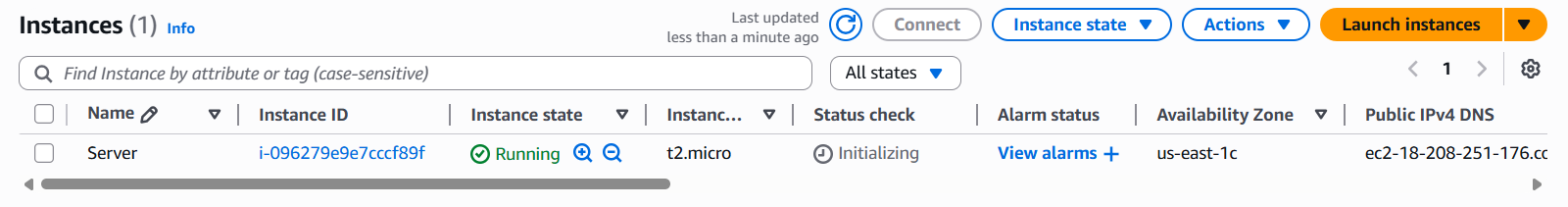
**MAIN PROJECT – DevOps Application Deployment**

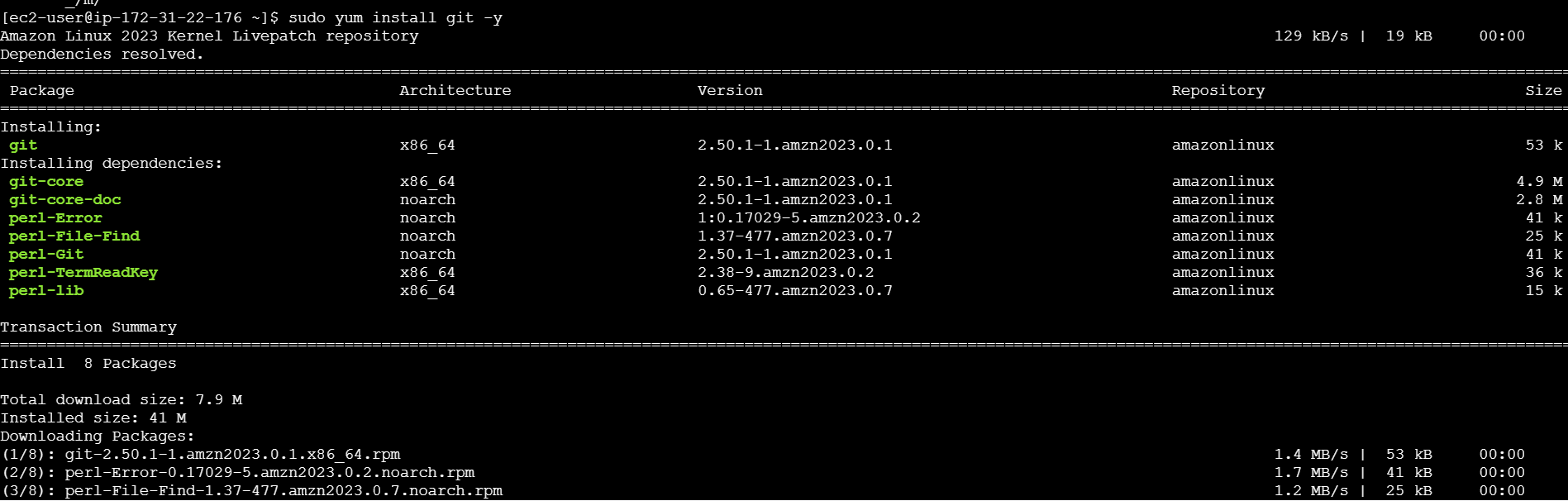
**Github Link:** [**https://github.com/Janani-977/DevOps-Project**](https://github.com/Janani-977/DevOps-Project)

Jenkins:

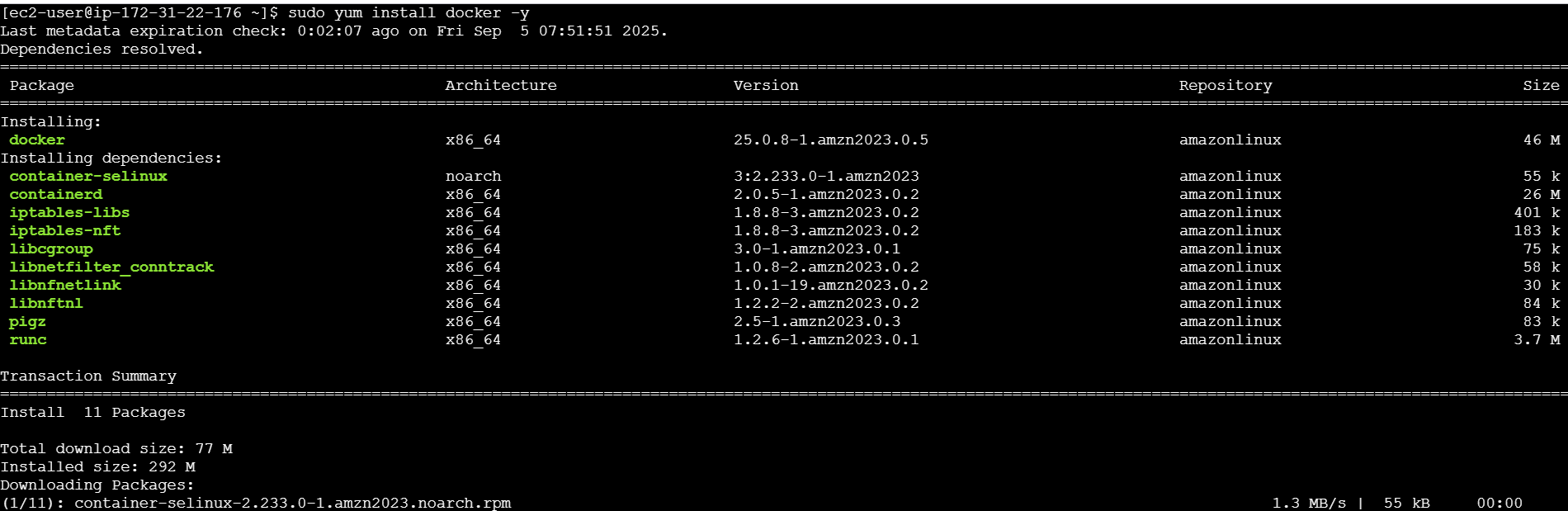
Launched an ec2 machine to host the application and Jenkins server  
  
  
  
Installing Dependencies:

Installing git and docker

# sudo yum install git -y

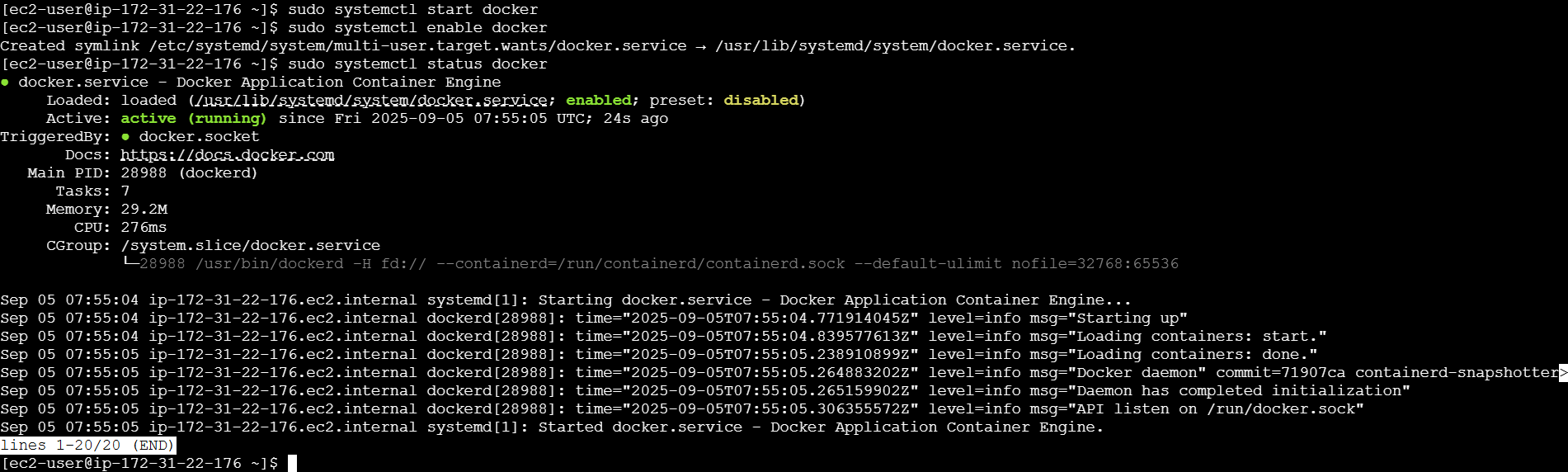


# sudo yum install docker -y



Starting and enable the docker service to remain service up even after the server boot

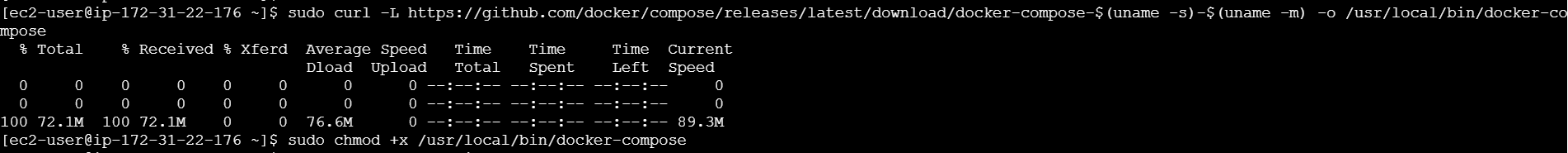
# sudo systemctl start docker  
# sudo systemctl enable docker  
# sudo systemctl status docker

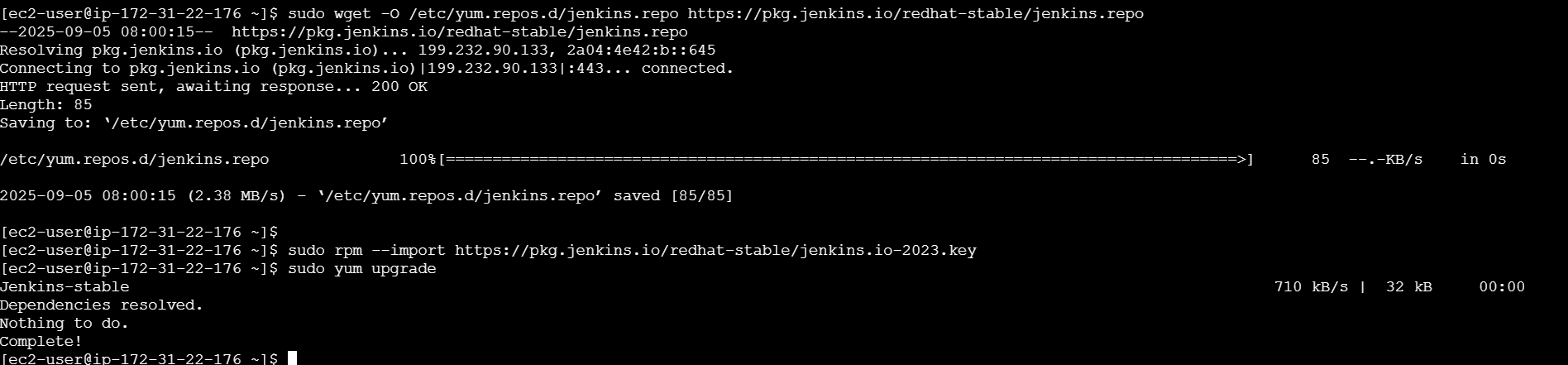


Installing the docker compose:

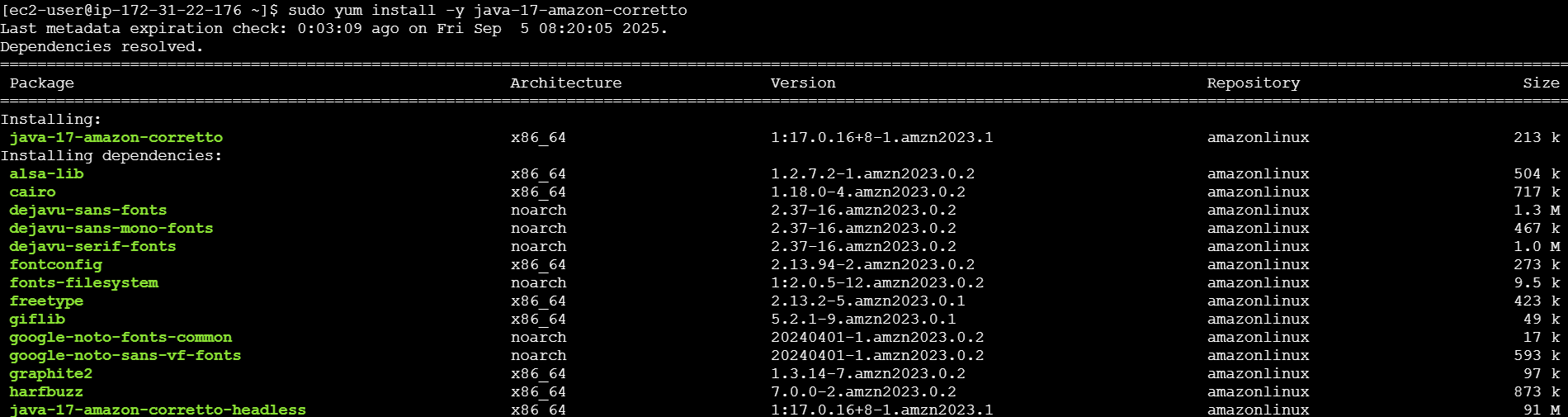
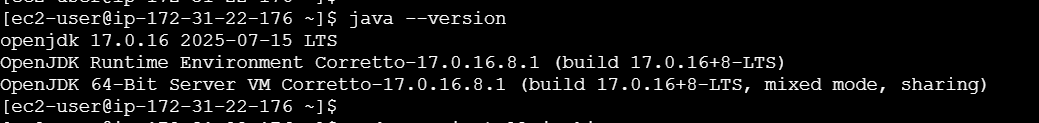
# sudo curl -L https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

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

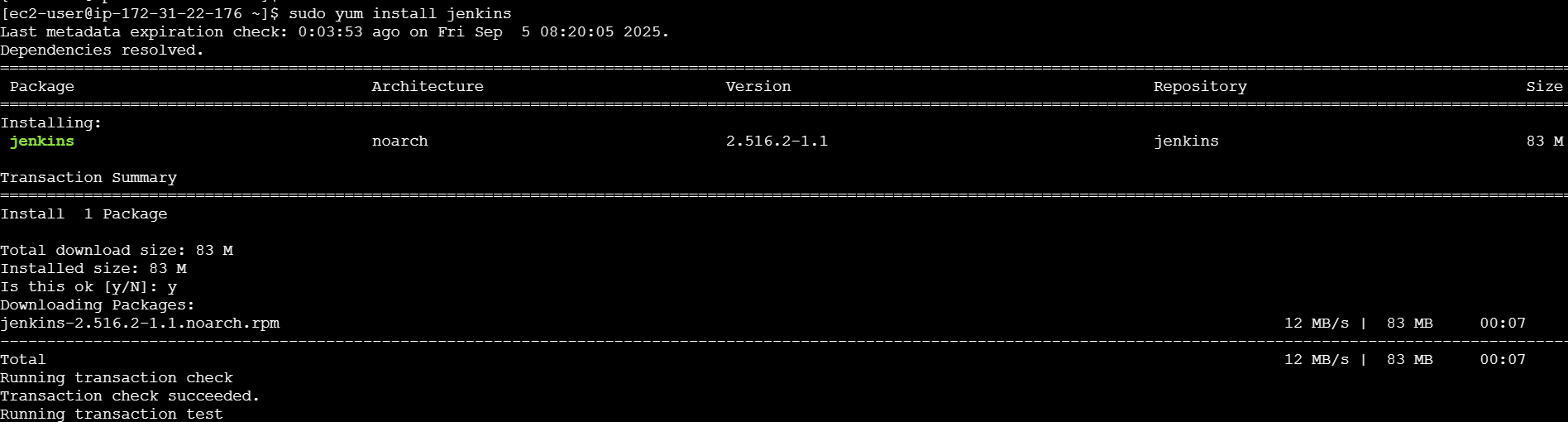




Installing java dependency to host the Jenkins.

