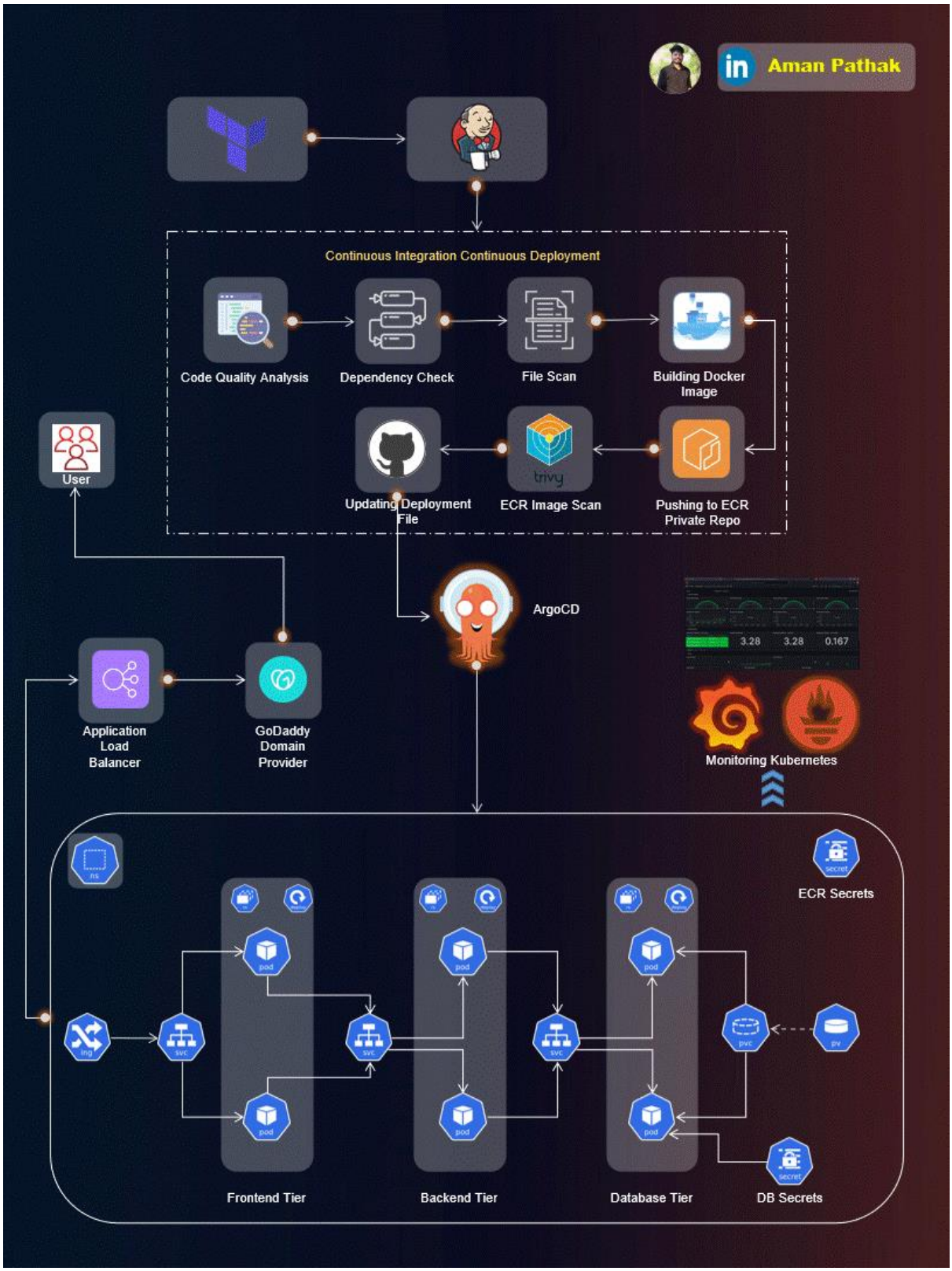
Advanced End-to-End DevSecOps Kubernetes Three-Tier Project using AWS EKS, ArgoCD, Prometheus, Grafana, and Jenkins



PHASE1: Infrastructure setup

PHASE2: Jenkins and Argocd setup

PHASE3: Monitoring setup

Phase1: Infrastructure Setup

Step1: create an ec2 instance and install docker, Jenkins, docker, terraform, kubectl, awscli, trivy, eksctl using user-data.

Allocate the storage 30gb and use min of 4 cpu instance

Tools-install URL: https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/blob/master/Jenkins-Server-TF/tools-install.sh

Step2: Access Jenkins:

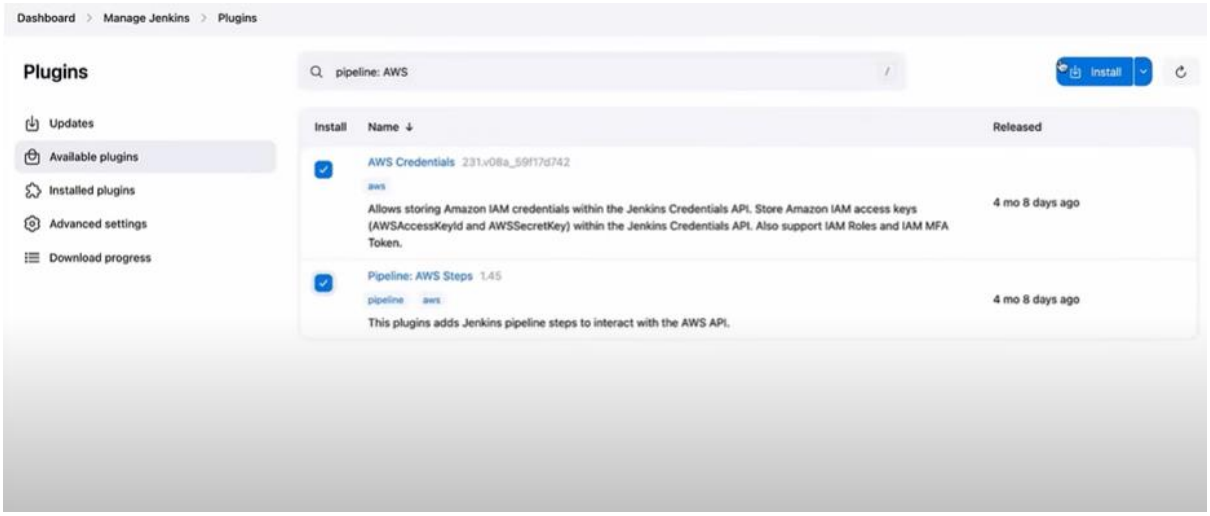
Open a web browser and go to **http://your_server_ip_or_domain:8080**.

You will see a page asking for the initial admin password. Retrieve it using:

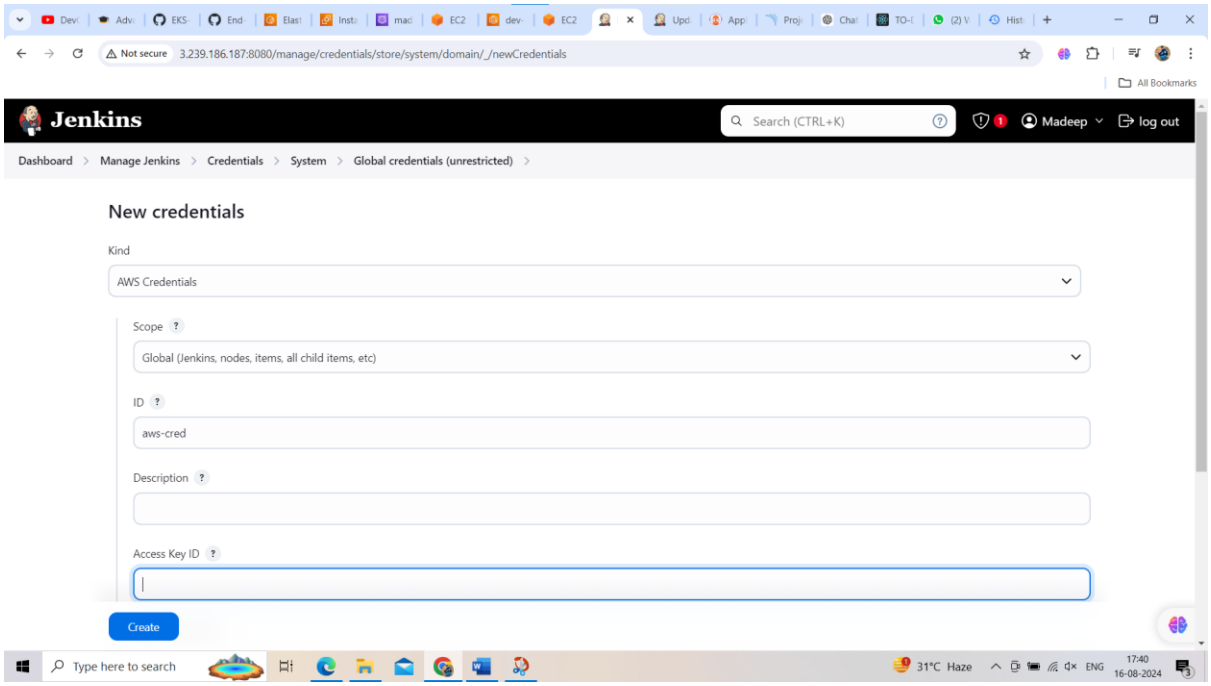
sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Enter the password, install suggested plugins, and create your first admin user.

