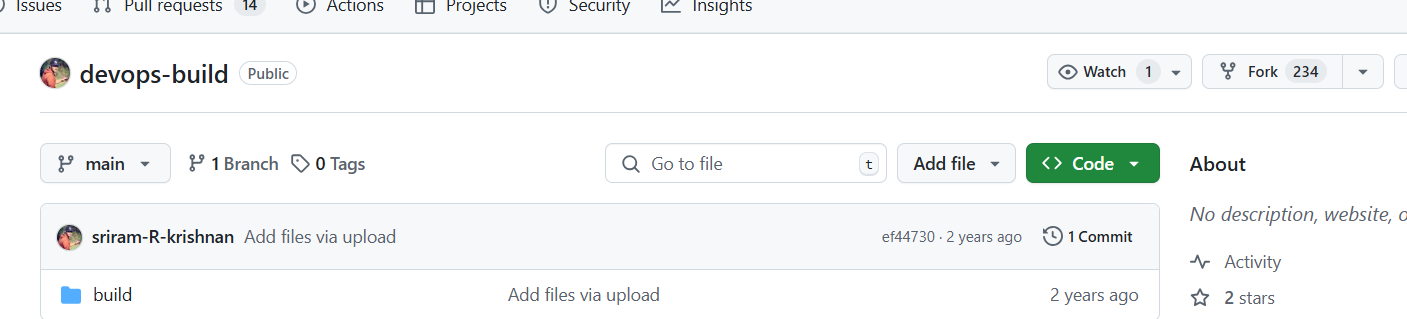
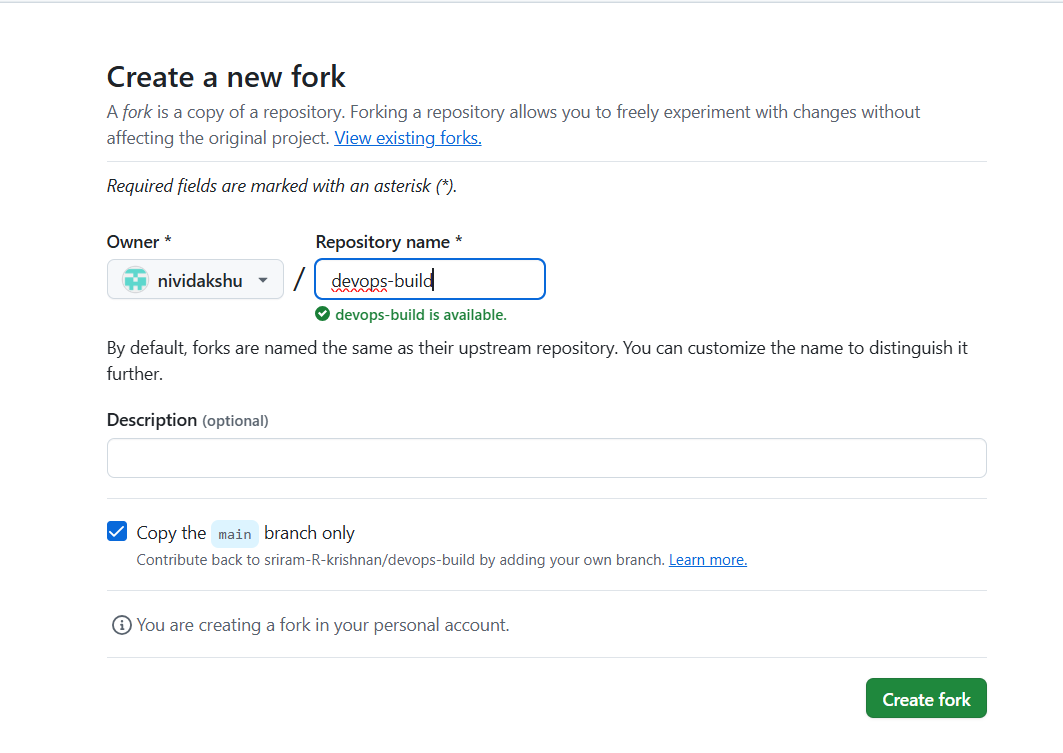
**Application Deployment**

*(Deploy the given React application to a production-ready state)*

**Step-1:**

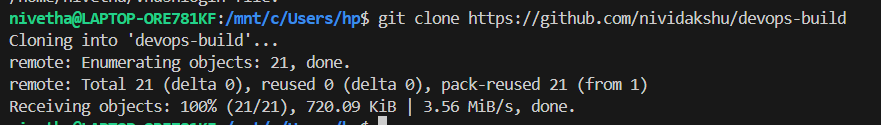
Create fork of the given repo : [sriram-R-krishnan/devops-build](https://github.com/sriram-R-krishnan/devops-build)





Now clone the github repository,

gitclone <https://github.com/nividakshu/devops-build>



cd devops-build

**Step-2:** Create Dockerfile and Dockercompose.yaml file

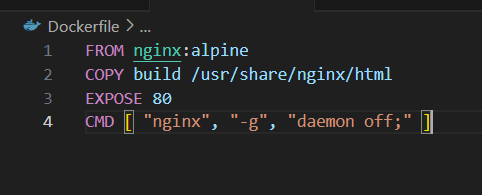
Dockerfile

FROM nginx:alpine

COPY build /usr/share/nginx/html

EXPOSE 80

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



* The -g, is used to override the default nginx configuration.
* Normally nginx will run in the background (daemon mode) without docker , but since we are using in docker, the container must keep running, so we are keeping the nginx running in the foreground to keep the container alive.

docker-compose.yaml

version: “3.8”

services:

  react-app:

    build: .

    container\_name: my-reactapp-container

    restart: always

    ports:

      - "80:80"

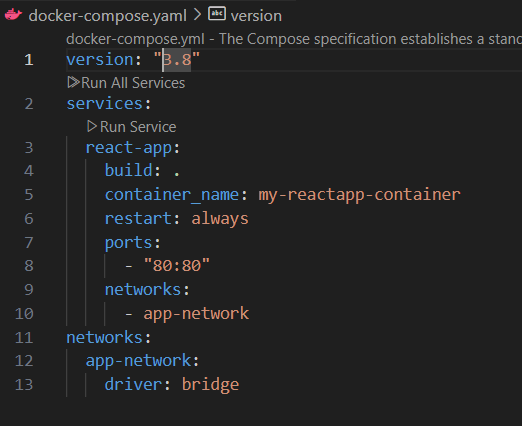
    networks:

      - app-network

networks:

  app-network:

    driver: bridge



**Step-3:** create build.sh (dev and prod) ; deploy.sh (dev and prod)

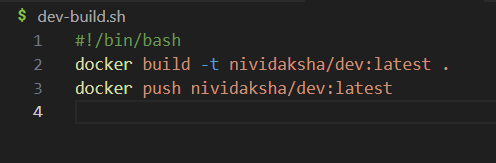
dev-build.sh

#!/bin/bash

docker build -t nividaksha/dev:latest .

docker push nividaksha/dev:latest

chmod +x dev-build.sh



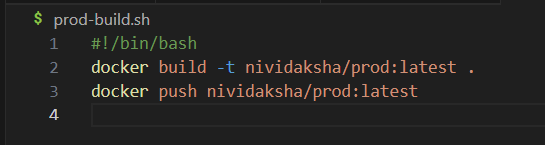
prod-build.sh

#!/bin/bash

docker build -t nividaksha/prod:latest .

docker push nividaksha/prod:latest

chmod +x prod-build.sh



dev-deploy.sh

#!/bin/bash

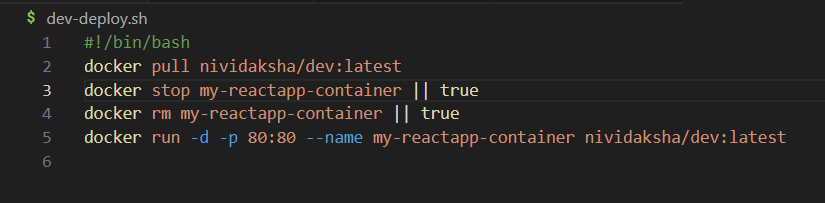
docker pull nividaksha/dev:latest

docker stop my-reactapp-container || true

docker rm my-reactapp-container || true

docker run -d -p 80:80 --name my-reactapp-container nividaksha/dev:latest

chmod +x dev-deploy.sh



prod-deploy.sh

#!/bin/bash

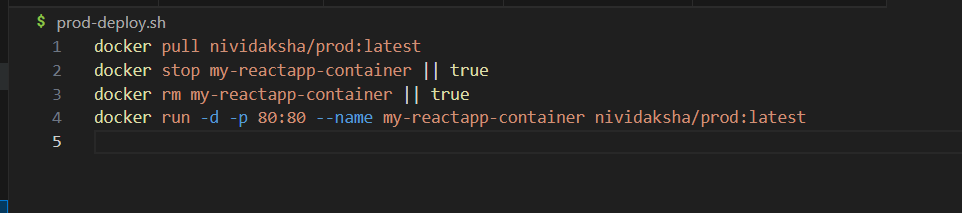
docker pull nividaksha/prod:latest

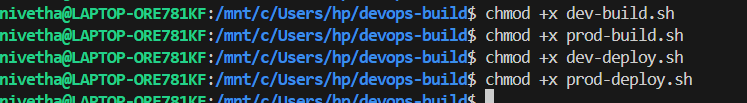
docker stop my-reactapp-container || true

docker rm my-reactapp-container || true

docker run -d -p 80:80 --name my-reactapp-container nividaksha/prod:latest

chmod +x prod-deploy.sh



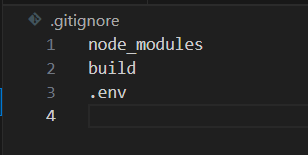


.gitignore

node\_modules

build

.env

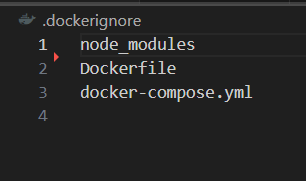


.dockerignore

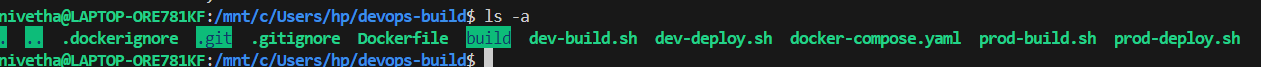
node\_modules

Dockerfile

docker-compose.yml



ls -a



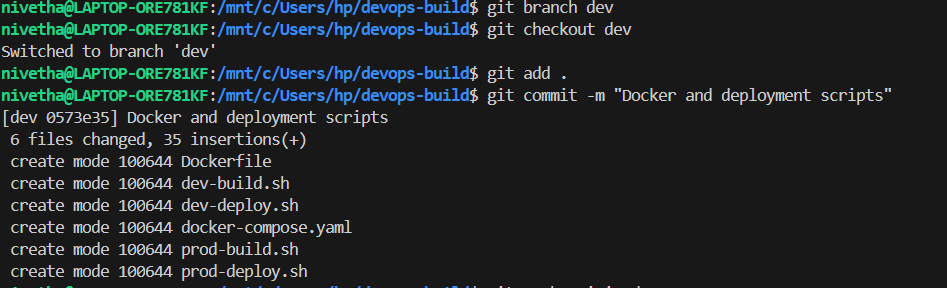
**Step-4:** push the files to github repo dev branch

git branch dev

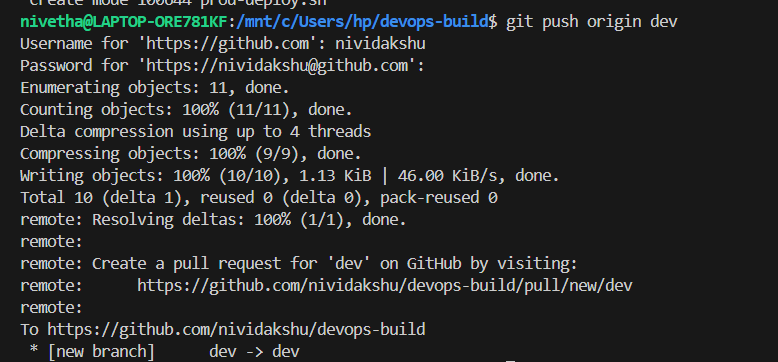
git checkout dev

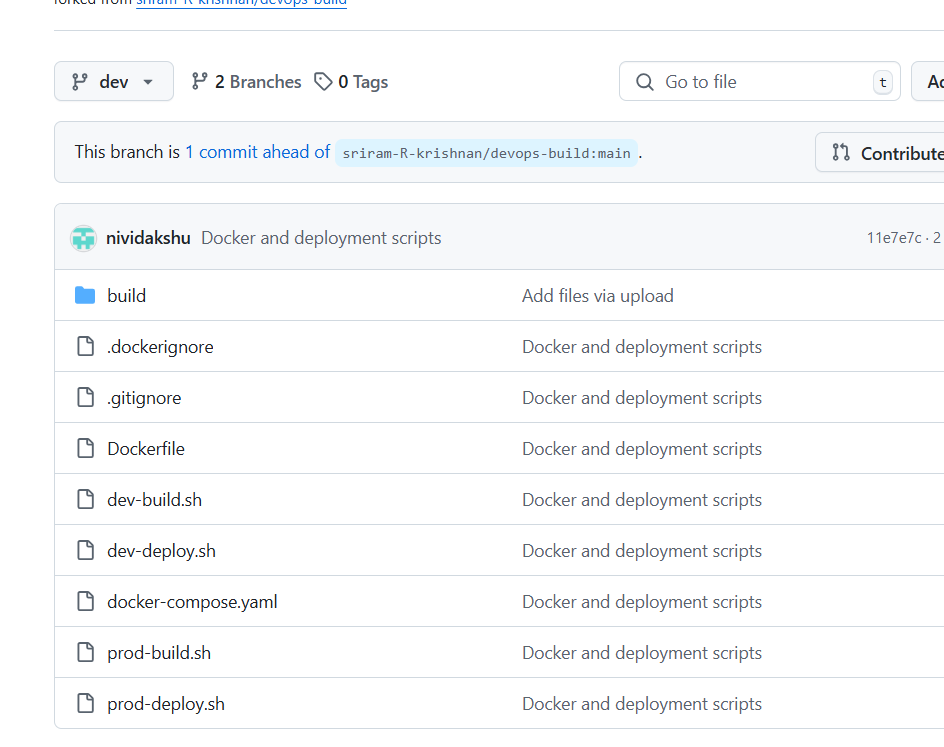
git add .

git commit -m “docker and deployment scripts”