Installing open-jdk-17   
  
  


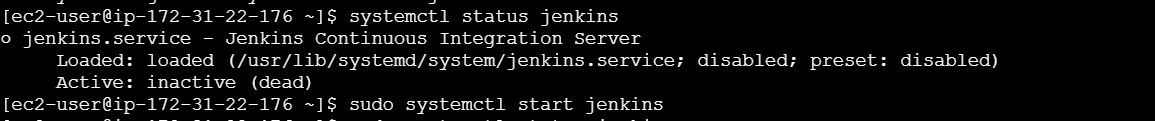
Installing Jenkins Server

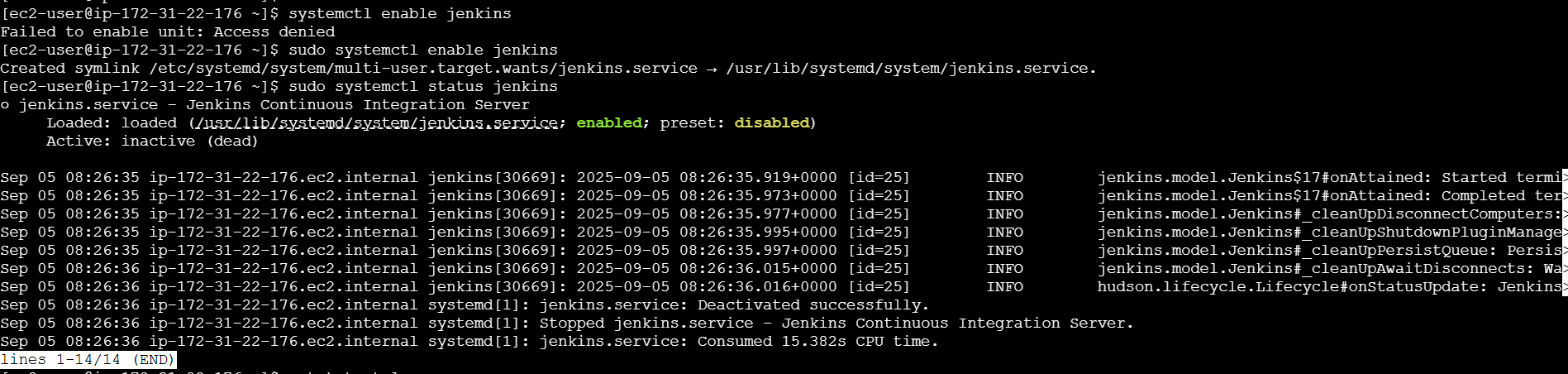


Reloading system daemon

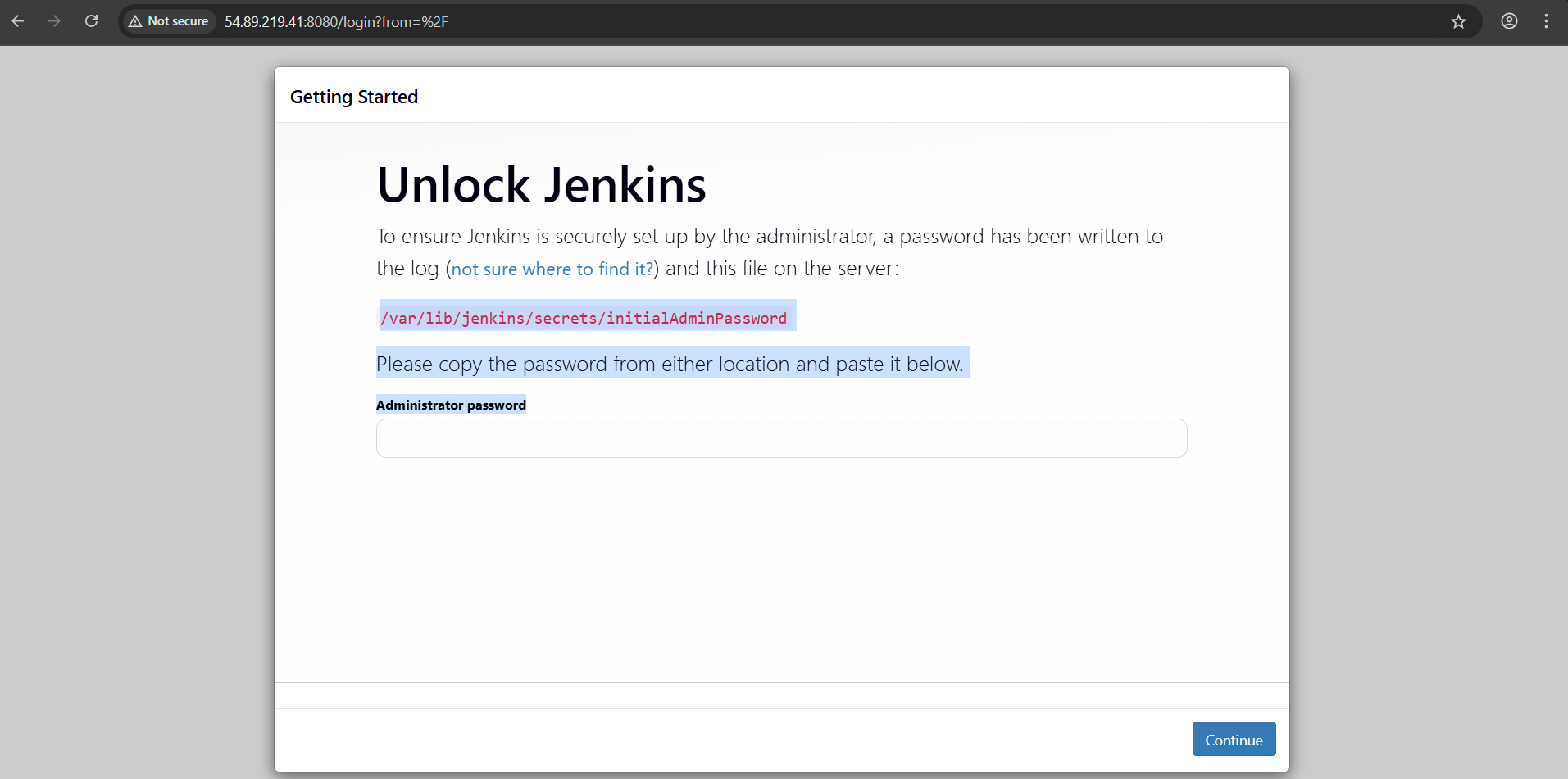


Starting and enabling service to make it permanent even after the reboot.

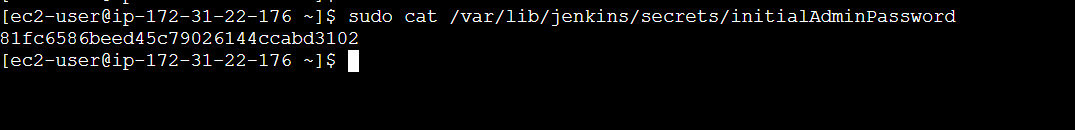


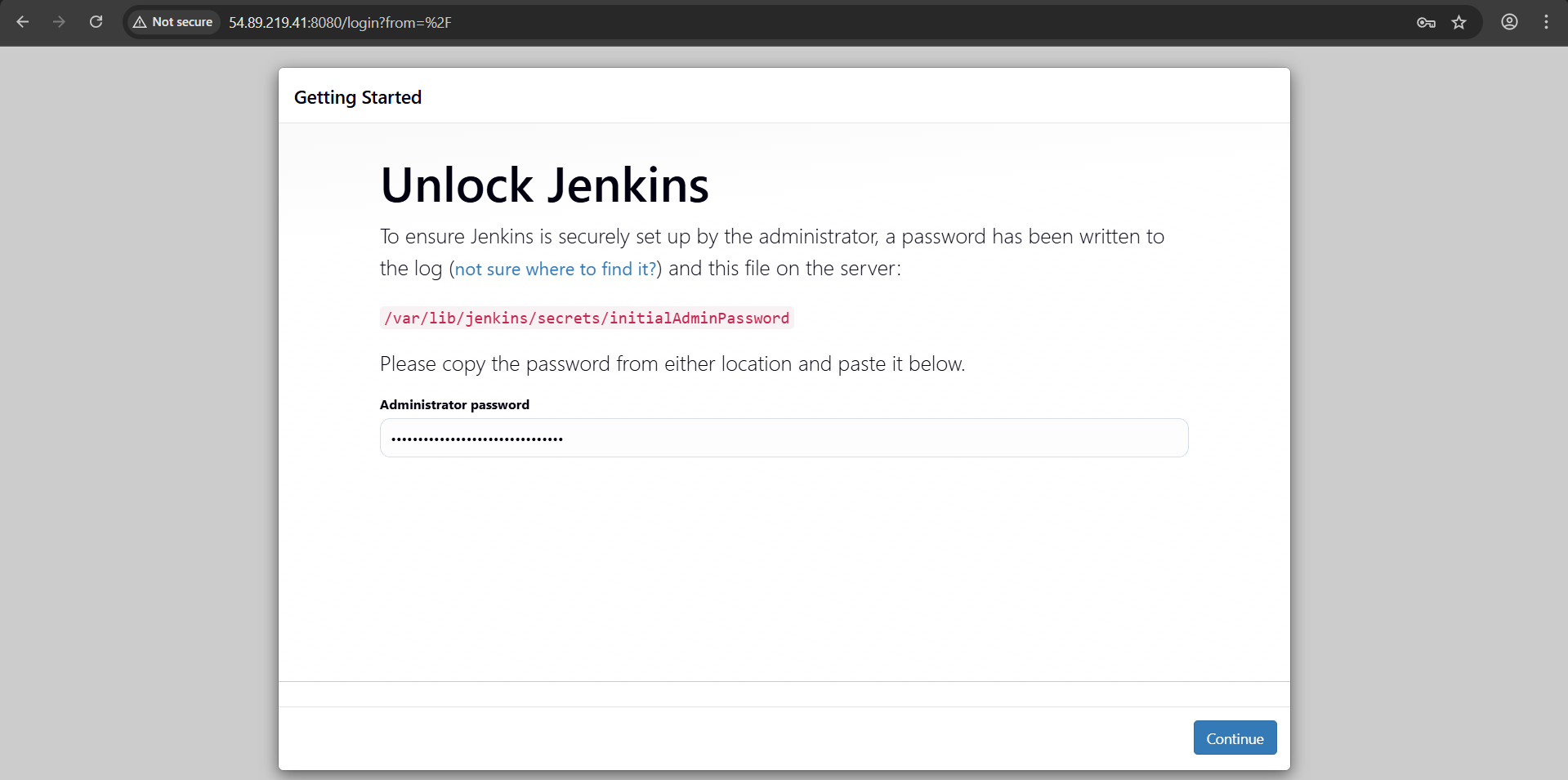


Jenkins will run on port num 8080. accessed it with the public ip address of Jenkins Server and port num 8080. To unlock Jenkins we have to pass the initial password. Which is available in the below path.



Logged into the Jenkins Server instance then copied the password to unlock the Jenkins Server.