Goto plugins and install **AWS Credentials , AWS Steps and Terraform**

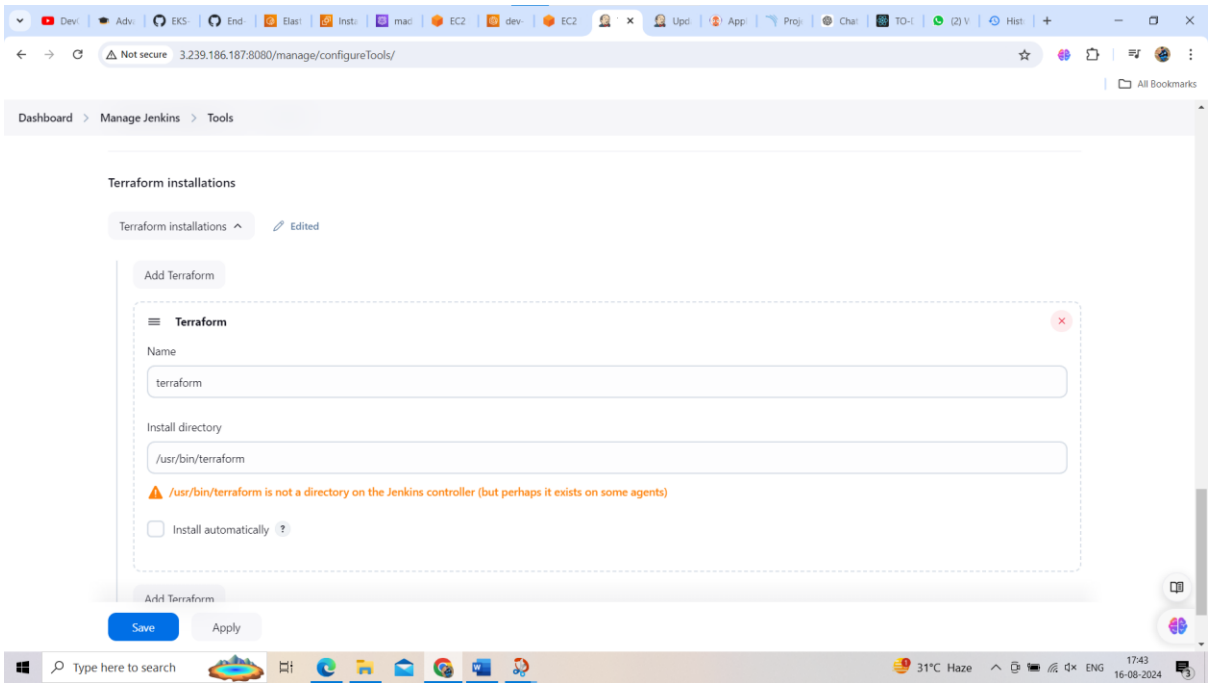


Step 3: Create Access key and Secret key for IAM user with Administrator access and Add those credentials in Jenkins credentials



Step4: Install terraform in Jenkins

Goto to tools → Terraform installations→ install from the local

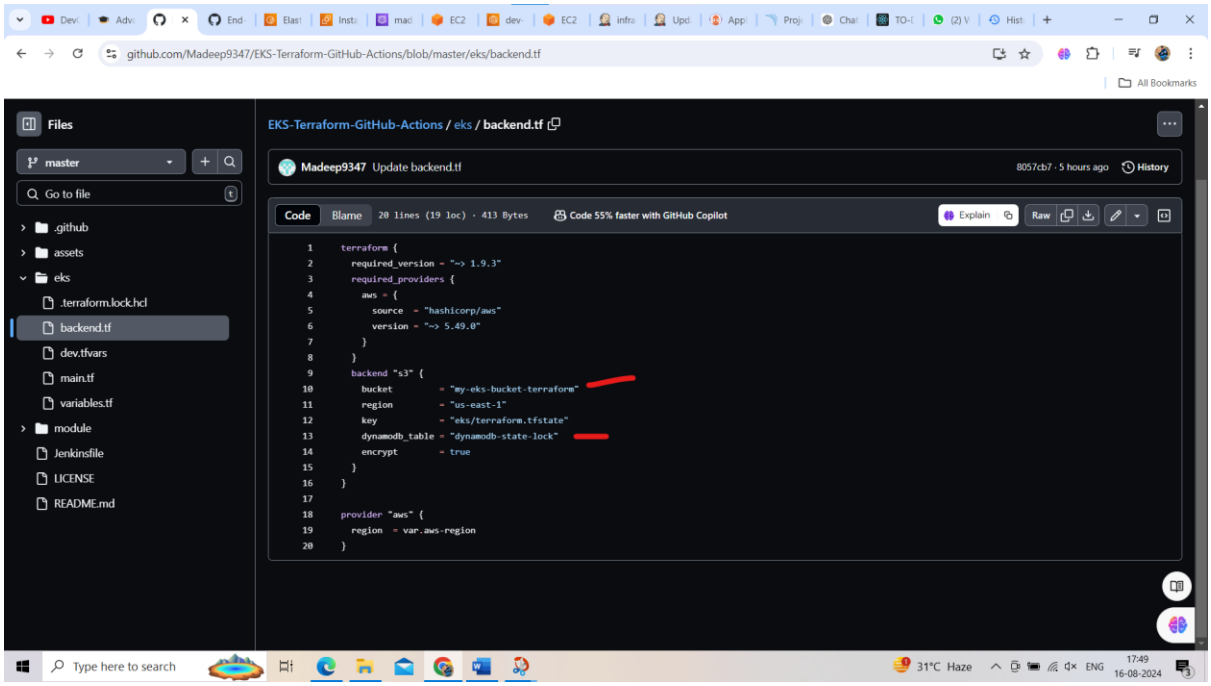


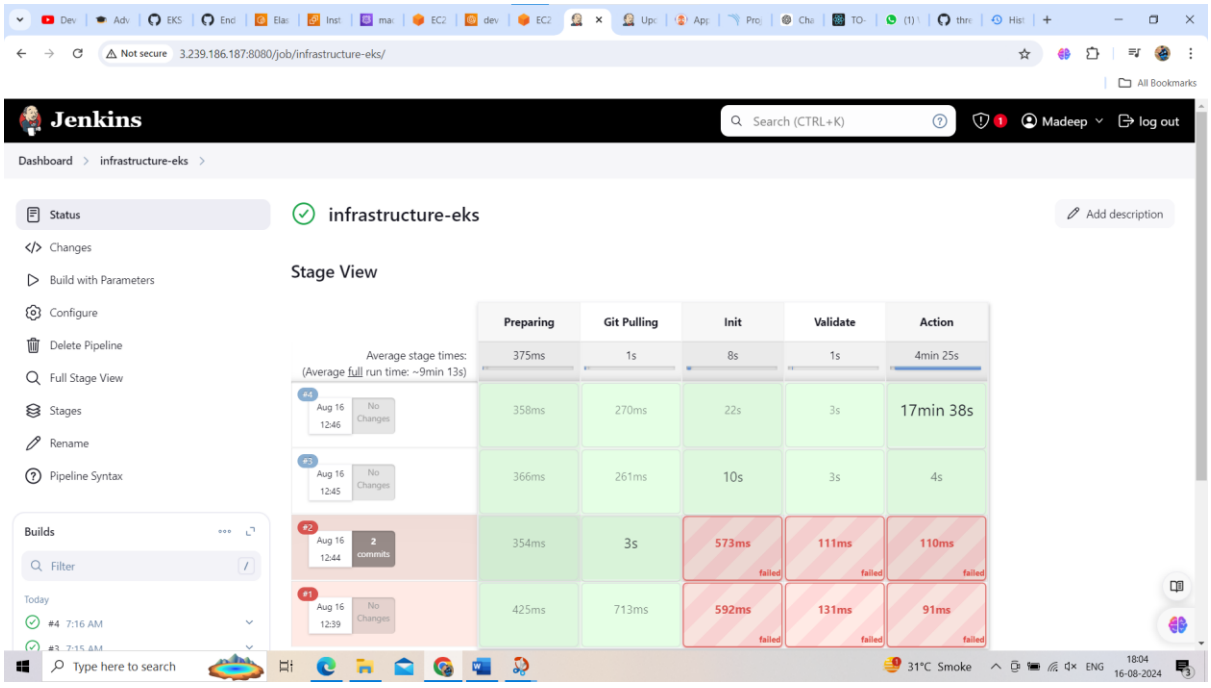
Step 5: Create Jenkins job

Create pipeline Jenkins job and use the jenkinsfile

https://github.com/Madeep9347/EKS-Terraform-GitHub-Actions/blob/master/Jenkinsfile

edit the backend configuration with your s3 bucket name and dynamo table





Step 6: Create an instance (Jump server) in the vpc created by Terraform and install the eksctl , awscli, kubectl, helm.

Link to tools : https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/blob/master/Jenkins-Server-TF/tools-install.sh

Step 7: configure the aws credentials in the jump server

```
ubuntu@ip-10-16-13-49:~$ aws configure

AWS Access Key ID [****************OSO4]: AWS Secret Access Key [****************72vV]:
Default region name [us-east-1]:
Default output format [None]:
ubuntu@ip-10-16-13-49:~$
```

Step 8: Update the Kubeconfig file

aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-1

check the jump-server able to communicate with the Kubernetes cluster

kubectl get nodes

Step 9: Now, we will configure the Load Balancer on our EKS because our application will have an ingress controller.

Download the policy for the LoadBalancer prerequisite.

curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam_policy.json

Step 10: Create the IAM policy using the below command

aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json

Step 11: Create a Service Account by using below command and replace your account ID with your one

eksctl create iamserviceaccount --cluster=dev-medium-eks-cluster --namespace=kube-system --name=aws-load-balancer-controller --role-name AmazonEKSLoadBalancerControllerRole --attach-policy-arn=arn:aws:iam::590184124026:policy/AWSLoadBalancerControllerIAMPolicy --approve --region=us-east-1

Check the load balancer is created or not

```
ubuntu@ip-10-16-13-49:~$ kubectl get sa -n kube-system
NAME                                 SECRETS   AGE
attachdetach-controller              0         5h36m
aws-cloud-provider                   0         5h36m
aws-load-balancer-controller         0         4h57m
```

Step 12: Run the below command to deploy the AWS Load Balancer Controller

sudo snap install helm --classic
helm repo add eks https://aws.github.io/eks-charts
helm repo update eks
helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=dev-medium-eks-cluster --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller

After 2 minutes, run the command below to check whether your pods are running or not.

kubectl get deployment -n kube-system aws-load-balancer-controller

Step 13: Create namespace for Argocd

kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml

kubectl get all -n argocd

kubectl get svc -n argocd

Step 14: Edit the argocd-server type Cluster ip to LoadBalance

```
  selector:
    app.kubernetes.io/name: argocd-server
  sessionAffinity: None
  type: LoadBalancer
```

