**Edureka Dev ops Final project 1**

**1. Business Challenge/Requirement**

ABC technologies is a leading online retail store. ABC has recently acquired a large retails offline business store. The business store has large number of stores across the globe but is following conventional pattern of development and deployment. As a result, it has landed to great loss and are facing below challenges.

* • low available
* • low scalable
* • low Performance
* • Hard built and maintained
* • Developed and deployed is time consuming

ABC will acquire the data from all these storage systems and plan to use it for analytics and prediction of the firm’s growth and sales prospect. In the first phase ABC has to create the servlets to Add a product and Display product details. Add servlet dependencies required to compile the servlets. Create an HTML page which will be used to add a product. Team is using git to keep all the source code.

ABC has decided to use DevOps model and Once source code is available in Github, we need to integrate it with Jenkins and provide continuous build generation for continuous Delivery, integrate with Ansible and Kubernetes for deployment. Use docker hub to pull and push images between ansible and Kubernetes.

**2. The Goal of the Project**

Below are some of the high-level goals of this project:

* • Implement CICD such that ABC Company to is able to be
* 1. Highly available
* 2. Highly scalable
* 3. Highly Performant
* 4. Easily built and maintained
* 5. Developed and deployed quickly

**Solution:**

Step 1: Use the git commands to VC the files locally and then push them to git hub

# git init

# git add .

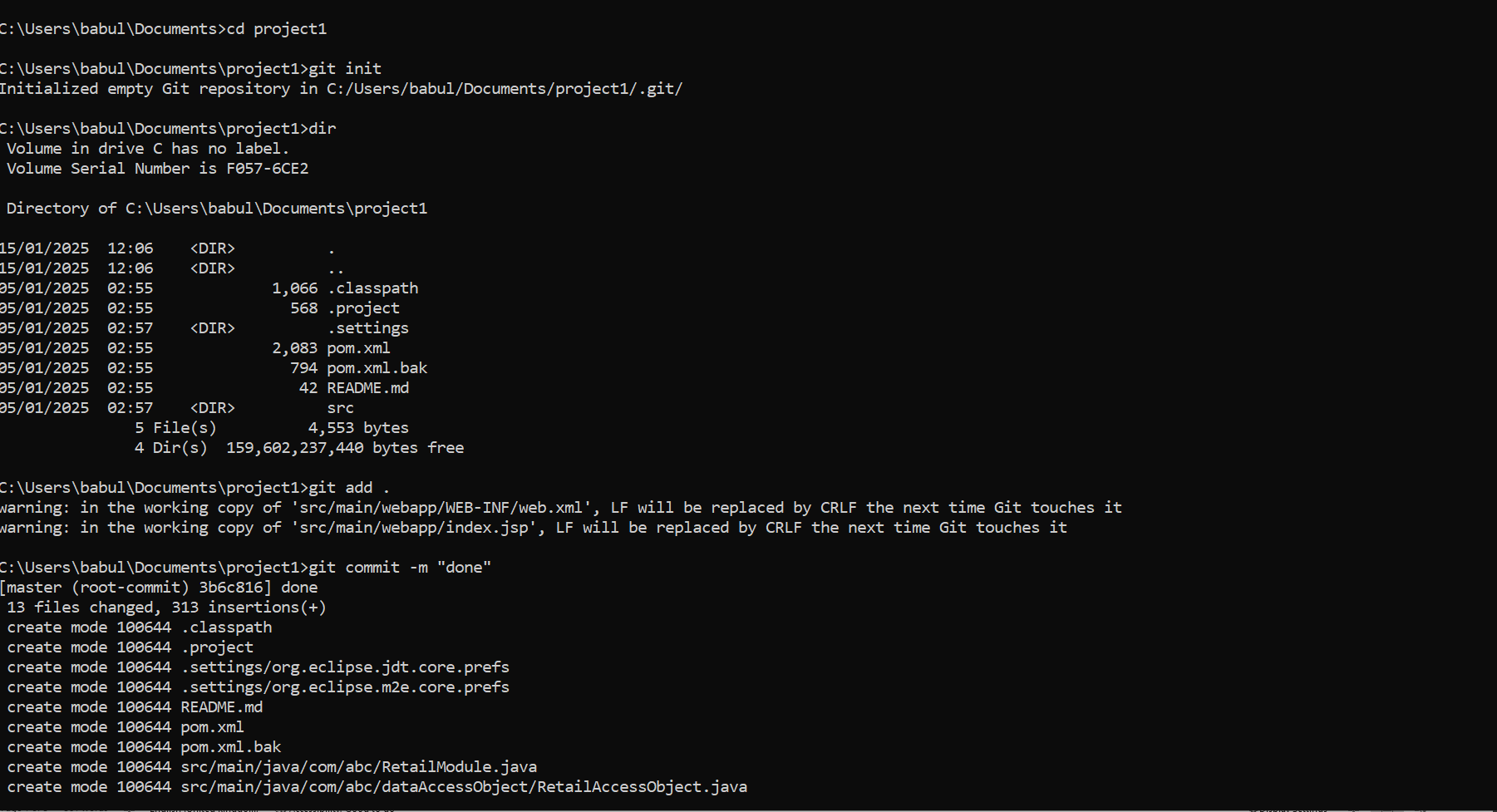
# git commit -m "done"