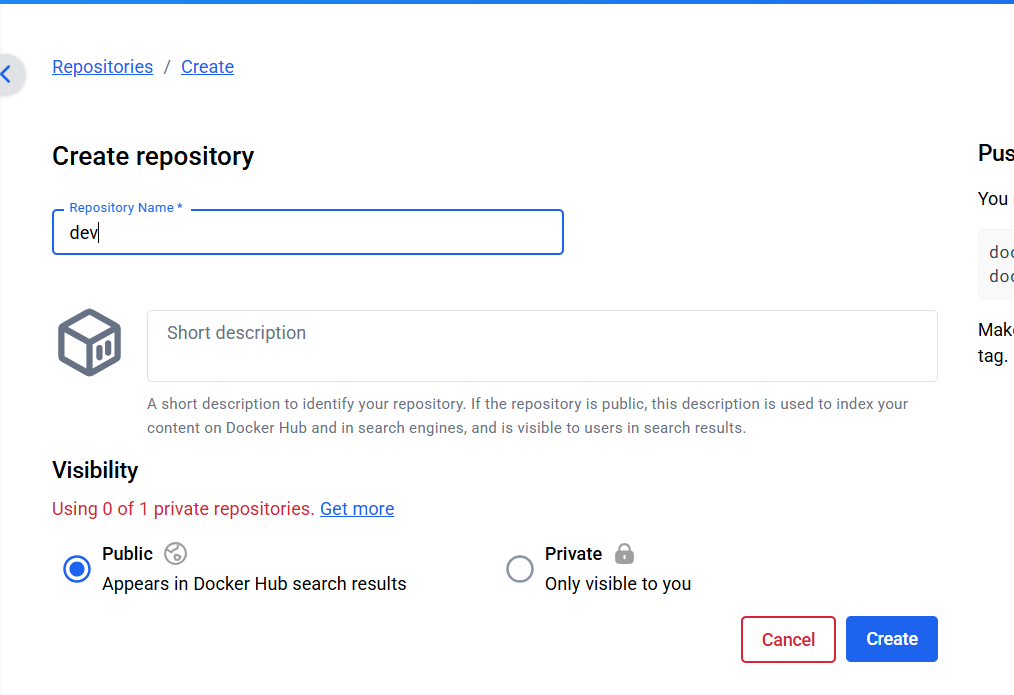


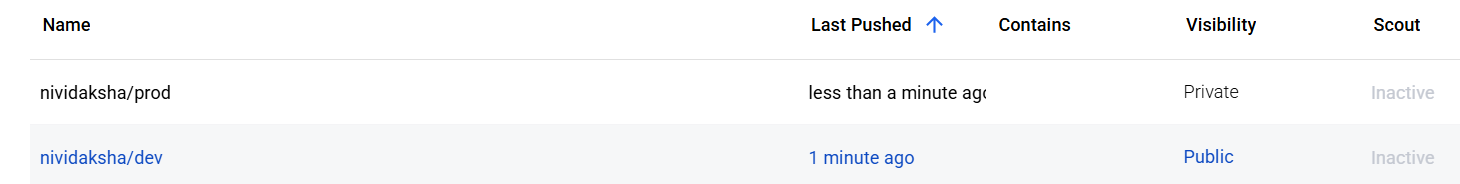
git push origin dev



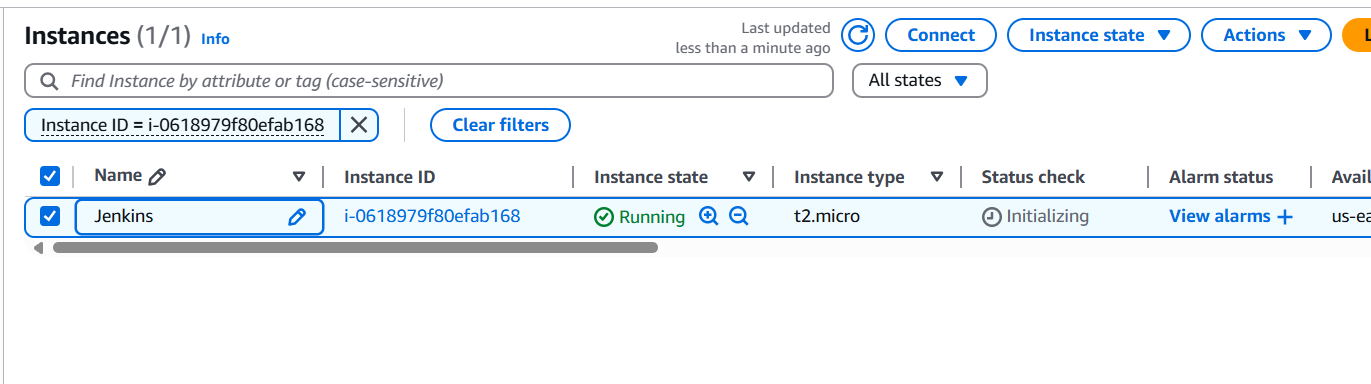


**Step-5:** In docker hub, create repo dev – public ; prod – private

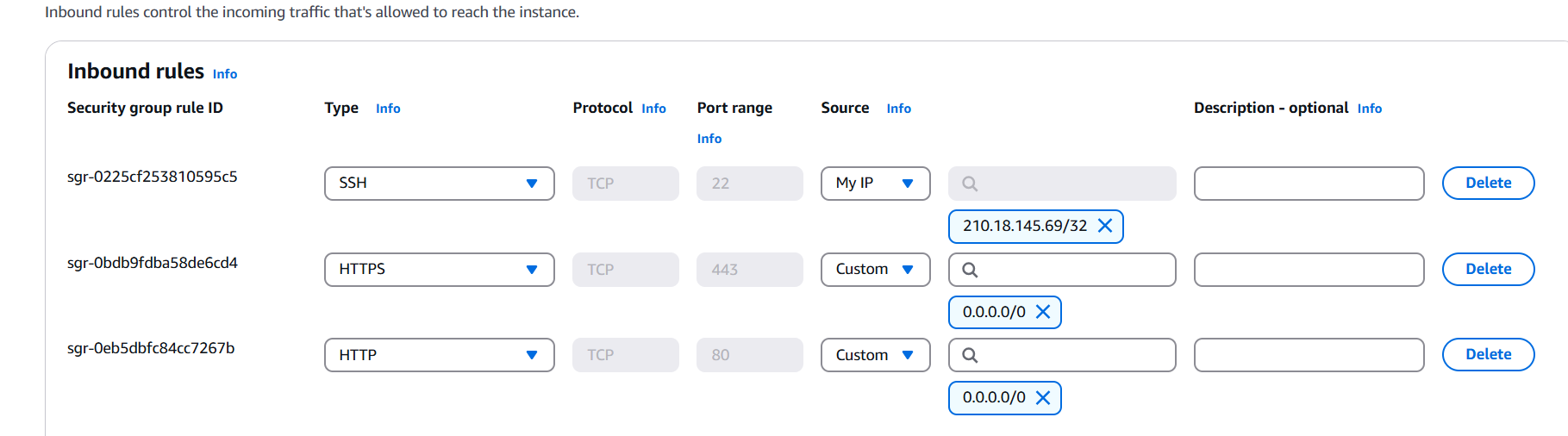




**Step-6:** Launch ec2 instance – chosen ubuntu machine and install jenkins



edit inbound rules as below



connecting via ec2 instance connect

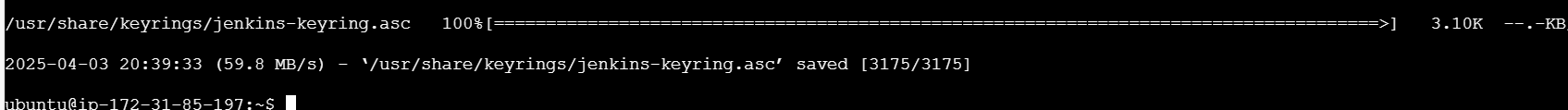
sudo apt update

sudo apt install fontconfig openjdk-17-jre



sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key



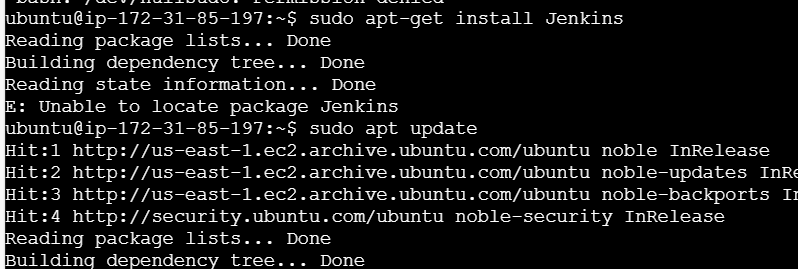
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list **>** /dev/null

sudo apt update

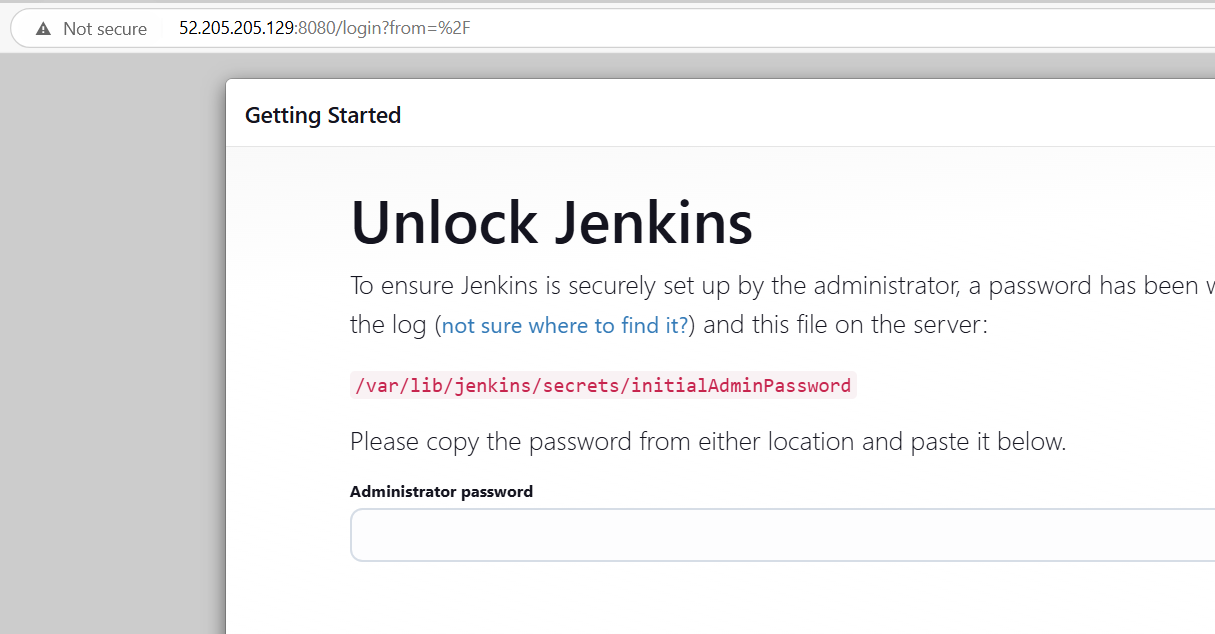
sudo apt-get install Jenkins



sudo systemctl enable jenkins

sudo systemctl start jenkins

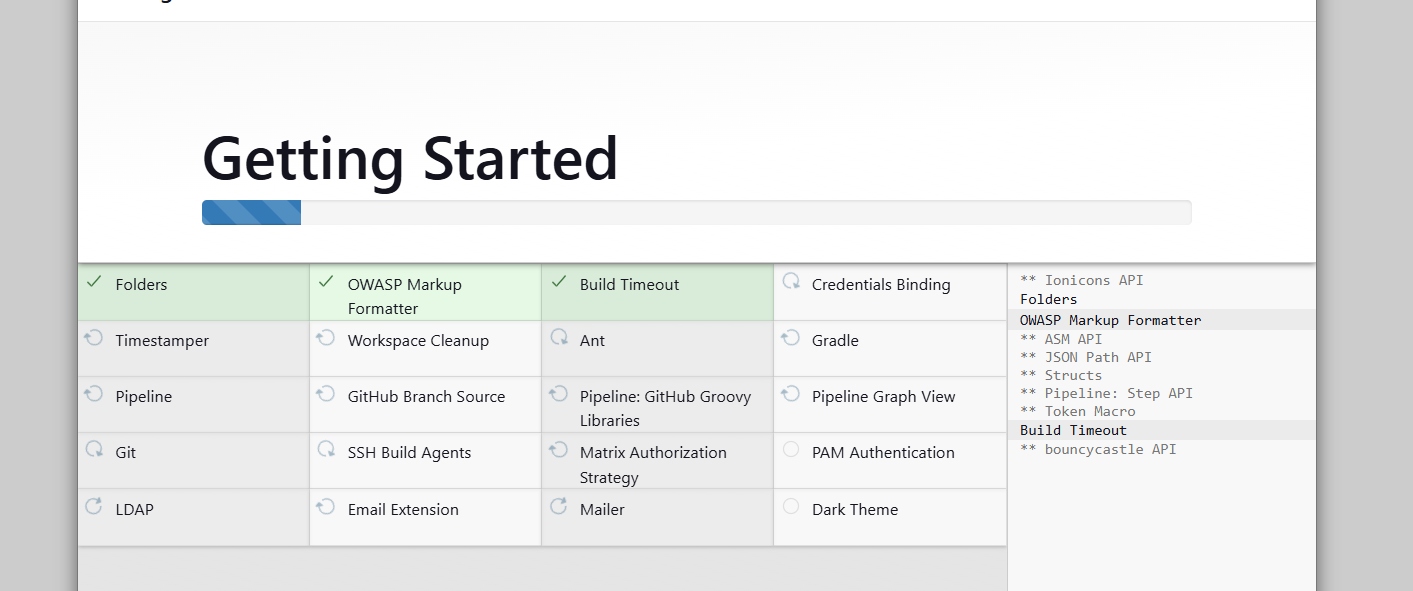
Access Jenkins via <http://52.205.205.129:8080>



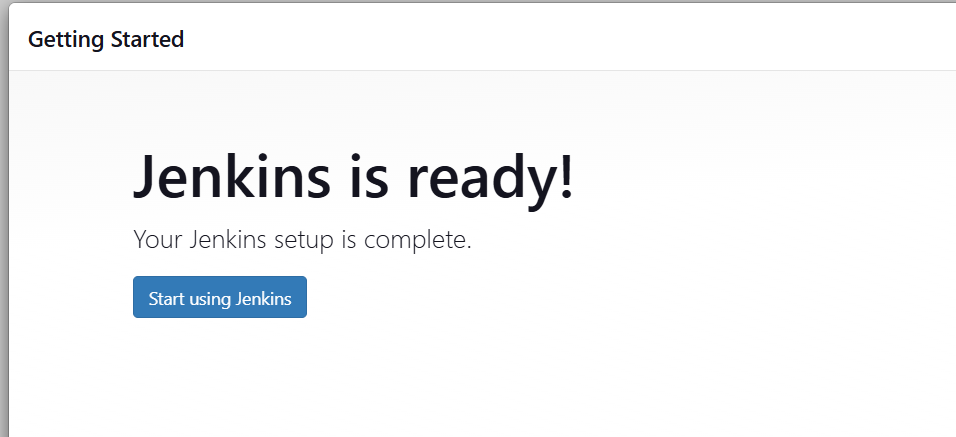
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



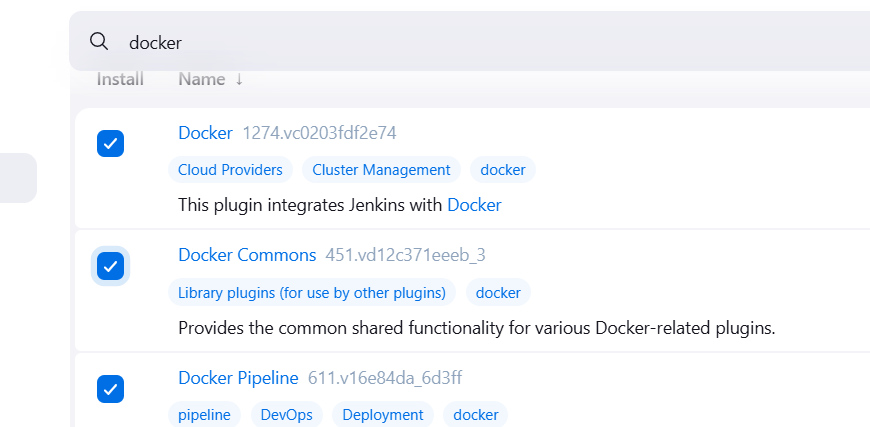
copy paste the initial admin pw



give the credentials

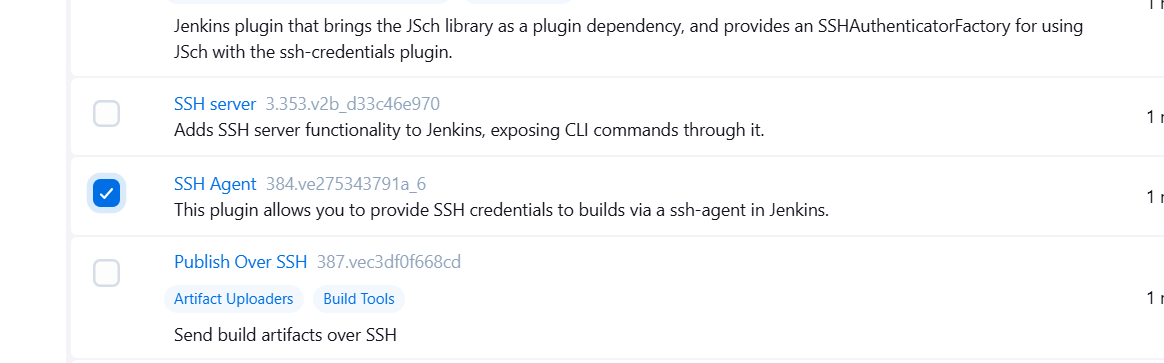


Install docker plugin >> manage Jenkins >> plugin >>Available plugins

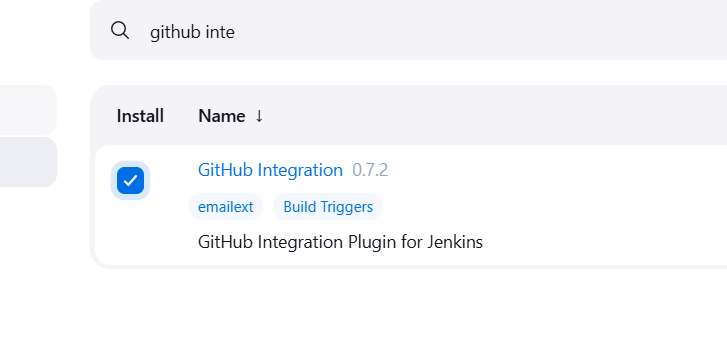


Check whether git, pipeline is installed or install it.

Install SSH Agent

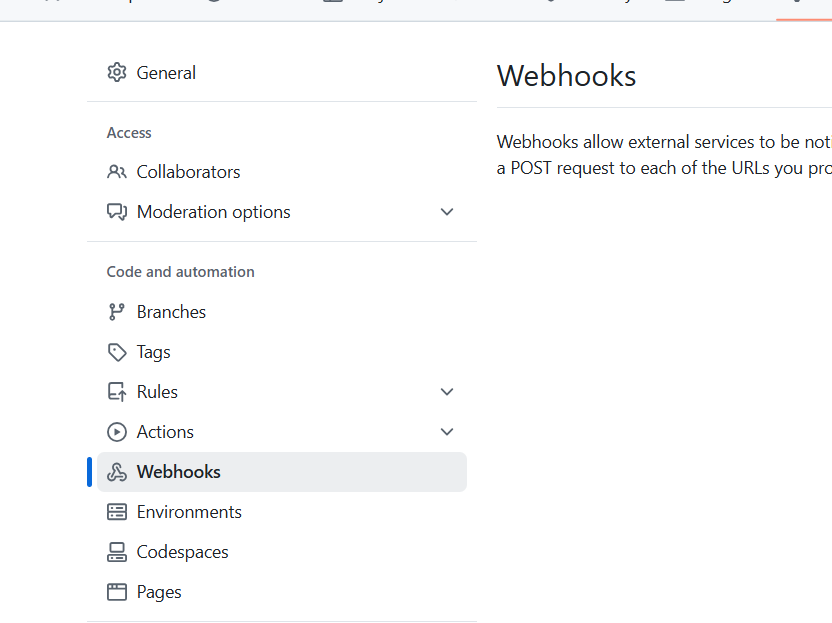


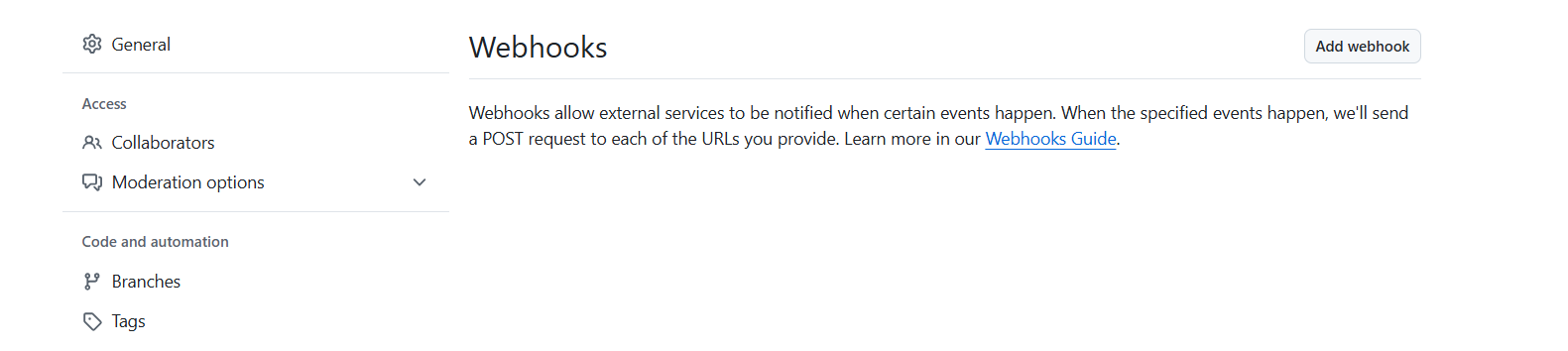
Install Github integration

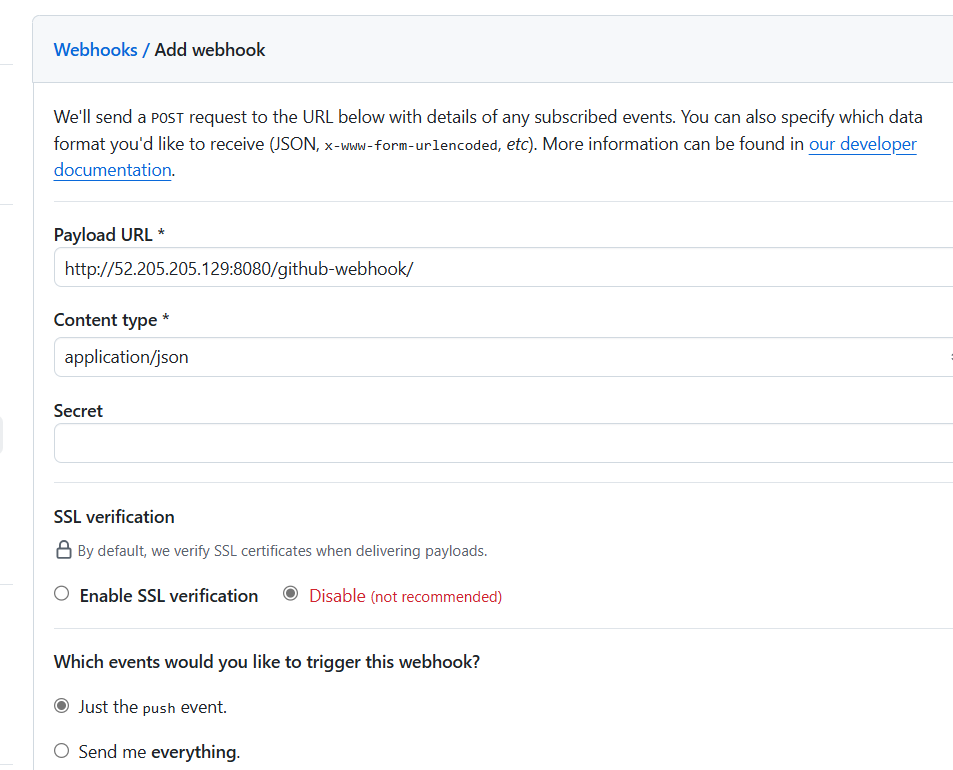
****

**Step 7:** add the webhook in the github so that when push occurs in the git, webhook trigger Jenkins to automate the process.