Step 15: Now access the Argocd with loadbalancer dns name

Default username: admin

Initial admin password is stored in

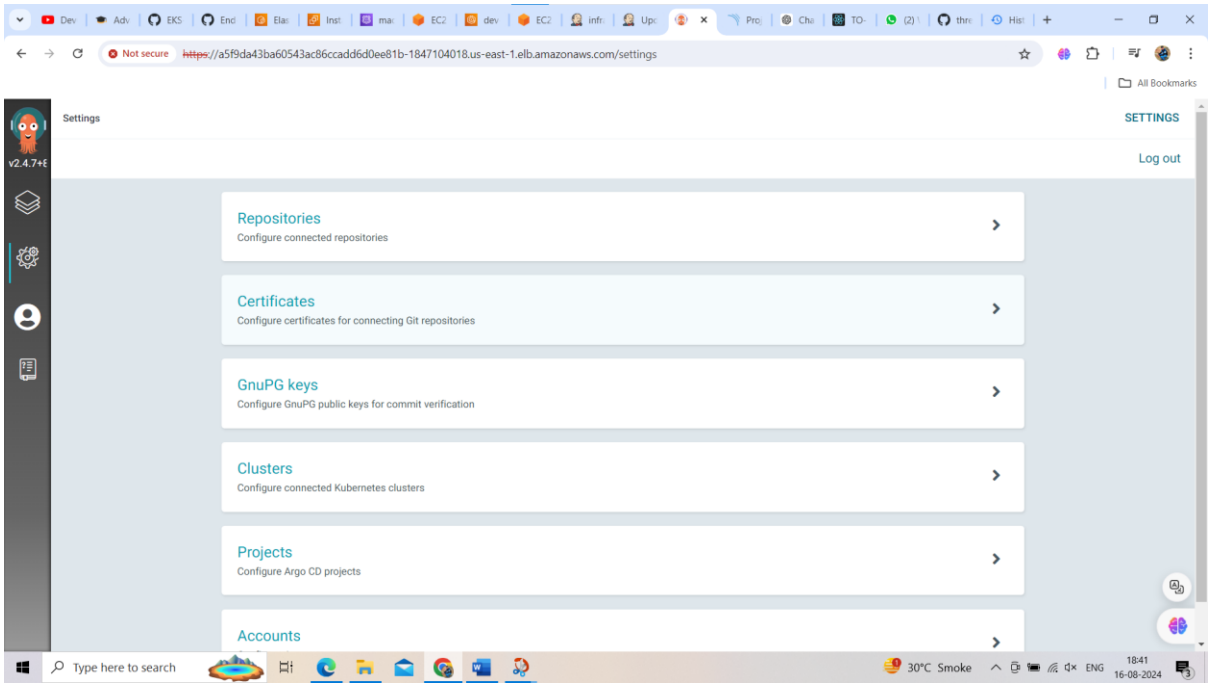
kubectl edit secret argocd-initial-admin-secret -n argocd

```
apiVersion: v1
data:
  password: eC1XZk1xVFdlakxqSHVxdg==
kind: Secret
```

get the password and decode with base64

```
ubuntu@ip-10-16-13-49:~$ echo eC1XZk1xVFdlakxqSHVxdg== | base64 --decode
x-WfMqTWejLjHuqvubuntu@ip-10-16-13-49:~$
```

Copy the password and login to the argocd



PHASE 2: Jenkins(Continuous Integration) and Argocd (Continuous Deployment/Delivery)

Step 1: Sonarqube setup

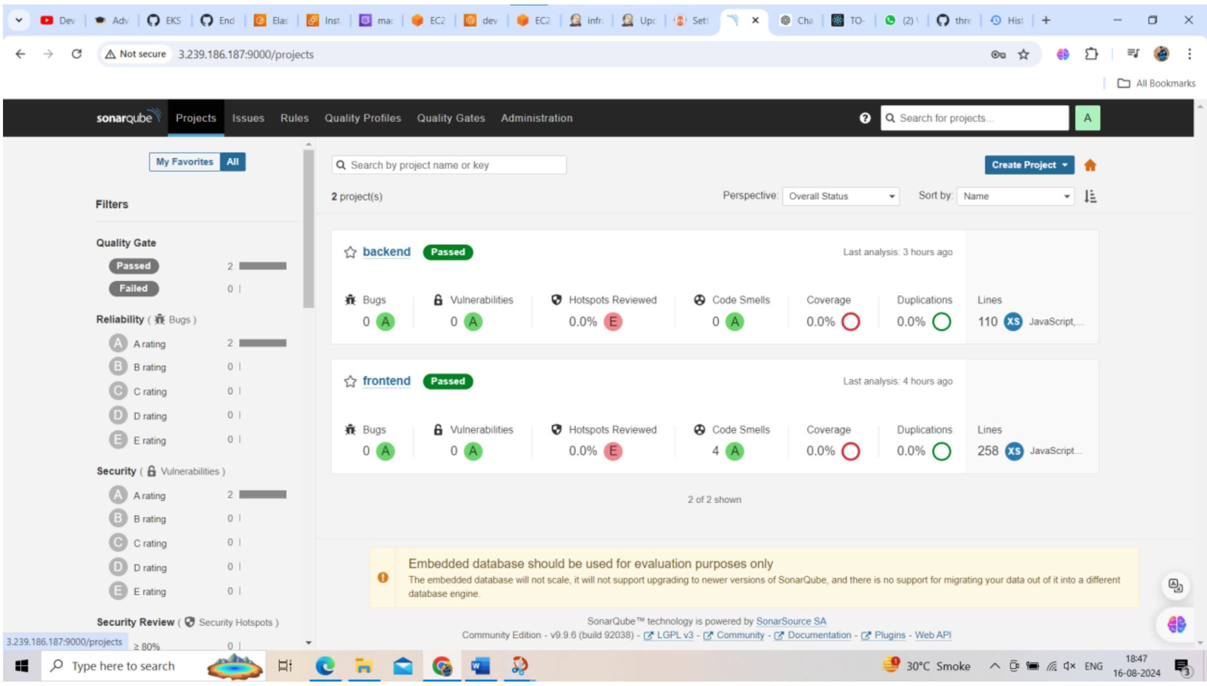
We are running the sonarqube in the port 9000 and use the same Jenkins ip address

```
jenkins@ip-10-0-25-107:~$ docker ps
CONTAINER ID   IMAGE                        COMMAND                  CREATED       STATUS       PORTS                                       NAMES
08be6e2d4d82   sonarqube:lts-community      "/opt/sonarqube/dock…"   7 hours ago   Up 7 hours   0.0.0.0:9000->9000/tcp, :::9000->9000/tcp   sonar
jenkins@ip-10-0-25-107:~$
```

Access SonarQube:

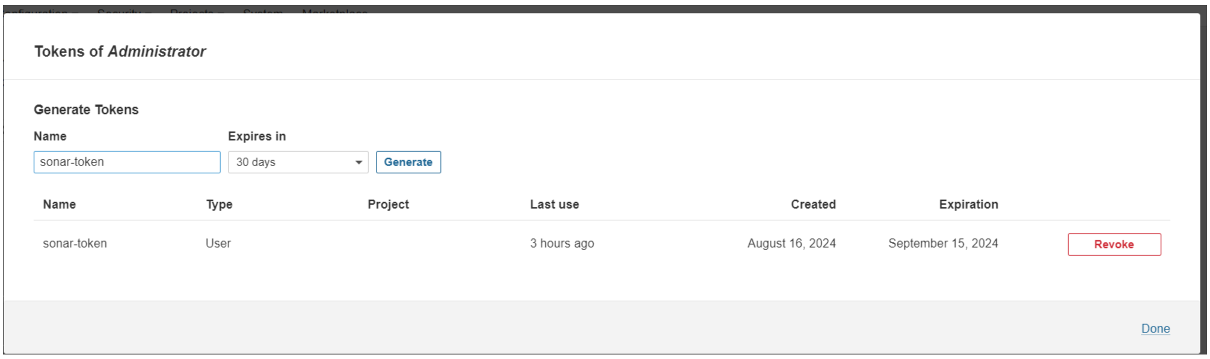
Open a web browser and go to http://your_server_ip_or_domain:9000.

The default username and password are both admin.