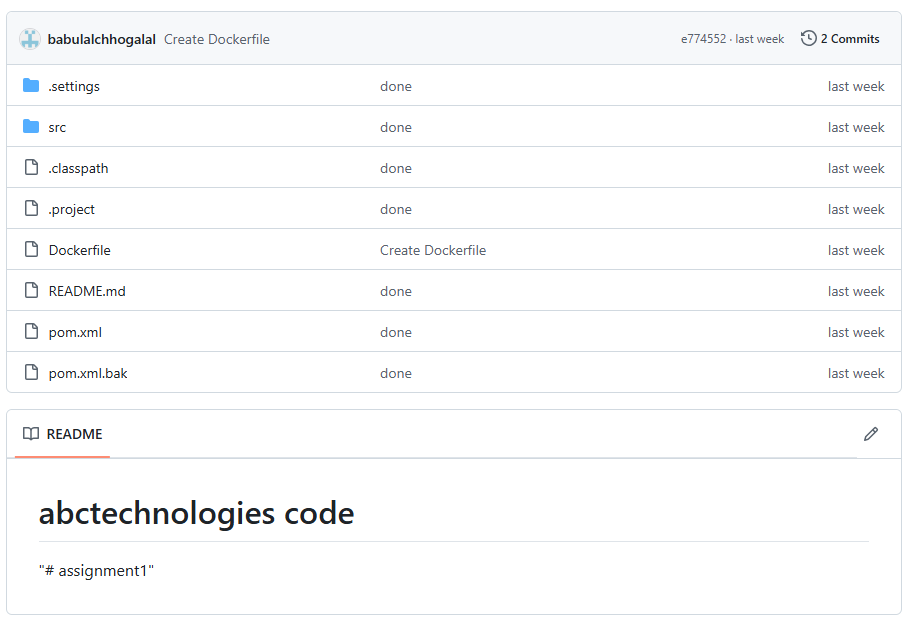
On github Create a repository

Execute below git commands on cmd prompt

# git remote add origin git@github.com:Babulalchhogalal/PGP-FinalProject-1.git

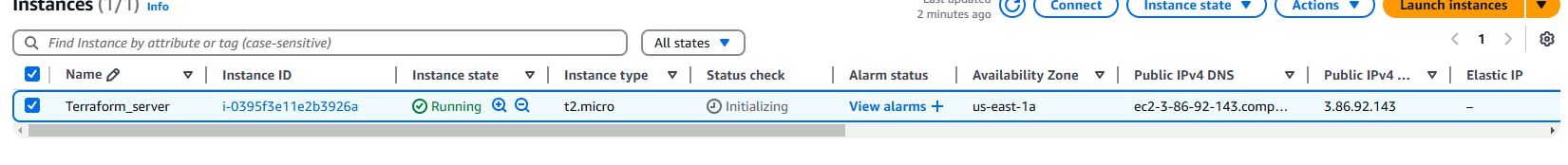
# git push origin master

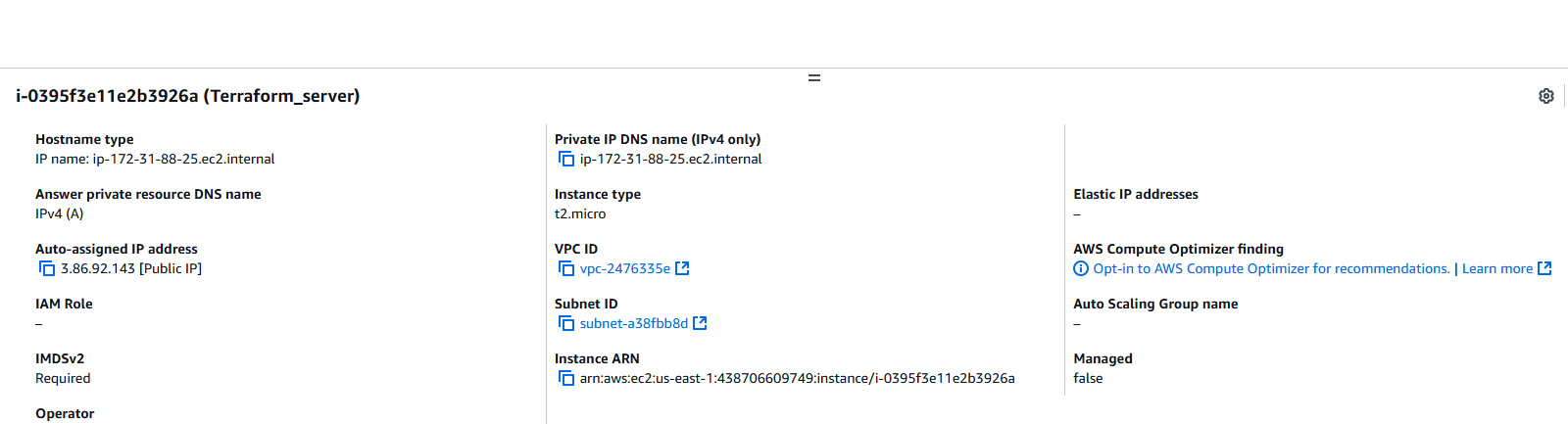




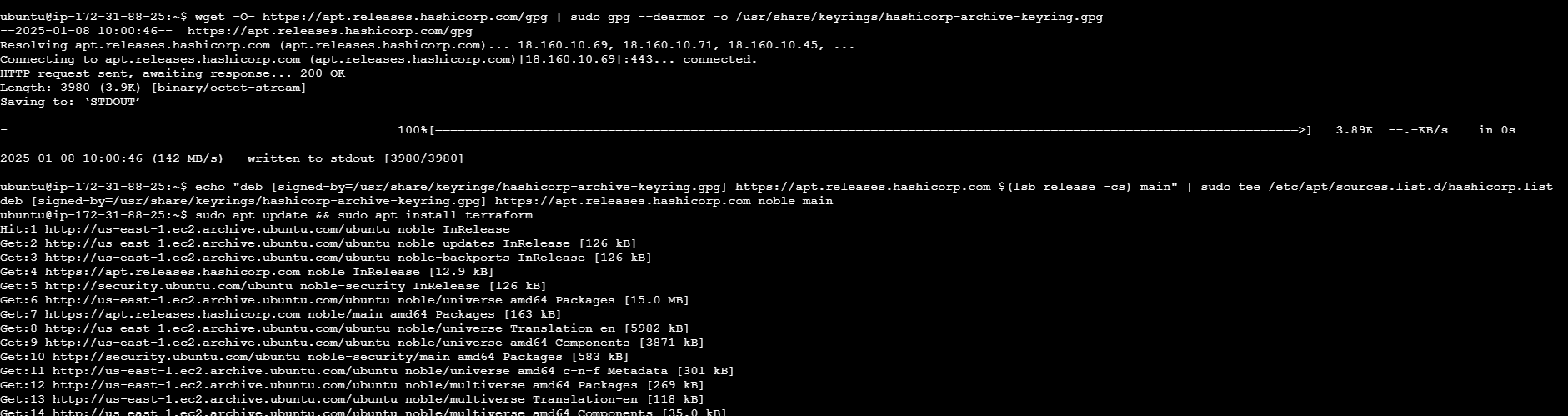
**Step 2**: Installed an EC2 instance on AWS and configured Terraform on it.

EC2 instance screenshot





Terraform installation:



Install AWS Cli using the below

$ **curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

**unzip awscliv2.zip**

**sudo ./aws/install**

**Step 3: Create a new Terraform file to create a new EC2 instance and we will have Ansible installed on it.**

**#infrastructure.tf**

provider "aws" {

region = "us-east-1"

shared\_credentials\_files = ["~/.aws/credentials"]

}

resource "tls\_private\_key" "mykey" {

algorithm = "RSA"

}

resource "aws\_key\_pair" "aws\_key" {

key\_name = "web-key"

public\_key = tls\_private\_key.mykey.public\_key\_openssh

provisioner "local-exec" {

command = "echo '${tls\_private\_key.mykey.private\_key\_openssh}' > ./web-key.pem"

}

}

resource "aws\_vpc" "sl-vpc" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "sl-vpc"

}

}

resource "aws\_subnet" "subnet-1"{

vpc\_id = aws\_vpc.sl-vpc.id

cidr\_block = "10.0.1.0/24"

depends\_on = [aws\_vpc.sl-vpc]

map\_public\_ip\_on\_launch = true

tags = {

Name = "sl-subnet"

}

}

resource "aws\_route\_table" "sl-route-table"{

vpc\_id = aws\_vpc.sl-vpc.id

tags = {

Name = "sl-route-table"

}

}

resource "aws\_route\_table\_association" "a" {

subnet\_id = aws\_subnet.subnet-1.id

route\_table\_id = aws\_route\_table.sl-route-table.id

}

resource "aws\_internet\_gateway" "gw" {

vpc\_id = aws\_vpc.sl-vpc.id

depends\_on = [aws\_vpc.sl-vpc]

tags = {

Name = "sl-gw"

}

}

resource "aws\_route" "sl-route" {

route\_table\_id = aws\_route\_table.sl-route-table.id

destination\_cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.gw.id

}

variable "sg\_ports" {

type = list(number)

default = [8080,80,22,443]

}

resource "aws\_security\_group" "sl-sg" {

name = "sg\_rule"

vpc\_id = aws\_vpc.sl-vpc.id

dynamic "ingress" {

for\_each = var.sg\_ports

iterator = port

content{

from\_port = port.value

to\_port = port.value

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

resource "aws\_instance" "myec2" {

ami = "ami-0e2c8caa4b6378d8c"

instance\_type = "t2.medium"

key\_name = "web-key"

subnet\_id = aws\_subnet.subnet-1.id

security\_groups = [aws\_security\_group.sl-sg.id]

tags = {

Name = "Project-EC2"

}

provisioner "remote-exec" {

connection {

type = "ssh"

user = "ubuntu"

private\_key = tls\_private\_key.mykey.private\_key\_pem

host = self.public\_ip

}

inline = [

"sudo apt update",

"sudo apt install software-properties-common",

"sudo add-apt-repository --yes --update ppa:ansible/ansible",

"sudo apt install ansible -y"

]

}

}

**Run the Below commands to execute Terraform file.**

# terraform init

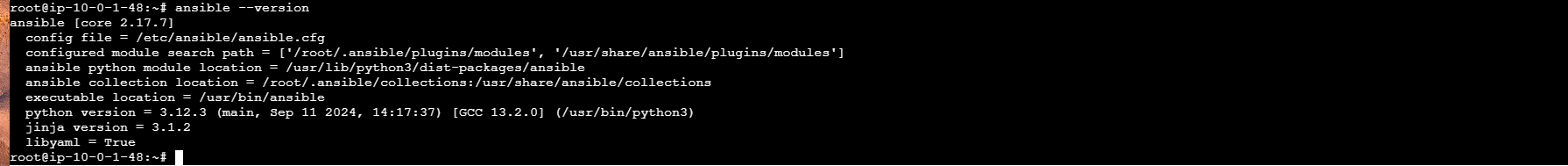
# terraform apply --auto-approve

**New EC2 instance is configured.**

**Step 4: We will login to EC2 instance and use Ansible playbook to configure all the CI/CD tools.**

Check if ansible is installed:

**# ansible –version**



**# sudo su –**

Run the below commands for installing Jenkins: [Always take the lastest steps for Jenkins]

Add Jenkins key to the server

# sudo wget -O /usr/share/keyrings/jenkins-keyring.asc https://pkg.jenkins.io/debian/jenkins.io-2023.key

Make Jenkins apt repo

# 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

Create an Ansible playbook

*# vim playbook1.yml*

*- name: Install and set up devops tools*

*hosts: localhost*

*become: true*

*tasks:*

*- name: Update the apt repo*

*command: apt-get update*

*- name: Install multiple packages*

*package: name={{item}} state=present*

*loop:*

*- git*

*- docker.io*

*-* *openjdk-17-jdk*

*- name: update apt-repo*

*command: sudo apt-get update*

*- name: install jenkins*

*command: sudo apt-get install jenkins -y*

*- name: start Jenkins and docker service*

*service: name={{item}} state=started*

*loop:*

*- jenkins*

*- docker*

Save the file and run the ansible playbook using below command

#Ansible-playbook playbook1.yml

Now go to the browser, take the public ip of the Ec2 server and use port 8080

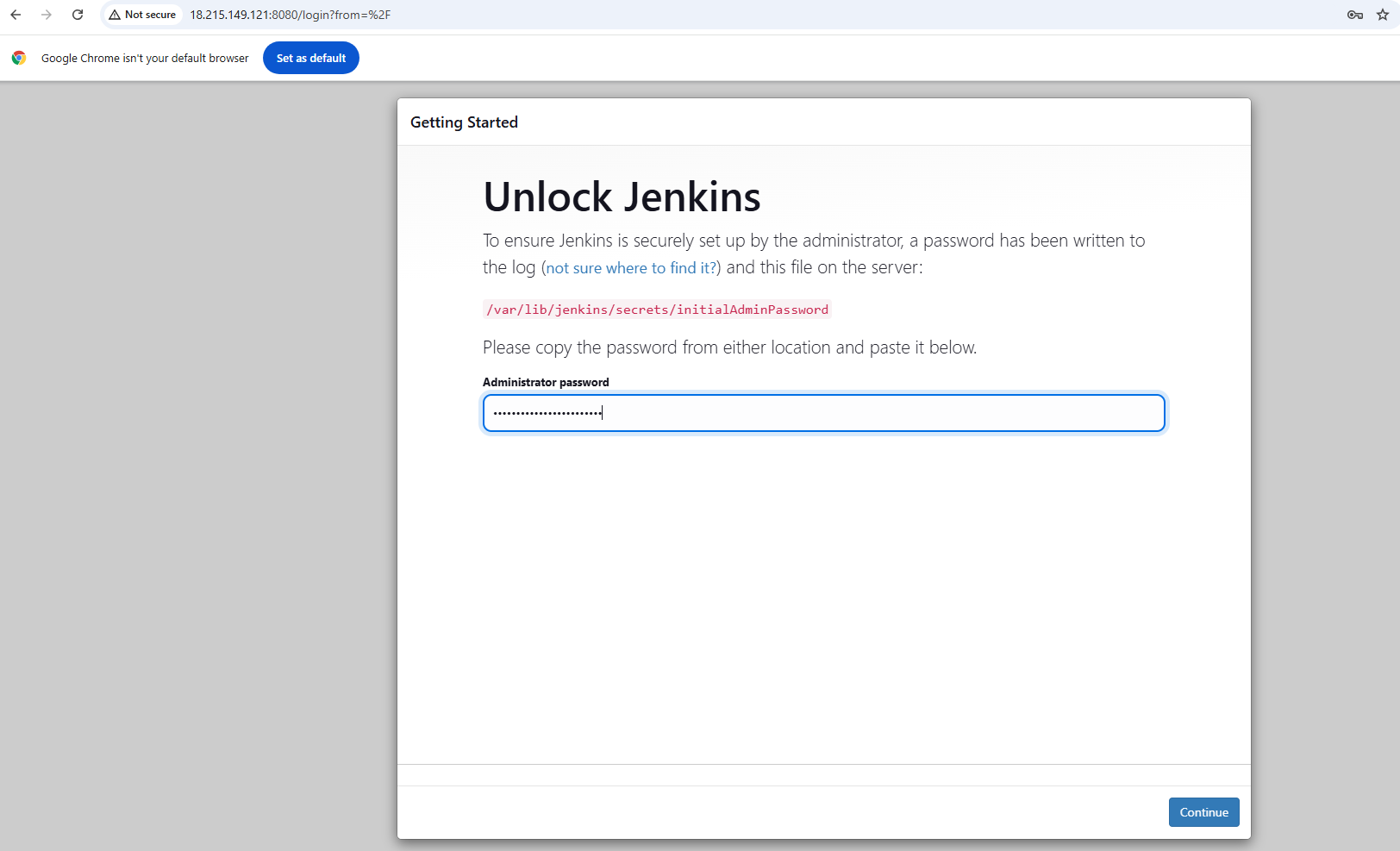
**Setup Jenkins dashboard and login to the Jenkins Dashboard.**

