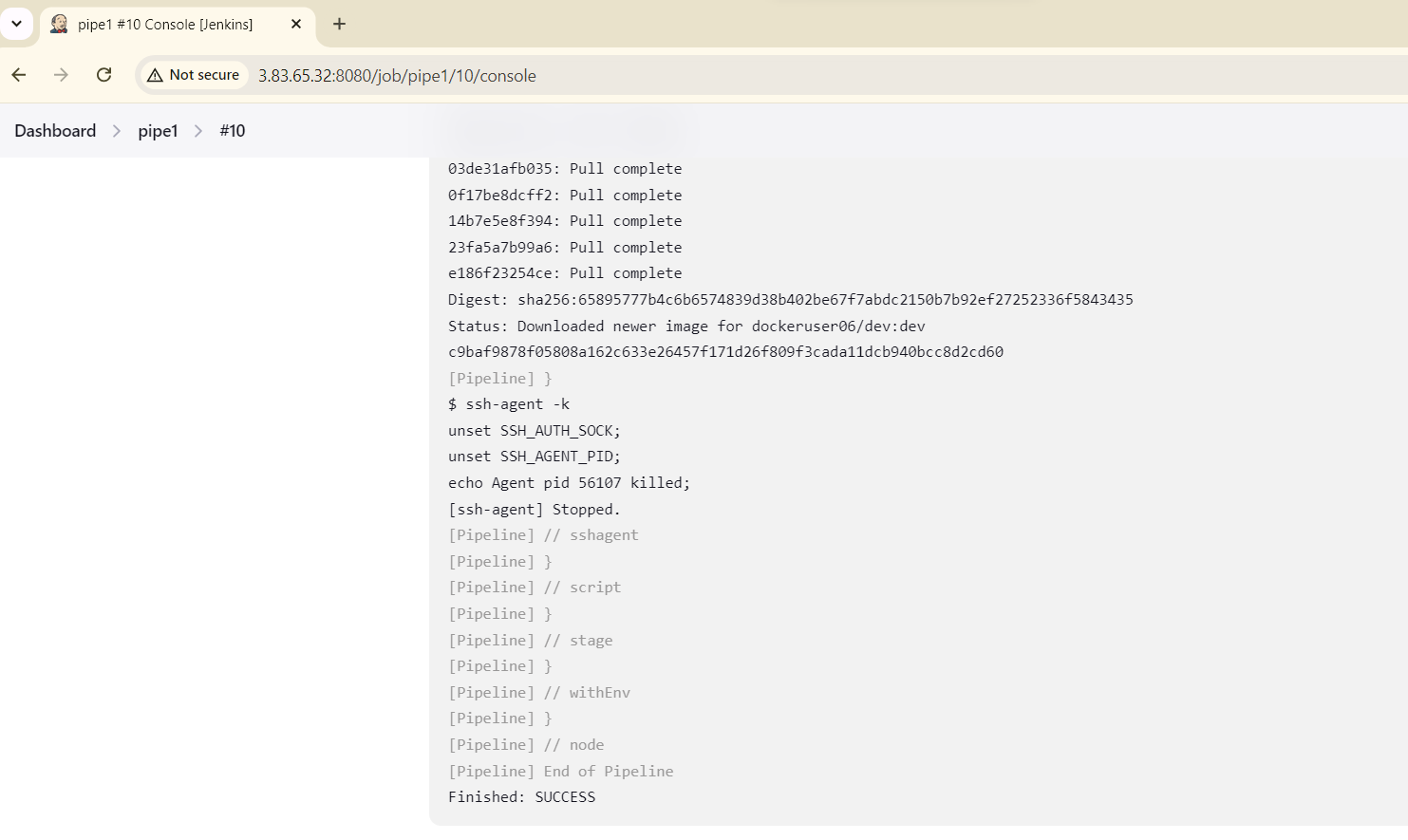
Here is the stage view of the pipeline:



And the logs for the 'deploy' stage:



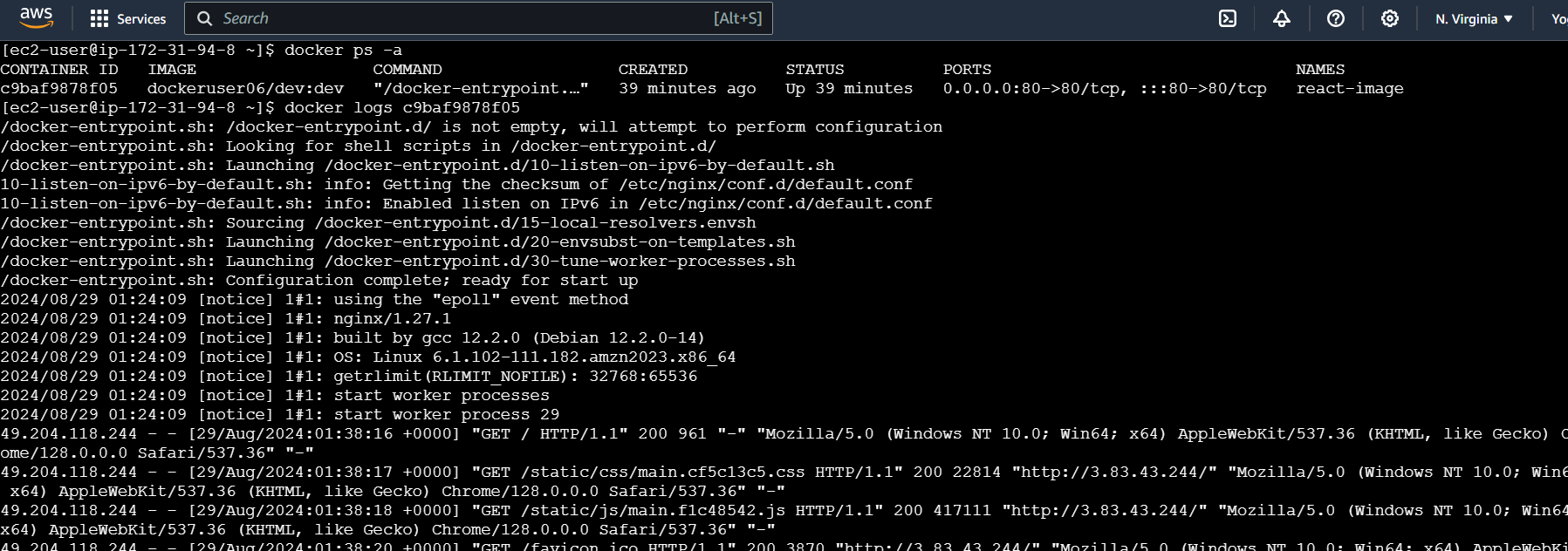






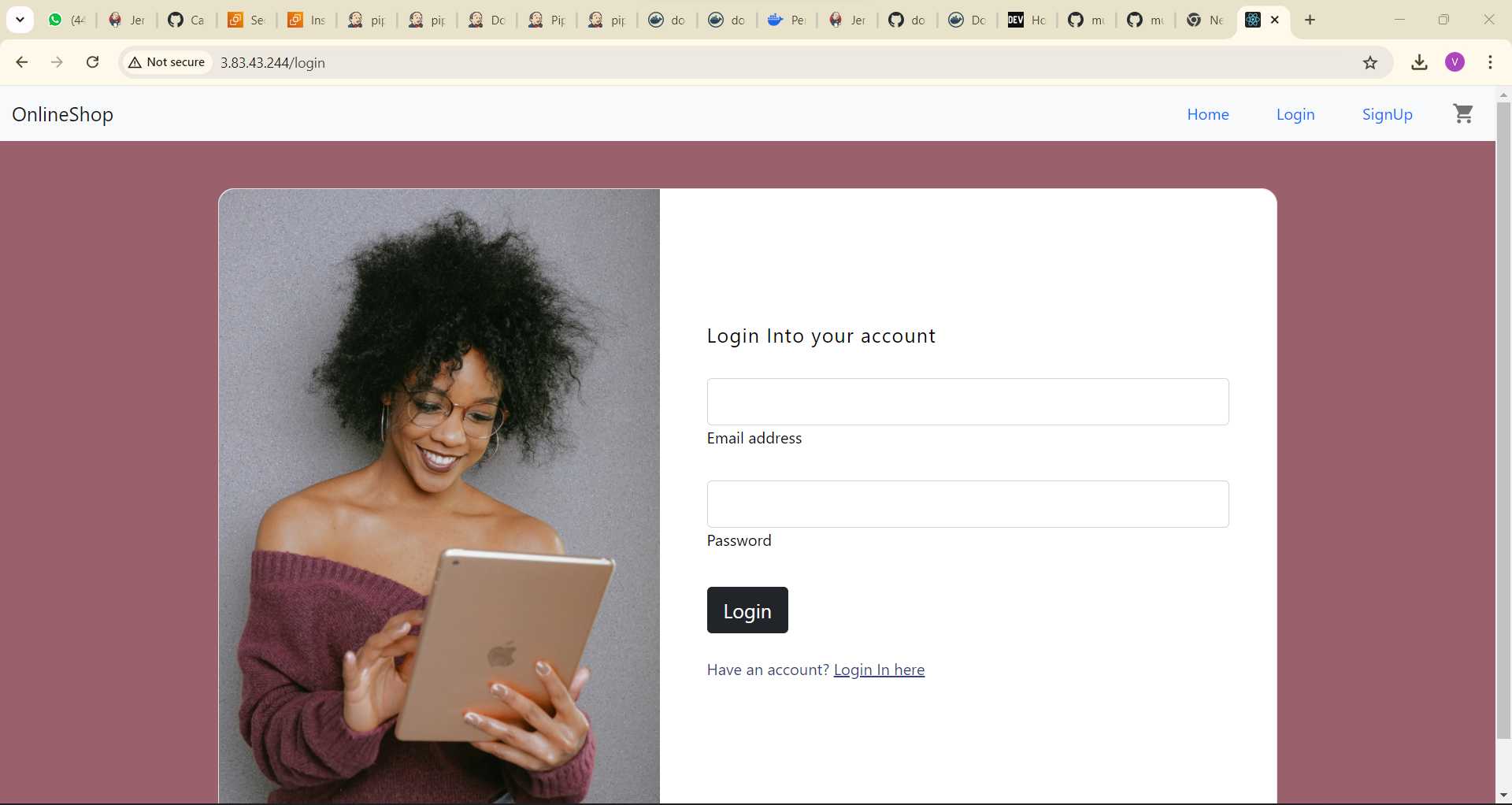
Now let’s check out the EC2 server:

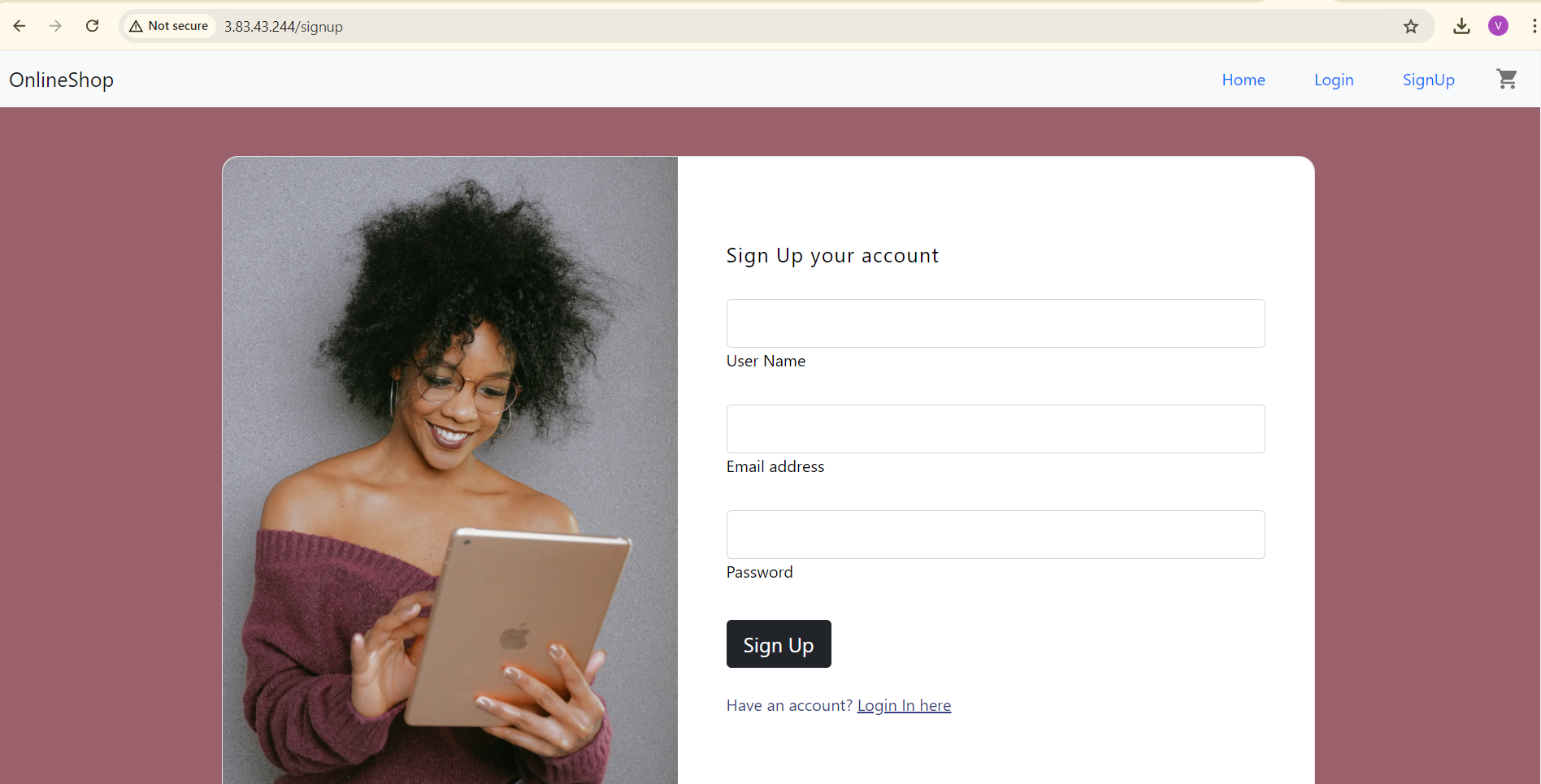
* Check Docker container status
* Check container logs
* Check Docker image



Access your application using the EC2 instance's public IP and port number. Verify that the application is functioning correctly.

**Deployed site page:**





**Monitoring health check status:**

**E**nable detailed monitoring