Goto githubrepo > settings > webhooks>add webhook>Jenkins url > application/json>just the push event



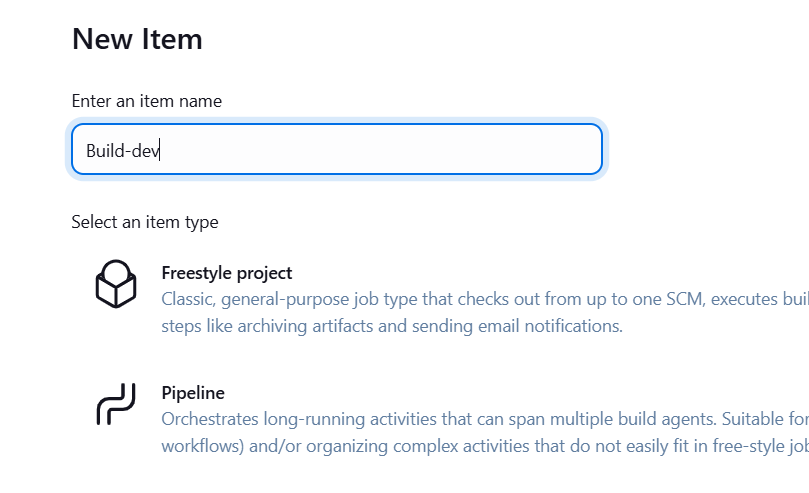




**Step 8:** Create jobs in Jenkins, 4 jobs (Build-dev, Build-prod, Deploy-dev, Deploy-prod)

* so when the dev repo in the git pushes the code, jenkins triggers the build-dev job and it build the image and push it to dev repo in the docker hub. Deploy-dev will be added in post build actions of Build-dev, so when the Build-dev job runs successfully, the Deploy-dev, will pull the pushed image from docker hub dev repo and deploys to ec2.
* The same thing happens when the dev branch is merged with main and push the code, Build prod will run and the upstream job is Deploy-prod.

1. build-dev

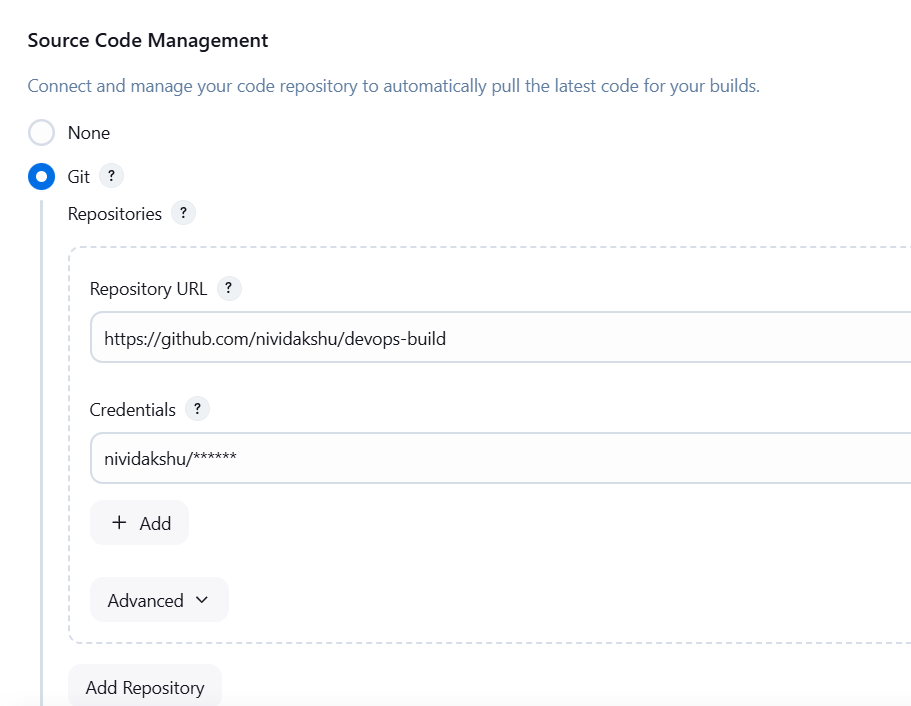


Freestyle Project → Name it: Build-Dev

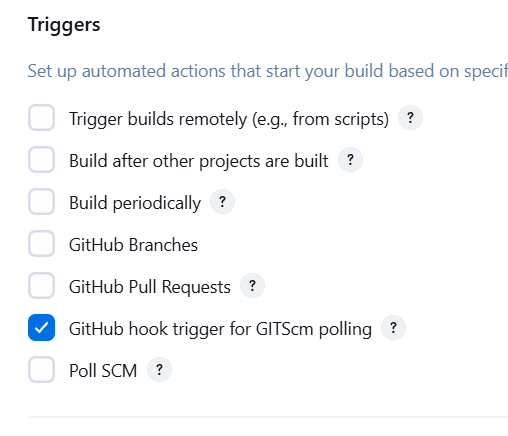
Source Code Management → Git

Repo URL: https://github.com/nividakshu/devops-build

Branch: dev



Triggers - Check GitHub hook trigger for GITScm polling



Build Steps → Execute Shell

docker login -u nividaksha -p Nivetha@1234

chmod +x dev-build.sh

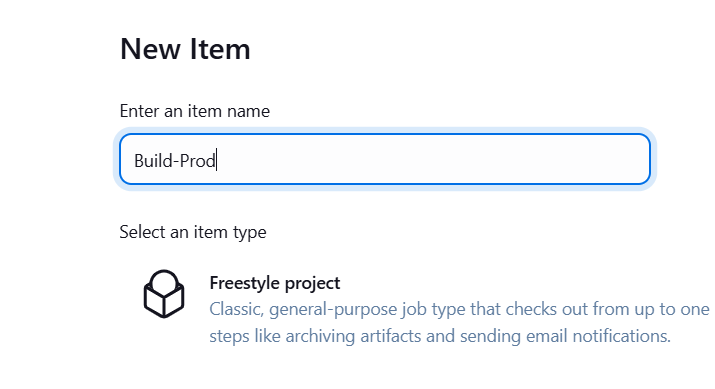
./dev-build.sh



Save

1. build prod

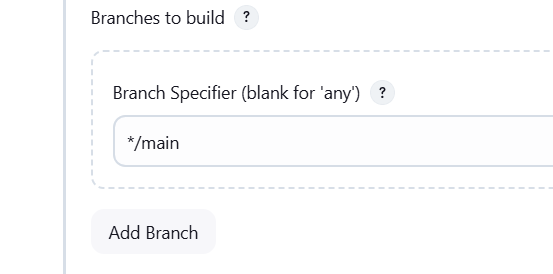
Freestyle Project → Name it: Build-Prod



Source Code Management → Git

Repo URL: https://github.com/nividakshu/devops-build

Branch: main



Build Triggers

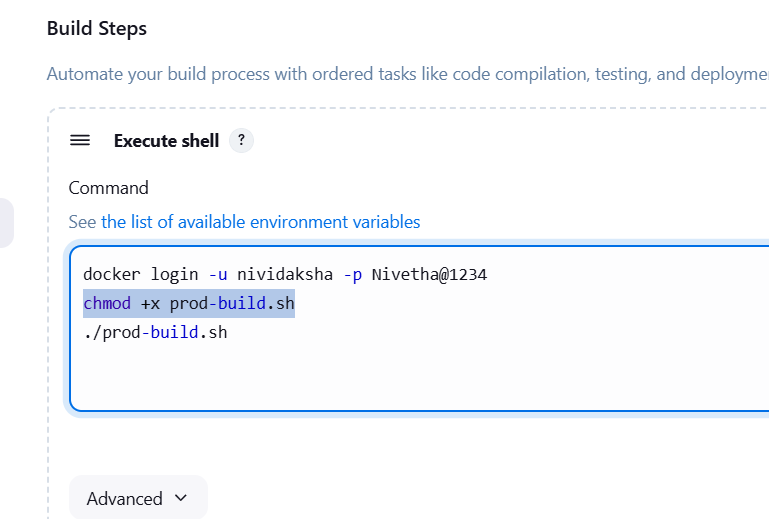
Check GitHub hook trigger for GITScm polling

Build Steps → Execute Shell

docker login -u nividaksha -p Nivetha@1234

chmod +x prod-build.sh

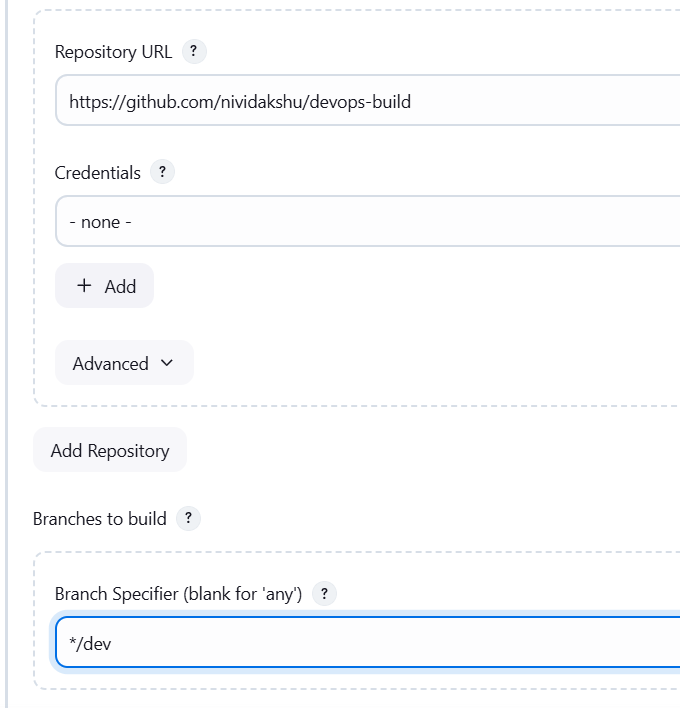
./prod-build.sh



1. Deploy dev to EC2

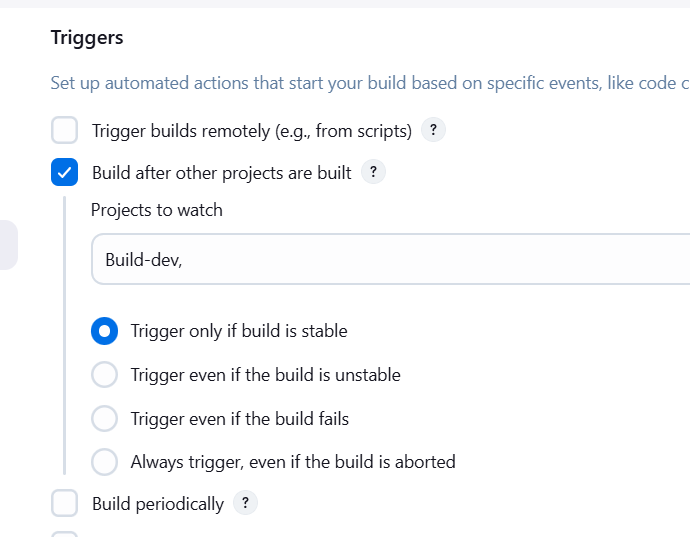
Freestyle Project → Name it: Deploy-dev





Build Triggers → Build after other projects are built

Choose Build-Dev



Build Steps → Execute Shell

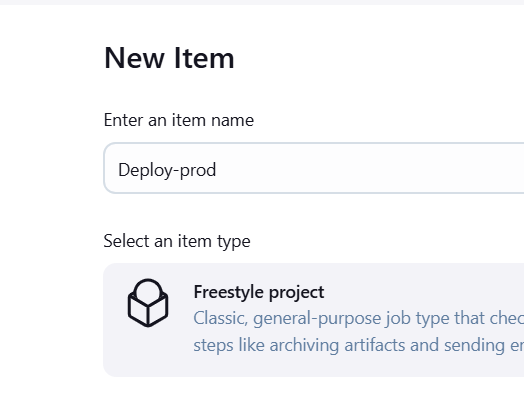
chmod +x dev-deploy.sh

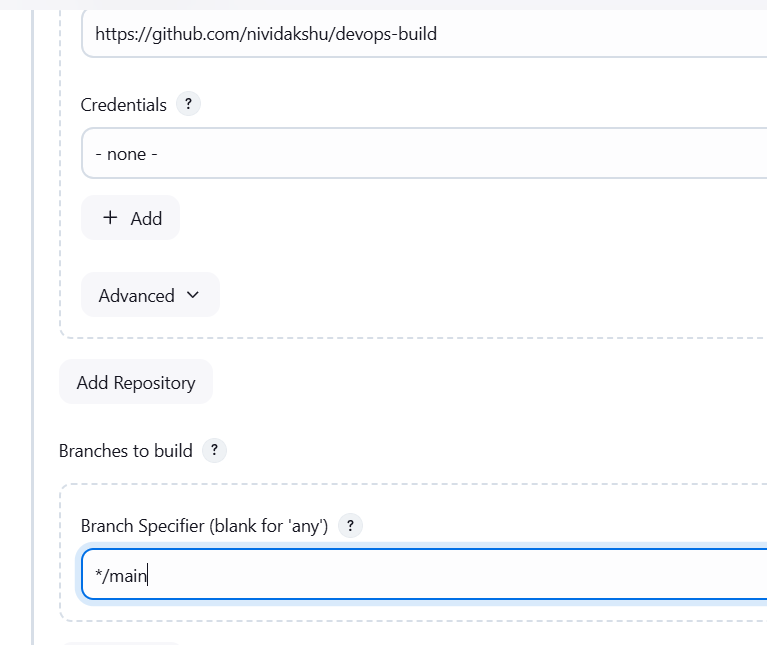
ssh -i ~/.ssh/Nivetha.pem -o StrictHostKeyChecking=no ubuntu@52.205.205.129 bash -s < dev-deploy.sh



1. Deploy prod to EC2

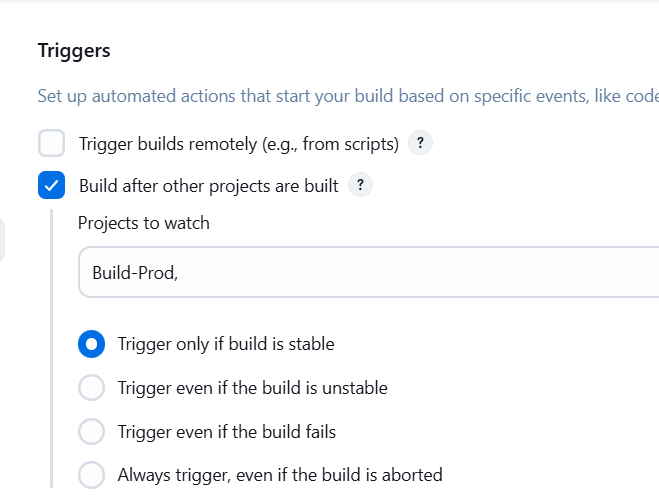
Freestyle Project → Name it: Deploy-Prod





Build Triggers → Build after other projects are built

Choose Build-Prod



Build Steps → Execute Shell

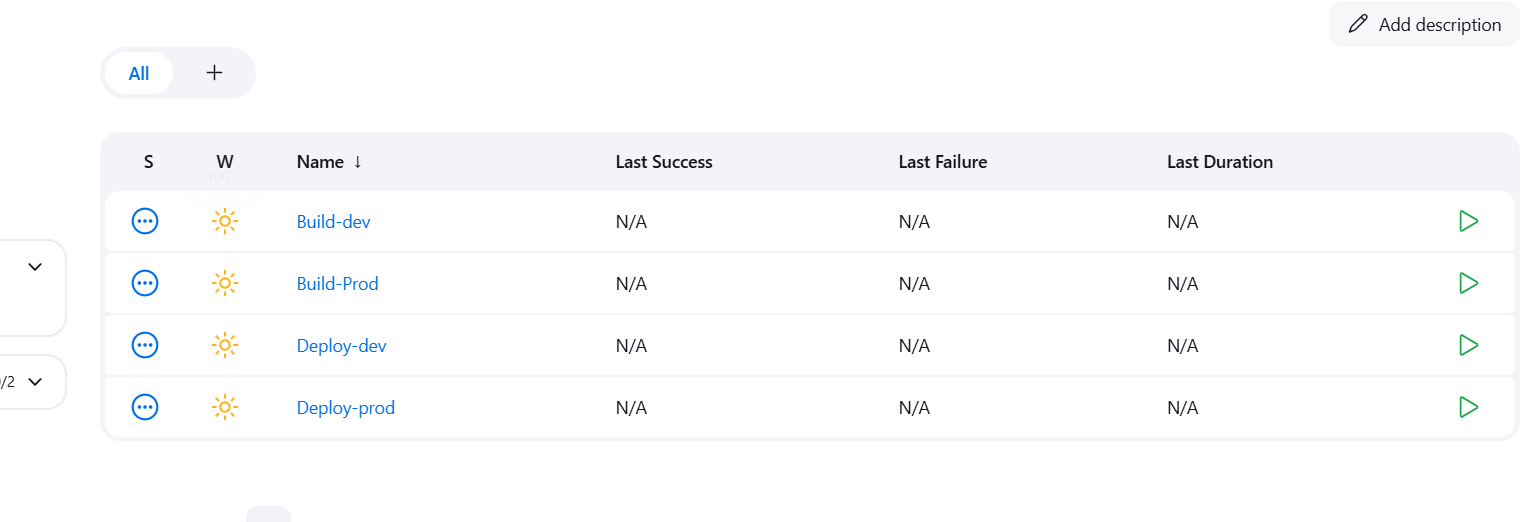
chmod +x prod-deploy.sh

ssh -i ~/.ssh/Nivetha.pem -o StrictHostKeyChecking=no ubuntu@52.205.205.129 bash -s < prod-deploy.sh



save

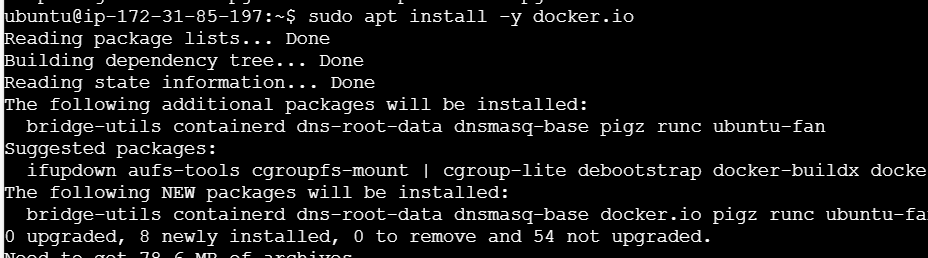
we created 4 jobs



**Step 9:** Install docker on ec2

sudo apt update

sudo apt install -y docker.io



sudo systemctl start docker

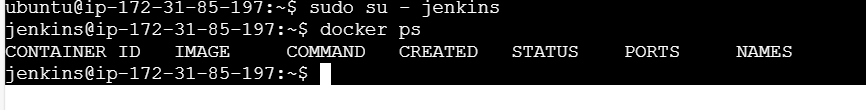
sudo systemctl enable docker



sudo usermod -aG docker Jenkins -- we are creating Jenkins user in docker because Jenkins has to configure docker. docker need sudo command to run but Jenkins usually wont be running as root user so we are creating user.

sudo usermod -aG docker ubuntu

go inside Jenkins and check whether it can access docker



To establish ssh connection,

create directory .ssh



cd .ssh

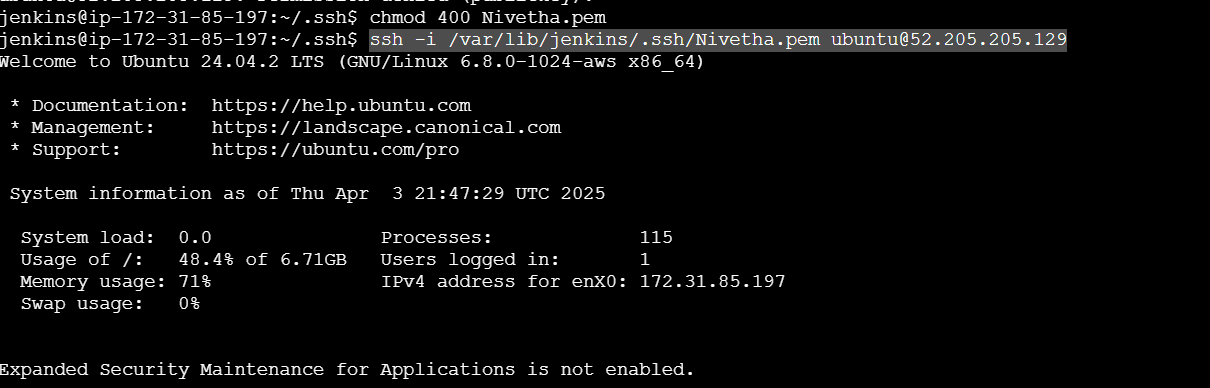
nano Nivetha.pem

copy the content from our pem file and paste it here



chmod 400 Nivetha.pem

then try ssh ssh -i /var/lib/jenkins/.ssh/Nivetha.pem ubuntu@52.205.205.129



Now Jenkins can ssh to ubuntu machine.