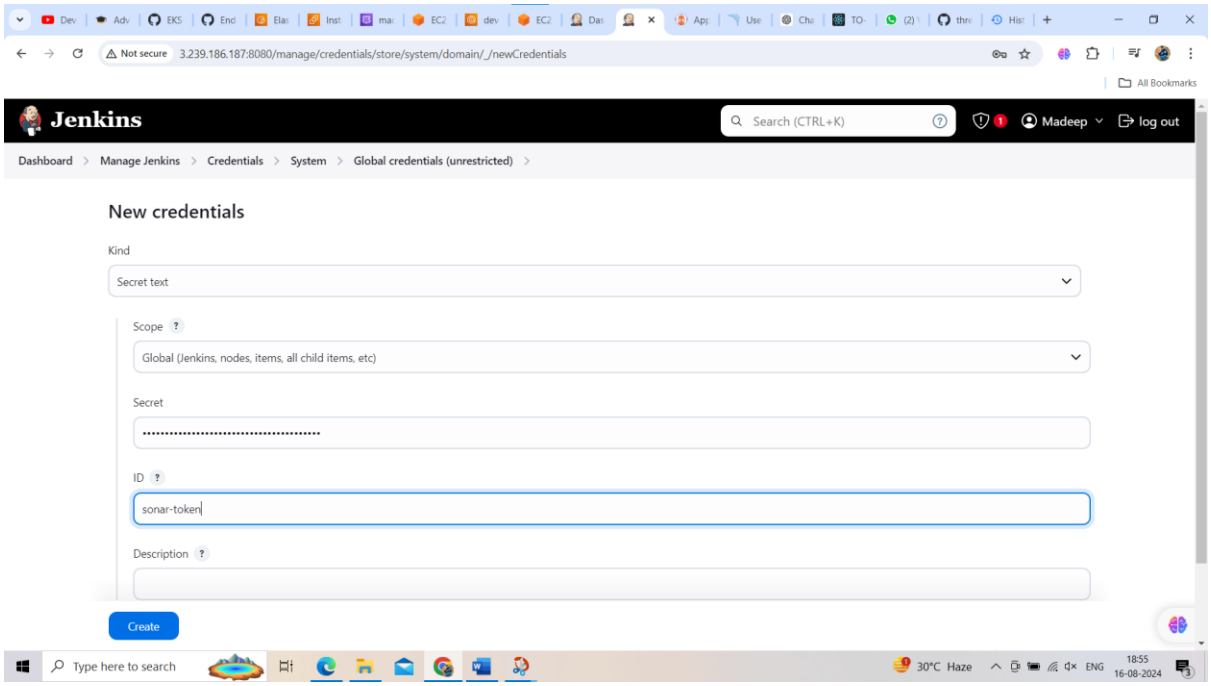


Step2: Create the Token

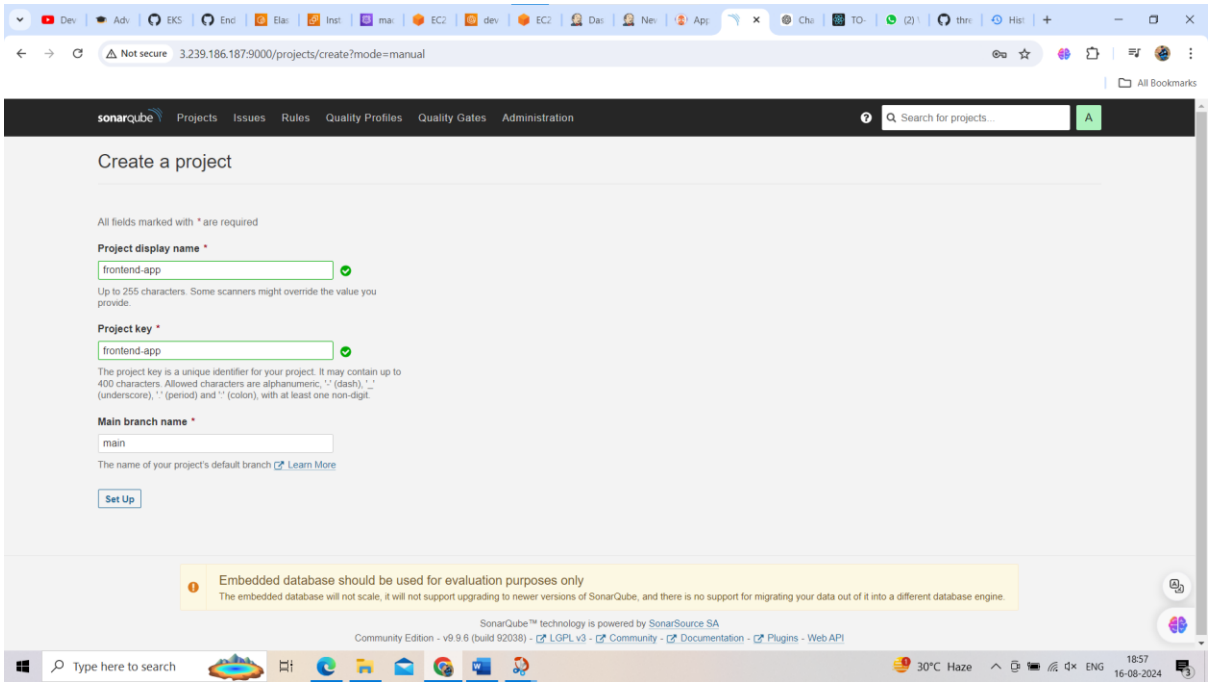
Goto administration → security → users→ generate token



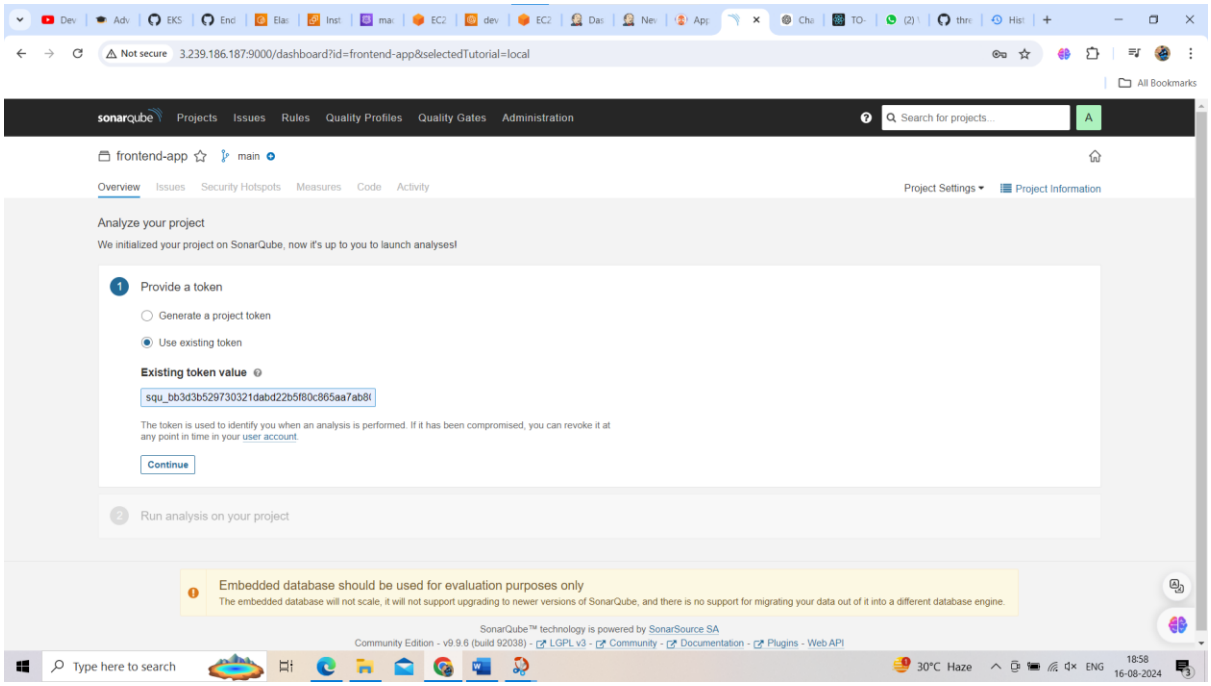
Copy the token and add this token in the Jenkins credentials

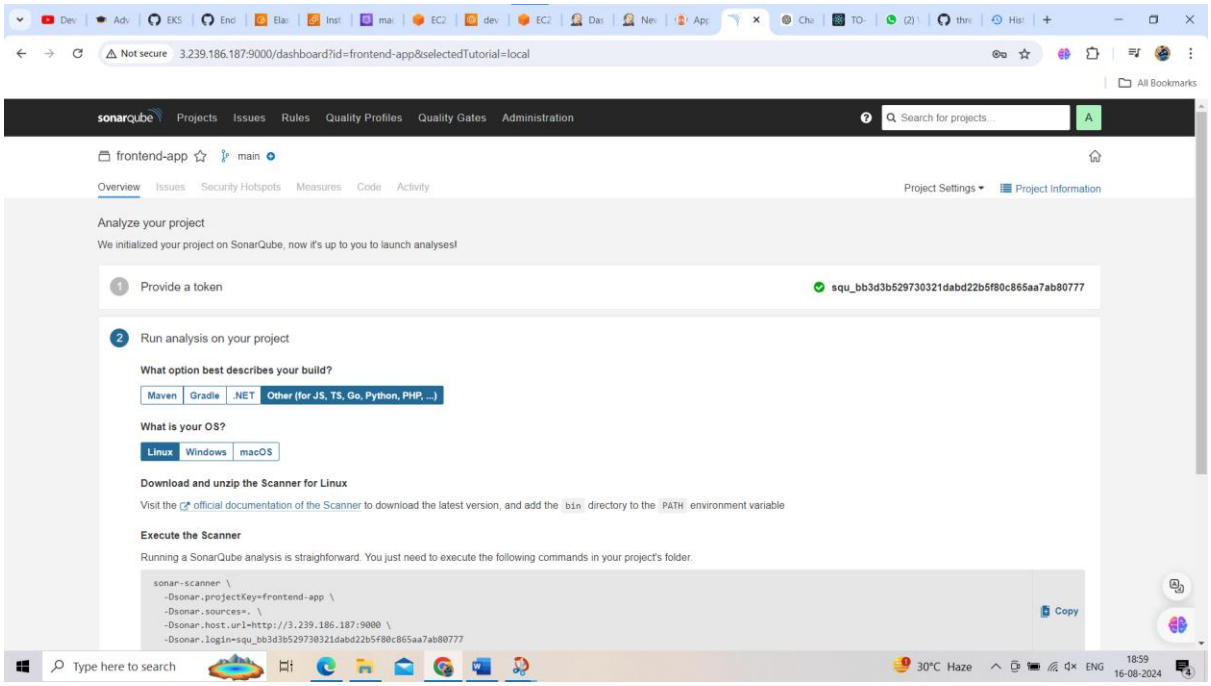


Step3: create the projects for frontend and backend



Click on setup and select manually and use the generated token

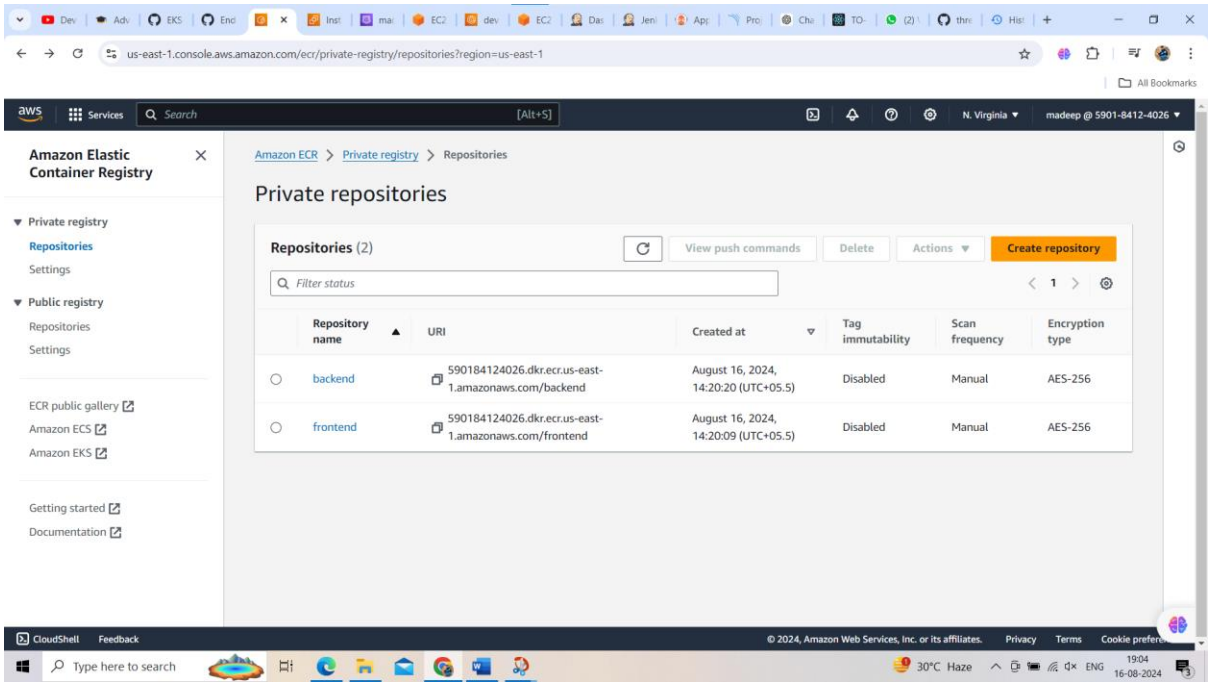




Step 5: Create the ECR repositories

Click on create repository → choose the private repo and enter the repository.