**Setup maven in tools section of Jenkins.**

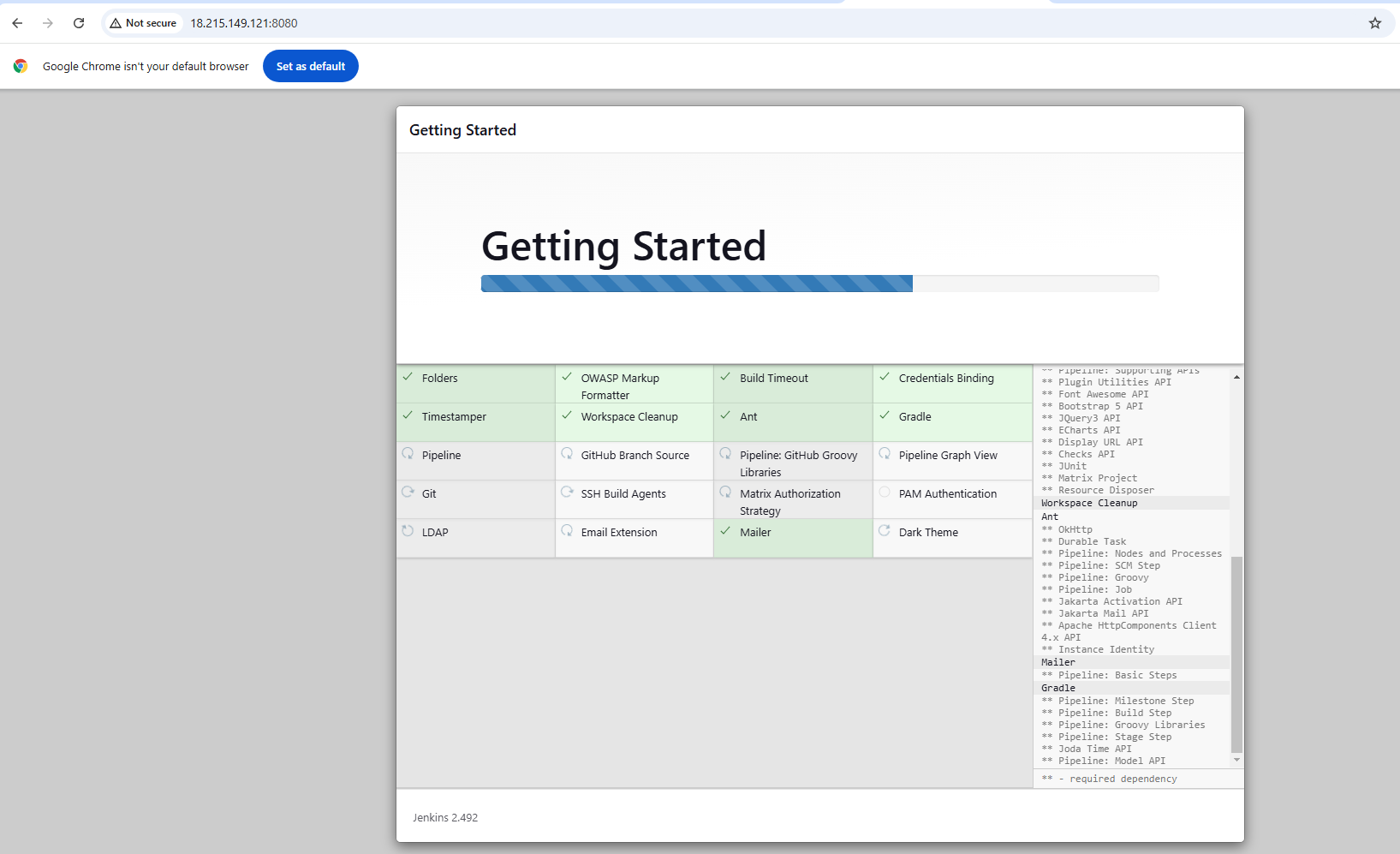
Setup Jenkin,

Goto the public address of the Jenkins server on port 8080 to configure Jenkins Dashboard

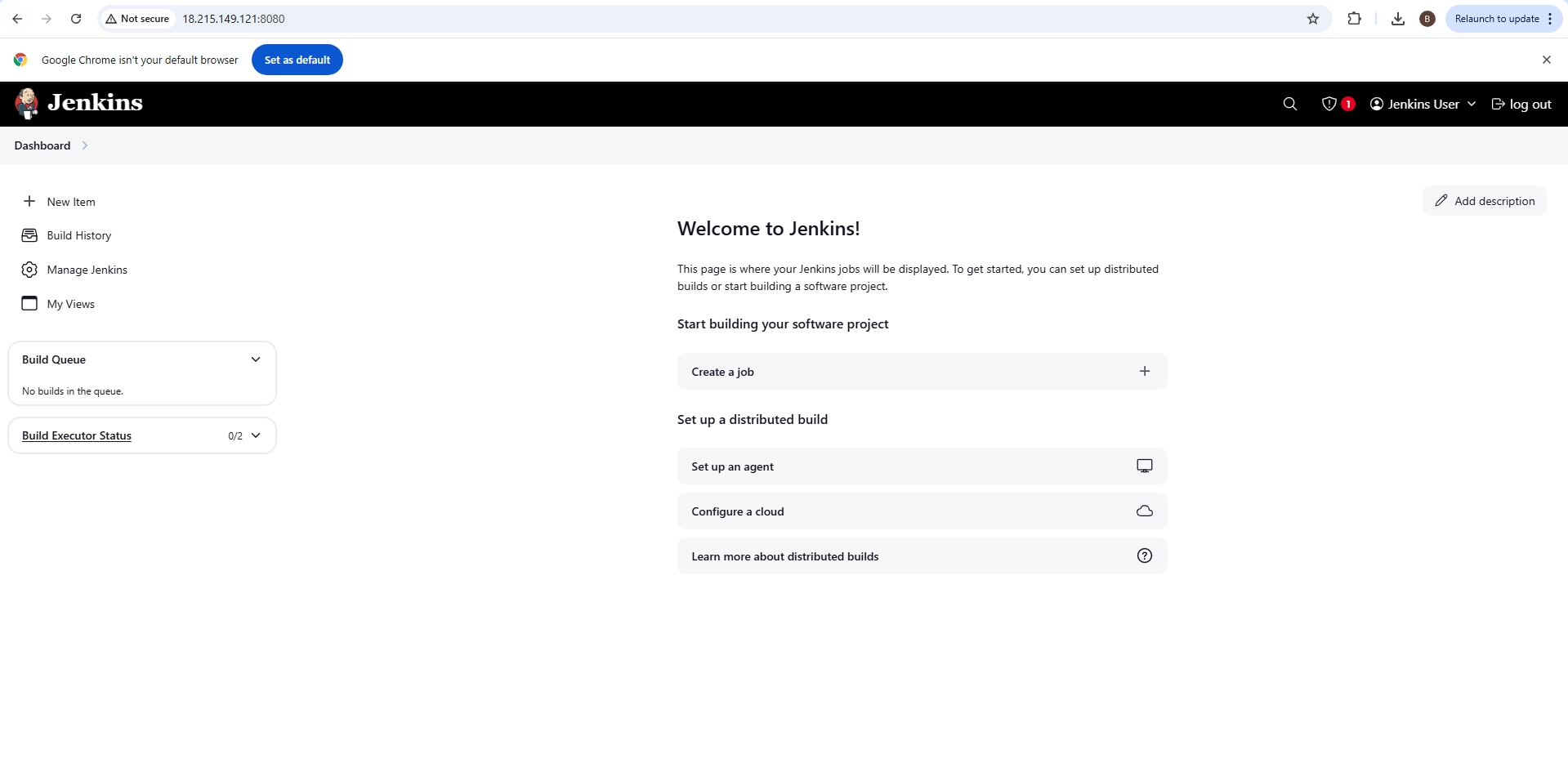
<http://18.215.149.121:8080/:8080>



Setup Jenkins Dashboard.