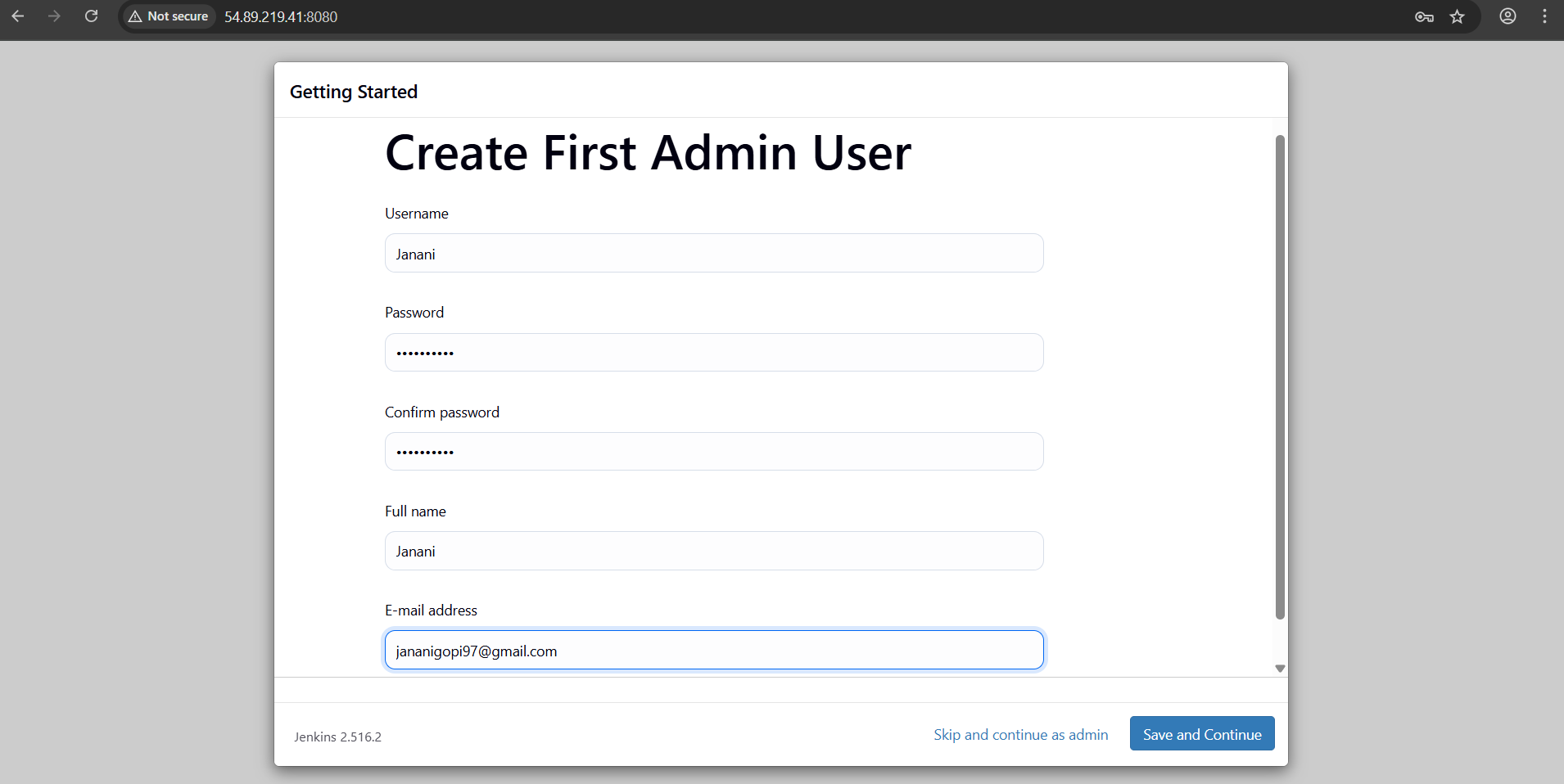




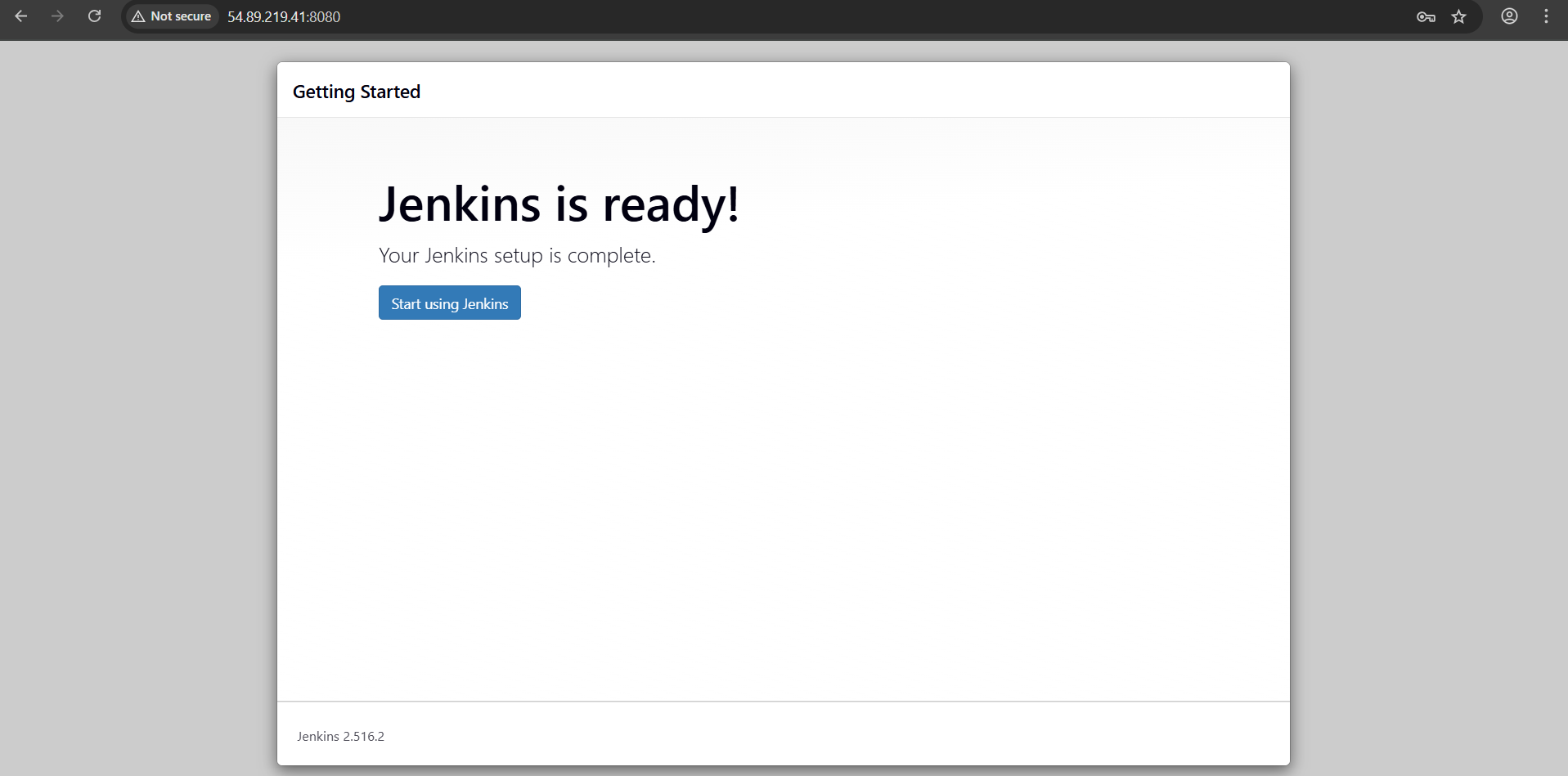
Installing plugins

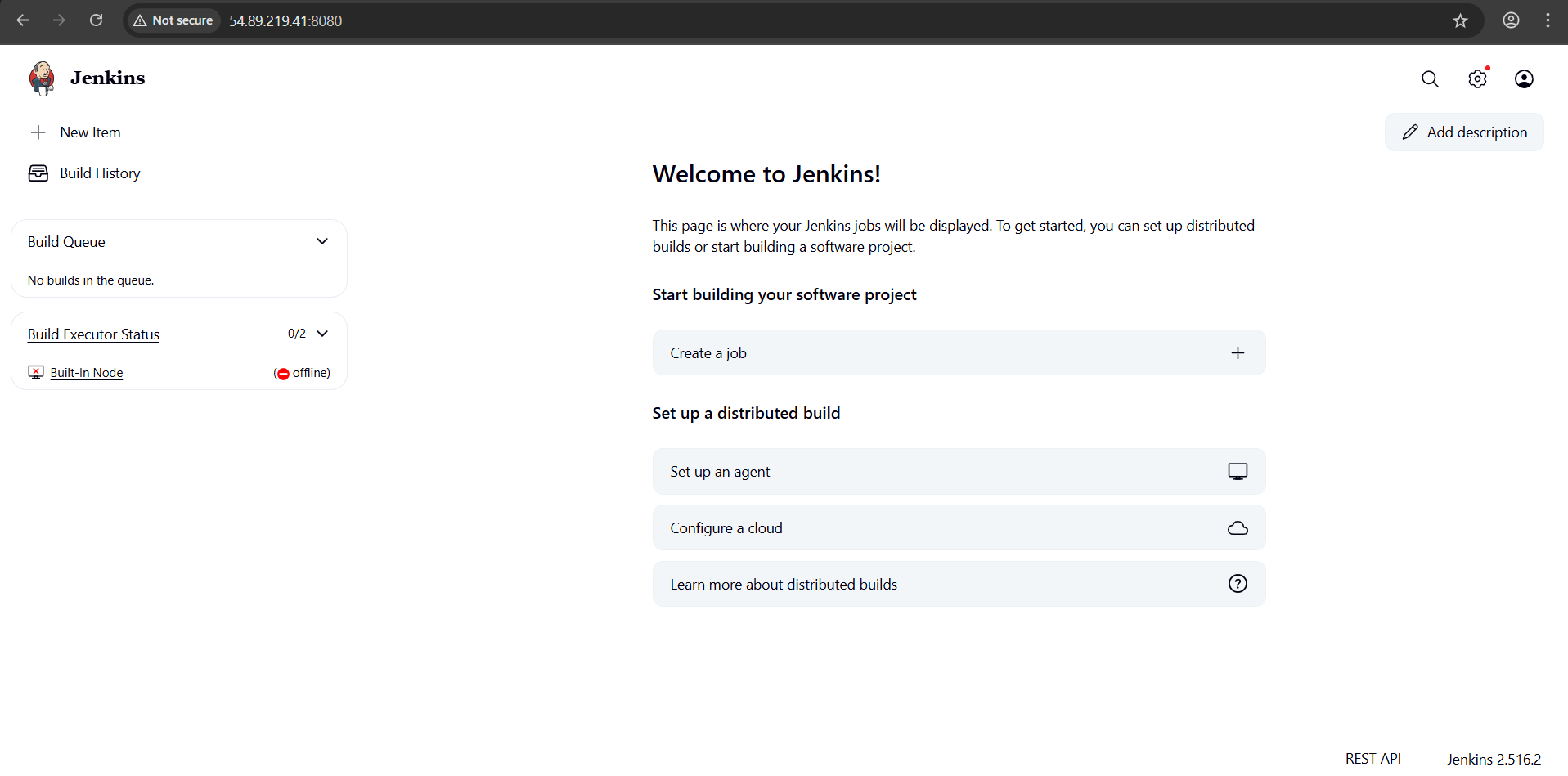


Creating an Admin user to access the Jenkins.

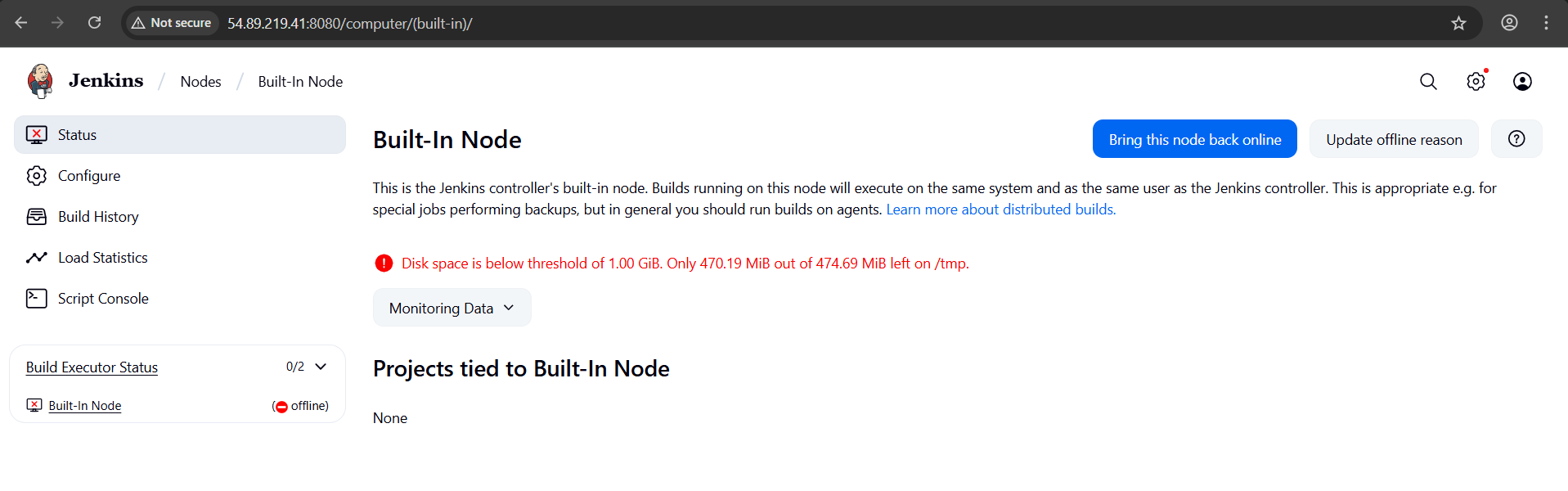


Jenkins is ready to use.





Currently master node is offline due to insufficient storage for the /tmp folder.

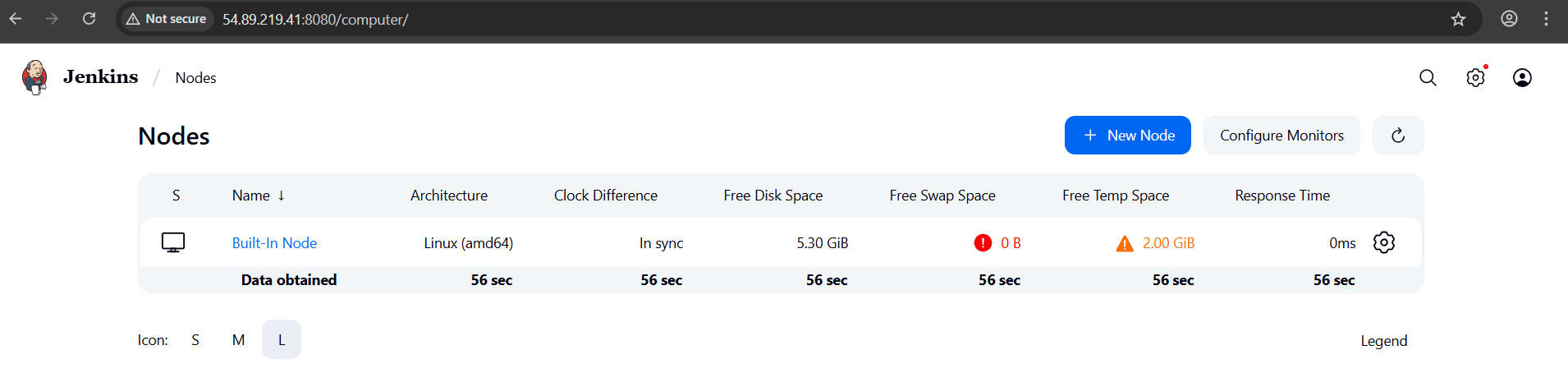


Increased size of tmp folder from 500 Mb to 2Gb

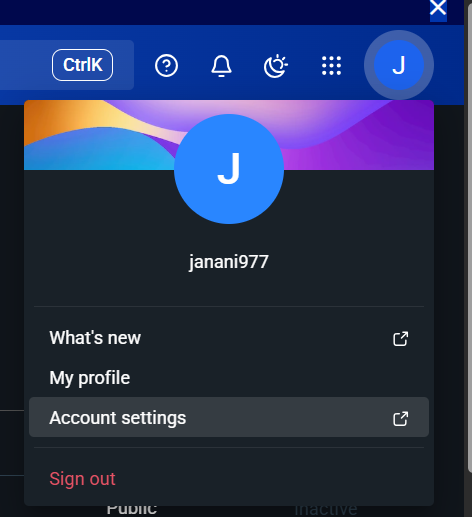


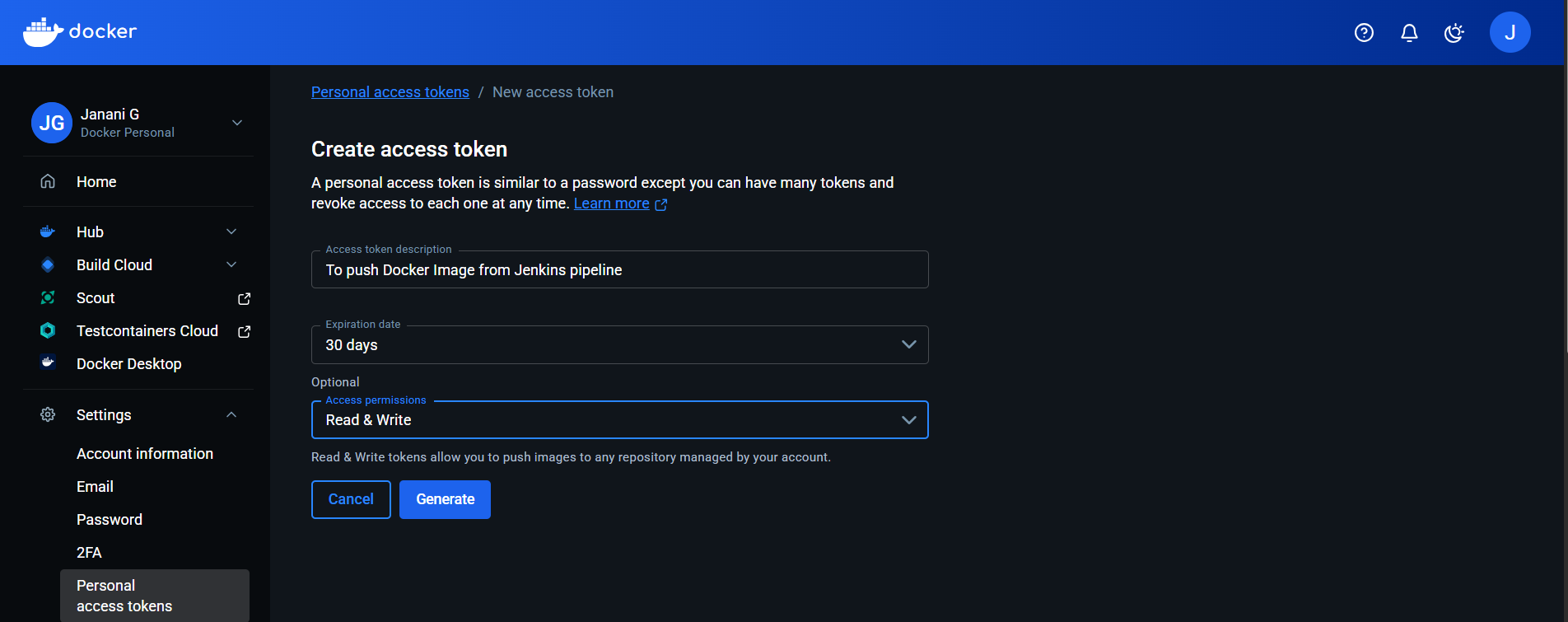


Now Jenkins master node is available to run the pipelines in it.