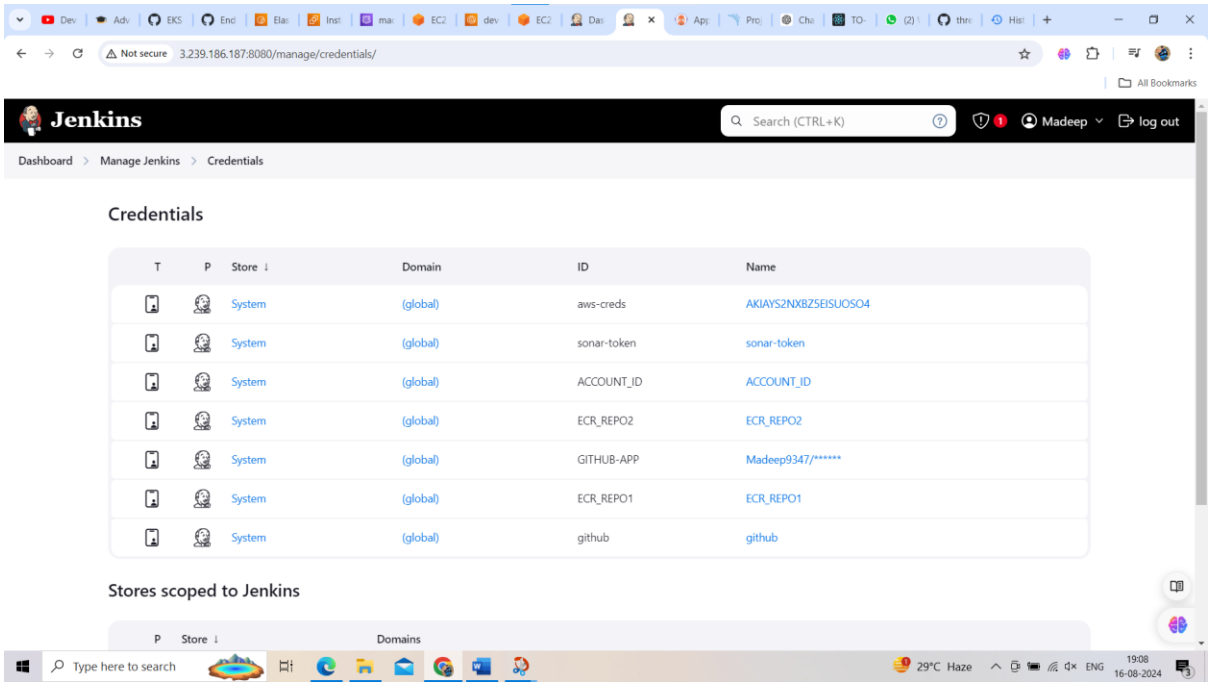
Here create the 2 repositories for frontend and backend



Step 6: Add the repositories, aws Account id in the jenkins credentials

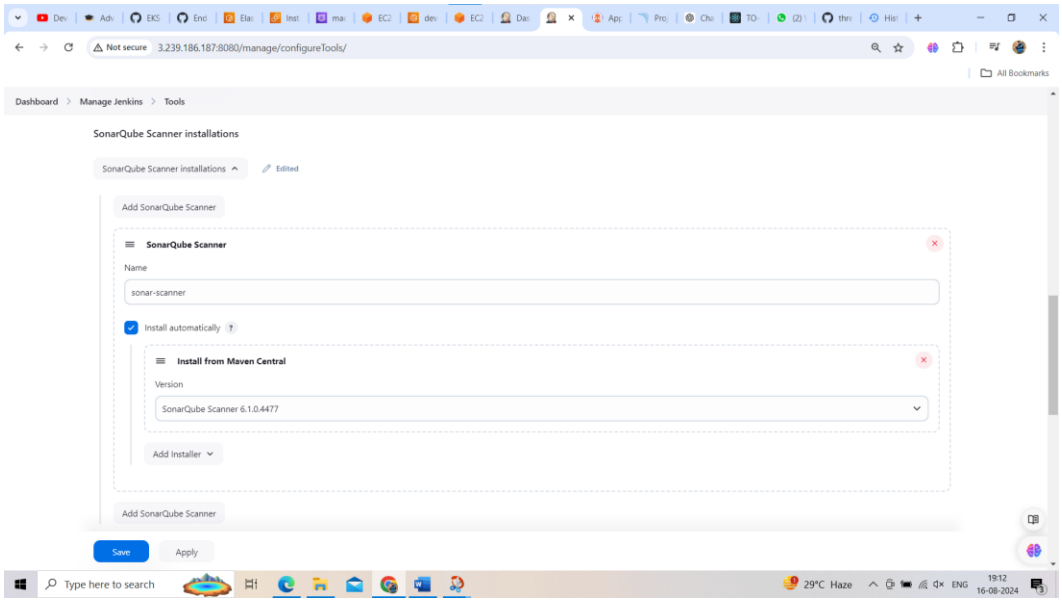
And also add the GitHub credentials and Personal Access Token (PAT) in the Jenkins Credentials.

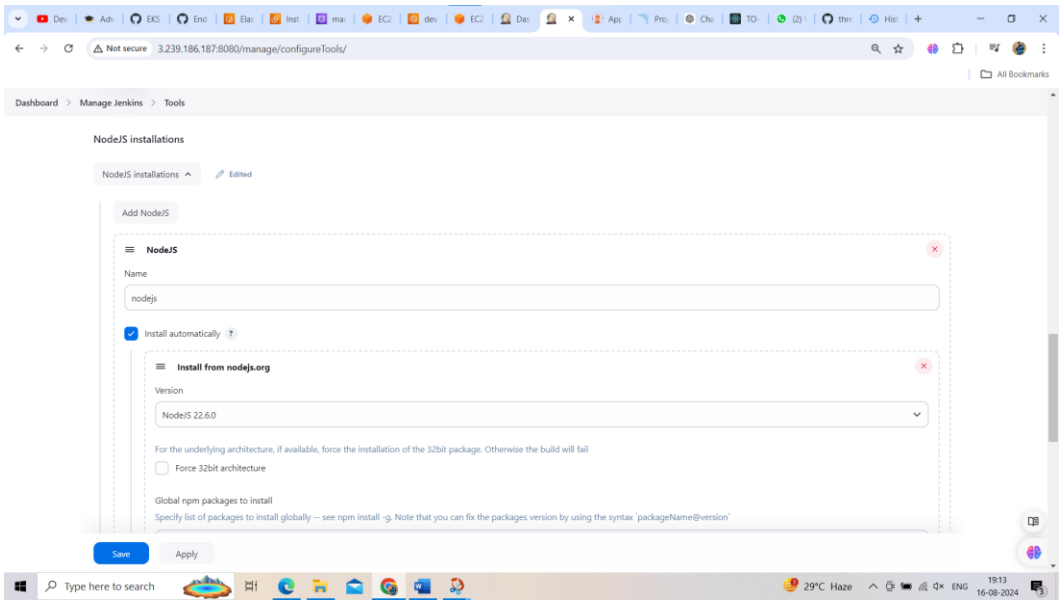
Final all the credentials looks like this