Once the dashboard is ready, try to access using the public ip on port 8080 with the new Jenkins user.



Step 5: Write the pipeline code to fetch code from Github, test and build the code using Maven

Commands

pipeline{

agent any

**tools{**

**maven 'mymaven'**

**}**

stages{

stage('Clone Repo')

{

steps{

git 'https://github.com/Sonal0409/PGP-FinalProject-2.git'

}

}

stage('Test Code')

{

steps{

sh 'mvn test'

}

}

stage('Build Code')

{

steps{

sh 'mvn clean install package'

}

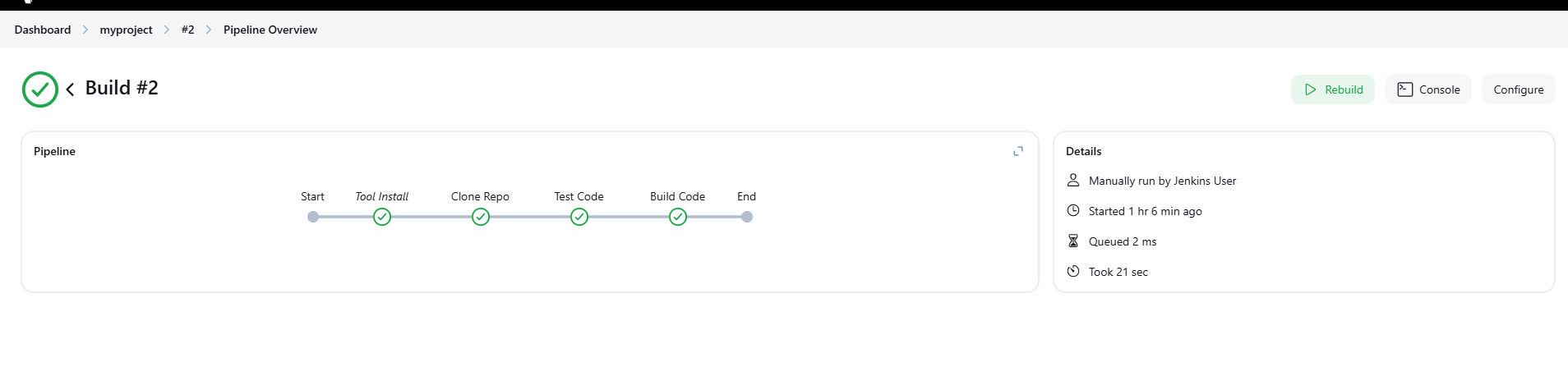
}

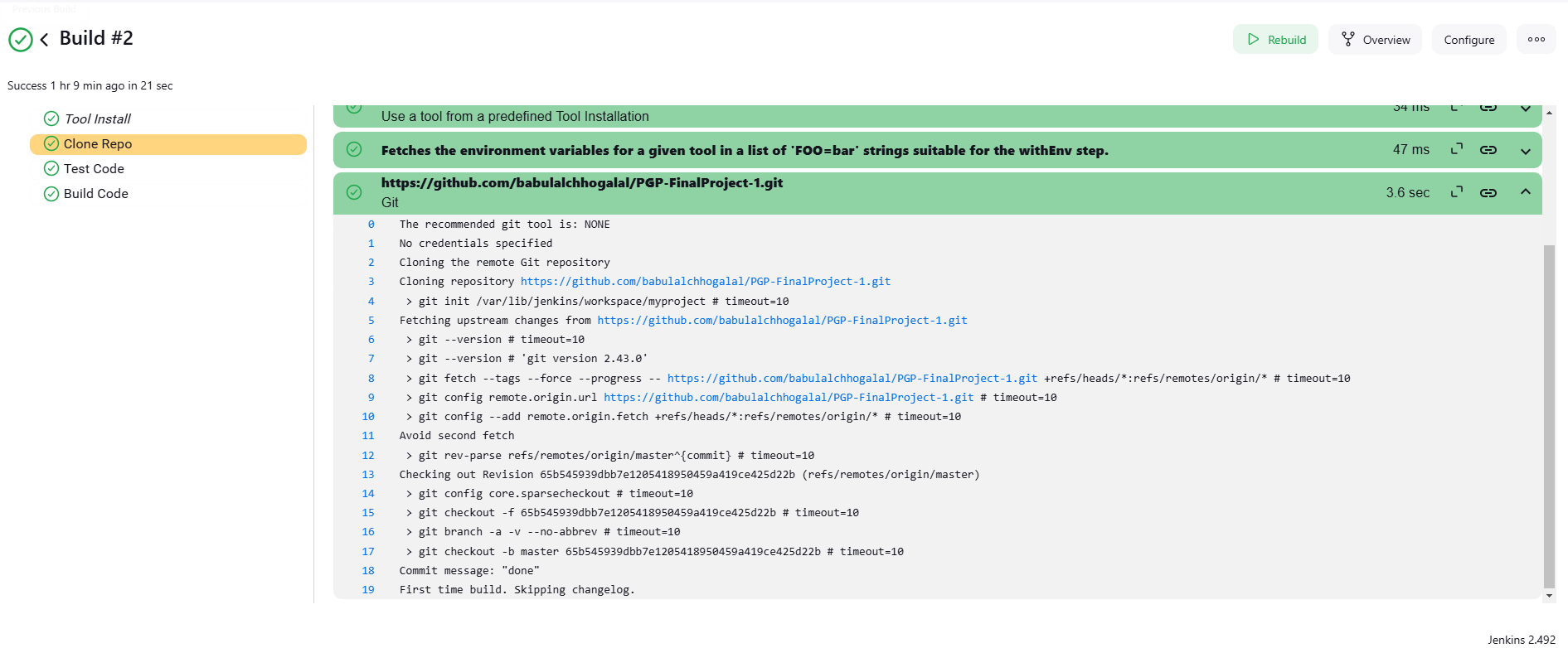
}

}

Execute the pipeline.

Below is the screenshot for successful execution of Pipeline.







**Below is the screenshot for target folder under workspace in Jenkins confirming the build file.**



Step 6: Containerize and implement microservice Architecture.

We will write docker file and save it in Github repo.

#Dockerfile

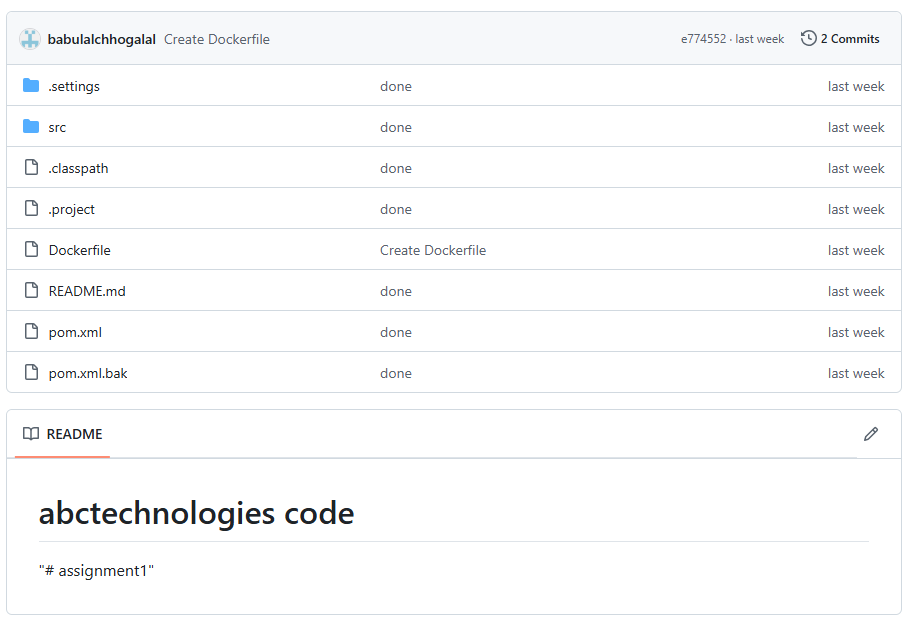
FROM tomcat:9

ADD \*\*/\*.war /usr/local/tomcat/webapps

CMD ["catalina.sh", "run"]

EXPOSE 8080

Add dockerfile to the repo



**We will go to the CICD pipeline in Jenkins add a stage to build dockerfile in to an Image**

We will run the image to deploy application on container.

Final Pipeline code:

pipeline{

agent any

tools{

maven 'mymaven'

}

stages{

stage('Clone Repo')

{

steps{

git 'https://github.com/babulalchhogalal/PGP-FinalProject-1.git'

}

}

stage('Test Code')

{

steps{

sh 'mvn test'

}

}

stage('Build Code')

{

steps{

sh 'mvn package'

}

}

stage('Build Image')

{

steps{

sh 'docker build -t myproject1:$BUILD\_NUMBER .'