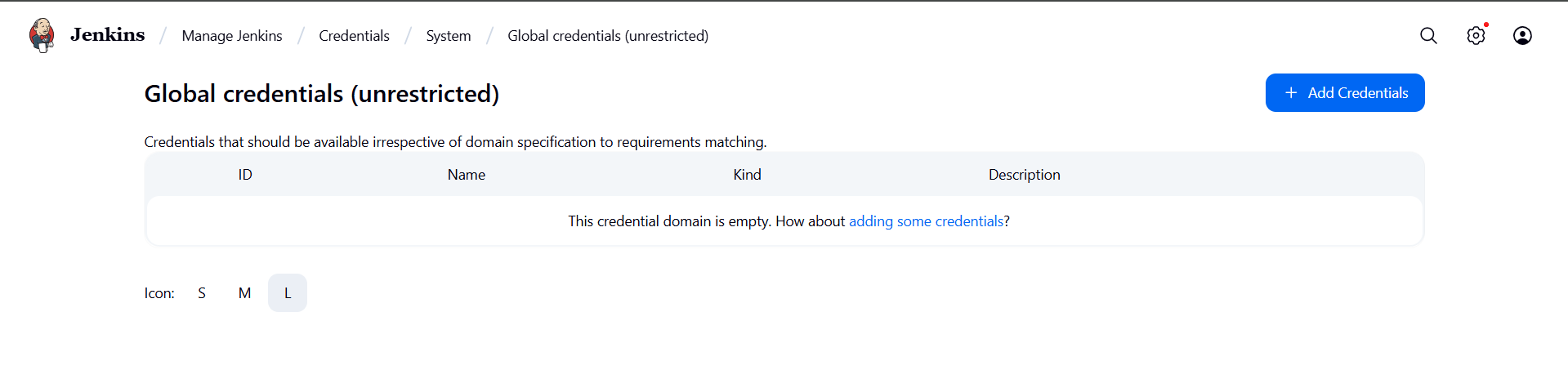


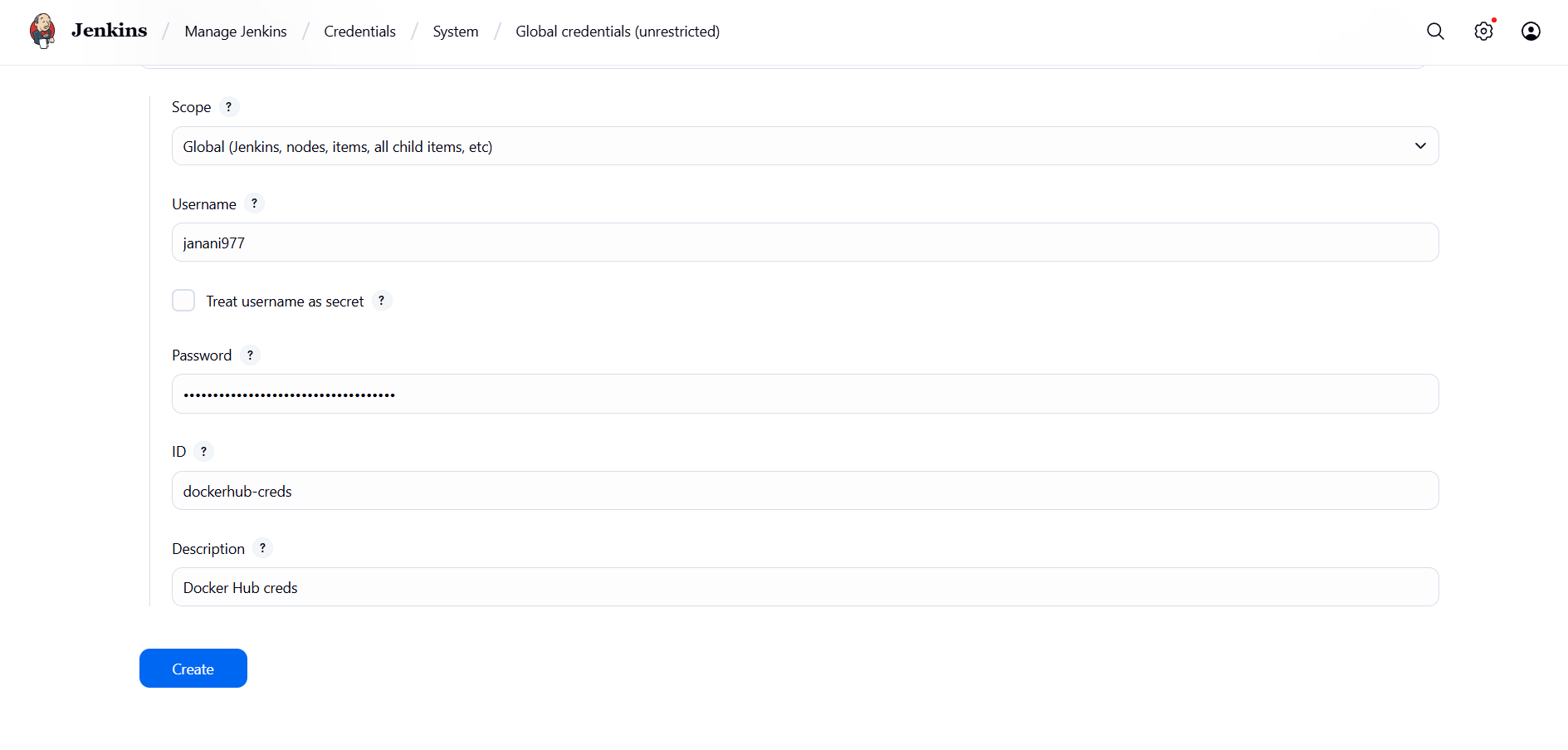
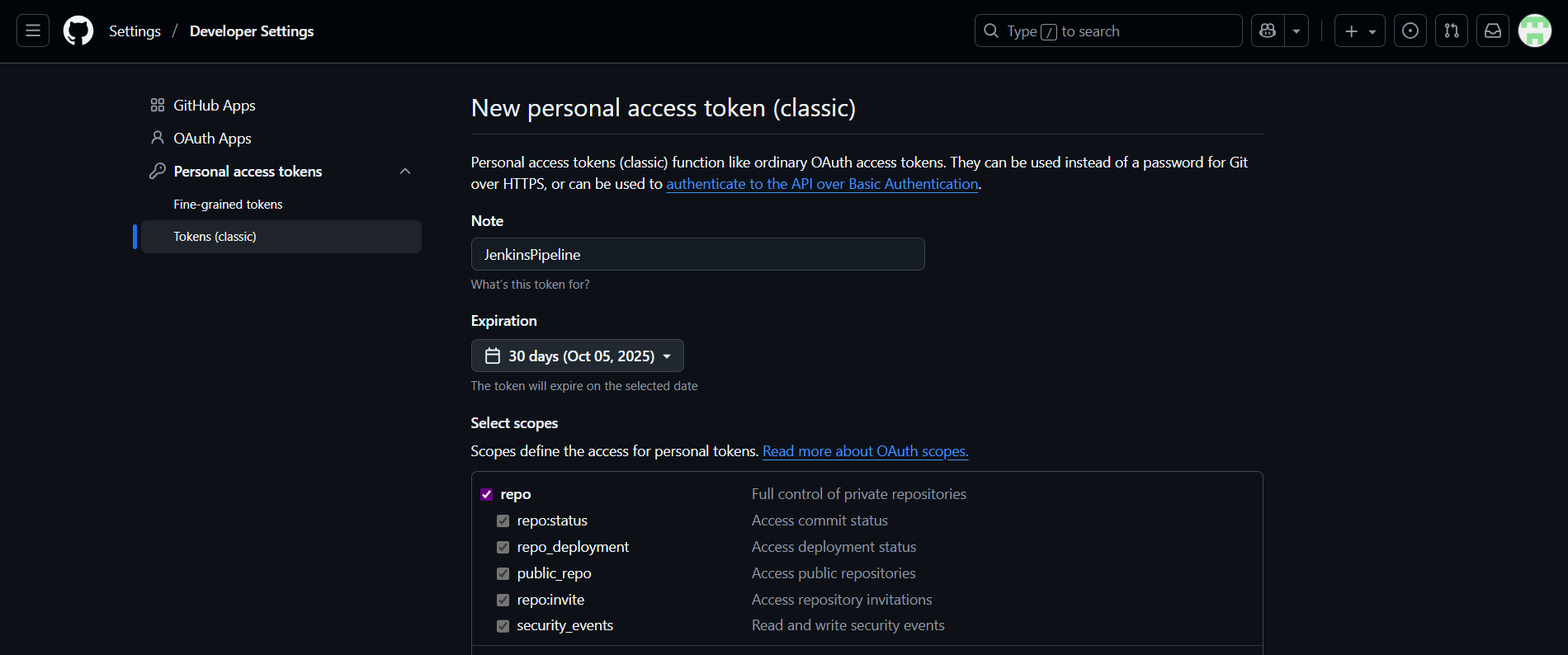
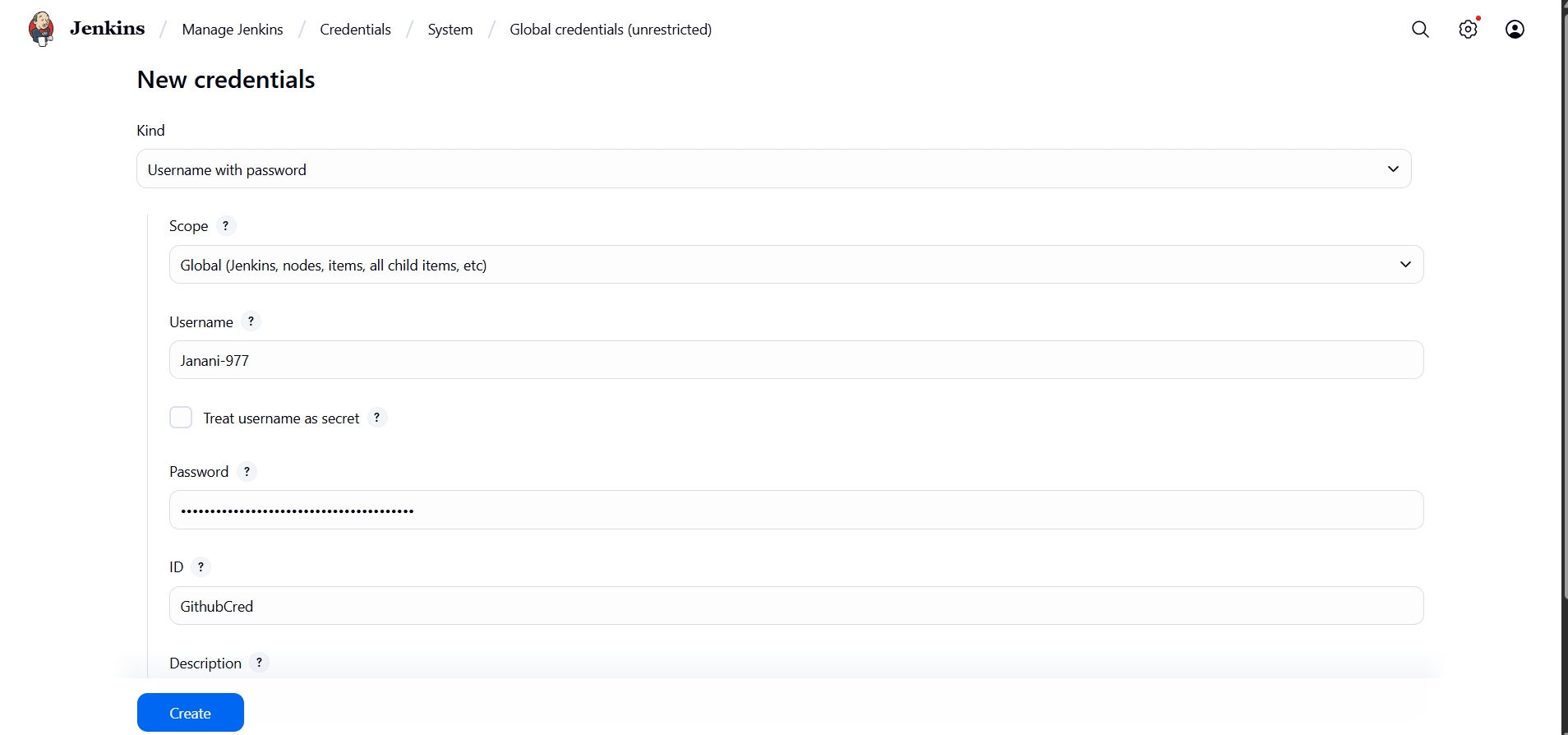
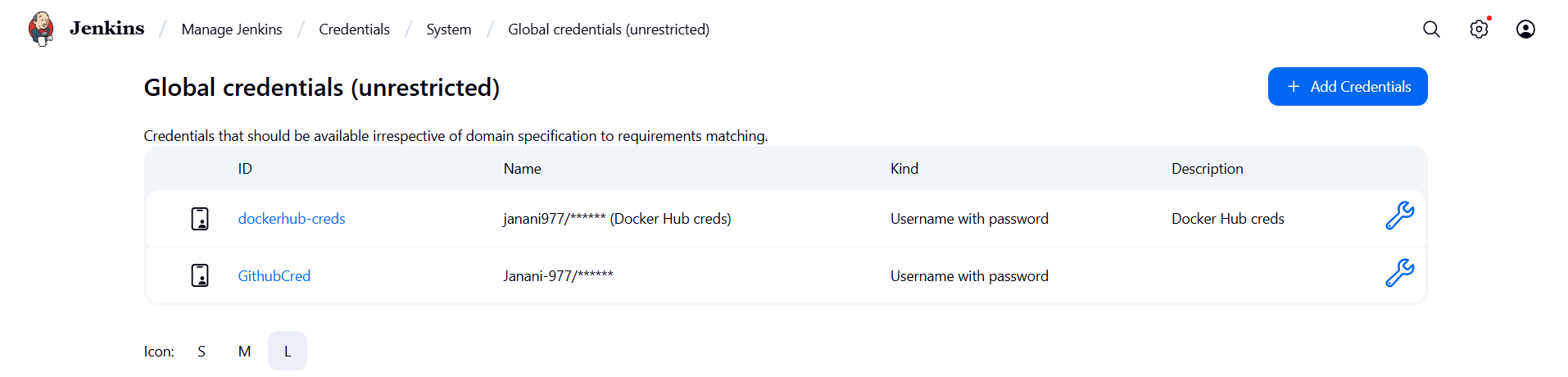
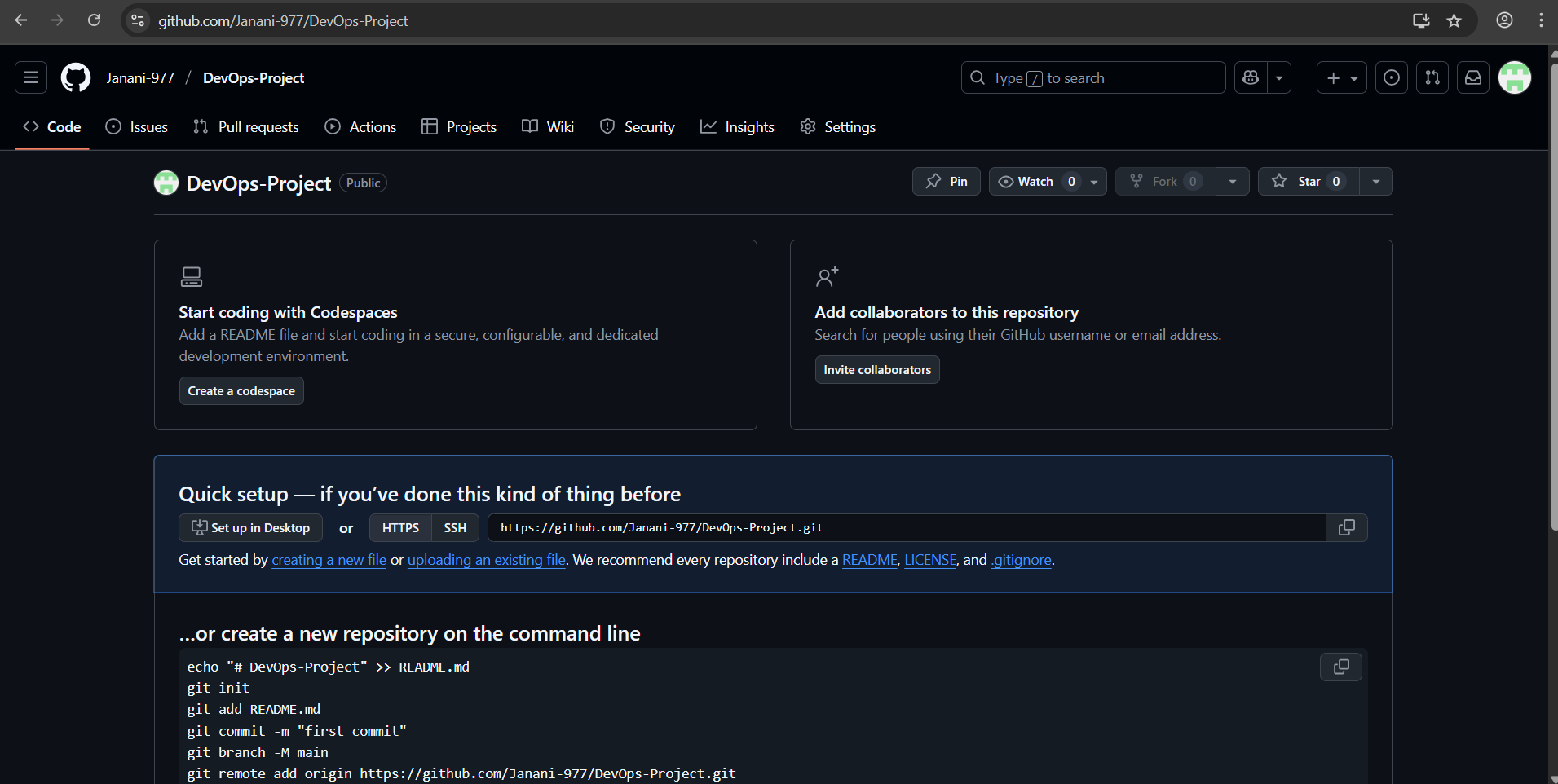
To push the docker image into the docker hub account. We have to perform Authorization to verify the identity of the user. For that in the user block click on account settings then click on personal token for the user.