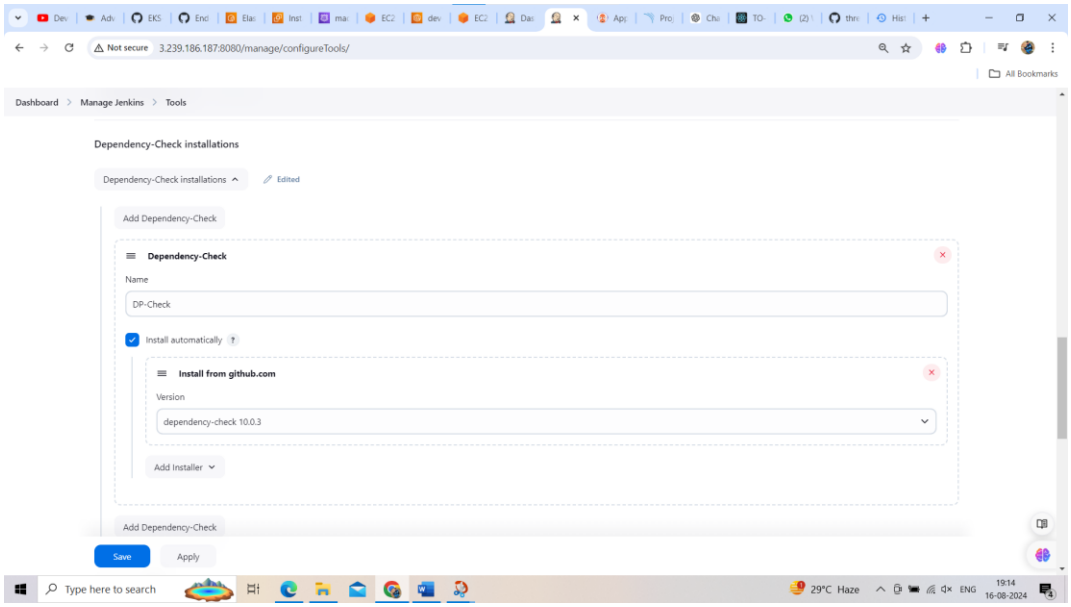


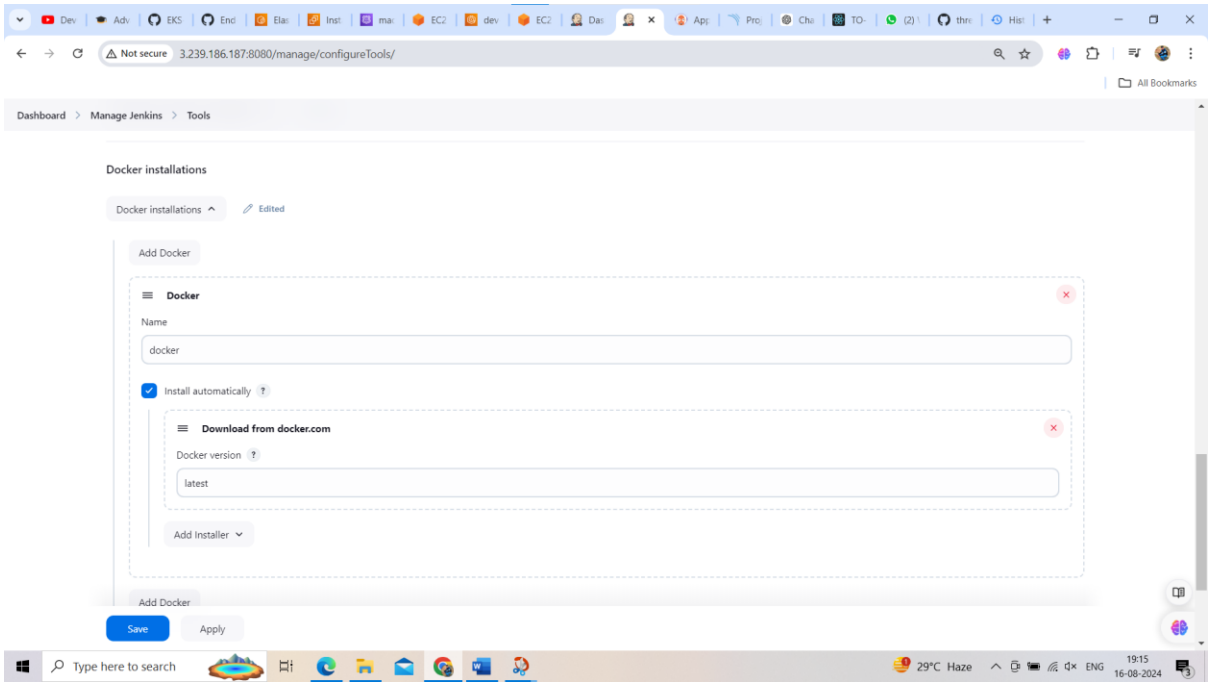
Step 7: Install plugins docker, sonar-scanner, OWASP, Nodejs

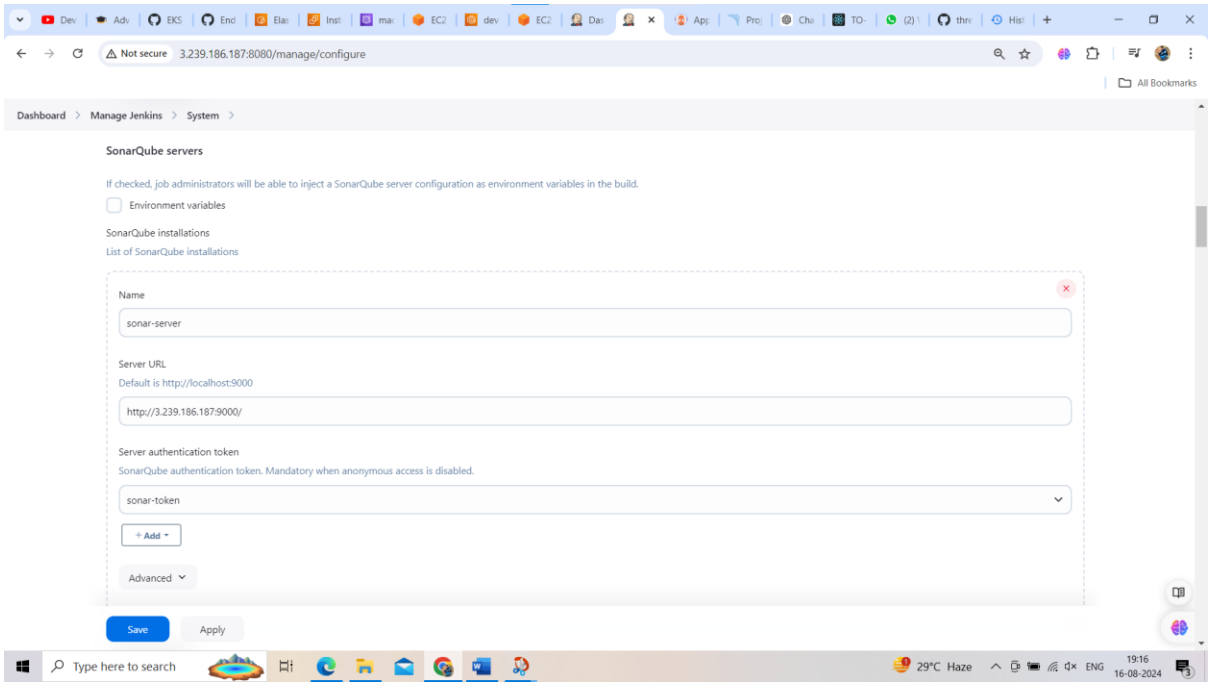
Step8: Setup the tools in Jenkins


Not secure 3.239.186.187:8080/manage/configureTools/
Dashboard > Manage Jenkins > Tools
SonarQube Scanner installations
SonarQube Scanner installations
Edited
Add SonarQube Scanner
SonarQube Scanner
Name
sonar-scanner
Install automatically
Install from Maven Central
Version
SonarQube Scanner 6.1.0.4477
Add Installer
Add SonarQube Scanner
Save
Apply
Type here to search
29°C Haze
ENG
19:12
16-08-2024


Not secure 3.239.186.187:8080/manage/configureTools/
Dashboard > Manage Jenkins > Tools
NodeJS installations
NodeJS installations
Edited
Add NodeJS
NodeJS
Name
nodejs
Install automatically
Install from nodejs.org
Version
NodeJS 22.6.0
For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail
Force 32bit architecture
Global npm packages to install
Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax `packageName@version`
Save
Apply
Type here to search
29°C Haze
ENG
19:13
16-08-2024

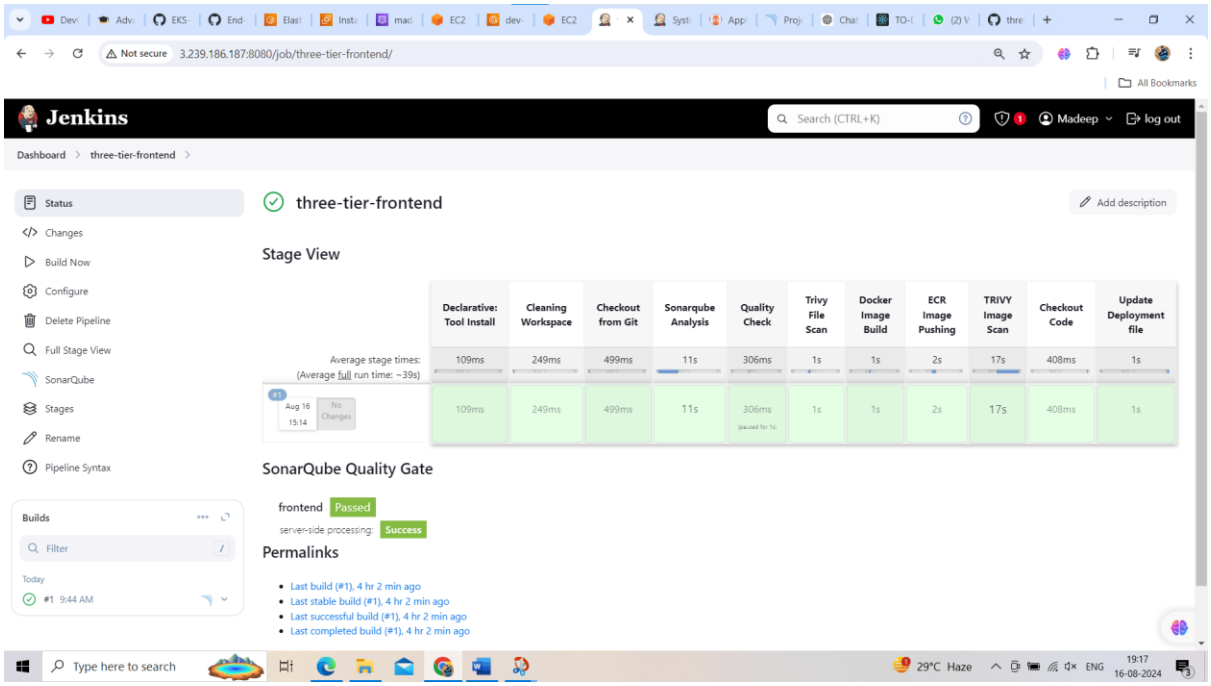

Not secure 3.239.186.187:8080/manage/configureTools/
Dashboard > Manage Jenkins > Tools
Dependency-Check installations
Dependency-Check installations
Edited
Add Dependency-Check
Dependency-Check
Name
DP-Check
Install automatically
Install from github.com
Version
dependency-check 10.0.3
Add Installer
Add Dependency-Check
Save
Apply
Type here to search
29°C Haze
ENG
19:14
16-08-2024