}

}

stage('Push the Image')

{

steps{

sh 'docker login -u babulalchhogalal -p Reeeter2091'

sh 'docker tag myproject1:$BUILD\_NUMBER Babulalchhogalal/myproject1:$BUILD\_NUMBER'

sh 'docker push babulalchhogalal/myproject1:$BUILD\_NUMBER'

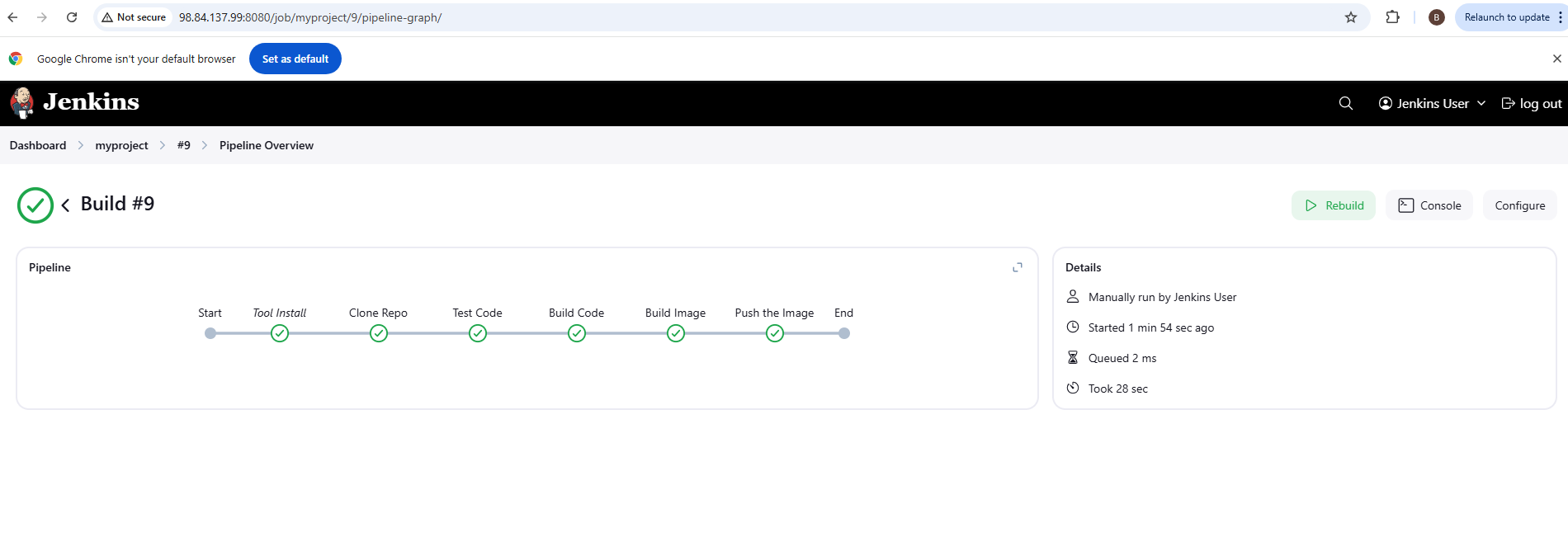
}

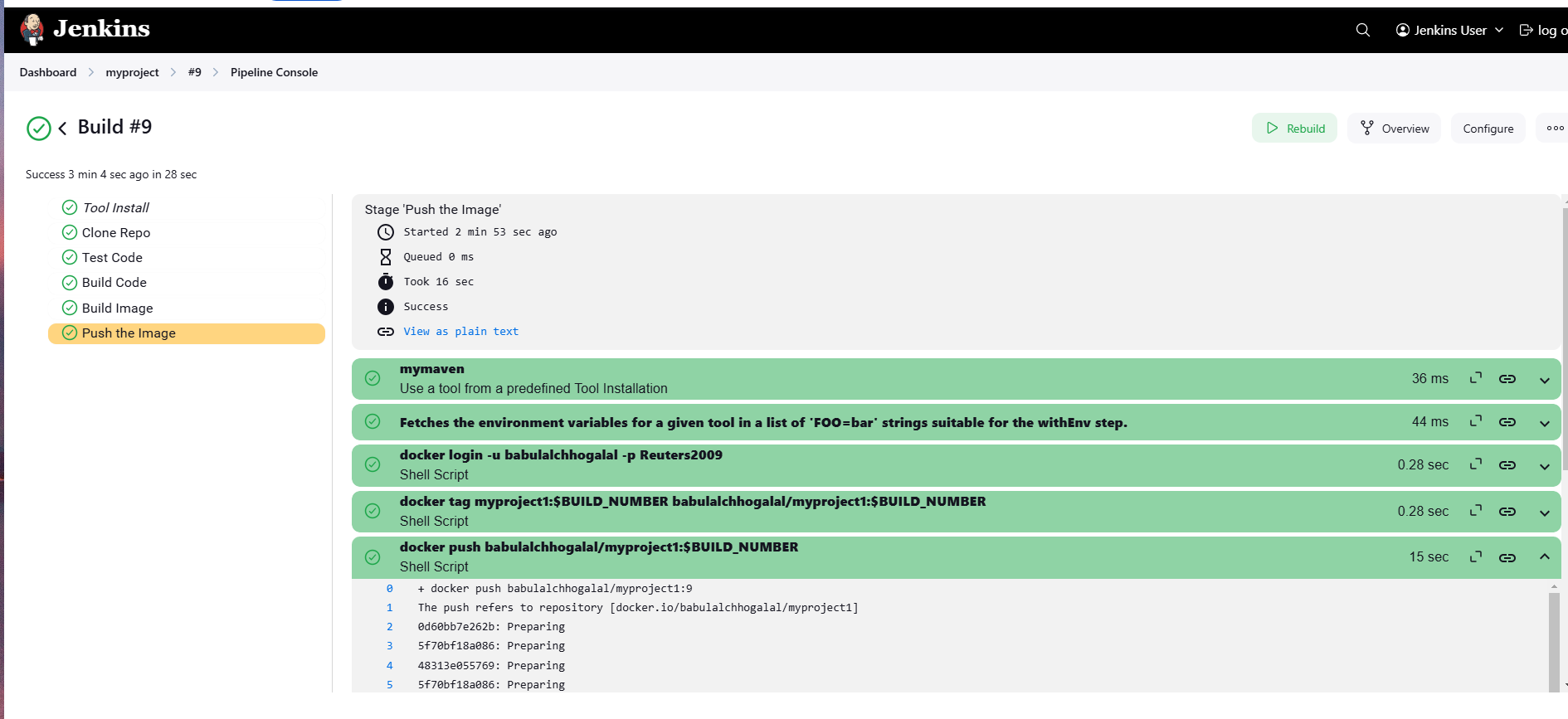
}

}

}

Rerun the build on Jenkins. Screenshots below

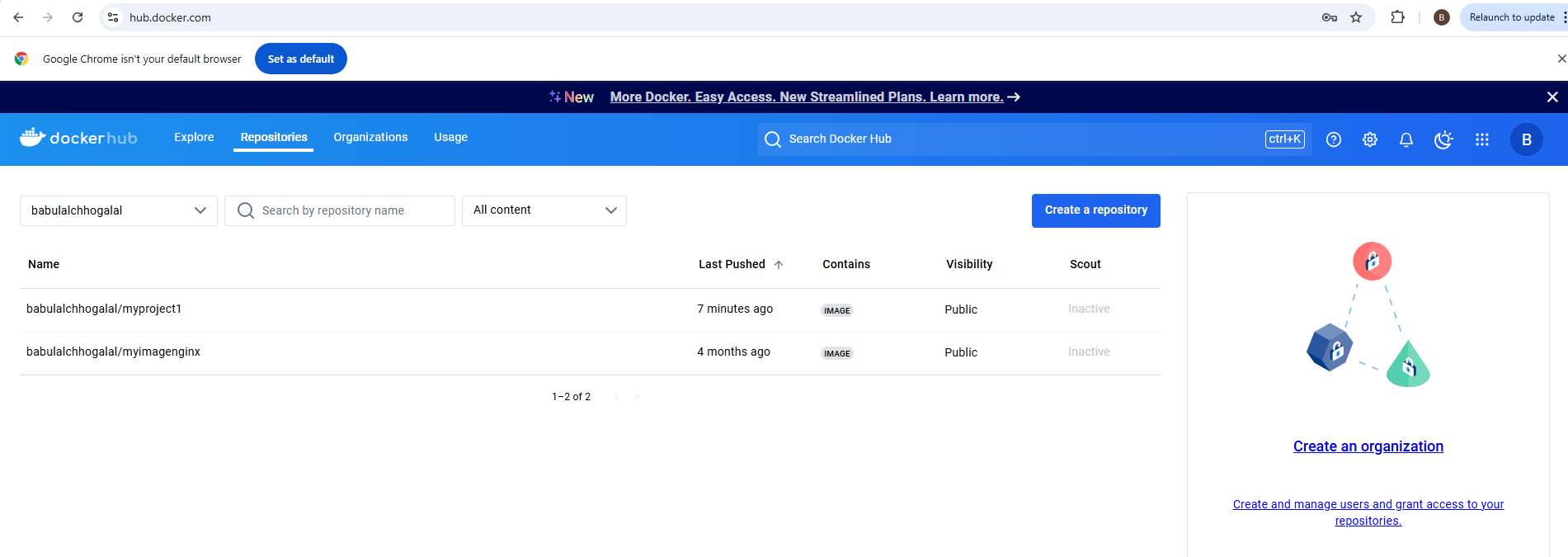




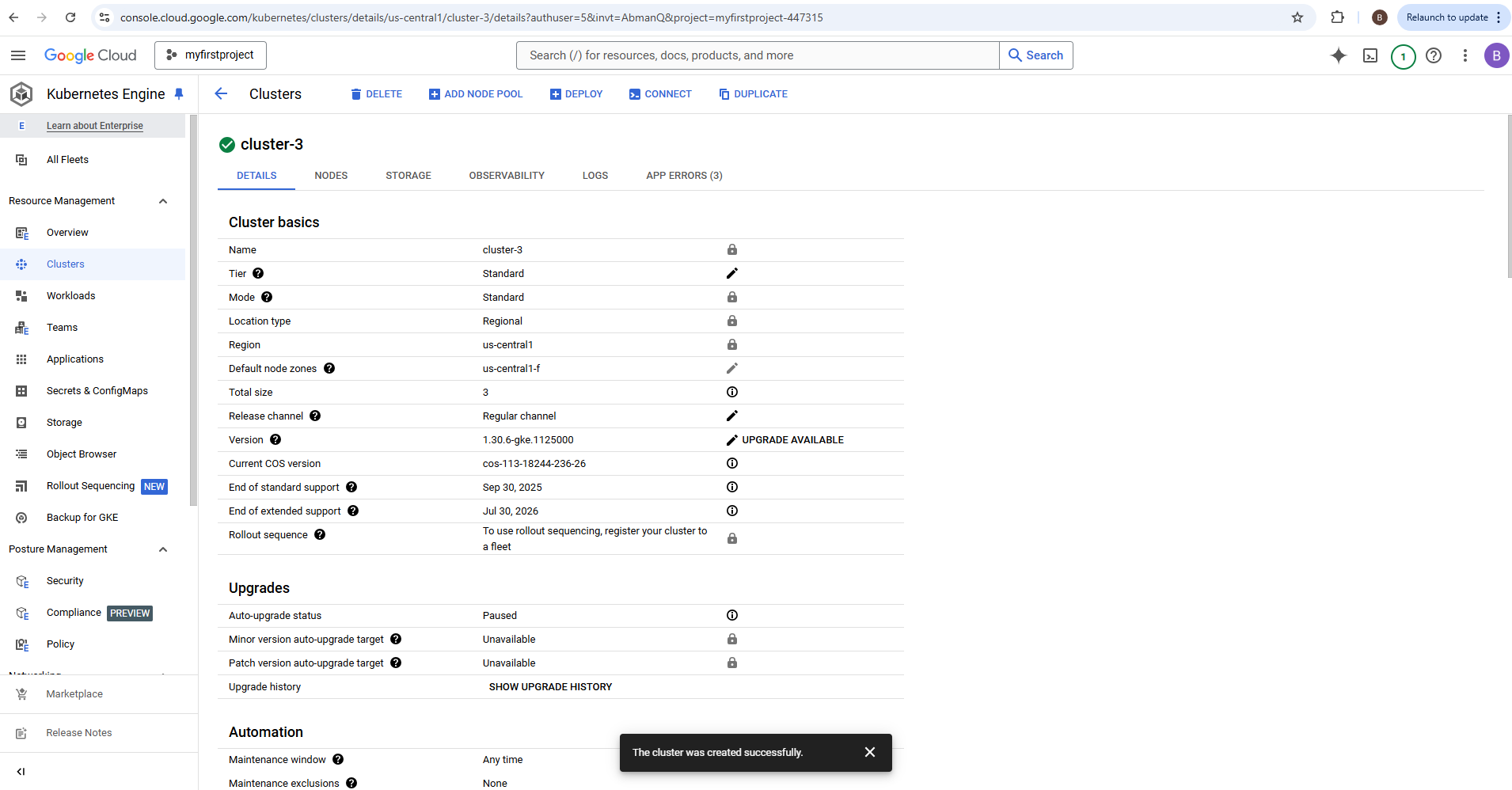
Docker image list



Docker hub screenshot confirming image was pushed to hub.



Step 7: Now that docker image is pushed to hub, we will create a GKE cluster and deploy image.



Once the GKE Cluster is ready. Create a the deployment yaml and service yaml to access the

deployed application

#Deployment.yml

kind: Deployment

apiVersion: apps/v1

metadata:

name: kubeserve

spec:

replicas: 3

minReadySeconds: 10 # wait for 45 sec before going to deploy next pod

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 1

maxSurge: 1 # max number of pods to run for the deployment