Adding the credentials to pass those in the pipeline to perform authorization before pushing the images to docker hub.



  
  
similarly to pull from the code after any event we need to pass the credentials to the pipeline for that generating the token to perform the Authorization.  
  
  
  
adding GitHub credentials in the Jenkins credentials.  
  
  
  
  
  
created a GitHub repo to push the code.  


Bash Scripting:

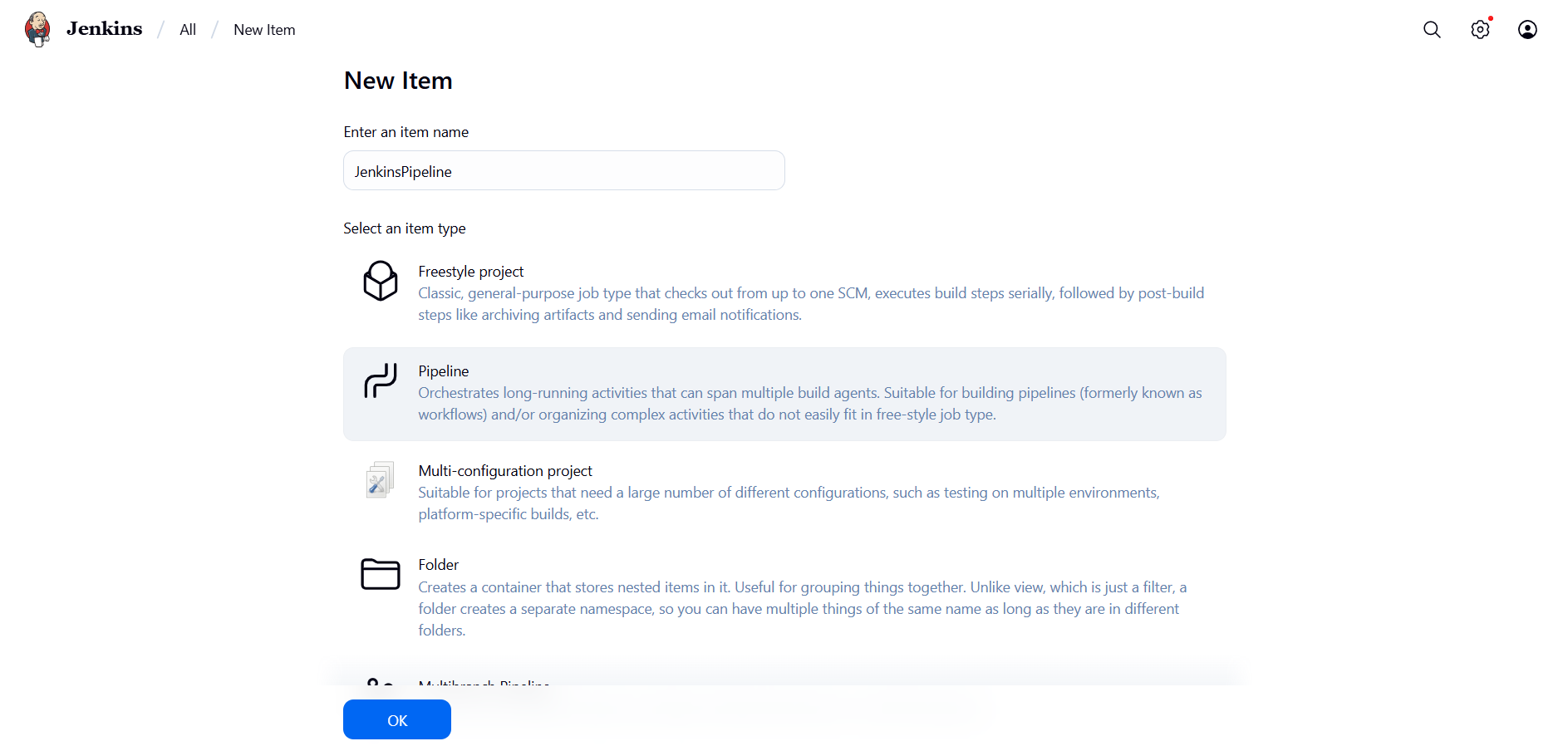
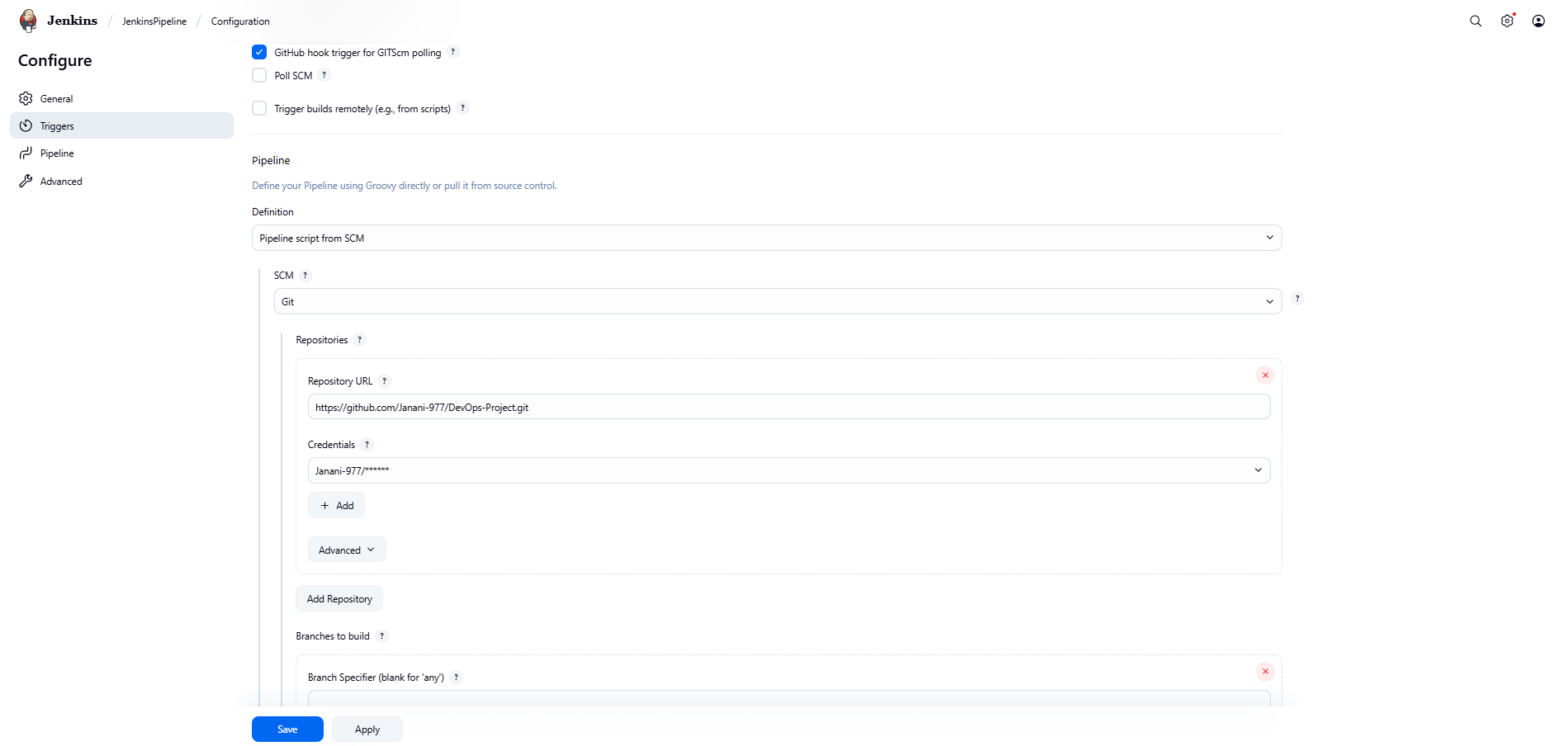
Build.sh



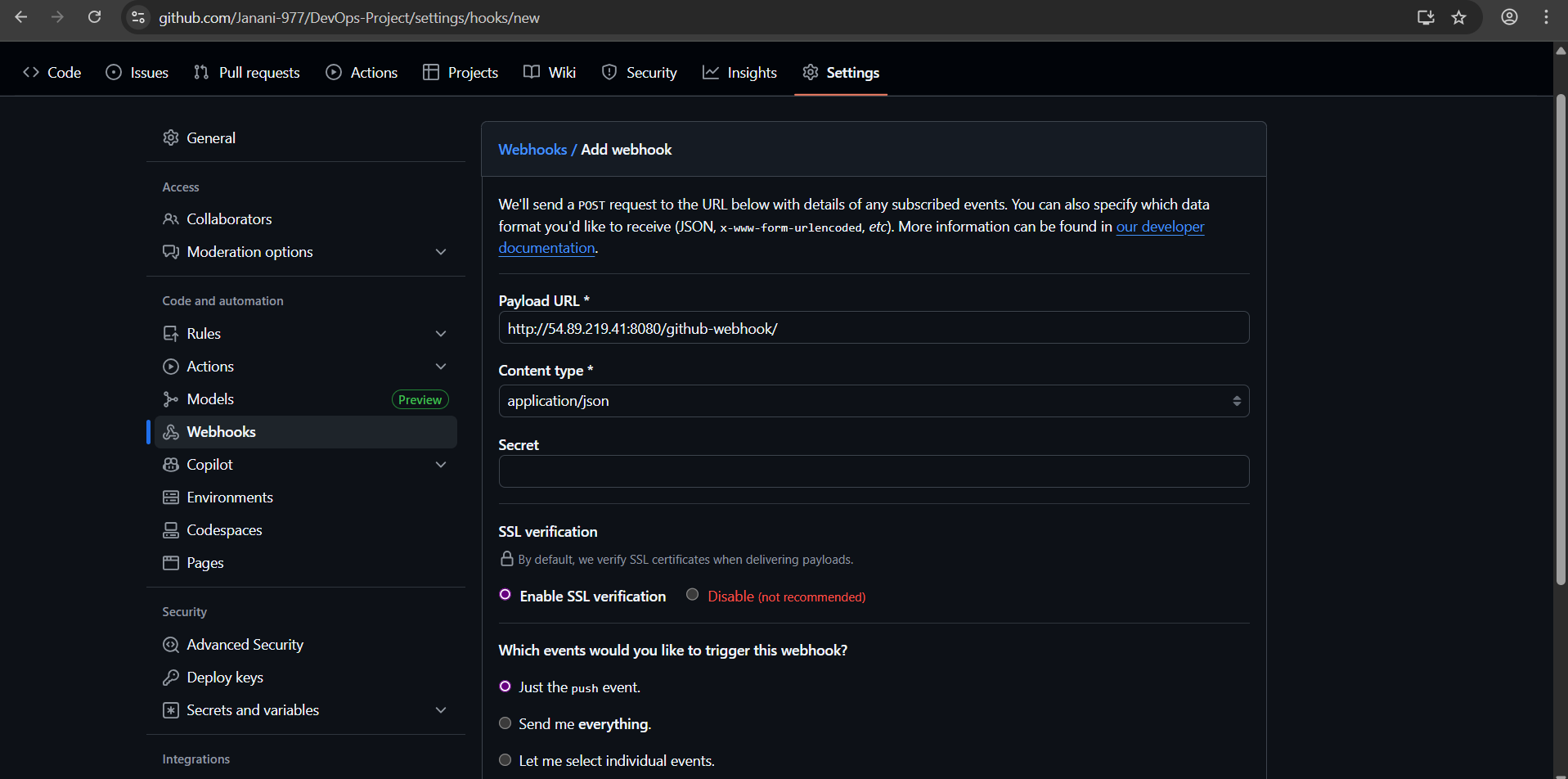
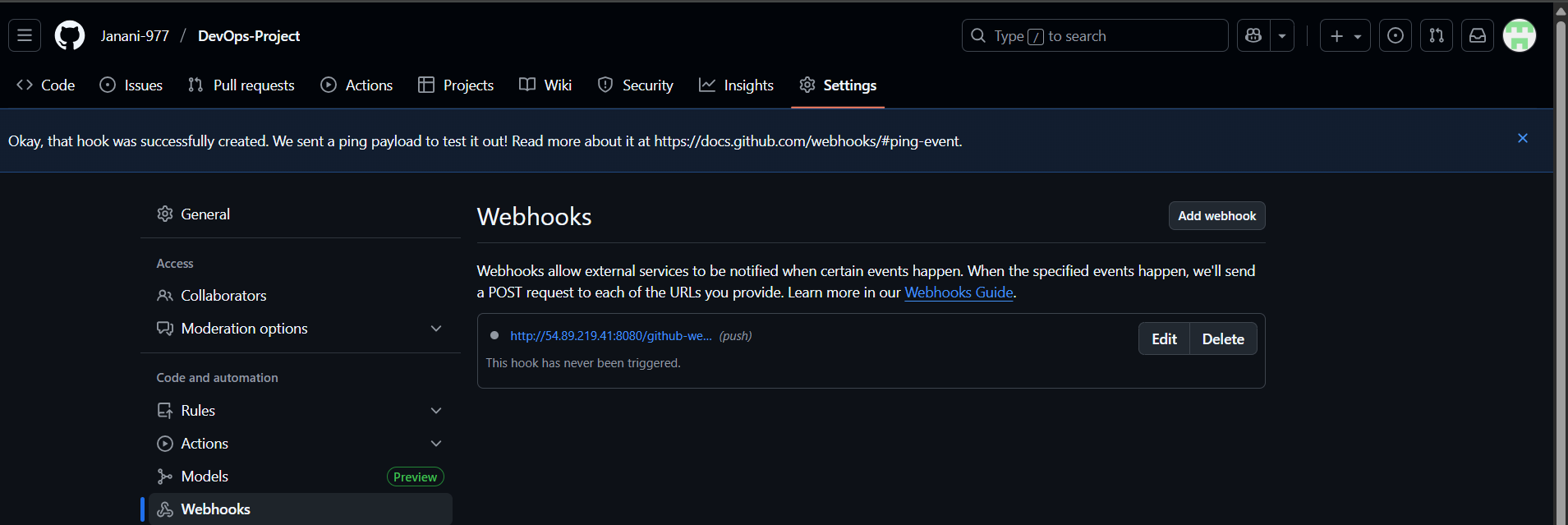
Deploy.sh



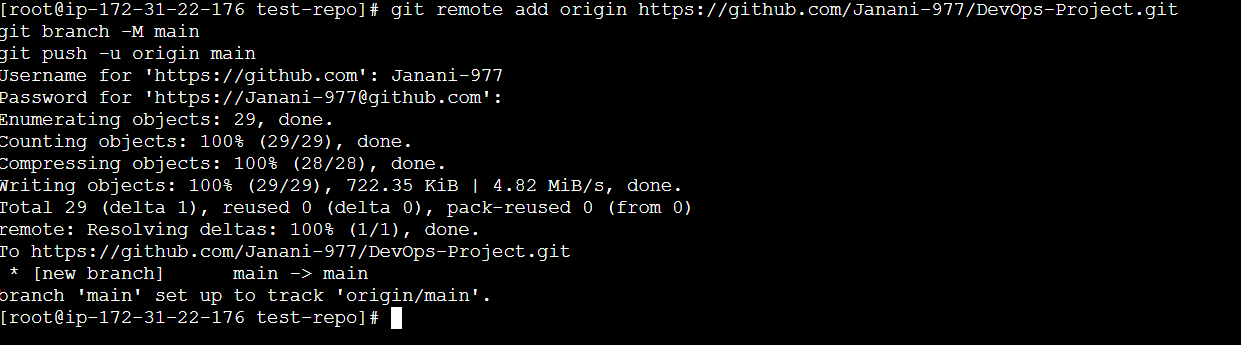
creating a pipeline to push the docker image whenever push events happen in the repo.