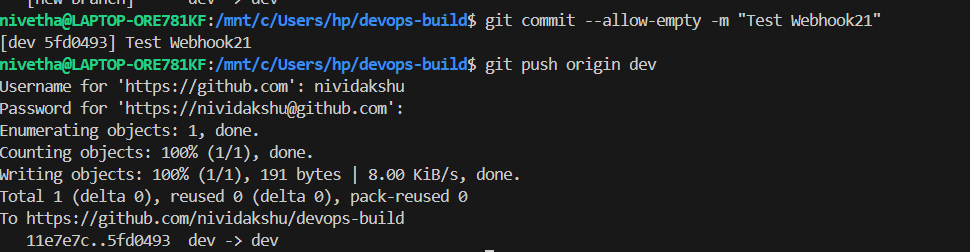
**Step 10:** Verification:

**Test Dev Deployment**

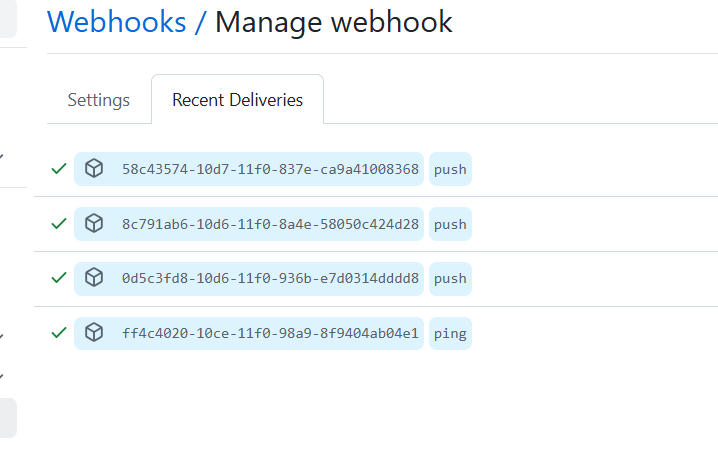
Pushing the code to dev branch

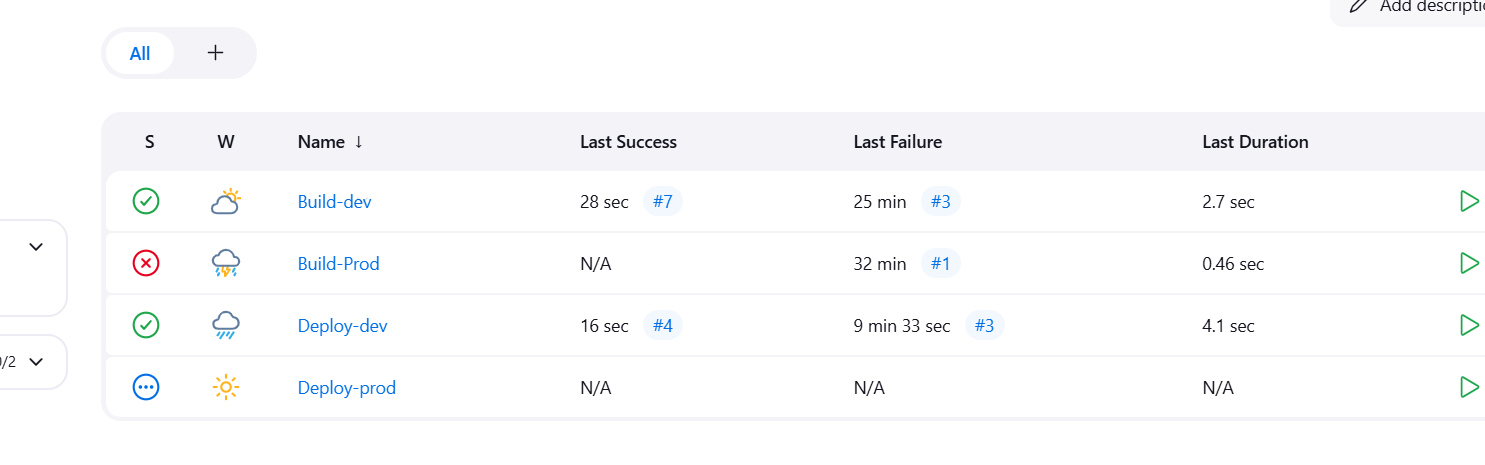
git commit --allow-empty -m "Test Webhook21"

git push origin dev

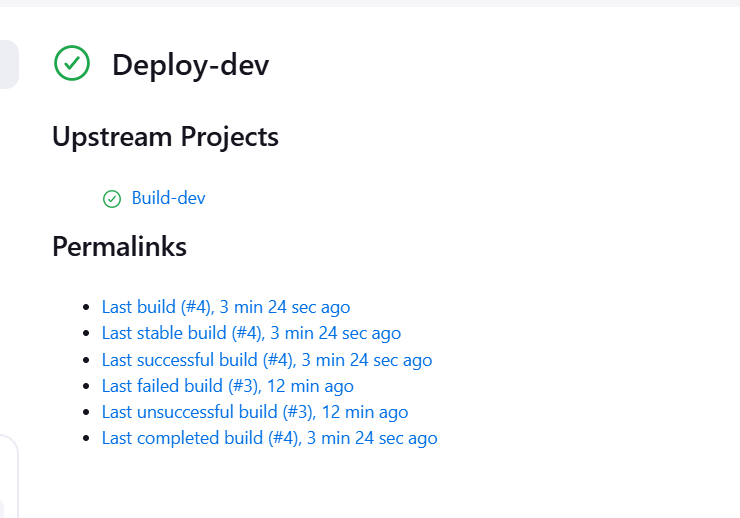


webhook is triggering the push



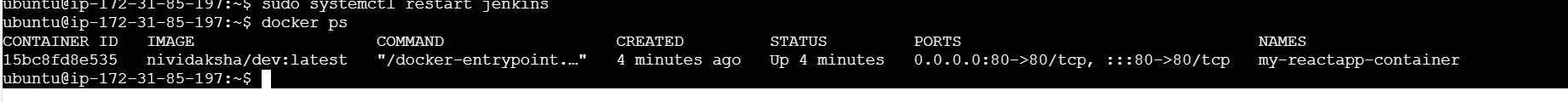


Jenkins is automatically start building the Build-dev job and upon completion, Deploy-dev is triggered.

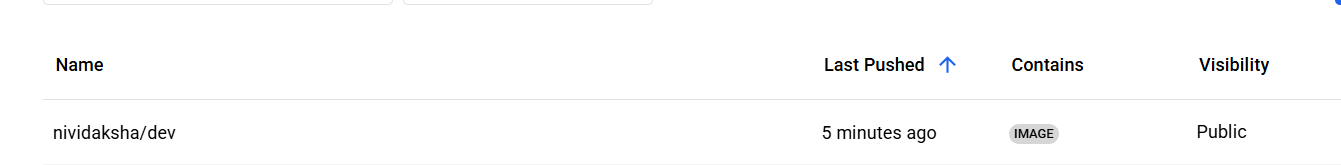


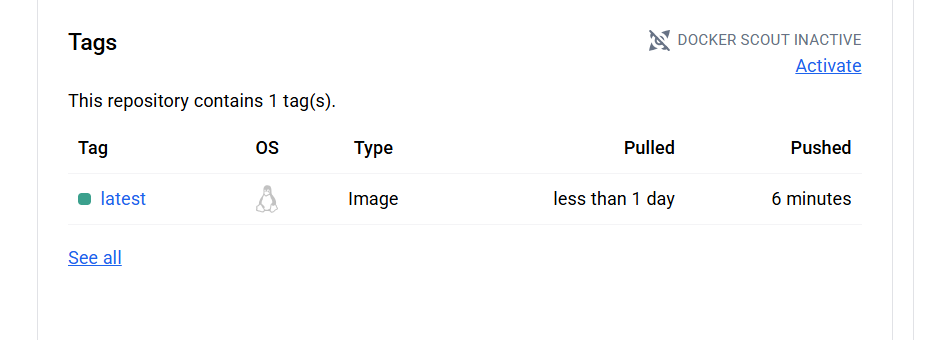


docker ps

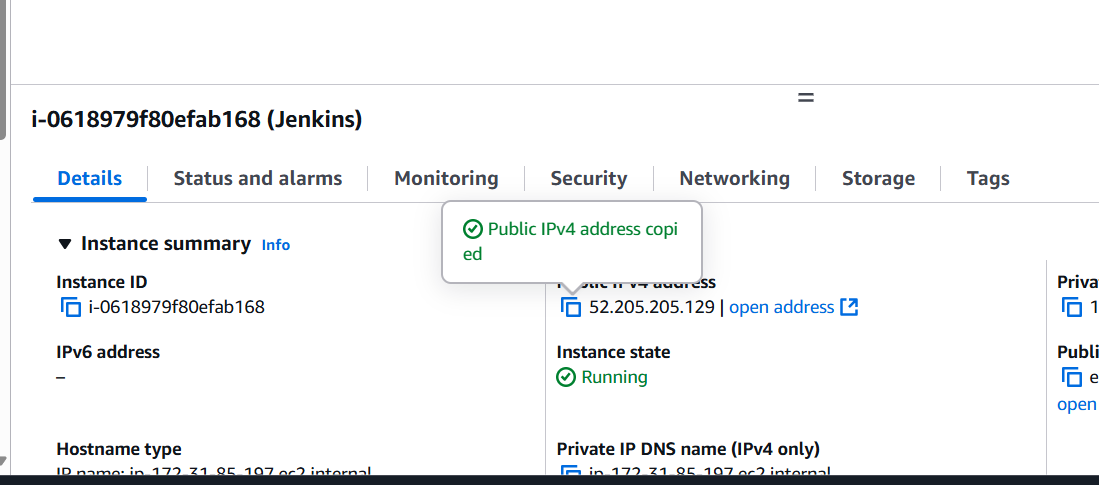


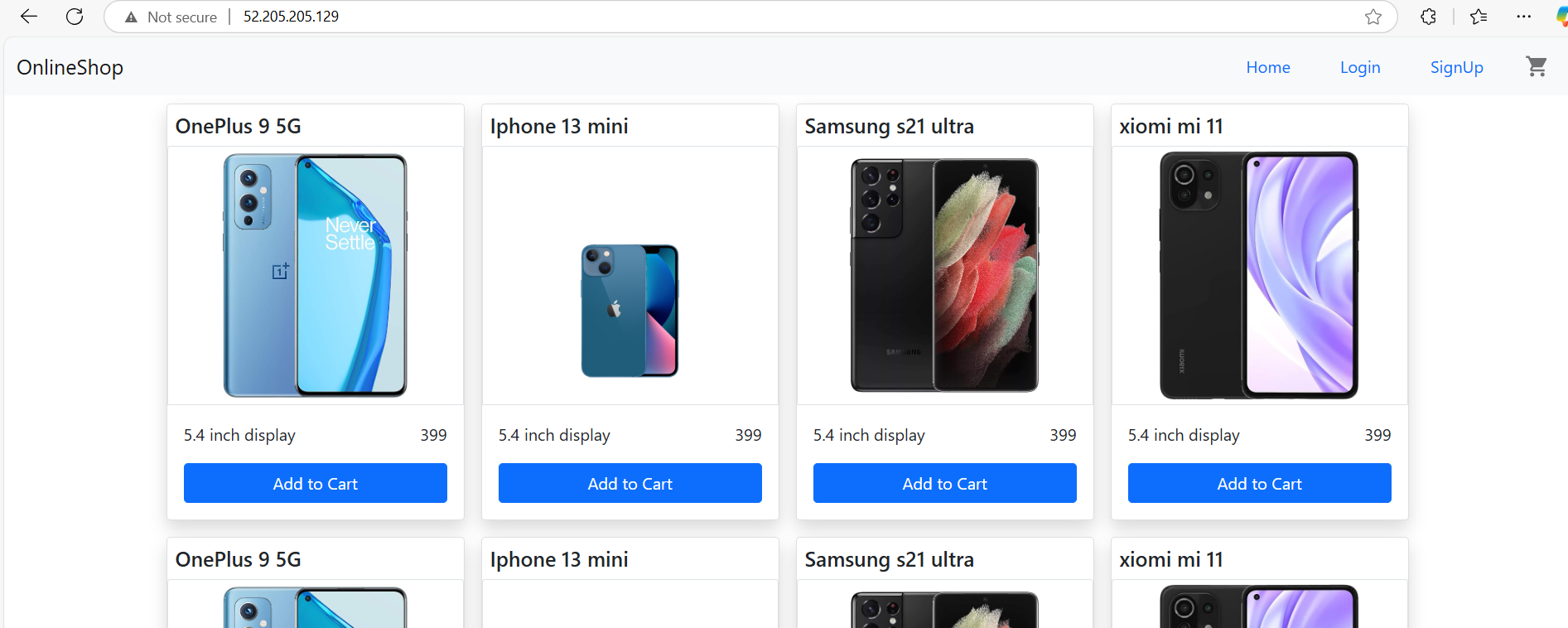
Docker hub – Image pushed to dev repo

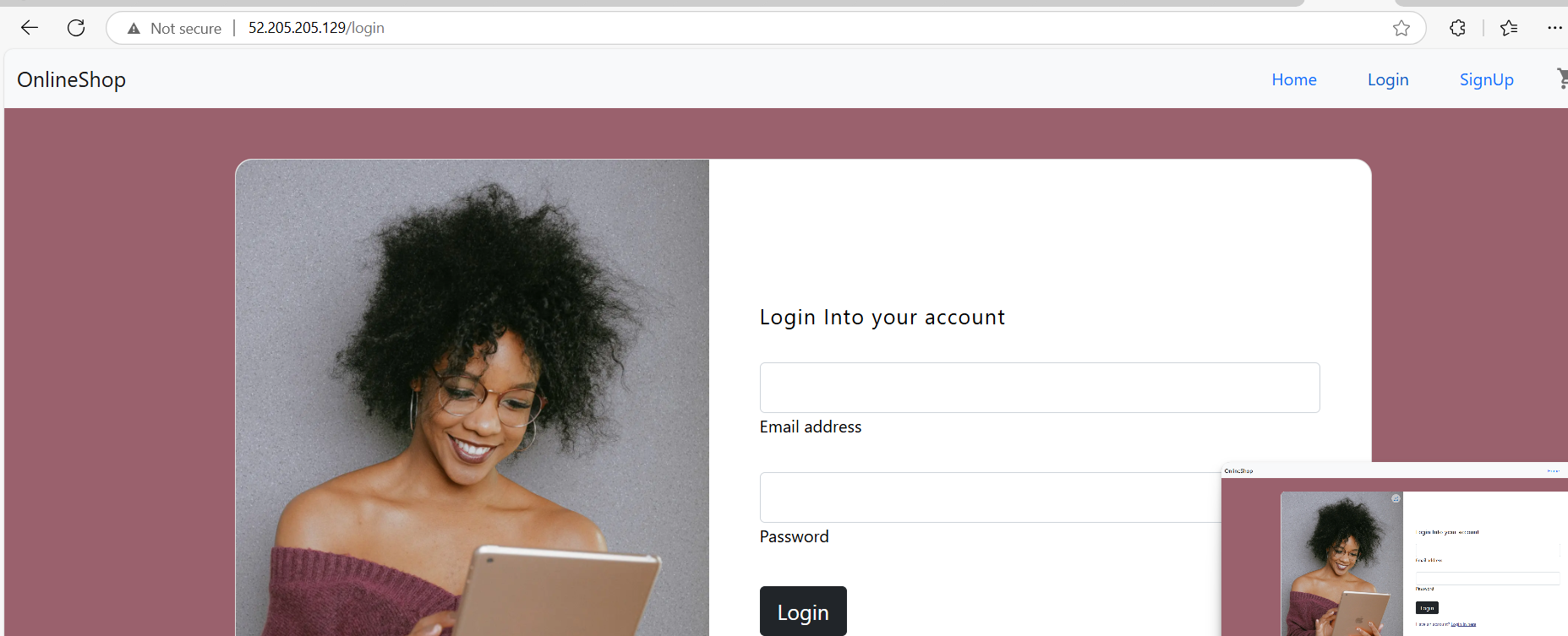




Deployment in ec2 instance

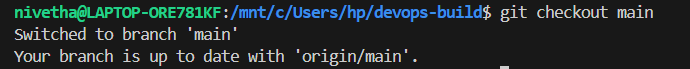




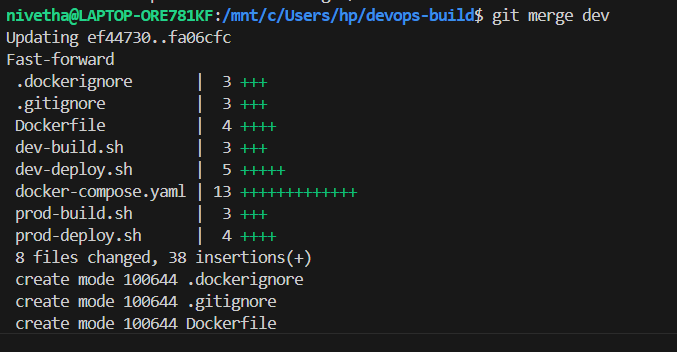


**Test Prod Deployment**

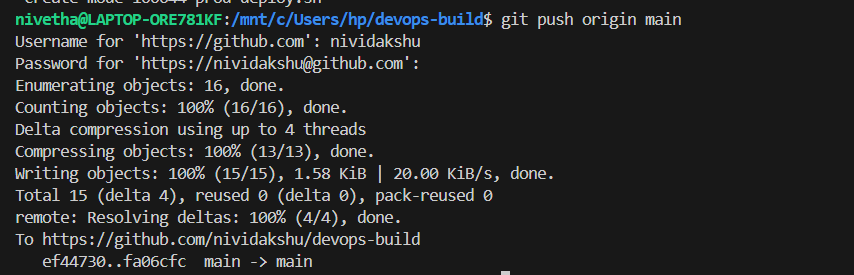
git checkout main



git merge dev



git push origin main



webhook triggered



In Jenkins all build were successful, and upon pushing, the build triggered automaticaly