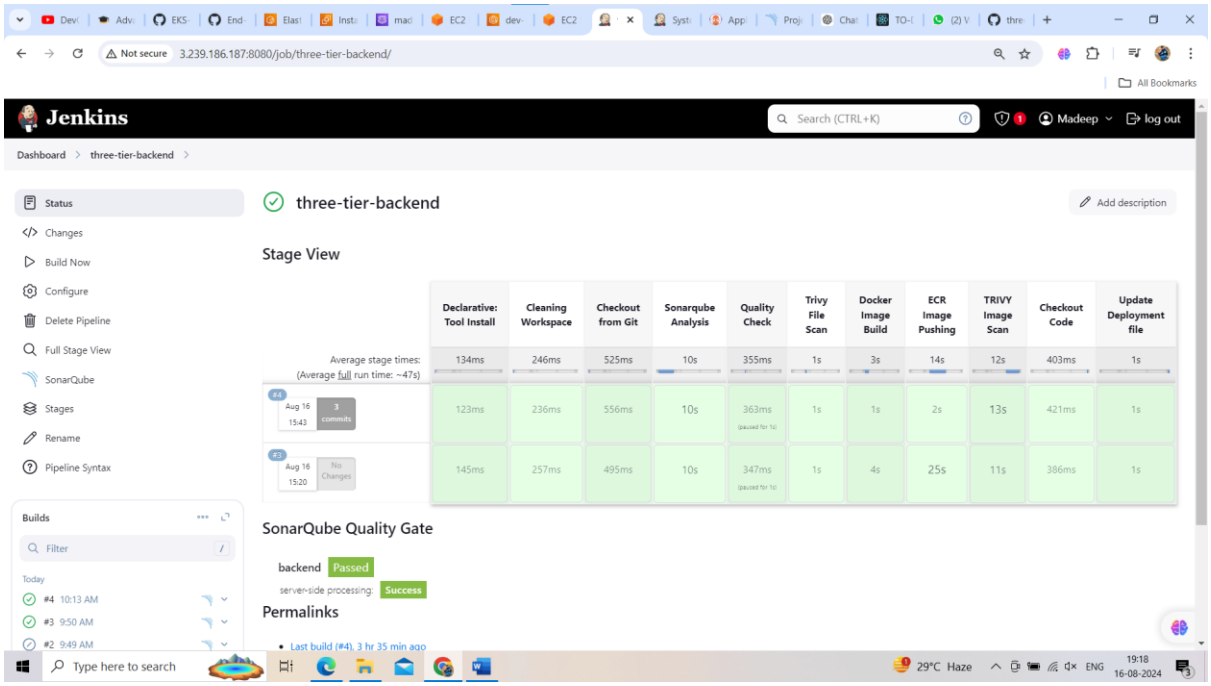


Step 9: create the Jenkins job for frontend application

Jenkinsfile: https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/blob/master/Jenkins-Pipeline-Code/Jenkinsfile-Frontend



Step 10 : Create the Jenkins job for Backend Application



Step 11: Install and Configure ArgoCD in the jump-server:

We will be deploying our application on a three-tier namespace. To do that, we will create a three-tier namespace on EKS.

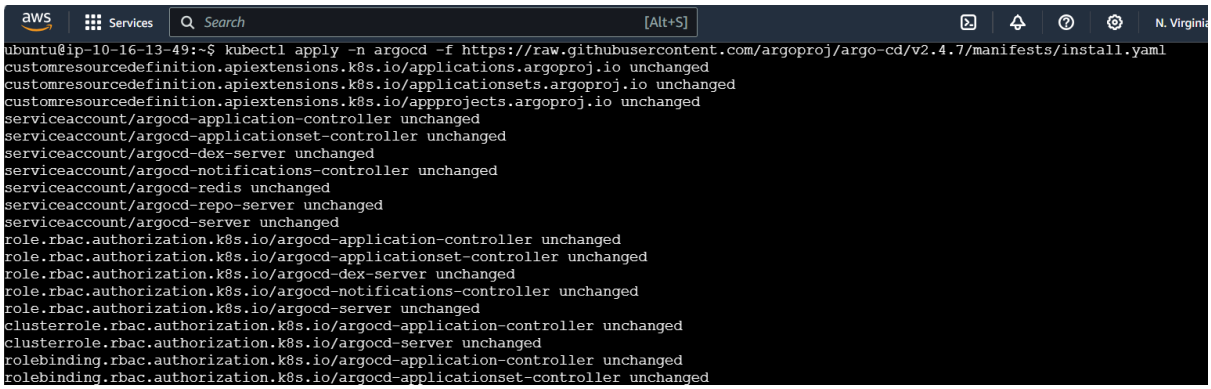
Create an separate namespace for argocd using the command:

kubectl create namespace three-tier

Now, we will install argoCD:

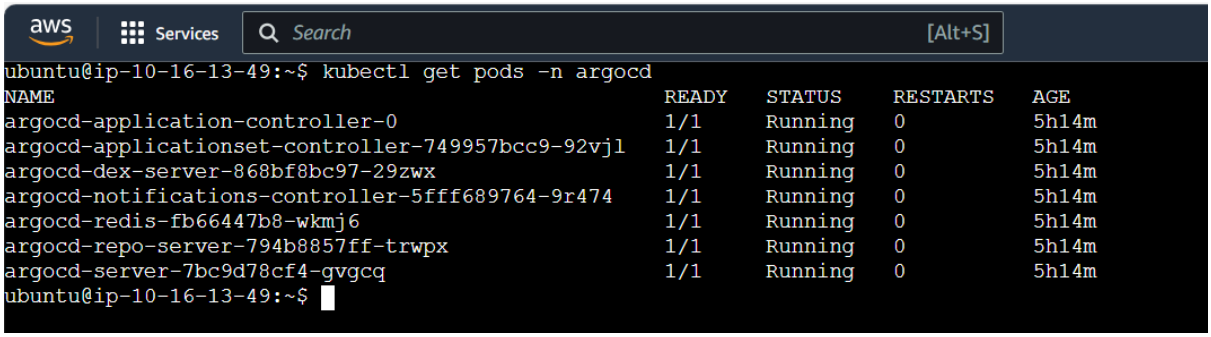
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml

these command will create all the required pods and services of the argocd server in the argocd namespace.



All pods must be running, to validate run the below command:

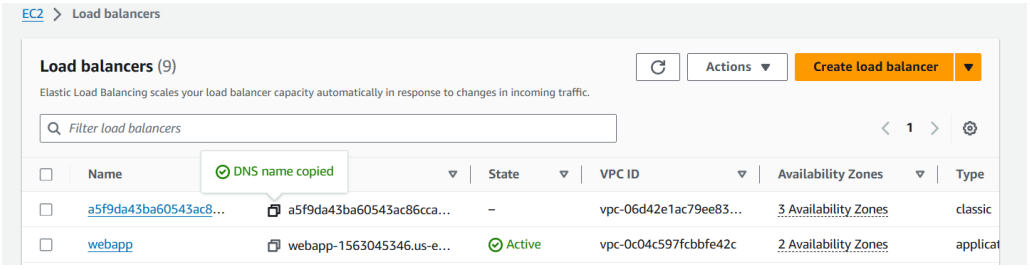
kubectl get pods -n argocd



Now, expose the argoCD server as LoadBalancer using the below command:

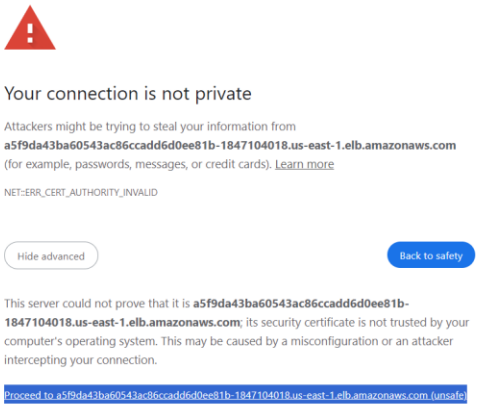
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

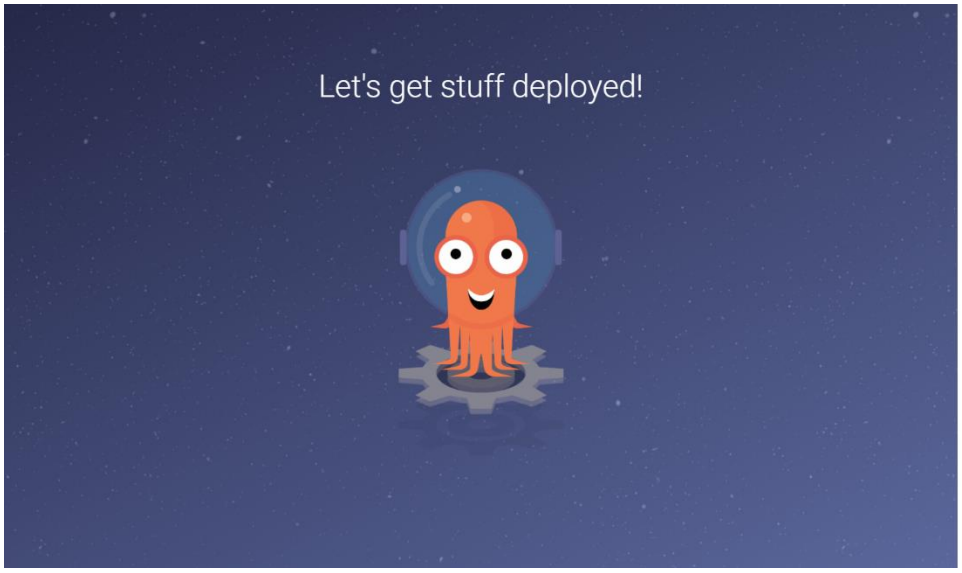
You can validate whether the Load Balancer is created or not by going to the AWS Console:



To access the argoCD, copy the LoadBalancer DNS and hit on your favorite browser.

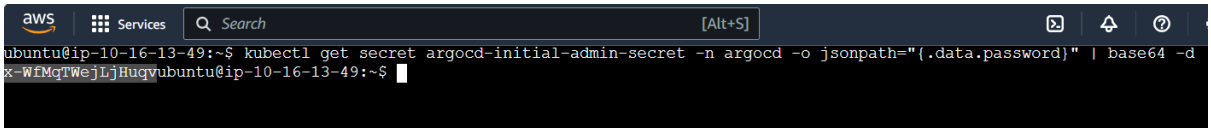
You will get a warning like Your connection is not private then Click on Advanced.