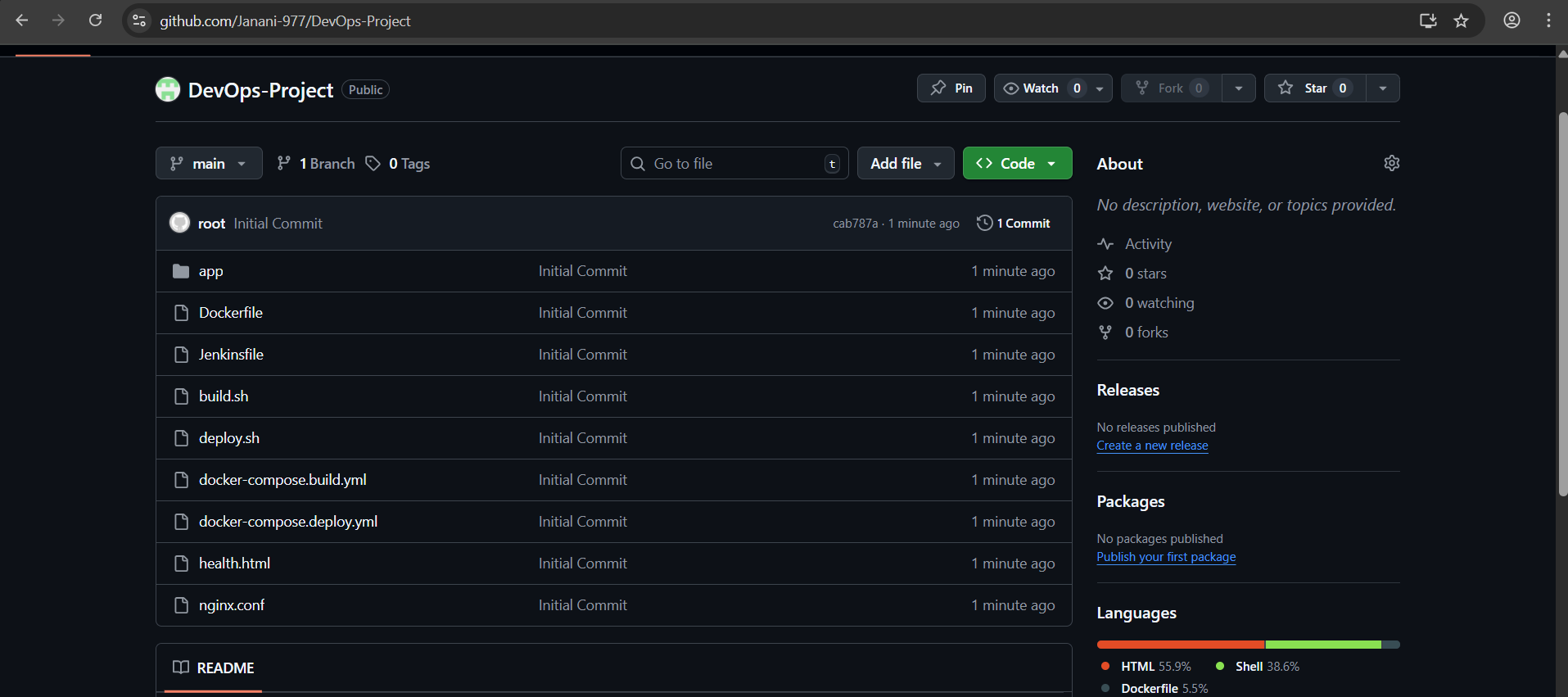
  
  
providing the repository details to the pipeline from where it has to pull to code of Jenkins pipeline.  
  




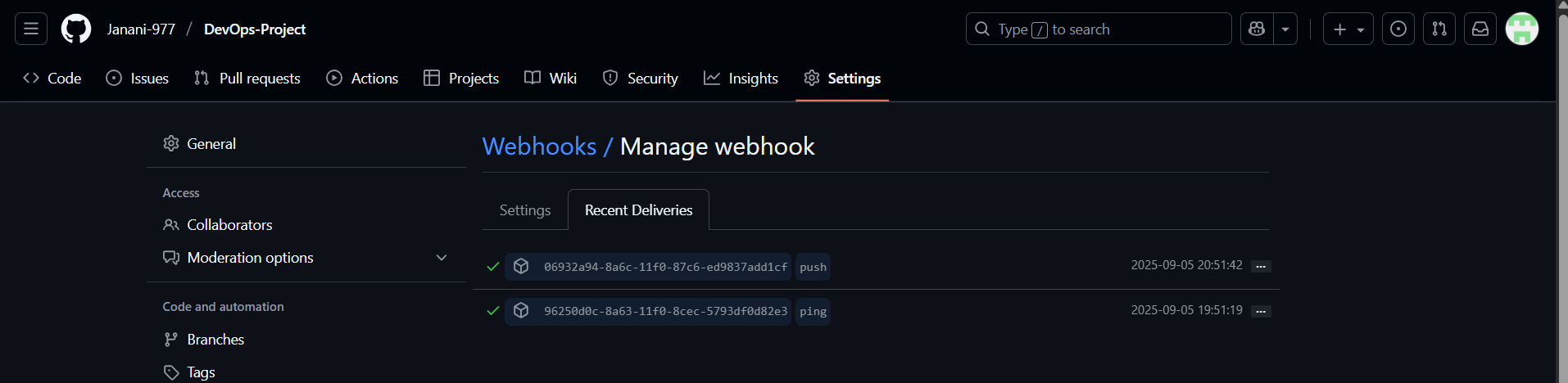
Configuring webhook to push event from GitHub to Jenkins.  
  
  
  
  


Pushing required code to push the docker image of application.

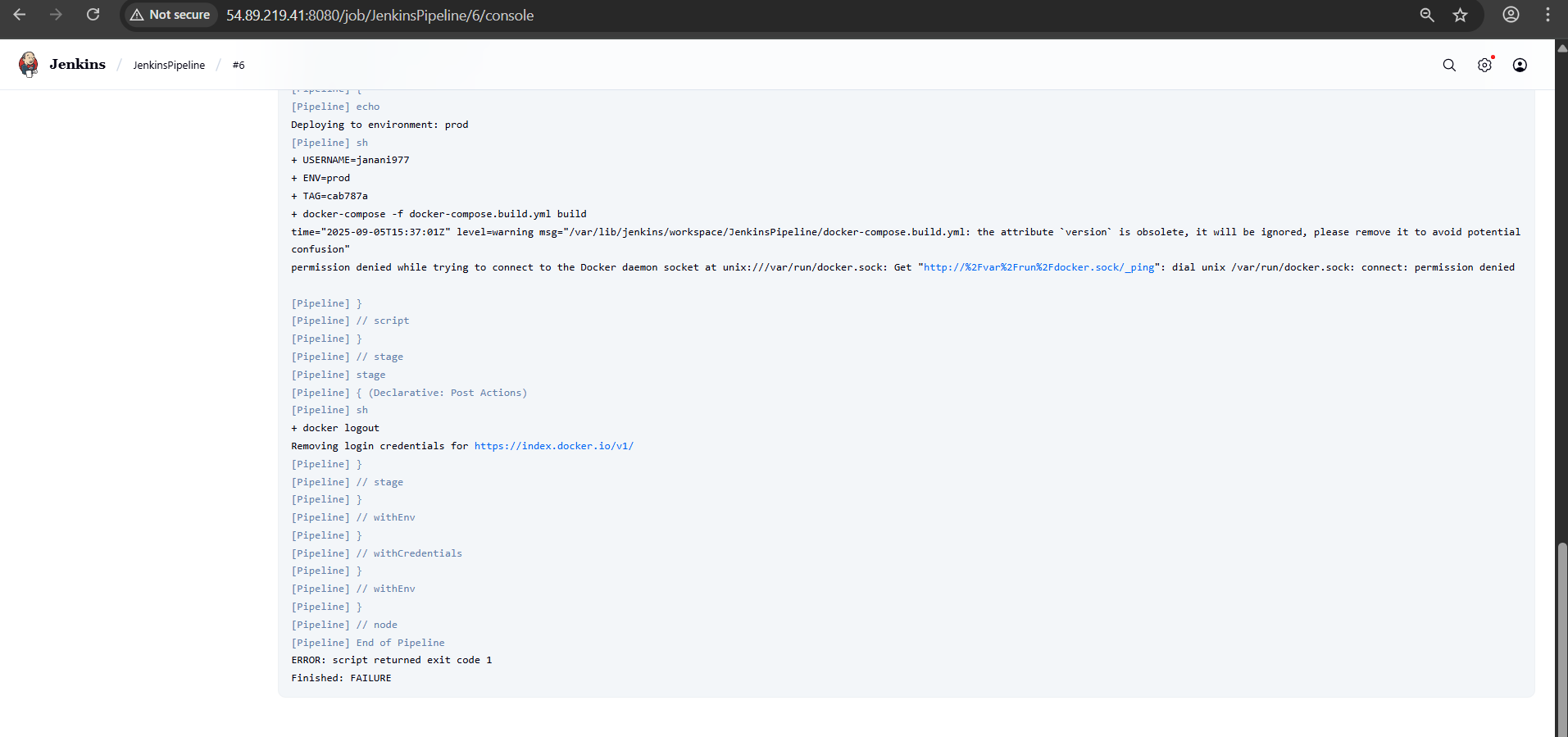




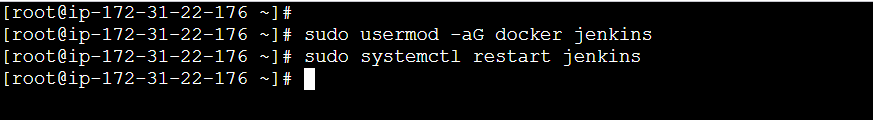
Webhook pushed the event from Jenkins to pipeline



Jenkins user doesn't have permission to run the docker command.