selector:

matchLabels:

app: kubeserve

template:

metadata:

name: kubeserve

labels:

app: kubeserve

spec:

containers:

- name: app

image: babulalchhogalal/myproject1:9

---

kind: Service

apiVersion: v1

metadata:

name: kubeserve-svc

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

selector:

app: kubeserve

---

spec:

hostPID: false

hostNetwork: false

---

volumes:

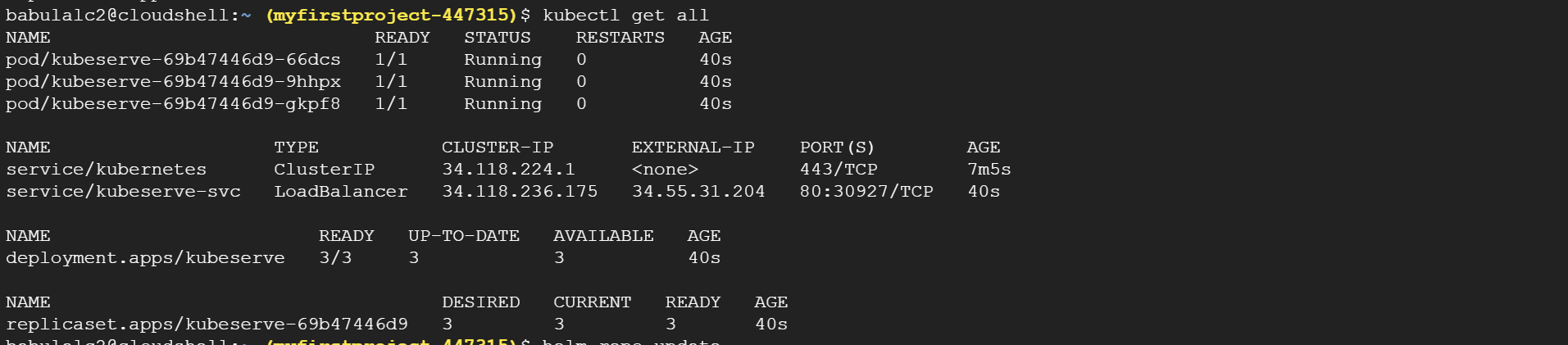
- name: logs-volume

persistentVolumeClaim:

claimName: logs-pvc

save the file and run the below command to deploy the pods and services

kubectl create -f deployment.yml



Monitoring using Prometheus/Grafana

**Install Prometheus and Grafana on Kubernetes cluster:**

We will be using Helm to install Prometheus:

* Add helm repo:

# helm repo **add prometheus-community** [**https://prometheus-community.github.io/helm-charts**](https://prometheus-community.github.io/helm-charts)

* Update helm repo:

# helm repo update

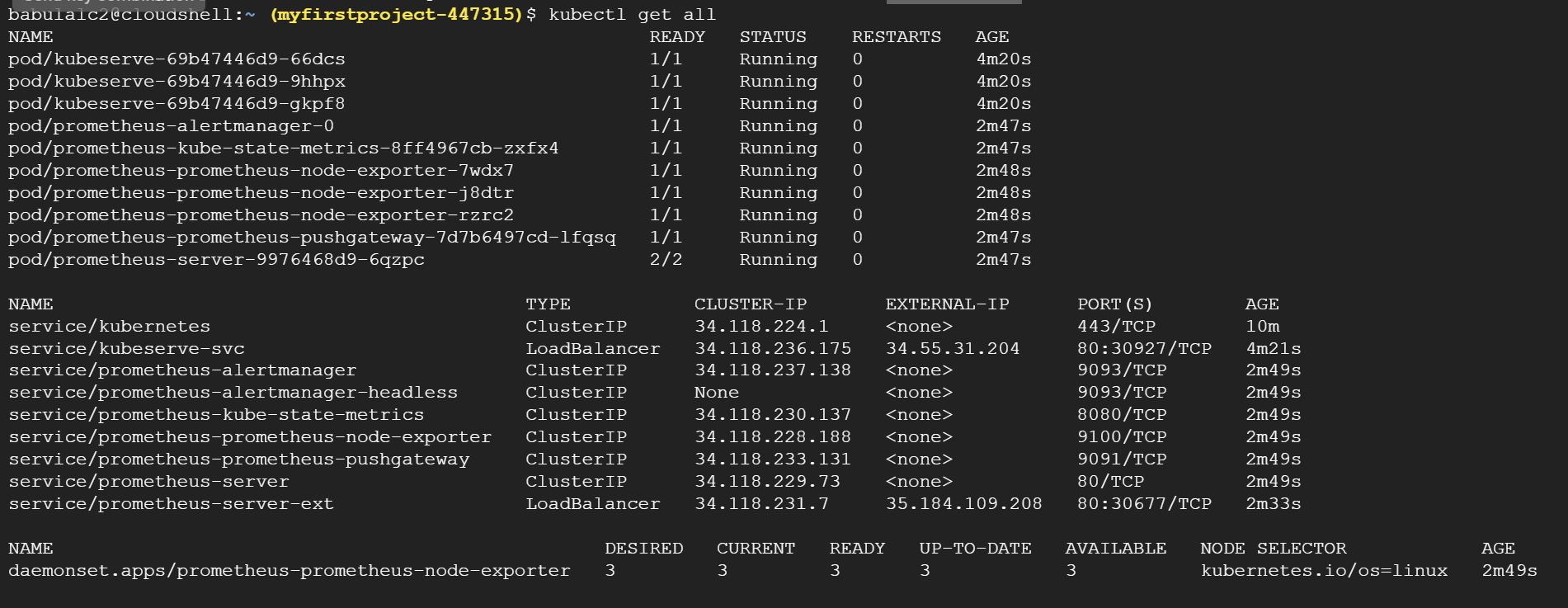
* Helm install command to install Prometheus