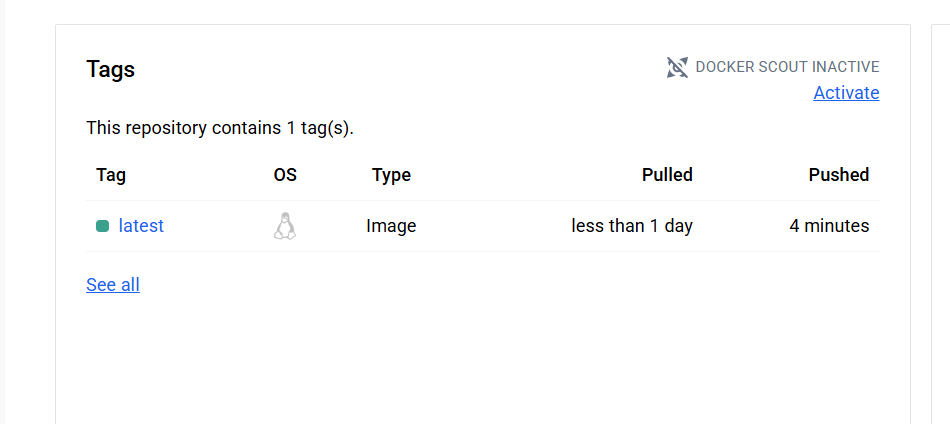


docker ps

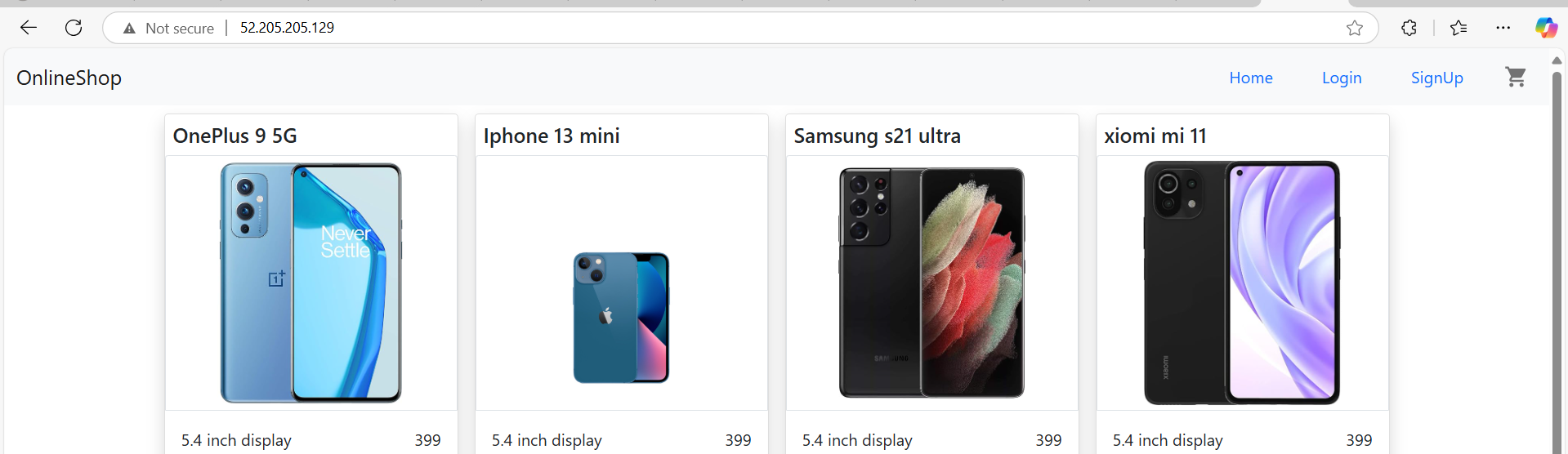


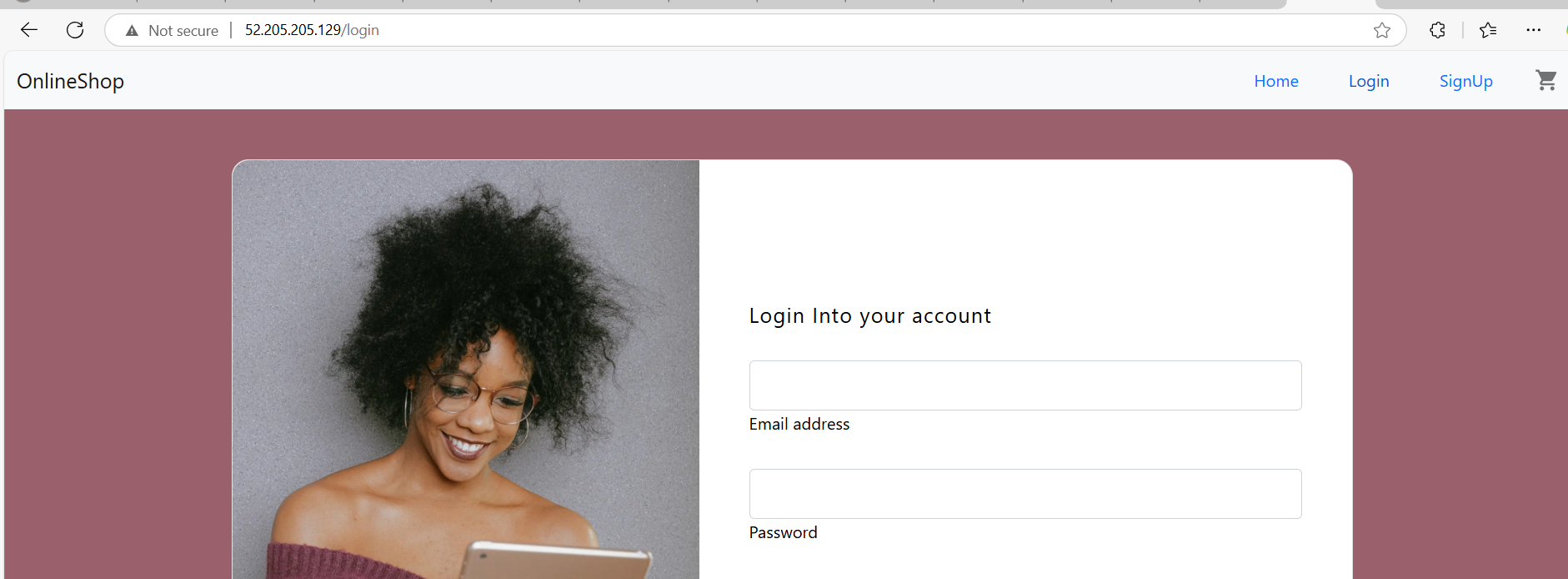
docker hub : image has been pushed to prod





Deployment to EC2:

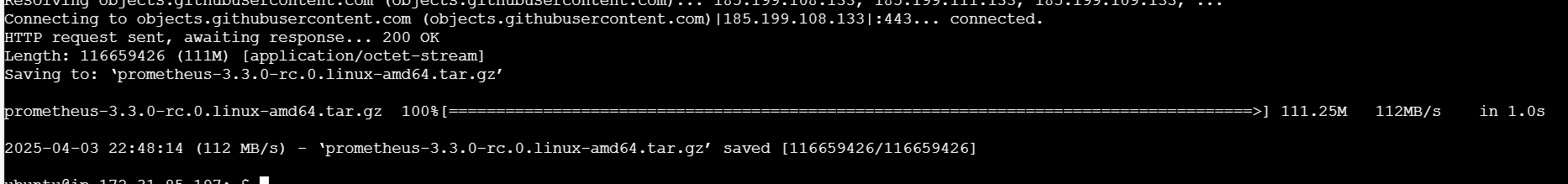




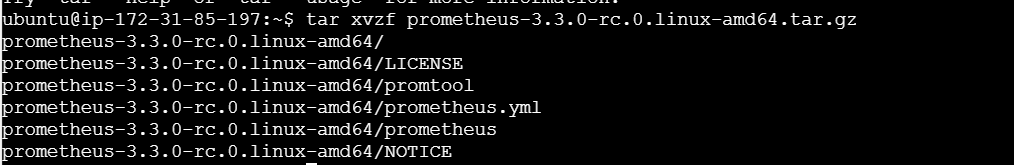
**Step 11:** Monitoring : using Prometheus, node\_exporter and alert manager

**Prometheus**

wget https://github.com/prometheus/prometheus/releases/download/v3.3.0-rc.0/prometheus-3.3.0-rc.0.linux-amd64.tar.gz



tar xvdf prometheus-3.3.0-rc.0.linux-amd64.tar.gz

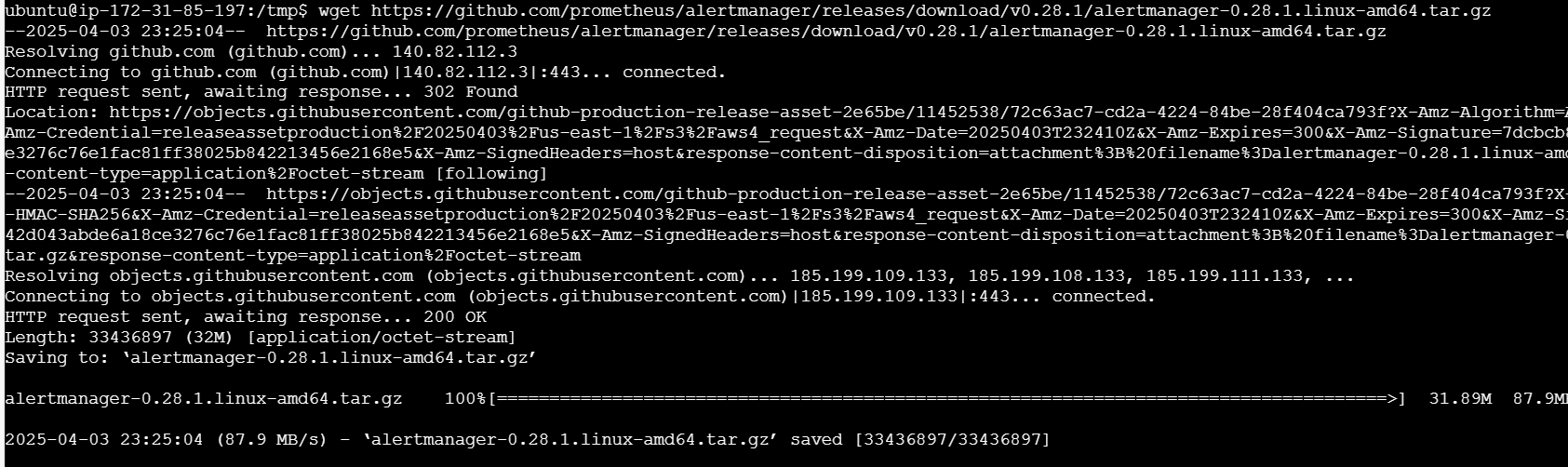


sudo mv prometheus-3.3.0-rc.0.linux-amd64 /etc/Prometheus

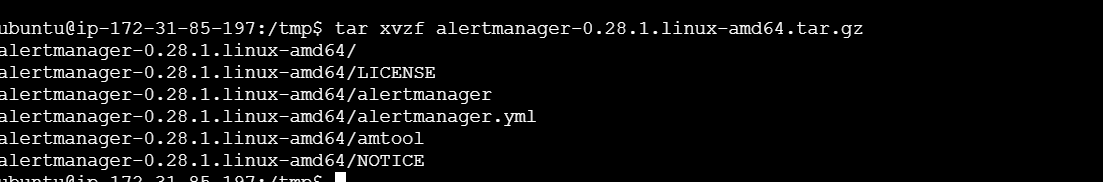
**Also download alert manager in same Prometheus doc**

cd /tmp/

wget <https://github.com/prometheus/alertmanager/releases/download/v0.28.1/alertmanager-0.28.1.linux-amd64.tar.gz>



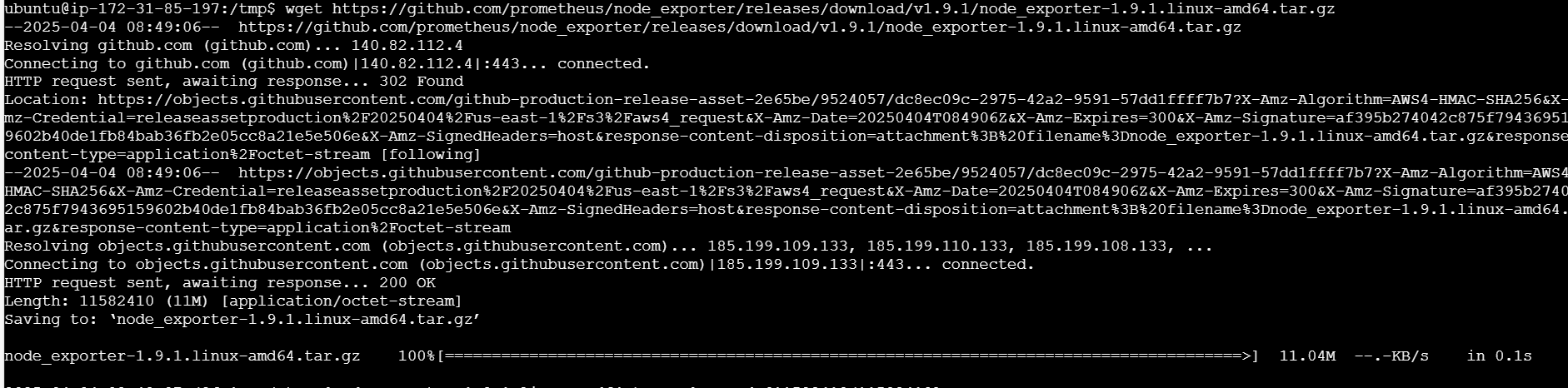
tar xvzf alertmanager-0.28.1.linux-amd64.tar.gz



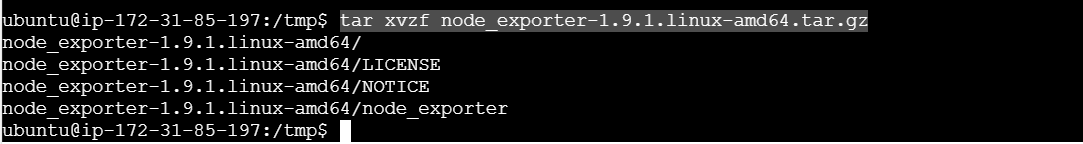
**Also download node\_exporter in same Prometheus doc**

* The reason of downloading node\_exporter is, when I run Prometheus without node exporter, it cannot scrape the react app metrics, it was throwing error.
* Prometheus is a **pull-based monitoring system** that collects metrics from targets that expose them in the **Prometheus format** but my ec2 instance was not exposing that metrics to Prometheus. Also our react app is a front end app and it is not generating any metrics , so Prometheus cannot scrape react app metrics in the required format.
* To overcome this installing node\_exporter , this exposes the metrics in Prometheus format so Prometheus scrape node exporter and monitor the metrics like health.

Wget <https://github.com/prometheus/node_exporter/releases/download/v1.9.1/node_exporter-1.9.1.linux-amd64.tar.gz>



tar xvzf node\_exporter-1.9.1.linux-amd64.tar.gz



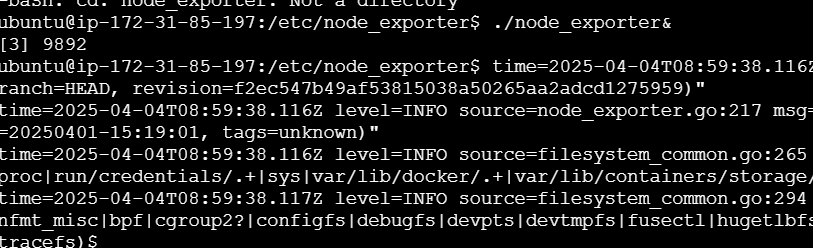
sudo mv node\_exporter-1.9.1.linux-amd64 /etc/node\_exporter