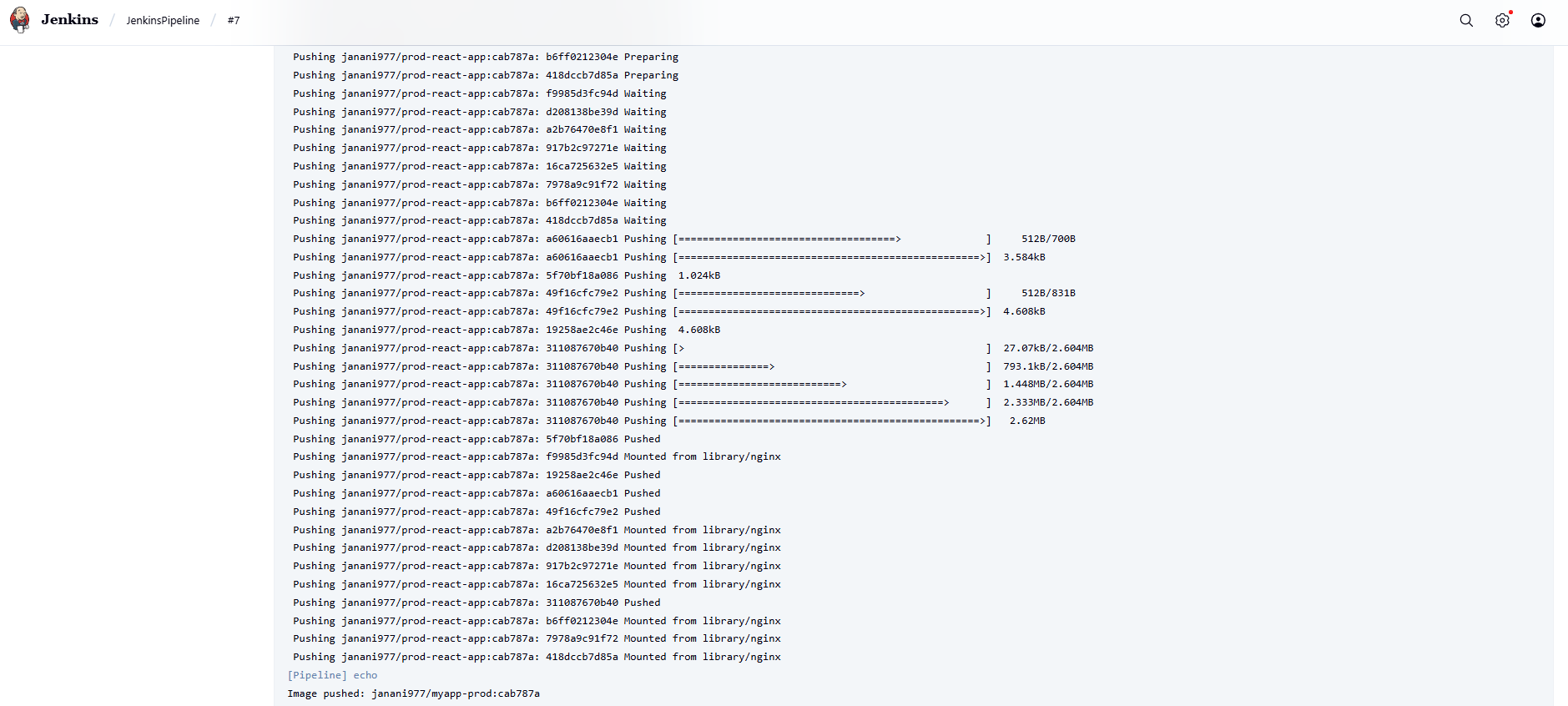


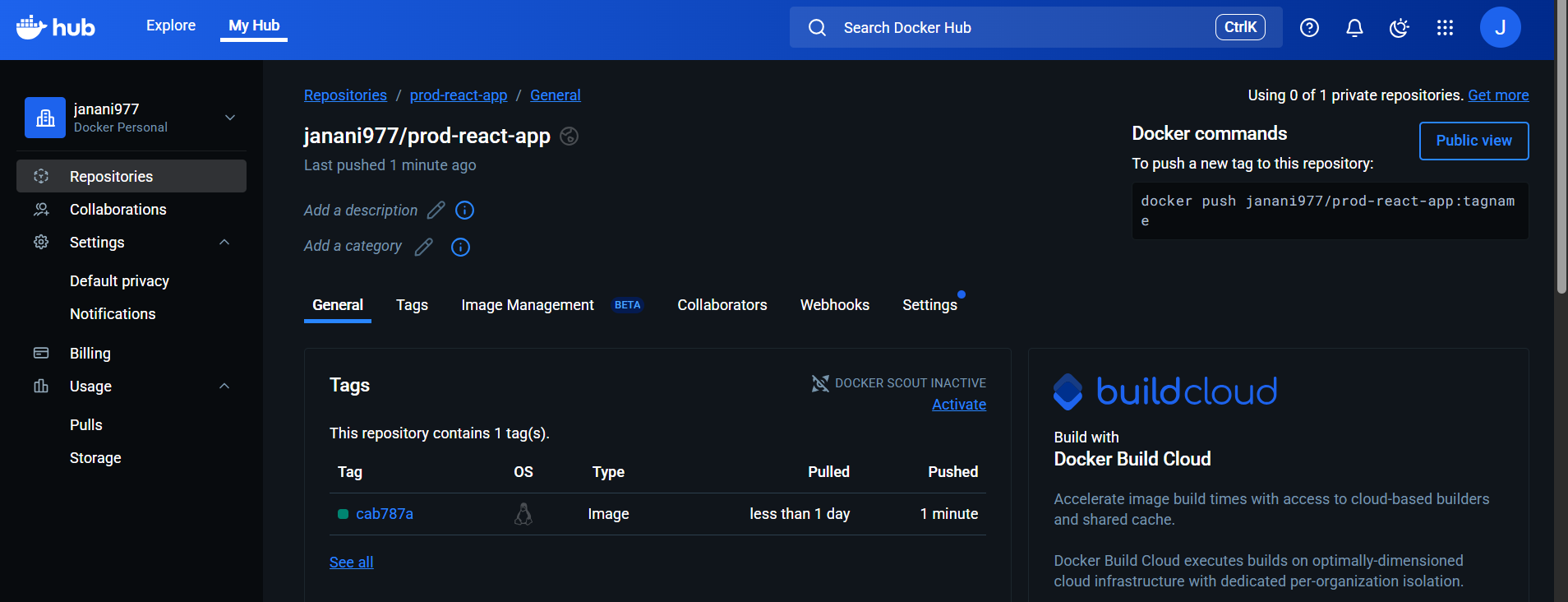
Providing the required permissions by adding the Jenkins user to docker user group. And restarted the Jenkins to reflect the changes.





DockerHub:

Docker image has been successfully pushed to docker hub repo.  
  


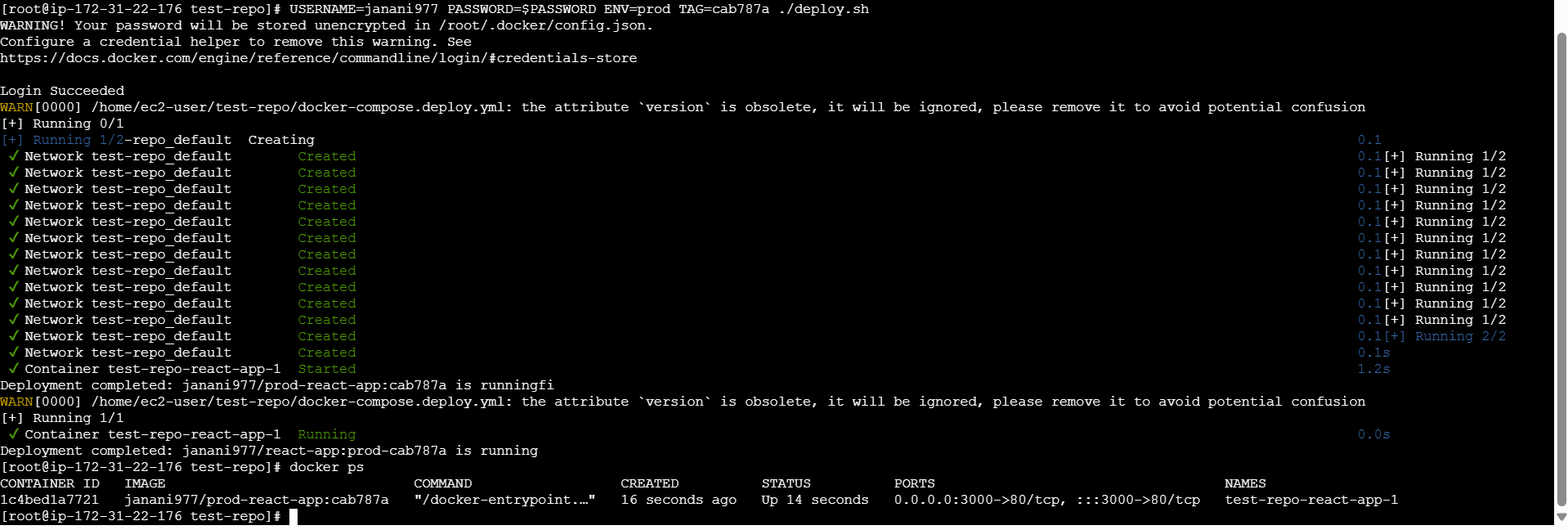


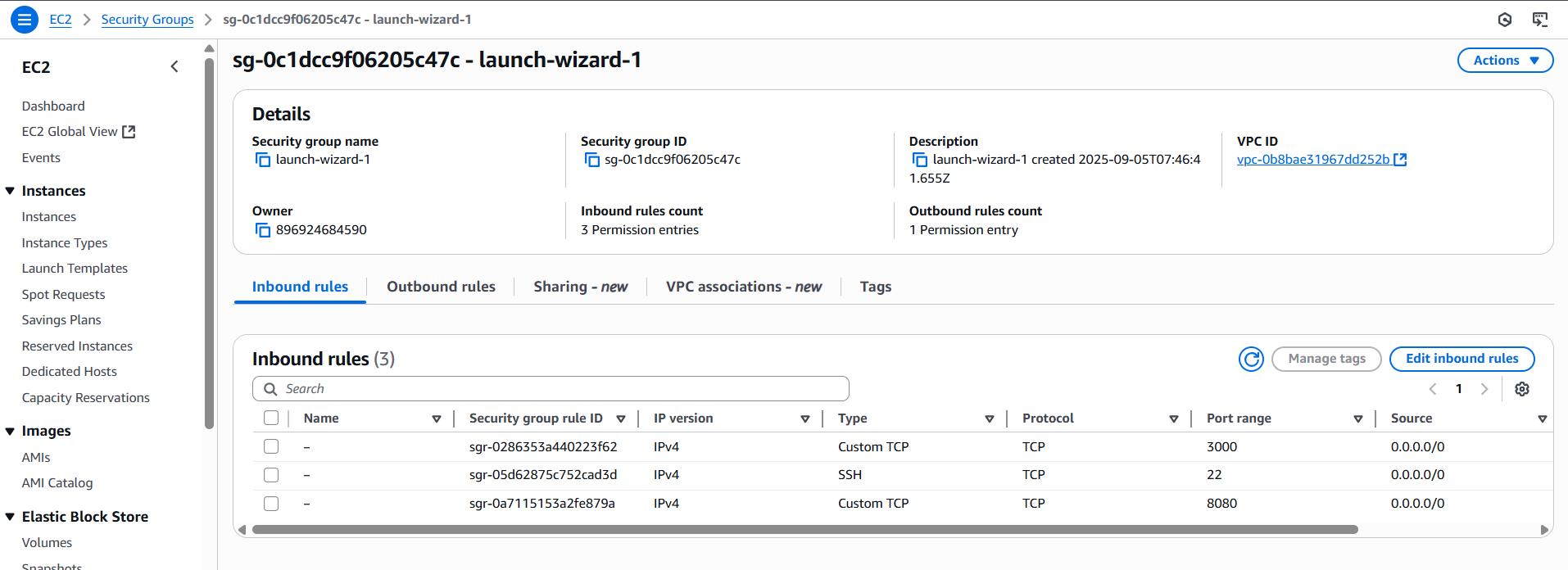
AWS:

Docker:

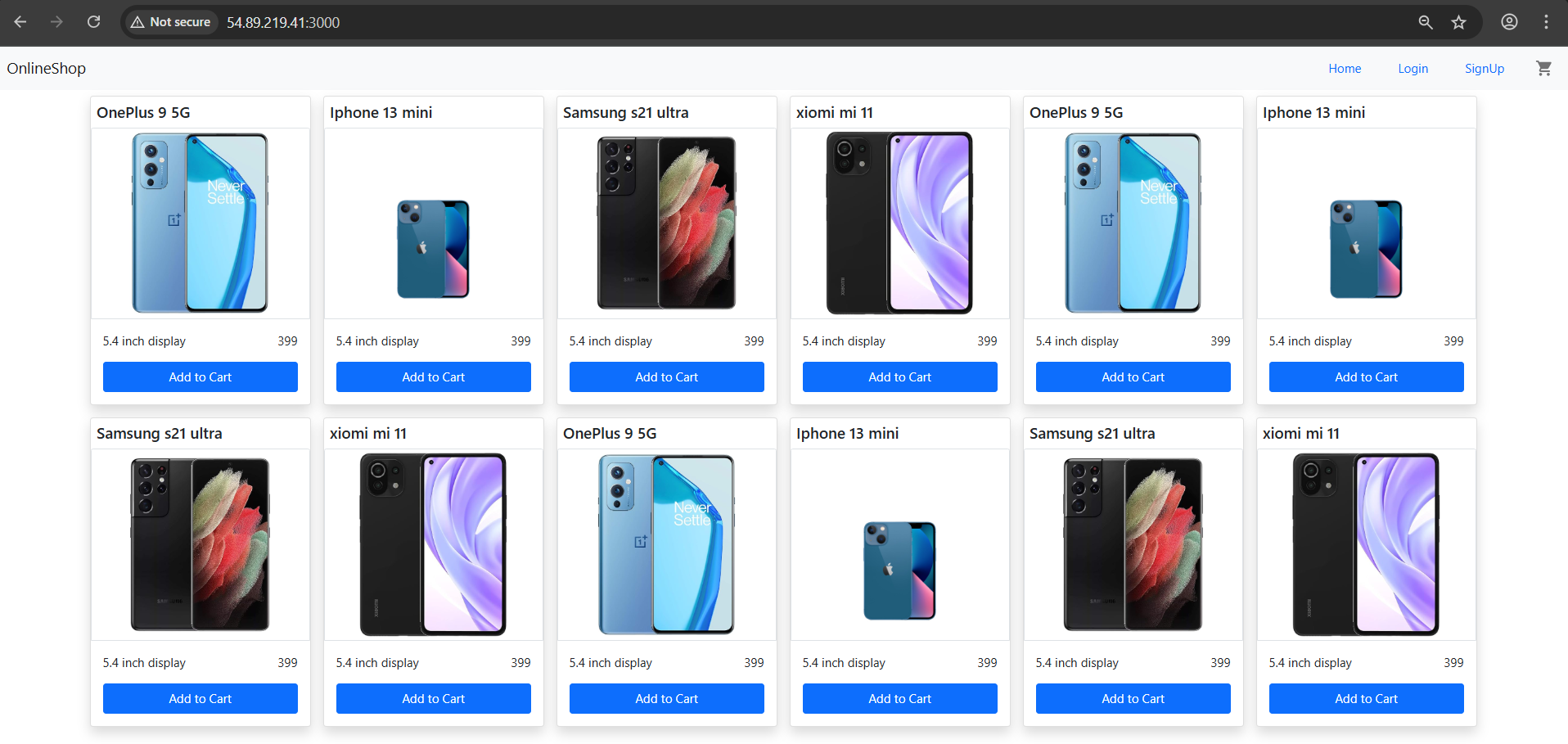
Deployed application with the docker image using docker compose.



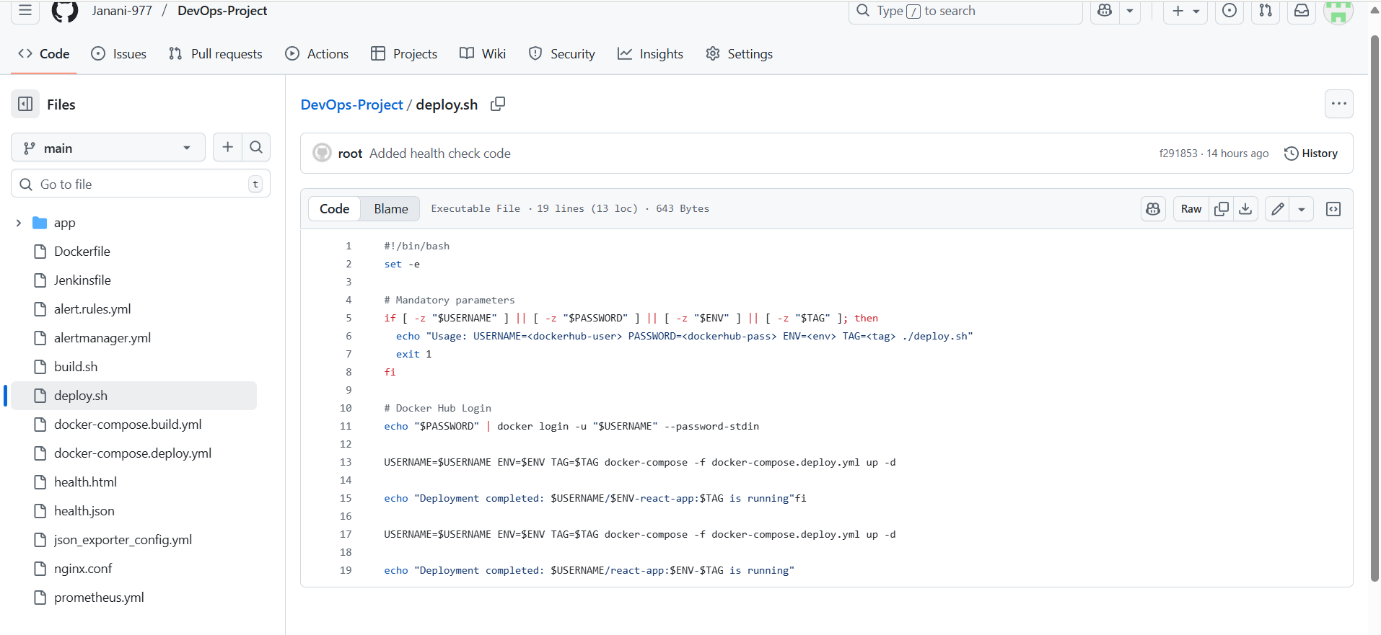




Able to view the application successfully in port 3000



Github:



Monitoring:

**Setting up health check for the application**

Building docker image with the application code

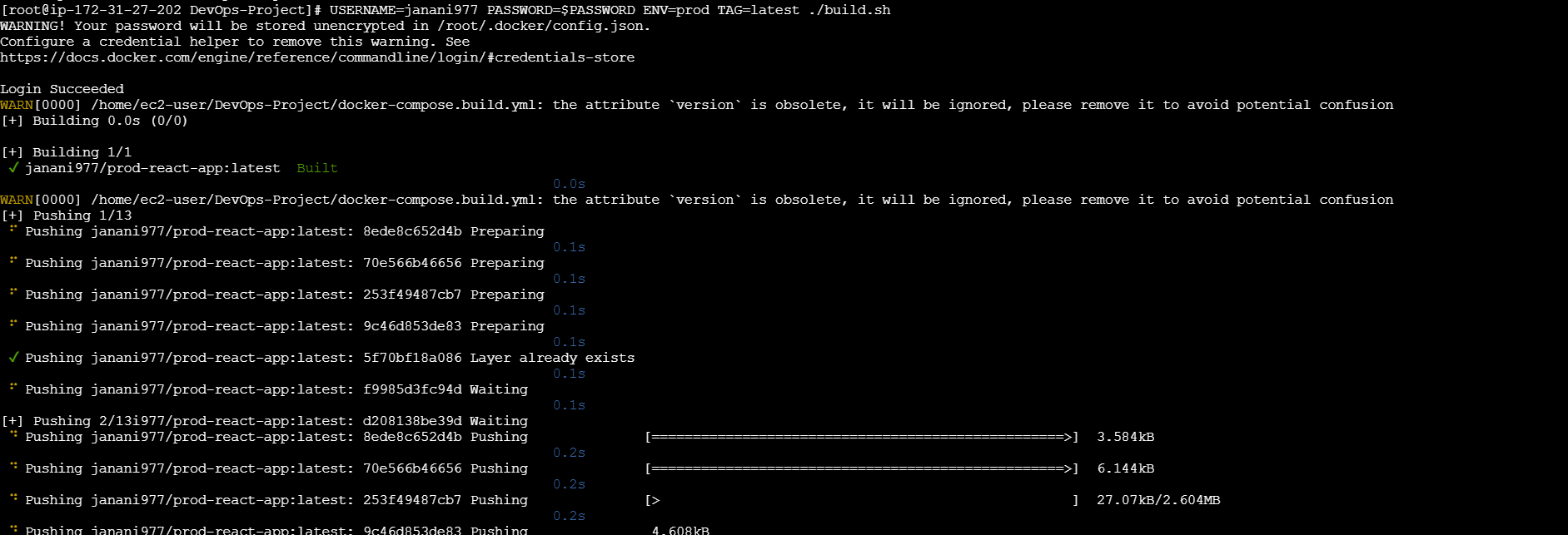
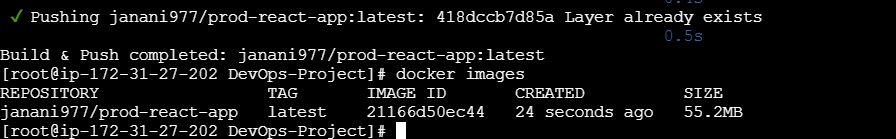
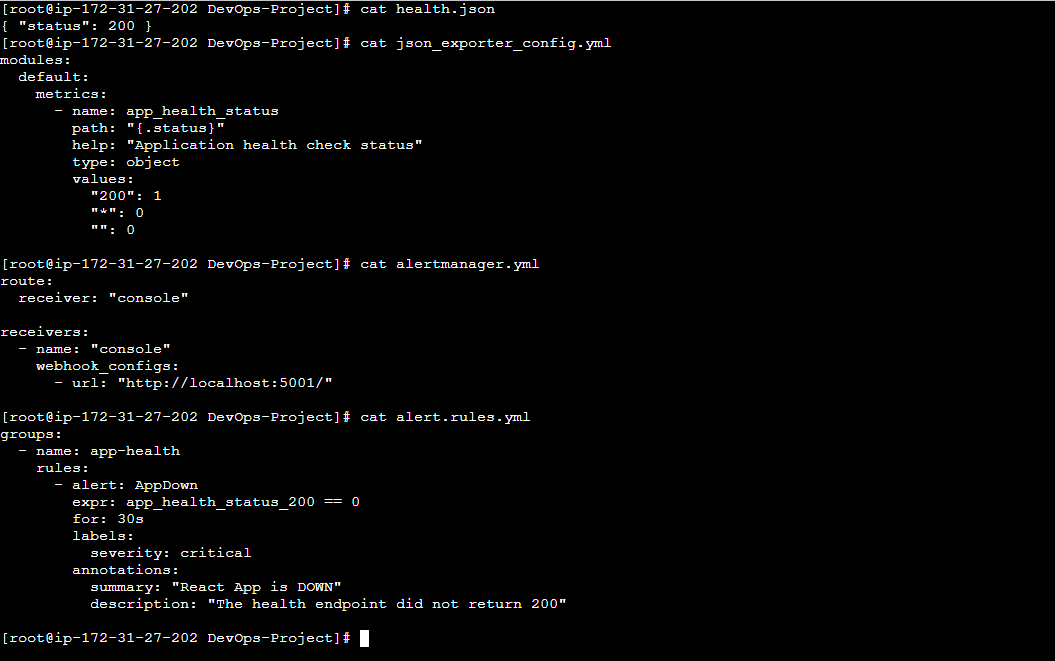


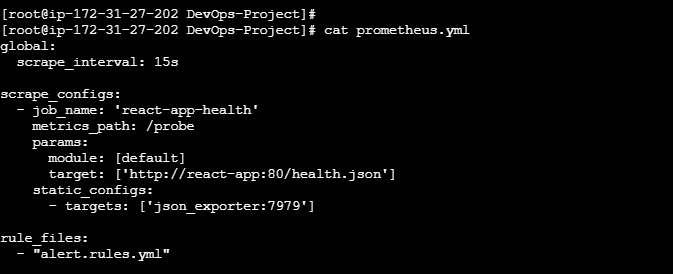
Image has been build using [build.sh](http://build.sh) file



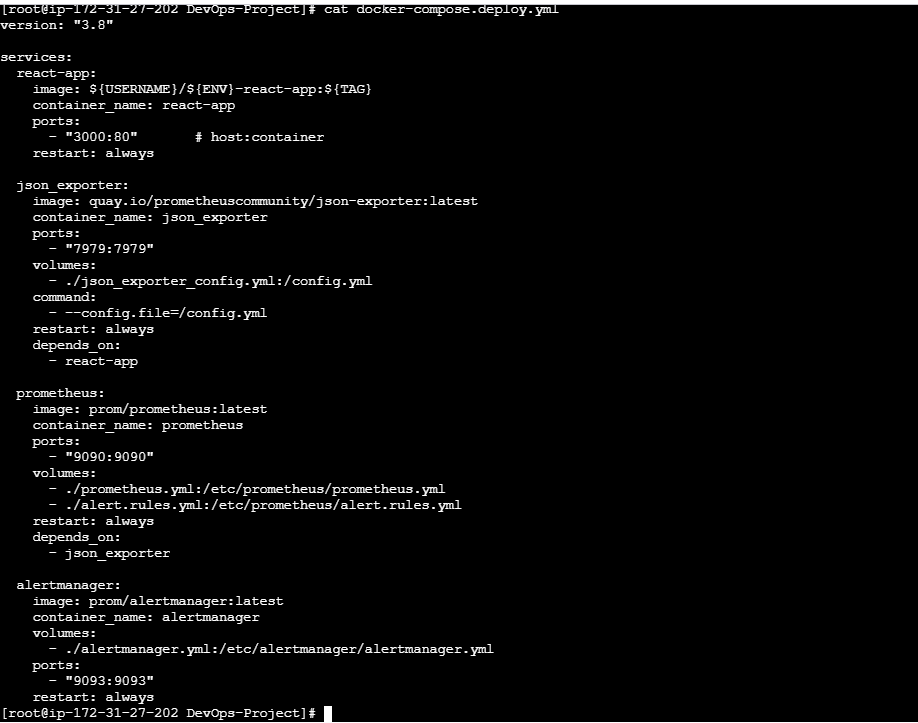
We are setting up a health check endpoint using the health.json file, which needs to be transformed into 0 and 1 values. Here, 0 indicates that the health check is failing, and 1 indicates that it is passing, so that Prometheus can properly interpret the data provided by the exporter. To achieve this, we are using the JSON exporter to transform the health check details. Additionally, we are configuring alerts in Prometheus such that if the status code is 0, an alert is triggered with a severity level of critical and an annotation message stating that the React application is down.



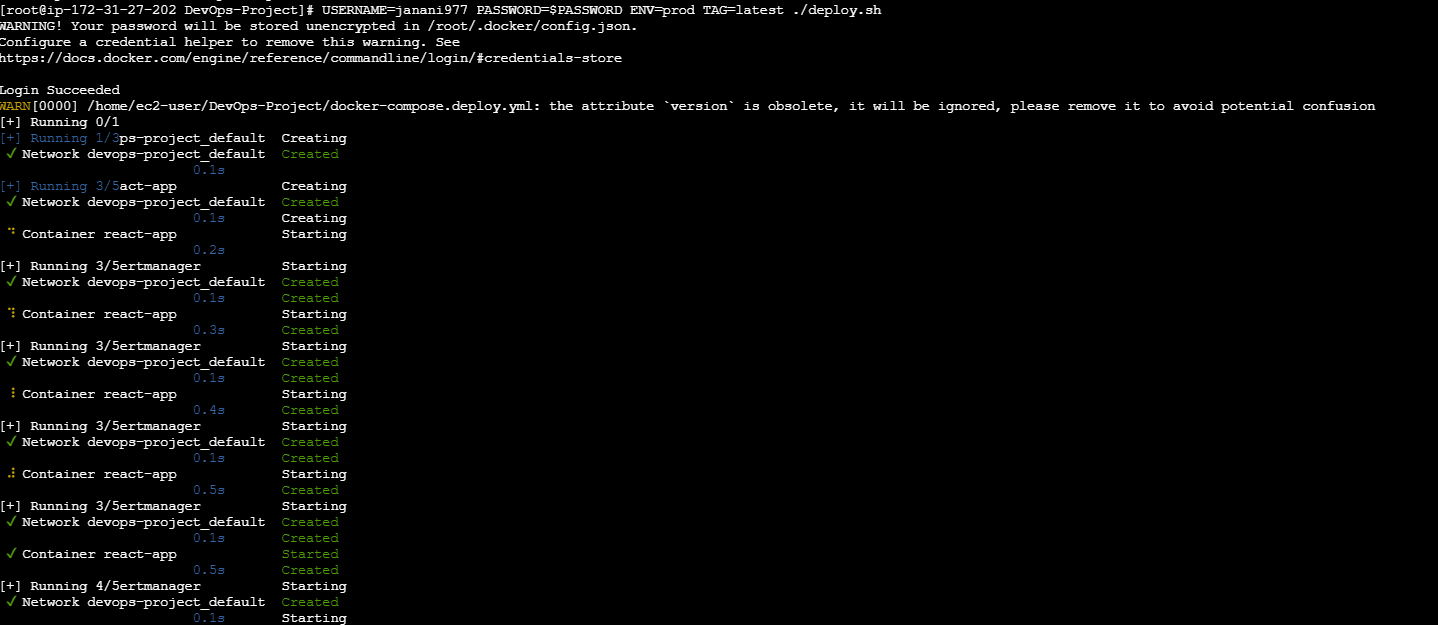
Prometheus will scrape data from the JSON exporter, and alerting rules are added in the configuration file.



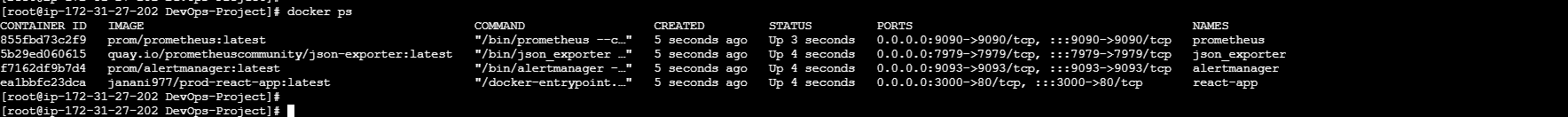
Docker Compose is used to mount the configuration and host the service on specific port numbers.



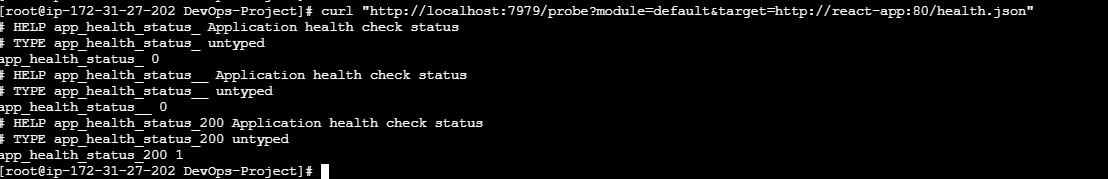
Deploying the application and metrics tool for the application



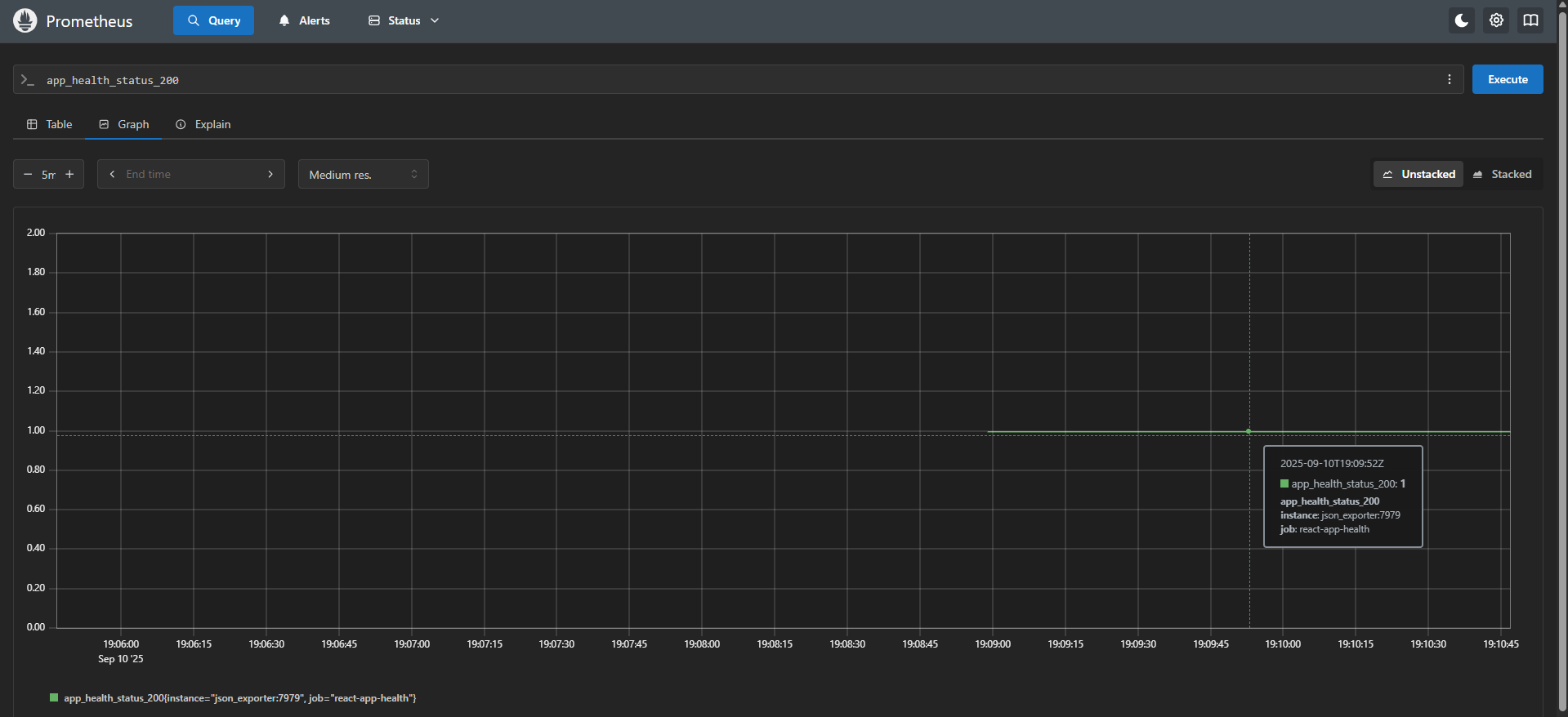
Containers were up and running



Hit the probe to check the health status, and it is passing.



The same metrics are also visible in Prometheus.



An alert was also added, indicating that the health check passed 55 seconds ago.