# helm install prometheus prometheus-community/Prometheus

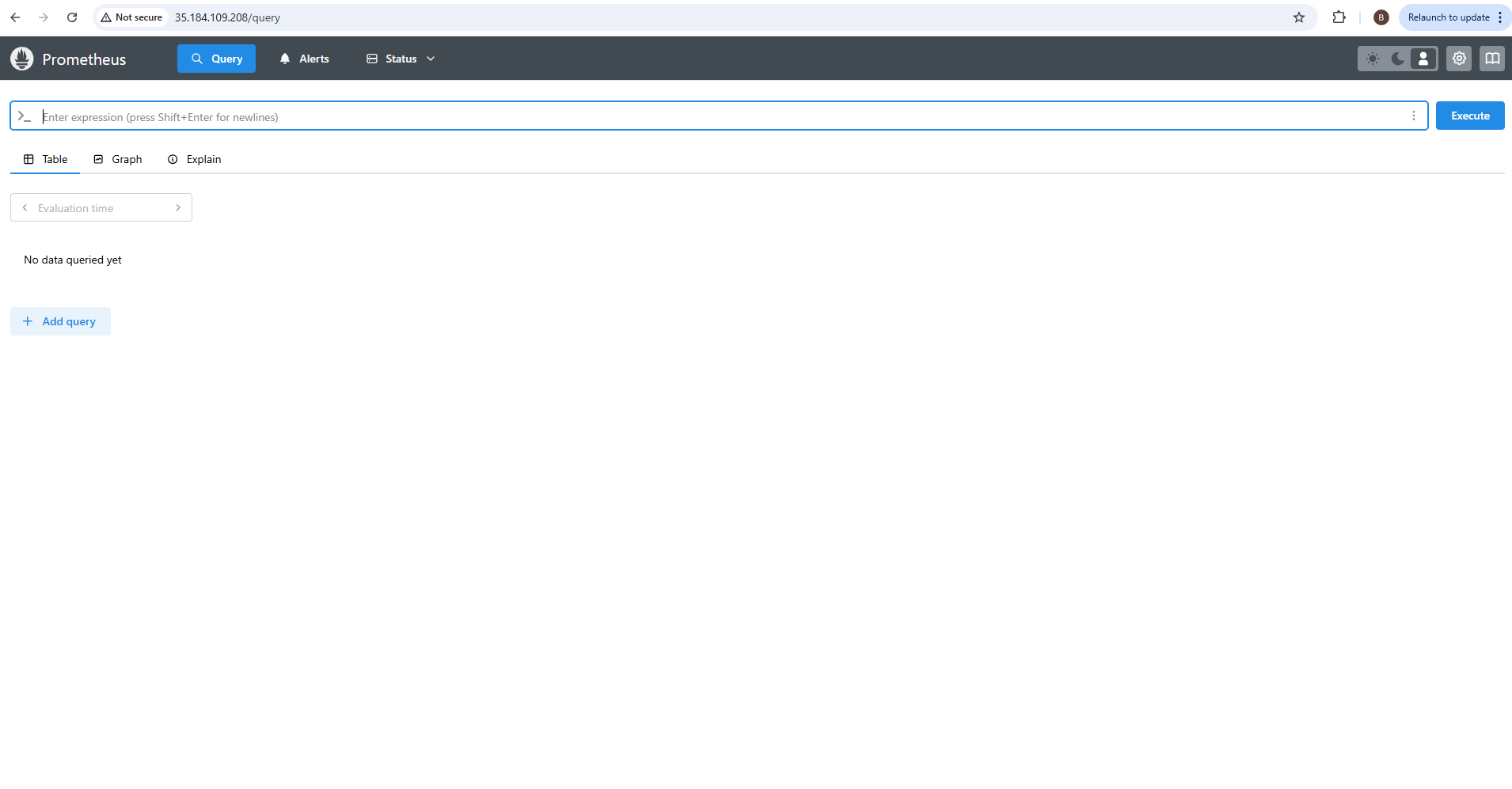
* Expose Prometheus service. This is required to access Prometheus server using browser

# kubectl expose service prometheus-server --type=LoadBalancer --target-port=9090 -- name=prometheus-server-ext

* Take the output for kubectl get all to check the Prometheus server POD running

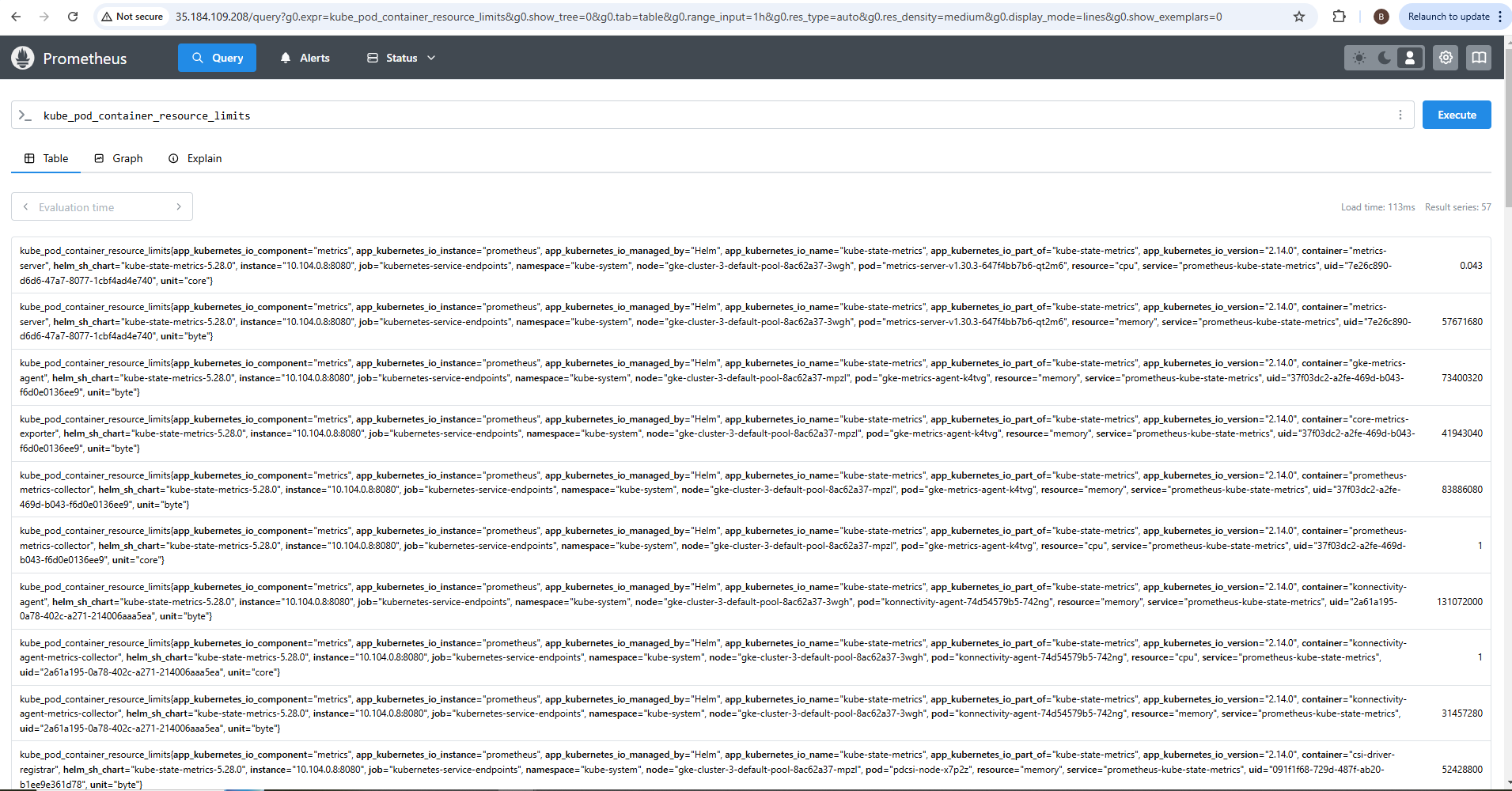


* Note the external ip of the Prometheus server and access it using Browser. It should load the Prometheus dashboard. Screenshot below



* Now execute the below query on the dashboard.

**Kube\_pod\_container\_resource\_limits**



**Installing Grafana through Helm**

* Add helm repo

helm repo add grafana https://grafana.github.io/helm-charts

* Update helm repo

helm repo update

* Install helm

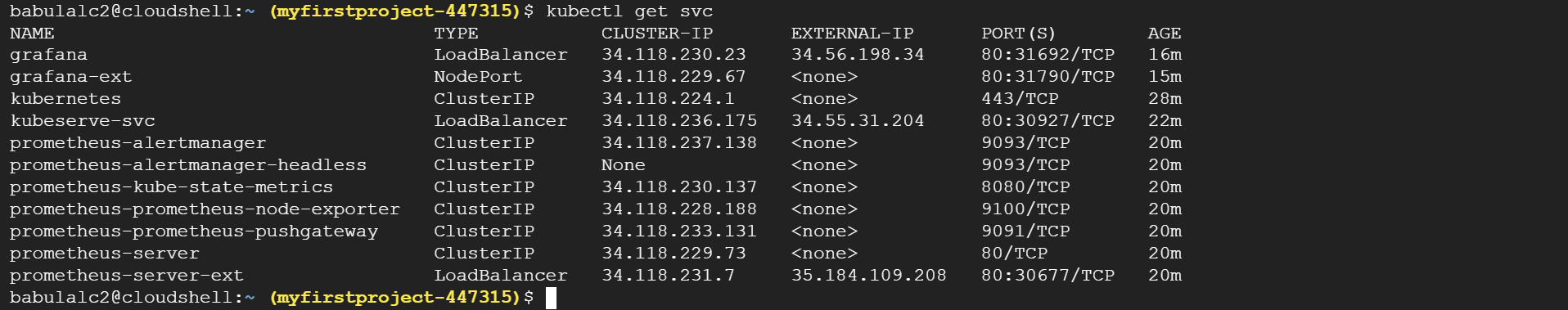
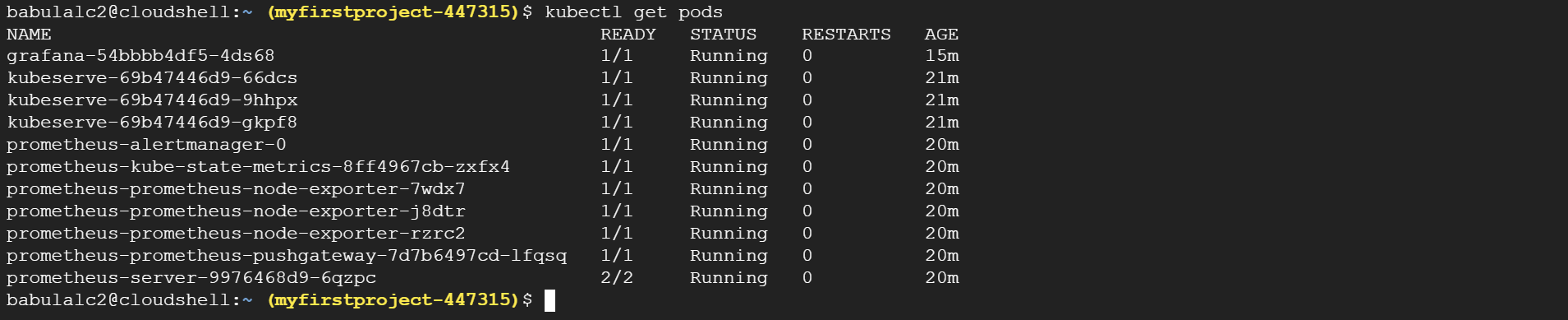
helm install grafana grafana/grafana