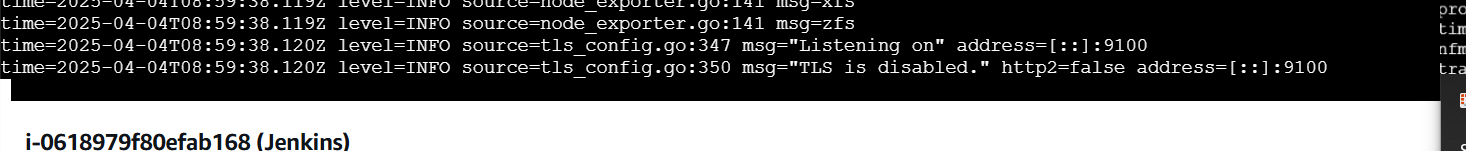
cd etc

cd node\_exporter

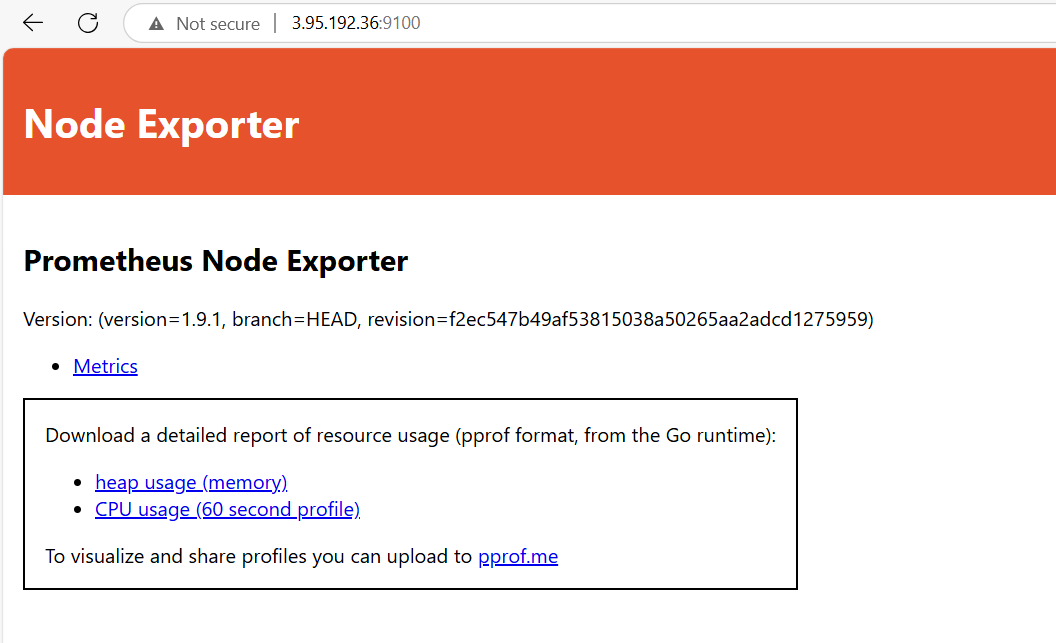
./node\_exporter &



Note the Port number - 9100

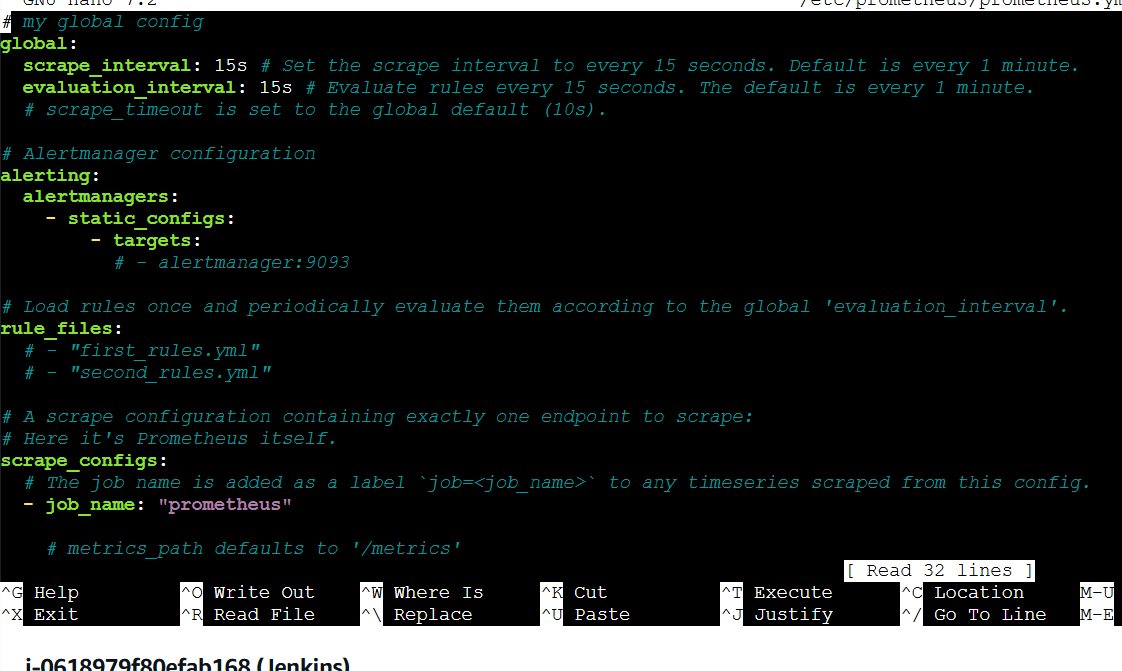


Verify it is running or not using http://3.95.192.36:9100

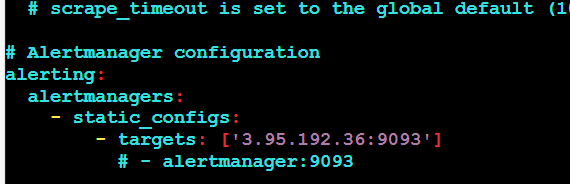


Now go to Prometheus.yaml file

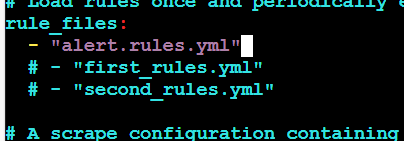
sudo nano /etc/prometheus/prometheus.yml



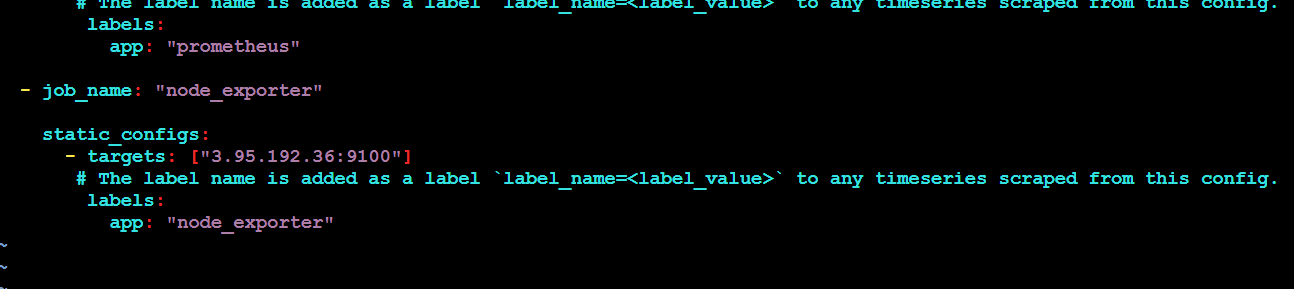
modify – add alertmanager port



add alert.rules.yml



Add the below job



then save and quit

cd /etc/Prometheus

sudo nano alert.rules.yml

groups:

- name: ApplicationAlerts

rules:

- alert: ApplicationDown

expr: up{app="my-react-app"} == 0

for: 1m

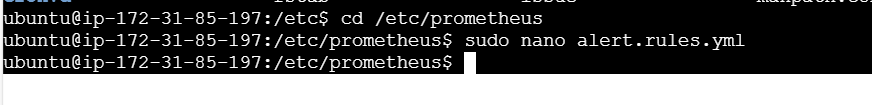
labels:

severity: critical

annotations:

summary: "Application is down"

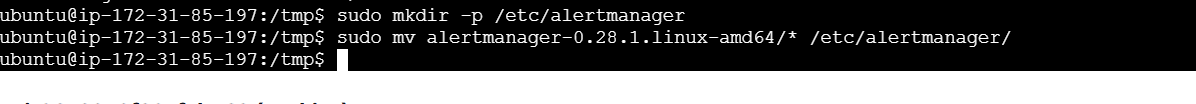
description: "The application {{ $labels.app }} is not responding."



(Note: I have stopped and started my instance , so public ip got changed)

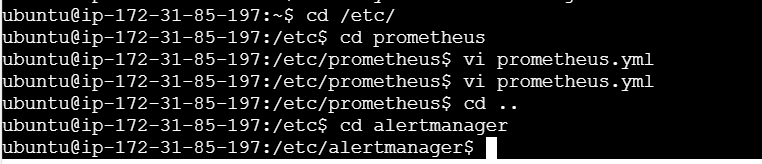
sudo mkdir -p /etc/alertmanager

sudo mv alertmanager-0.28.1.linux-amd64/\* /etc/alertmanager/



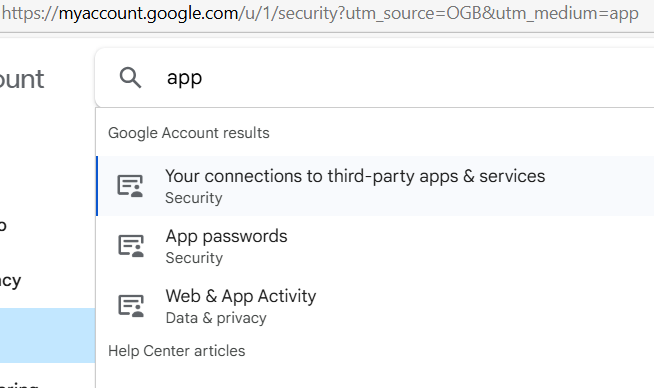
Since logged back in, changing the target host in Prometheus.yml then,

Cd alertmanager



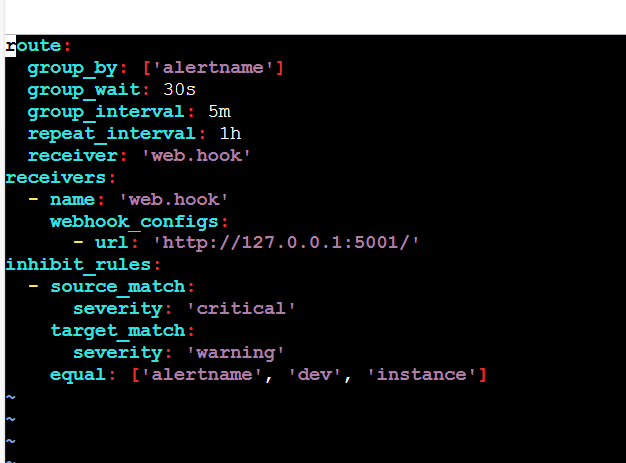
Vi alertmanager.yml

Before editing this, go to gmail and generate app password,



My app pw : enwv pqvn lxbt zalv

Vi alertmanager.yml



Here the webhook config is given, we can add email config alongside or delete webhook and add email config.

Modify

Add the below content

route:

receiver: "email"

receivers:

- name: "email"

email\_configs:

- to: "nivethadhaksha@gmail.com"

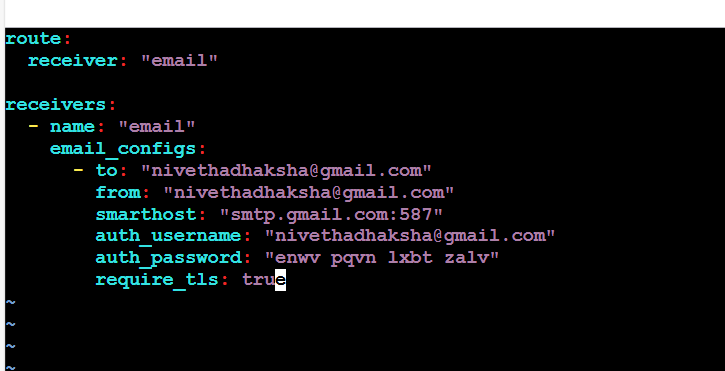
from: "nivethadhaksha@gmail.com"

smarthost: "smtp.gmail.com:587"

auth\_username: "nivethadhaksha@gmail.com"

auth\_password: "enwv pqvn lxbt zalv"

require\_tls: true



Save

Here auth pw can be set as env variable as well for security purpose

auth\_password: "{{ env.EMAIL\_PASSWORD }}"

export EMAIL\_PASSWORD="enwv pqvn lxbt zalv"

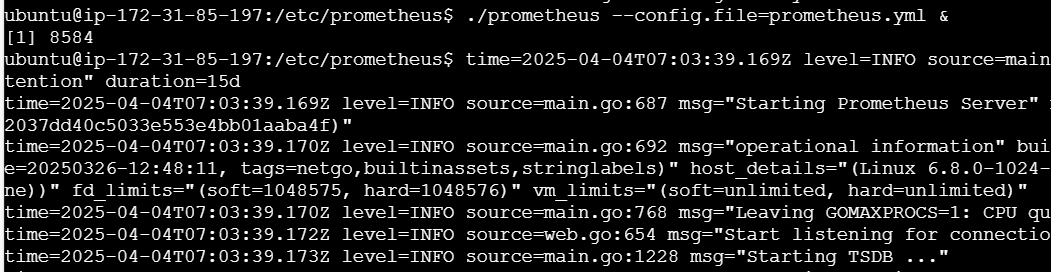
Now **prometheus.yml and alert.rules.yml ➝ prometheus directory.**

**alertmanager.yml ➝ alertmanager directory.**

**Node\_exporter ➝ node\_exporter directory**

Starting Prometheus

./prometheus --config.file=prometheus.yml &



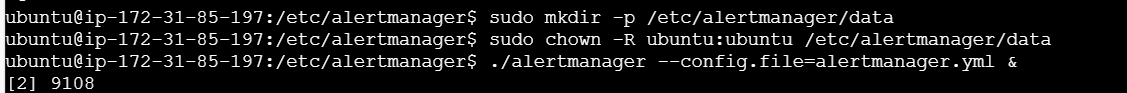
Cd etc

Cd alertmanager

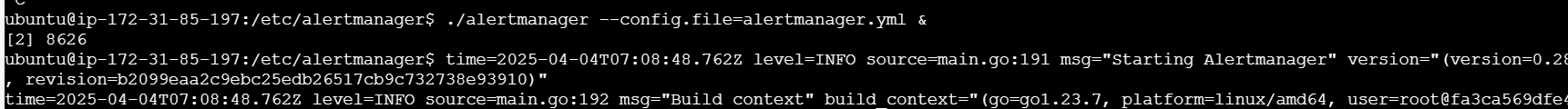
sudo mkdir -p /etc/alertmanager/data

sudo chown -R ubuntu:ubuntu /etc/alertmanager/data

we are giving alertmanager permission to create the data



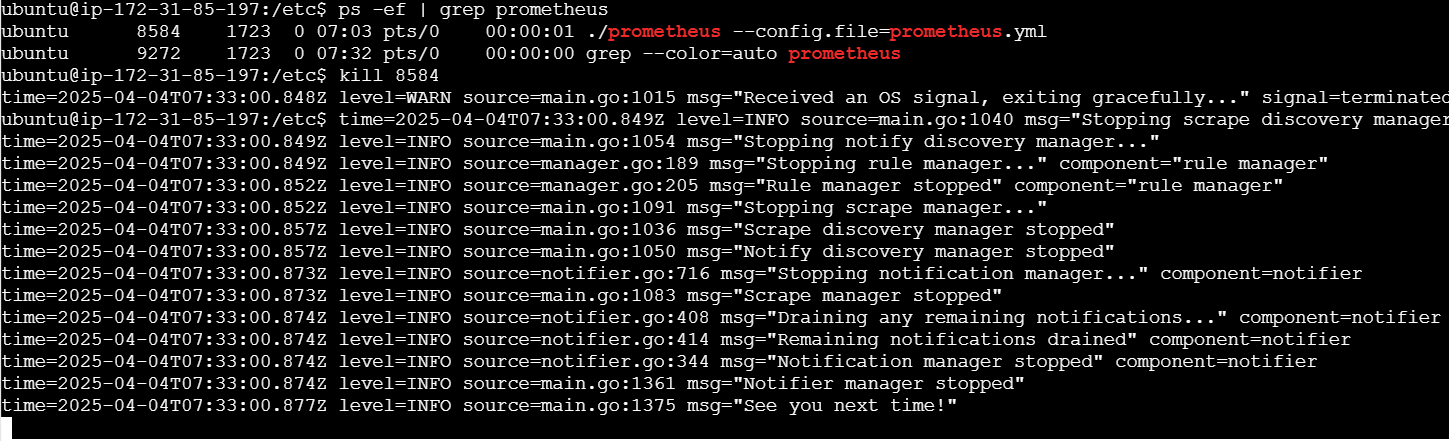
./alertmanager --config.file=alertmanager.yml &



Restarting Prometheus

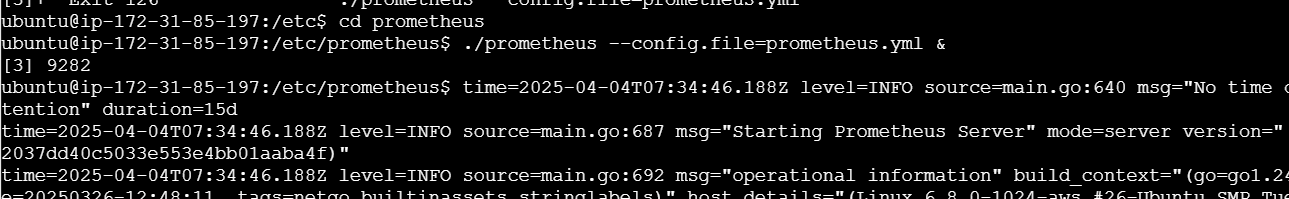
Ps -ef | grep Prometheus

Kill 8584

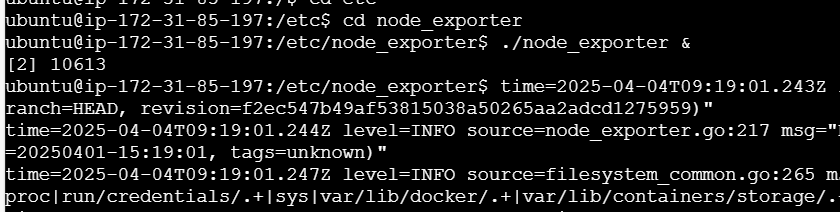


Cd Prometheus

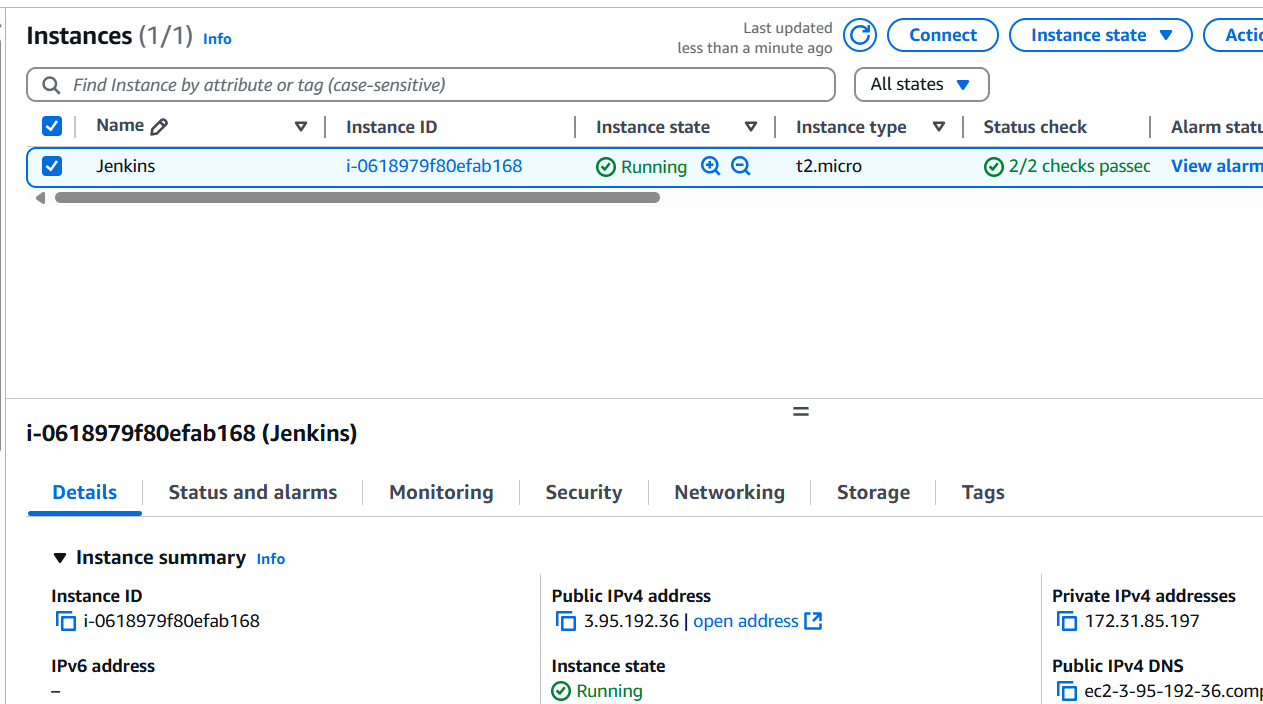
Running it again



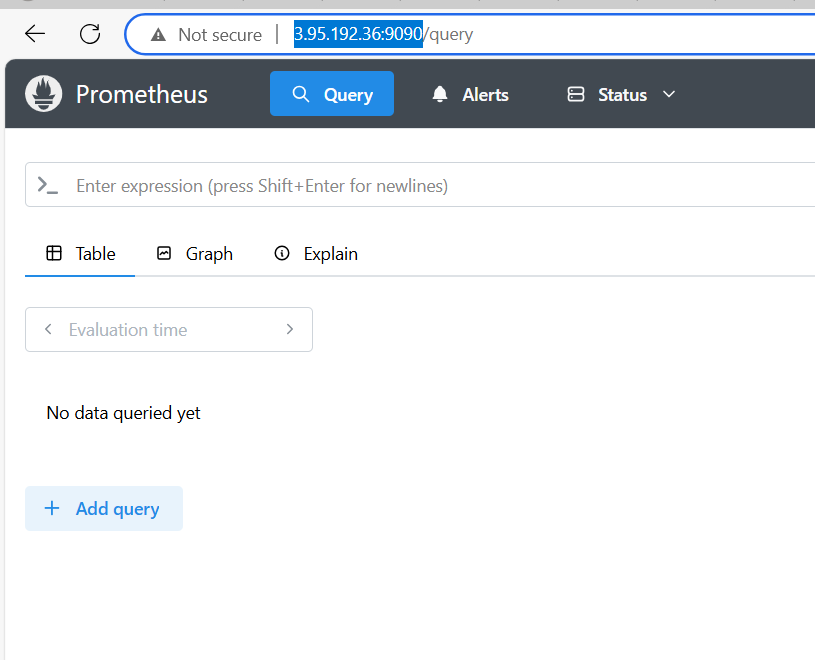
Navigate to node exporter, run node exporter again