* Username is admin and get password by running below command:

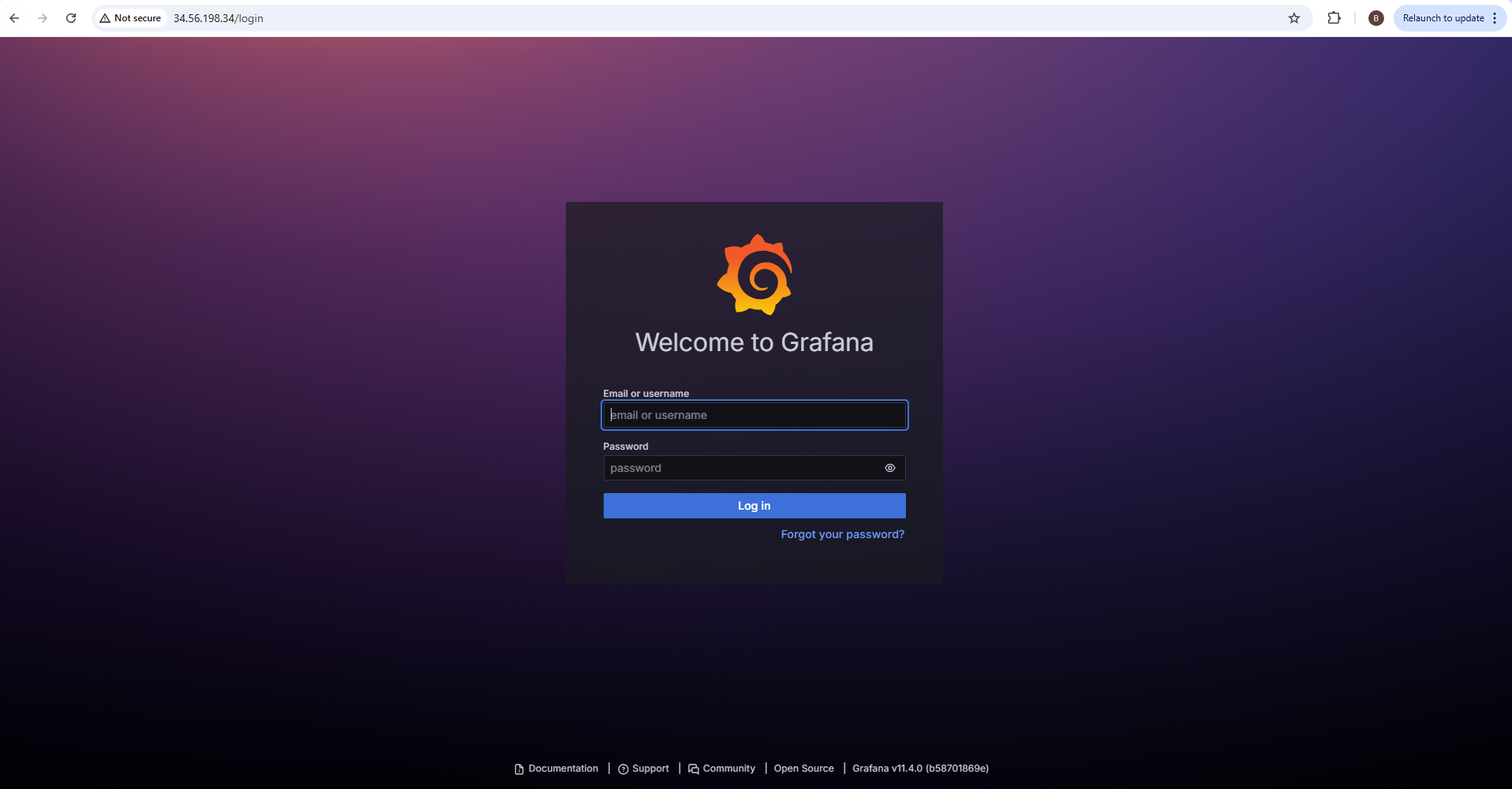
kubectl get secret --namespace default grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

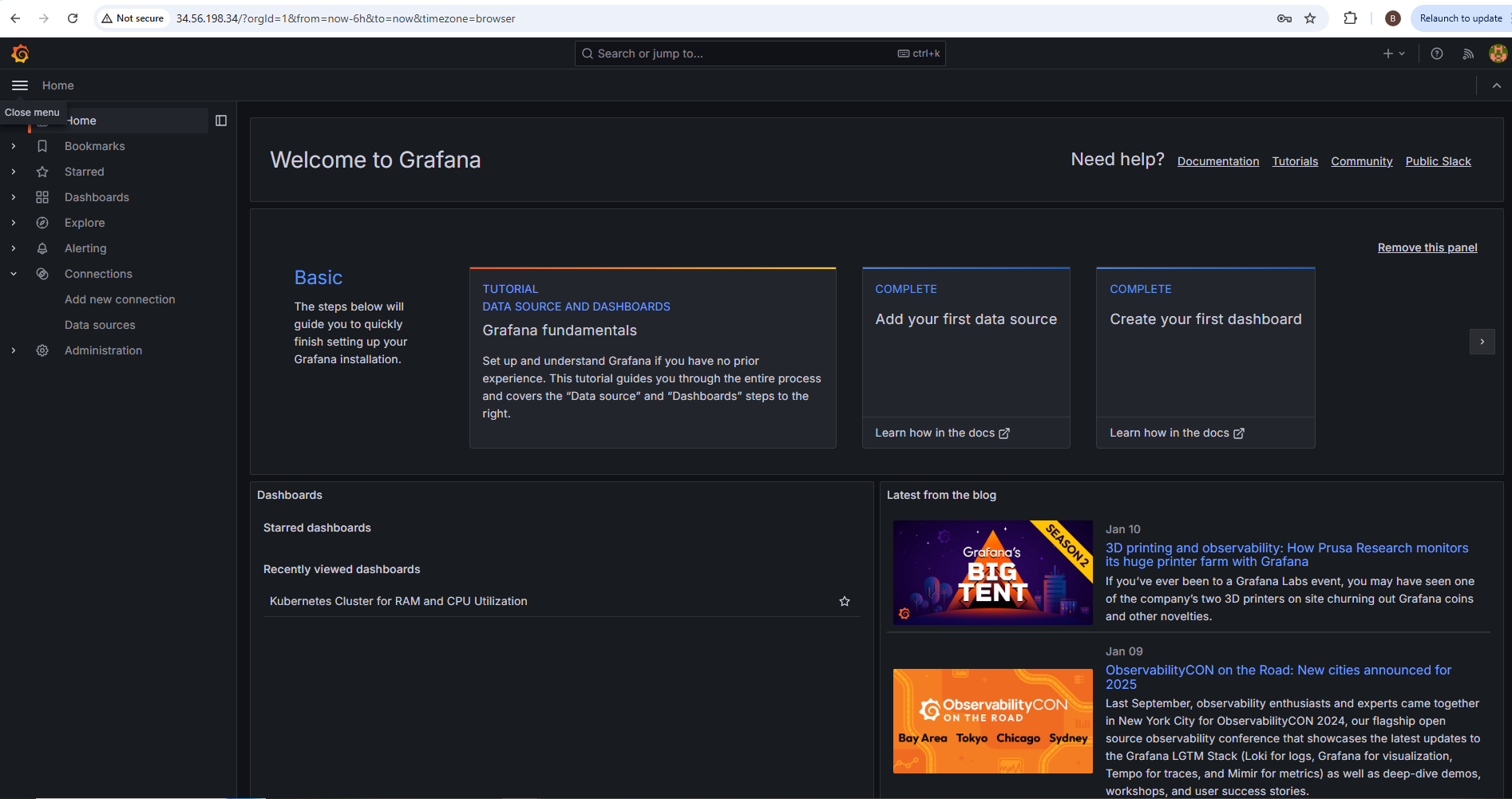
* Expose Grafana Service

kubectl expose service grafana --type=LoadBalancer --target-port=3000 --name=grafana-ext



Log into grafana: username will be admin and password will be the text that was retrieved earlier.





In Grafana we will setup the datasource as Prometheus using the Prometheus IP address:portnumber

Click on Data source --> scroll down -- to add prometheous URL field. Copy the prometheous dashboard URL -> https://ipaddress:portnumber

Scroll down --> click on save and test

Grafana will now be able to fetch data from prometheous

Import the dashboard to display data

Go to Homepage of grafana --> click on Dashboards --> Click on Import dashboard --> the id of dashboard is 16734 and click on load

Scroll down and select prometheous data source -> click on Import

Use dashboard, 16734 and add the alert