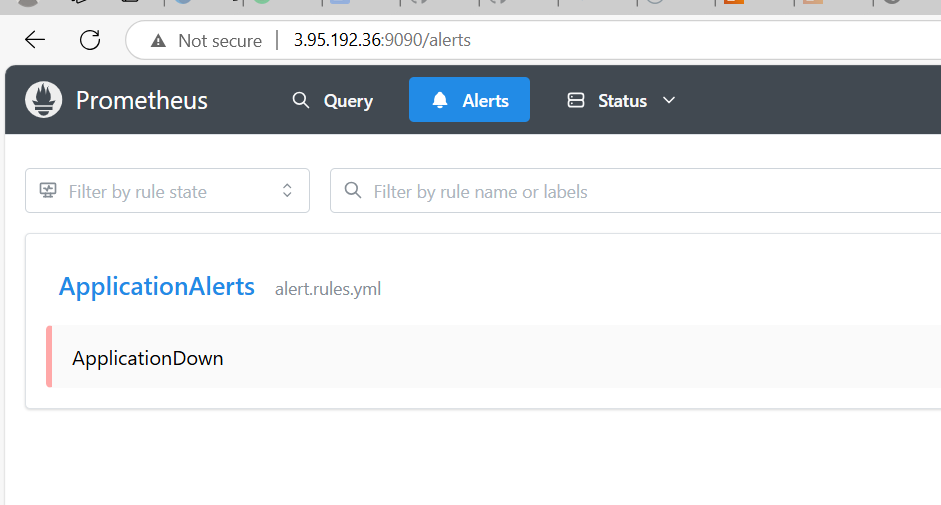


**Prometheus url**

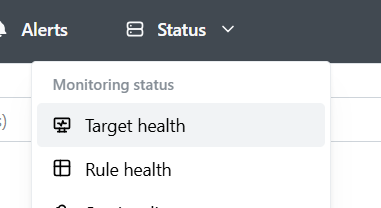
****

<http://3.95.192.36:9090/>

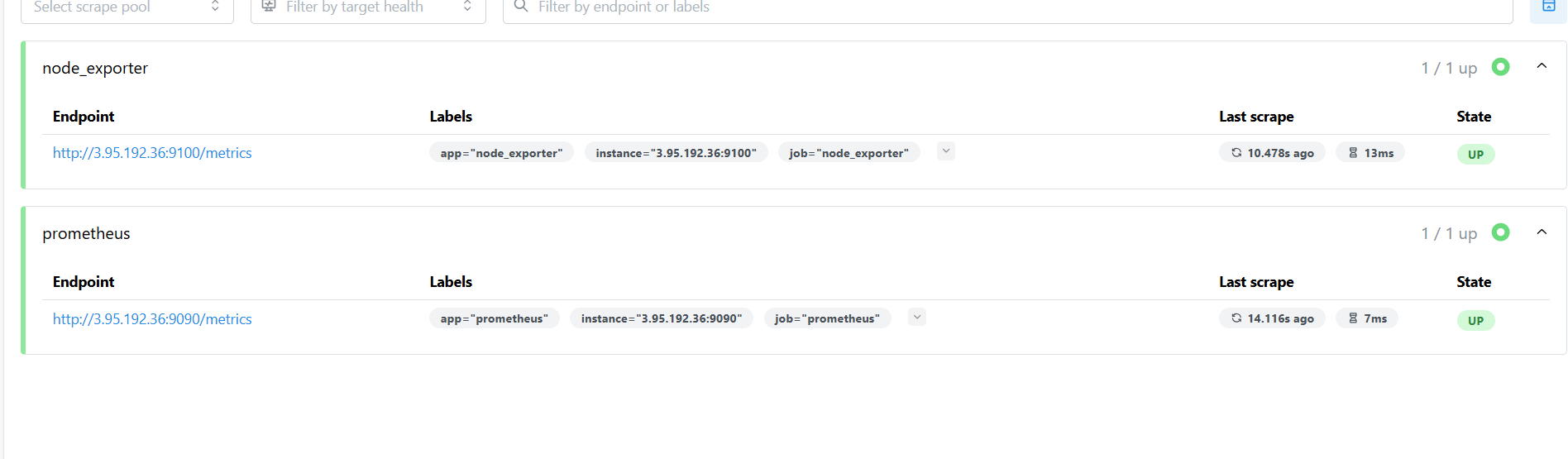




Choose status >> target health

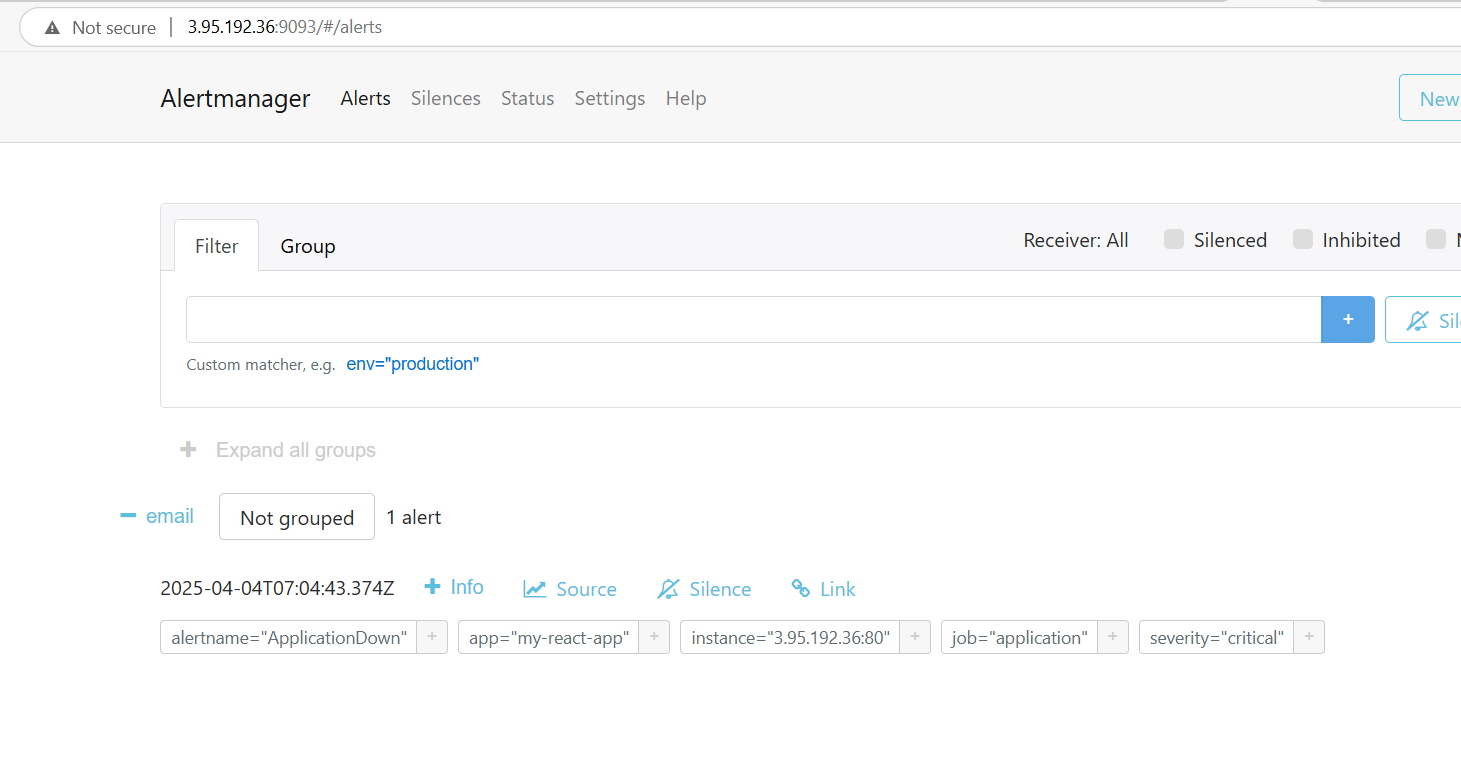


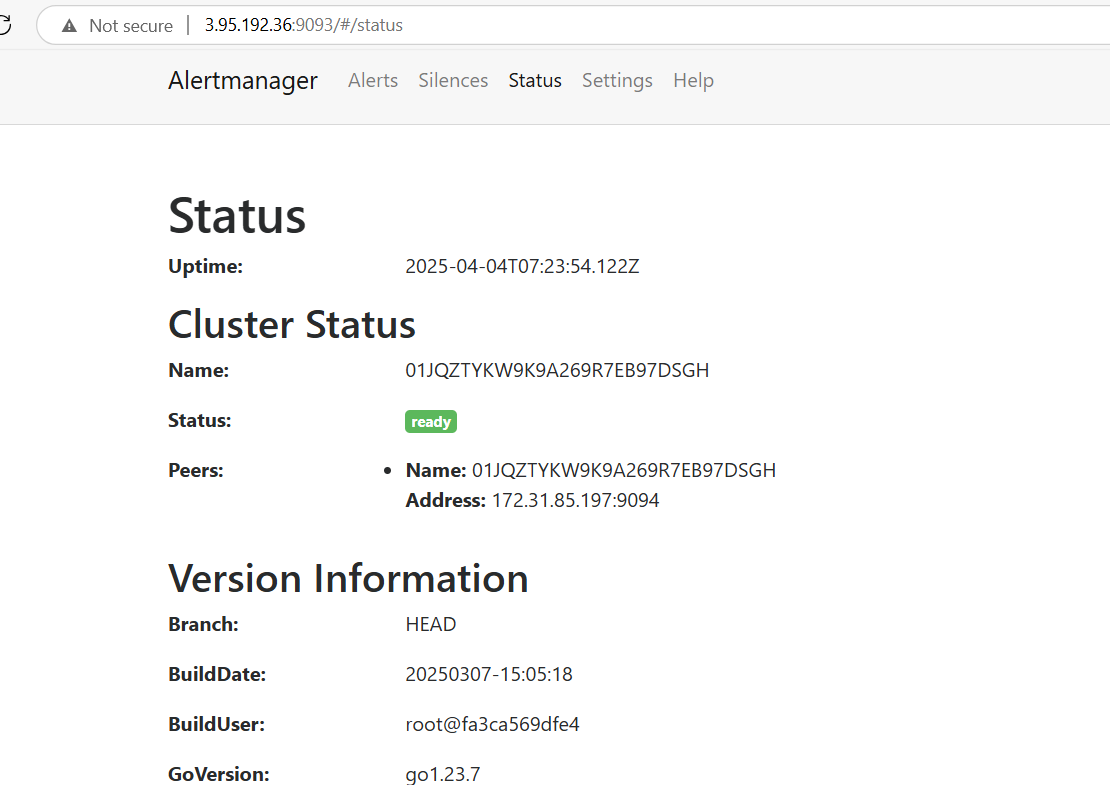
Monitoring the health

****

**Alertmanager url**

<http://3.95.192.36:9093/>



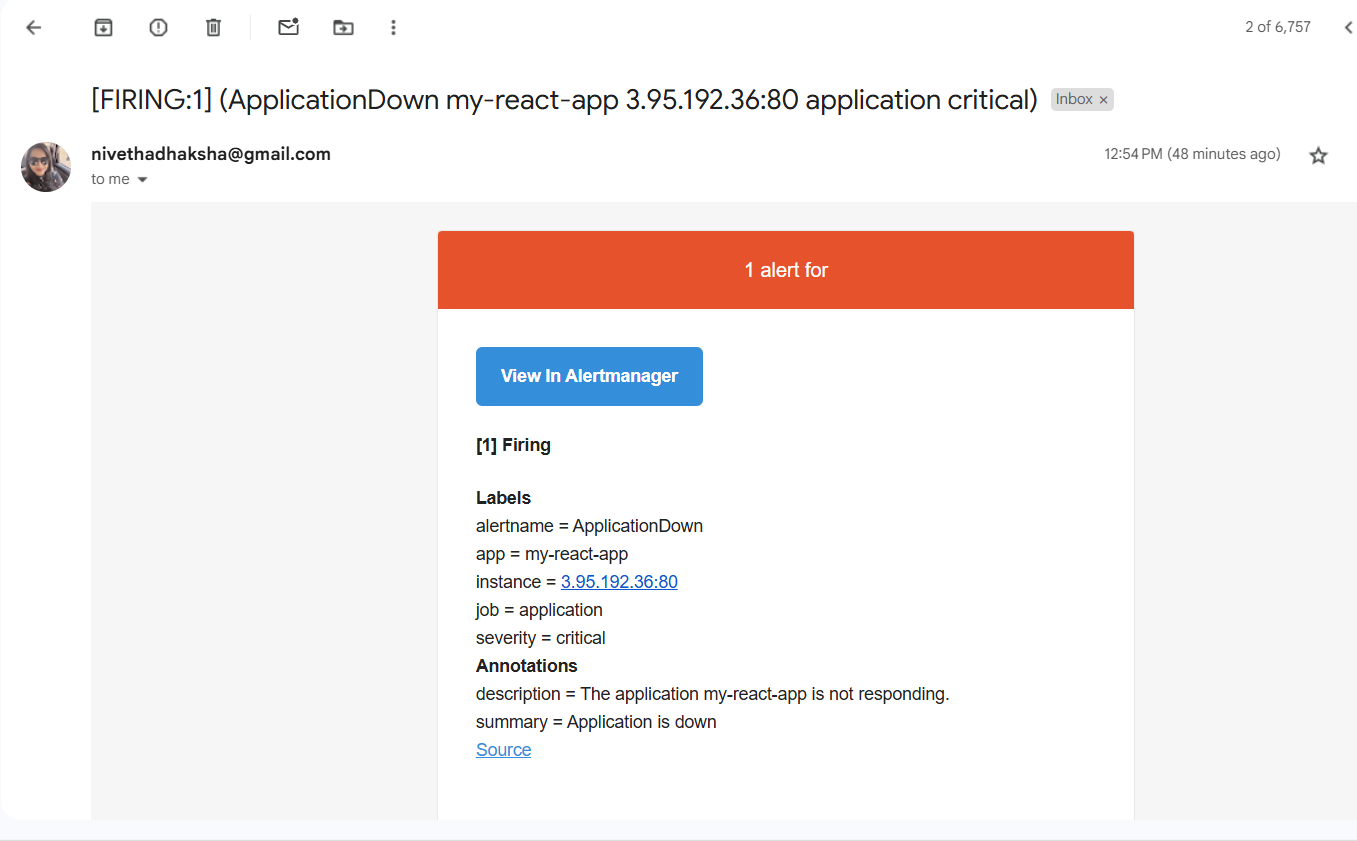


Docker ps

Docker stop cfeb



I stopped my docker container to check whether my alert manager is triggering the mail.



**Ports: Jenkins -8080**

**Prometheus – 9090**

**Node-exporter – 9100**

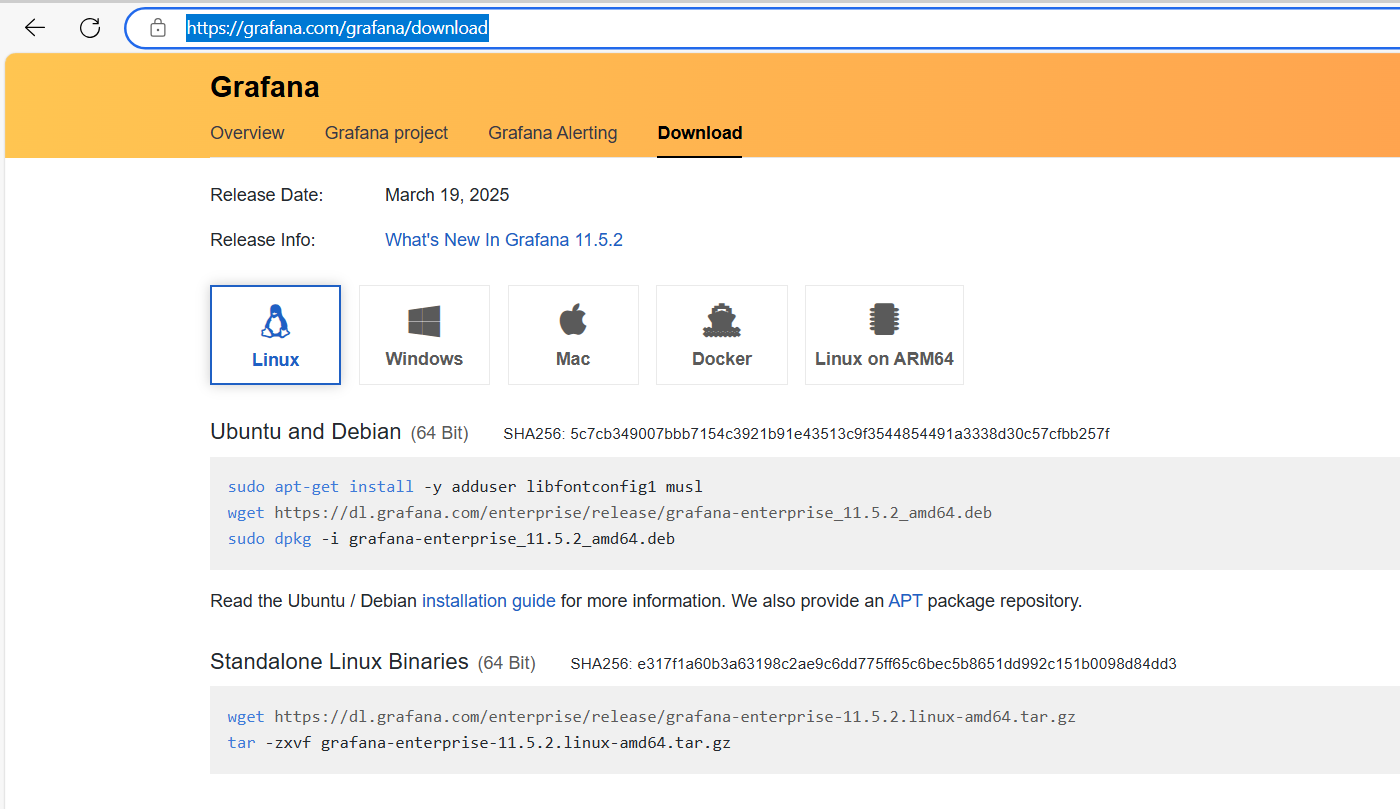
**Alert manager -9093**

**Checklist :**

1. **Jenkins (login page, configuration settings, execute step commands). - done**
2. **AWS (EC2 console, SG configurations) - done**
3. **Docker Hub repo with image tags - done**
4. **Deployed site page -done**
5. **Monitoring health check status – done (along with the mail trigger)**

**Optional**: Have installed Grafana as well for visualization

Visit the website [Download Grafana | Grafana Labs](https://grafana.com/grafana/download) to get the Grafana download link

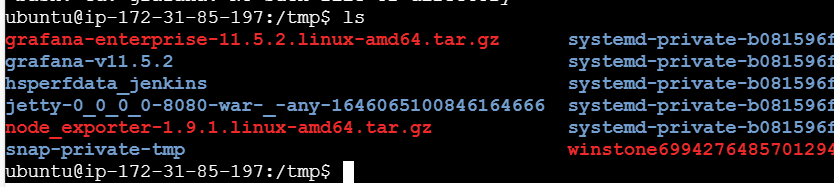


wget <https://dl.grafana.com/enterprise/release/grafana-enterprise-11.5.2.linux-amd64.tar.gz>



tar -zxvf grafana-enterprise-11.5.2.linux-amd64.tar.gz

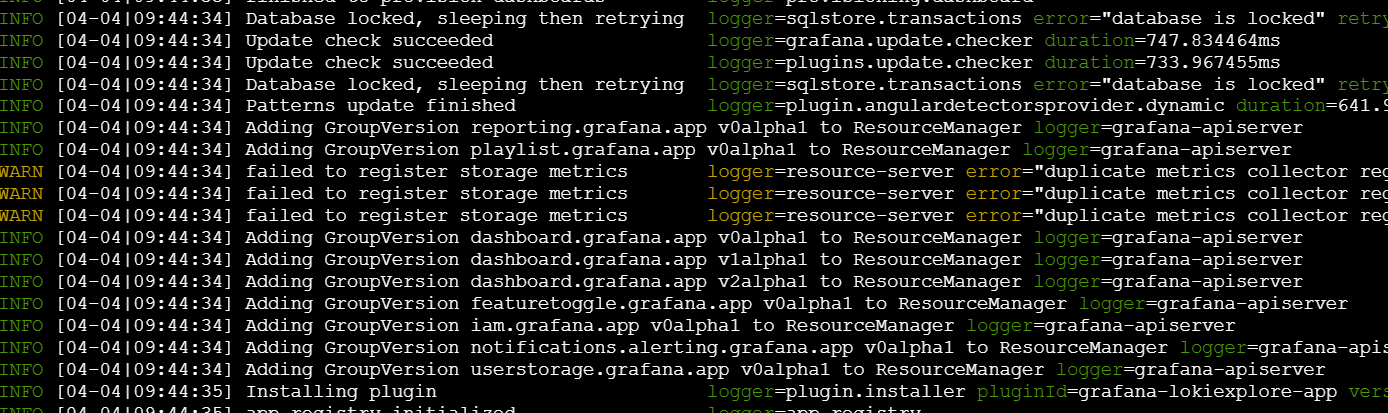
ls



cd grafana-v11.5.2

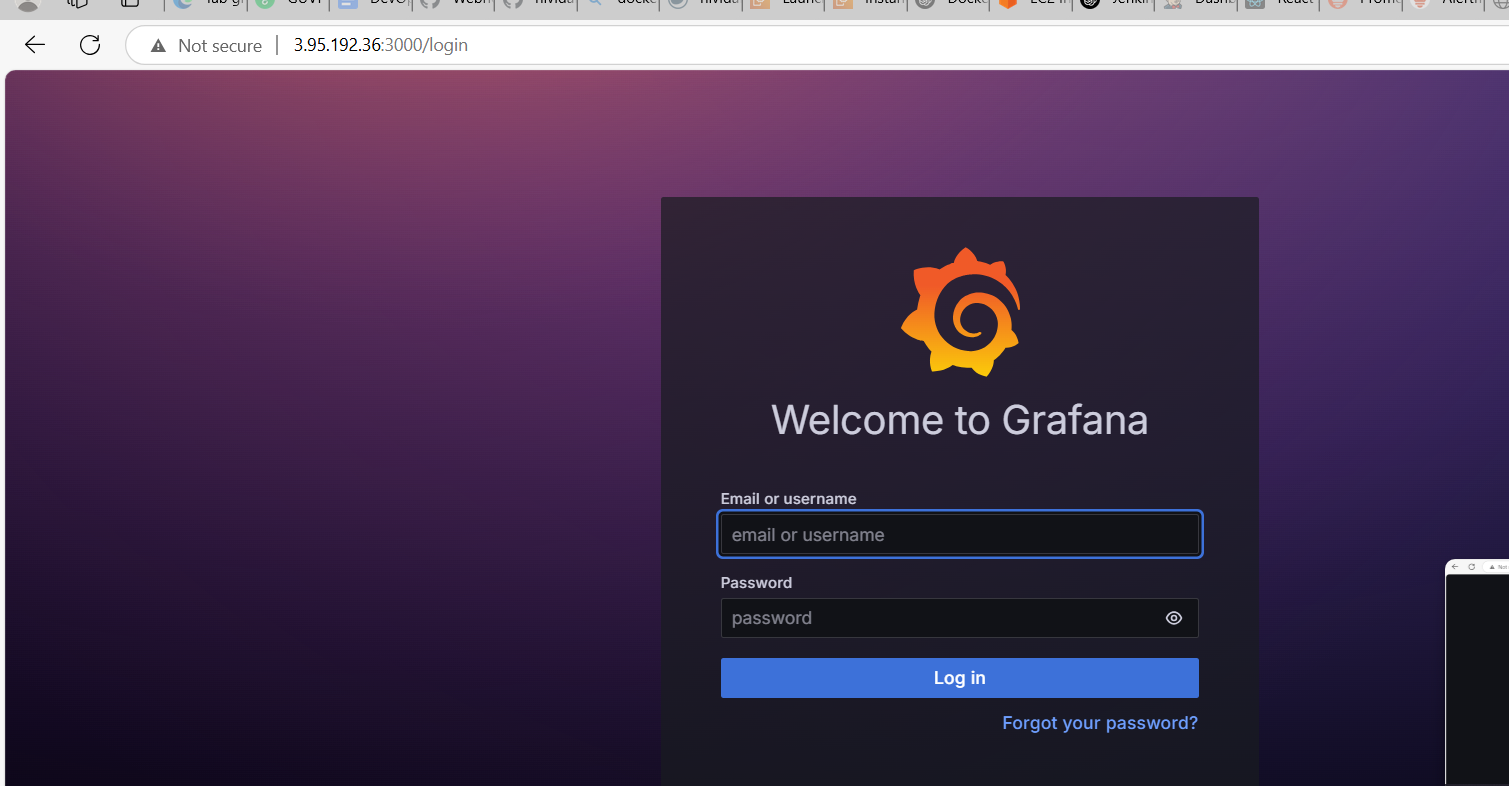
cd bin

./grafana-server &

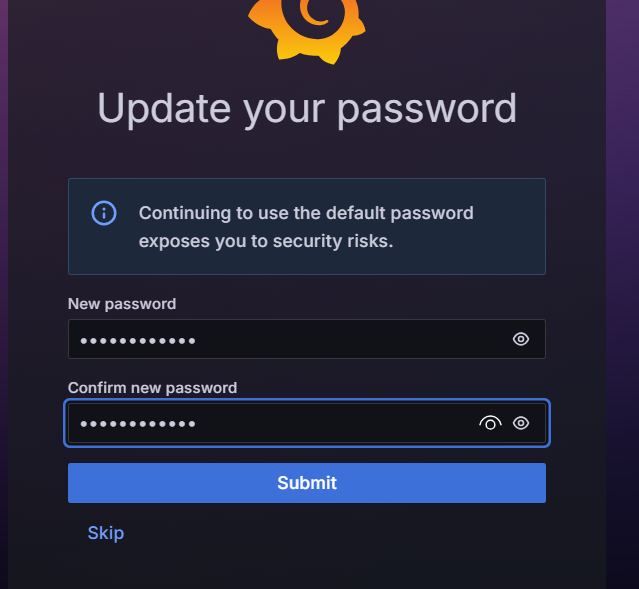


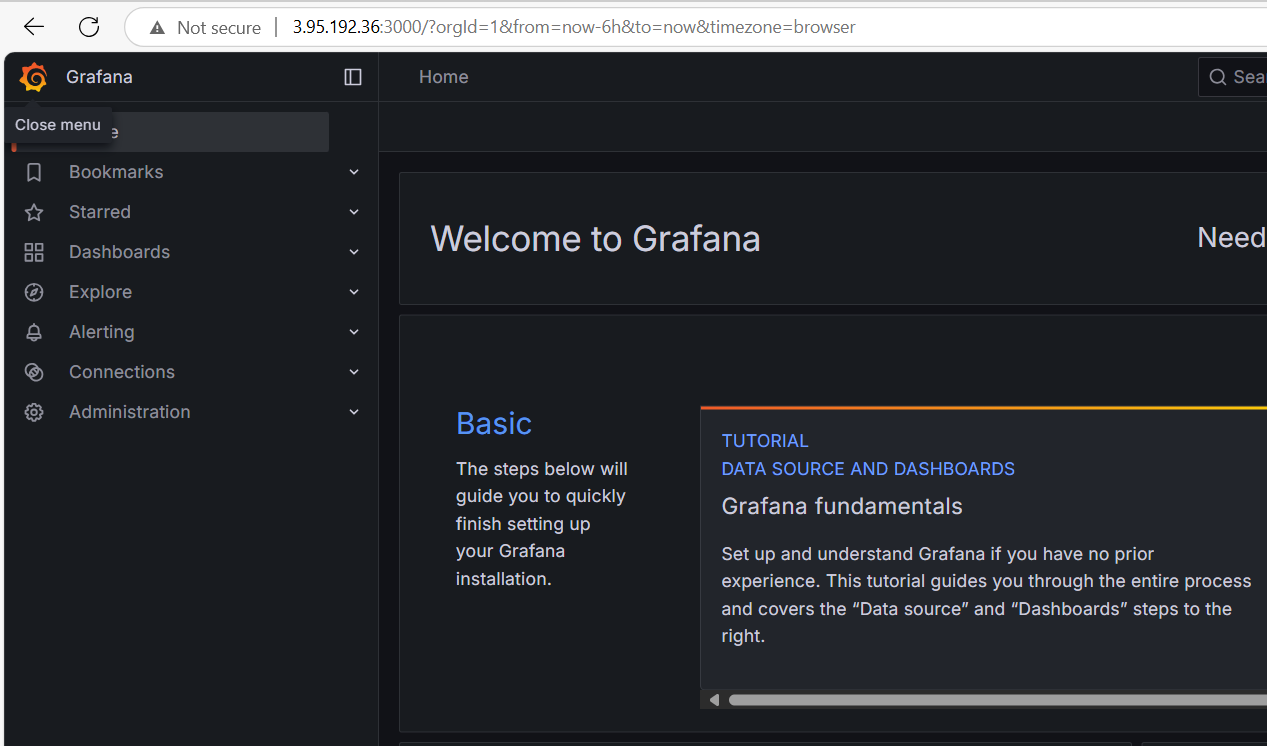
**Grafana port : 3000**

http://3.95.192.36:3000/

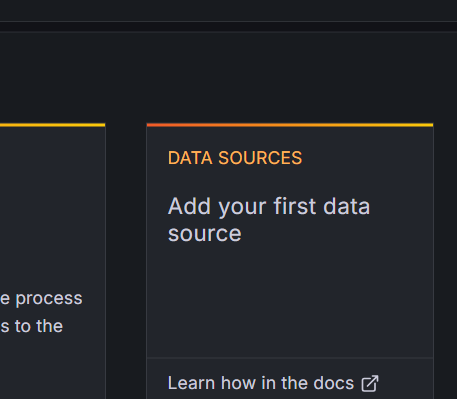
****

Login using credentials, use admin as the default password. Then change the pw.

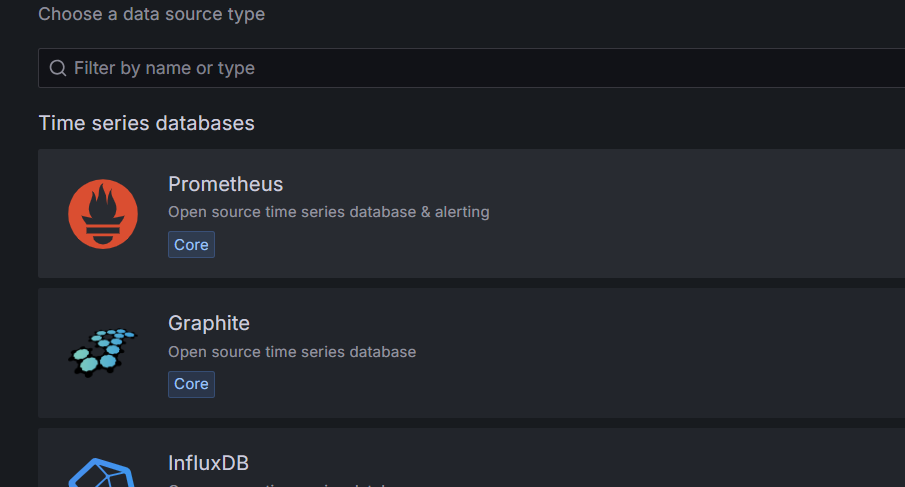
****



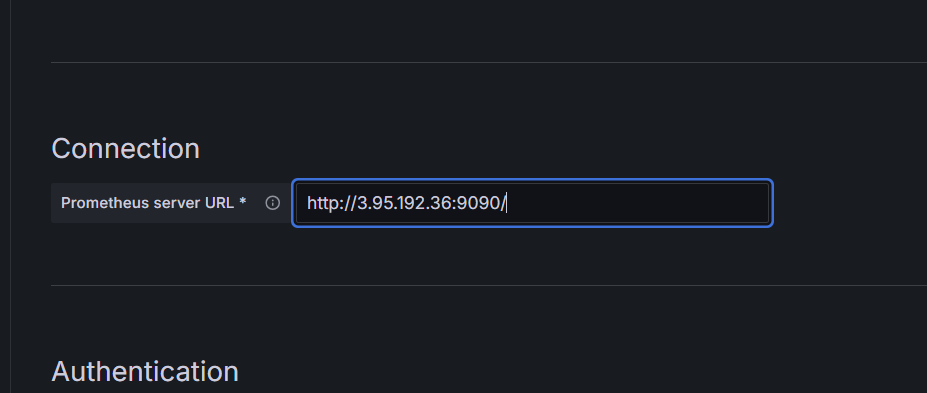
Add the data sources

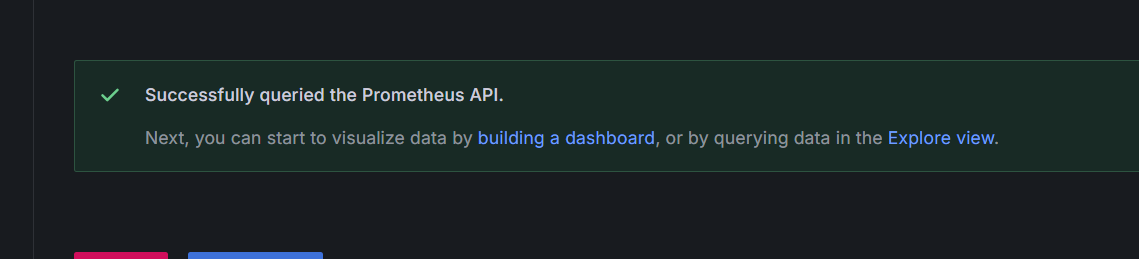


Choose Prometheus

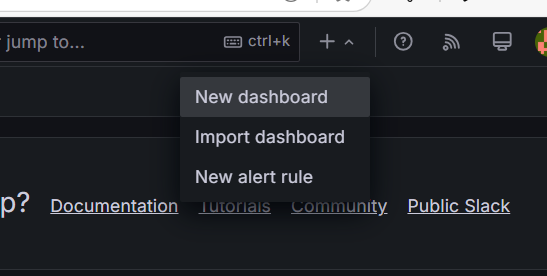


Give the Prometheus url





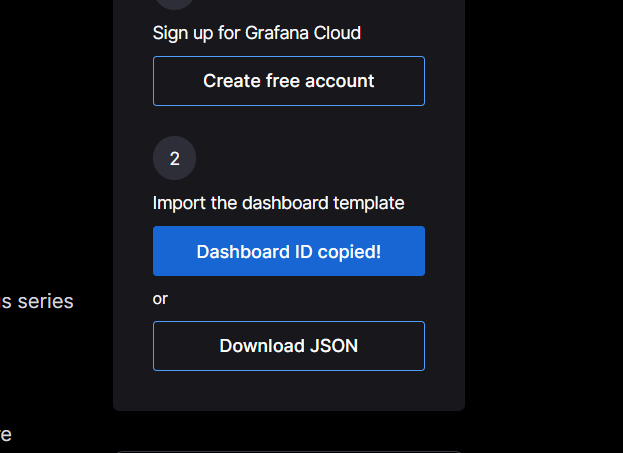
Then go to home> new dashboard or import dashboard



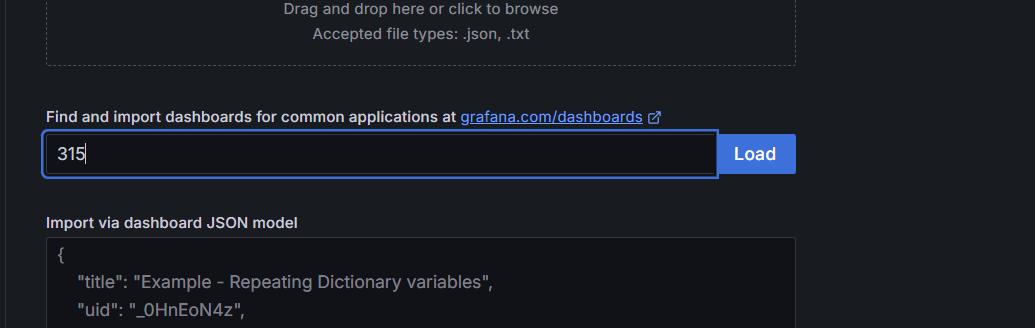
For import dashboard

Visit Grafana labs >> choose the desired dashboard and copy the id





Paste the id and load



Set the query to view the dashboard



Explore different virtualization options