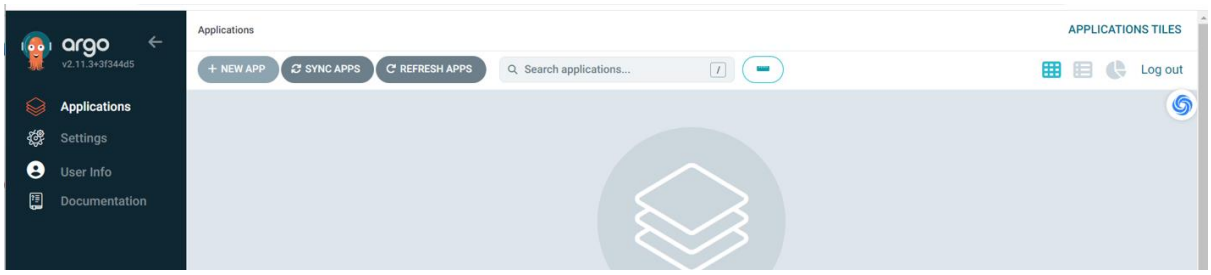
Now, we need to get the password for our argoCD server to perform the deployment.

kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 –d



Enter the username (admin) and password in argoCD and click on SIGN IN.

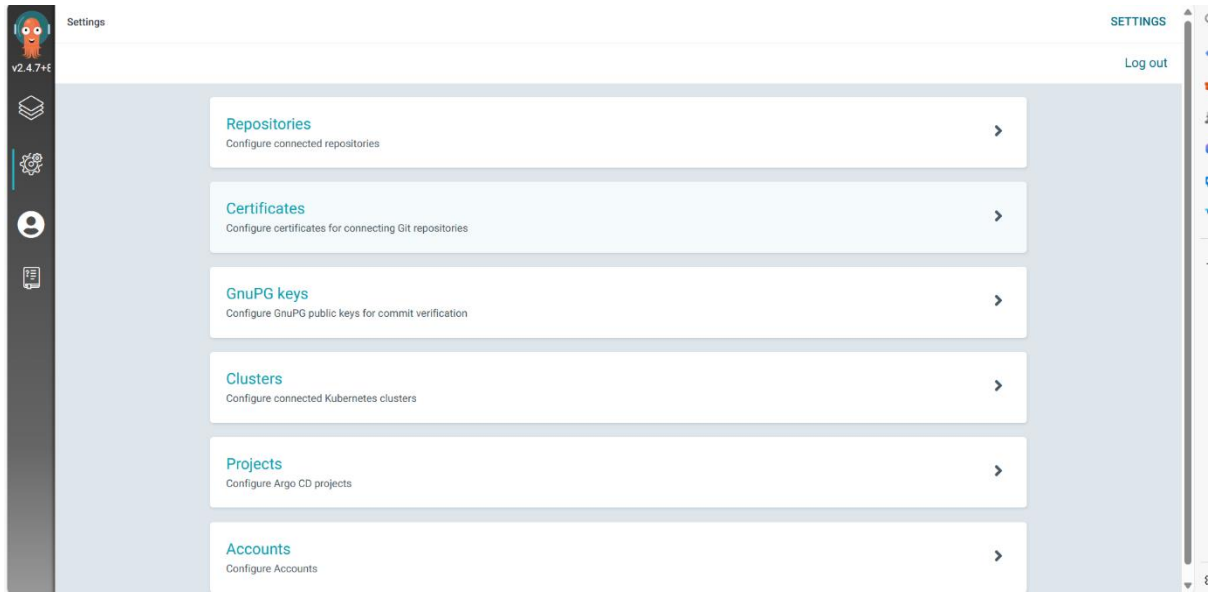
Here is our ArgoCD Dashboard.



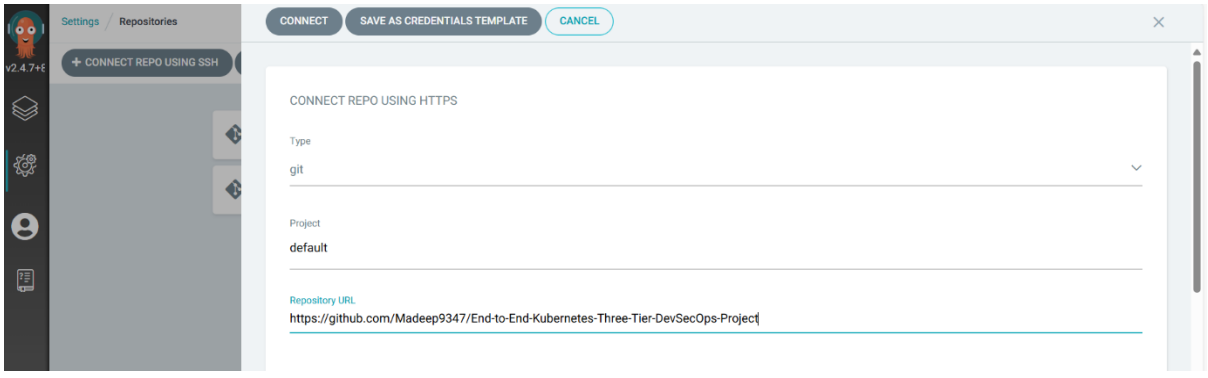
Step12:Now we need to add the Github Repository to the argocd by going to the setting.

Connect the github repository using HTTPS

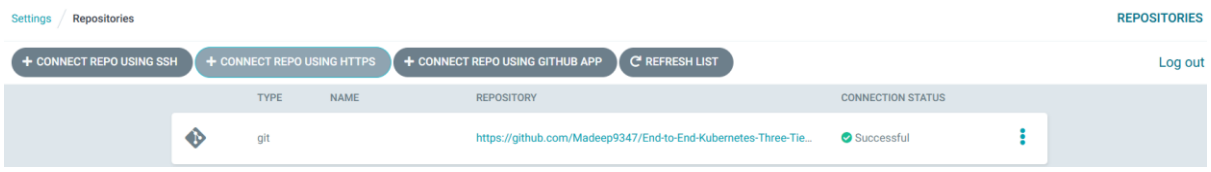
Now we need to add the Github Repository to the argocd by going to the setting.



Go to the settings add the repo using the connect repo using the HTTPS and add the github repository URL.

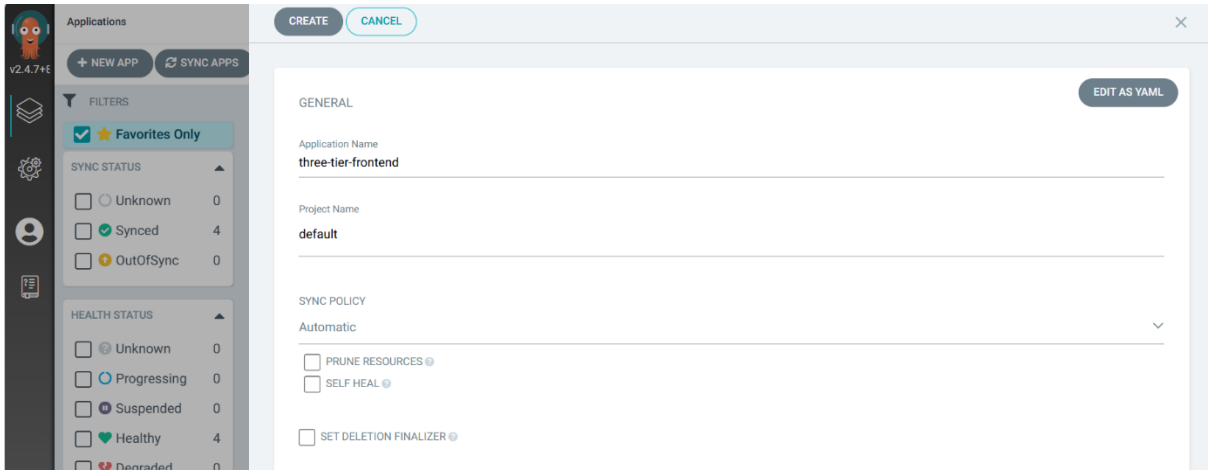


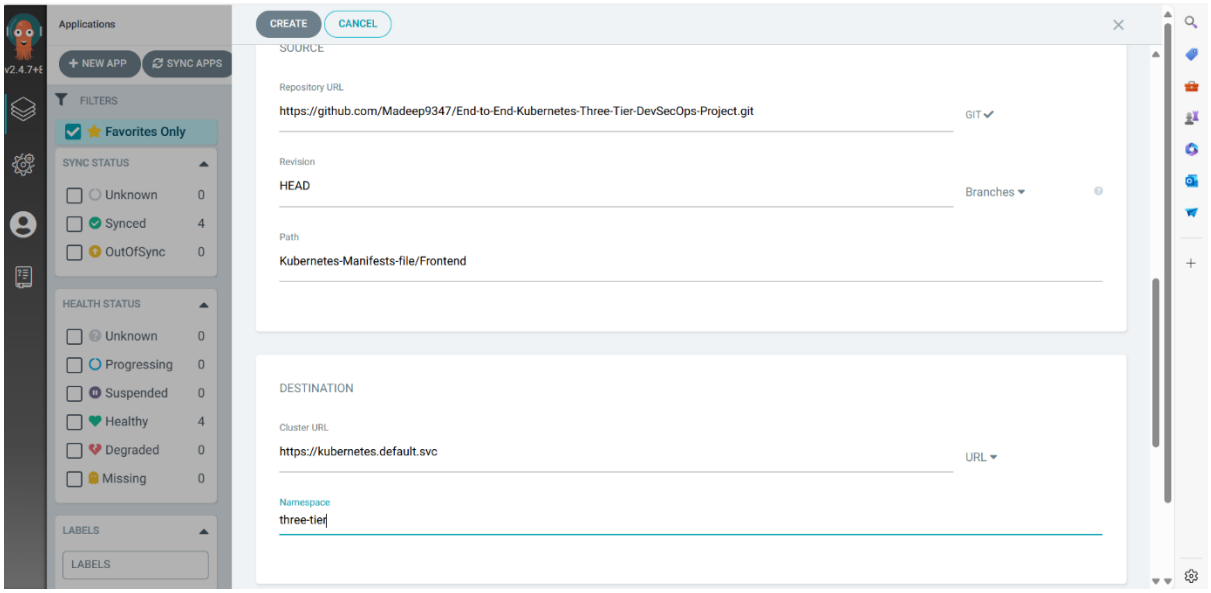
And Click on connect then the repo will be connected to the argocd:



Step 13:Now create the separate apps for Frontend, Backend, Database and the ingress manifest files:

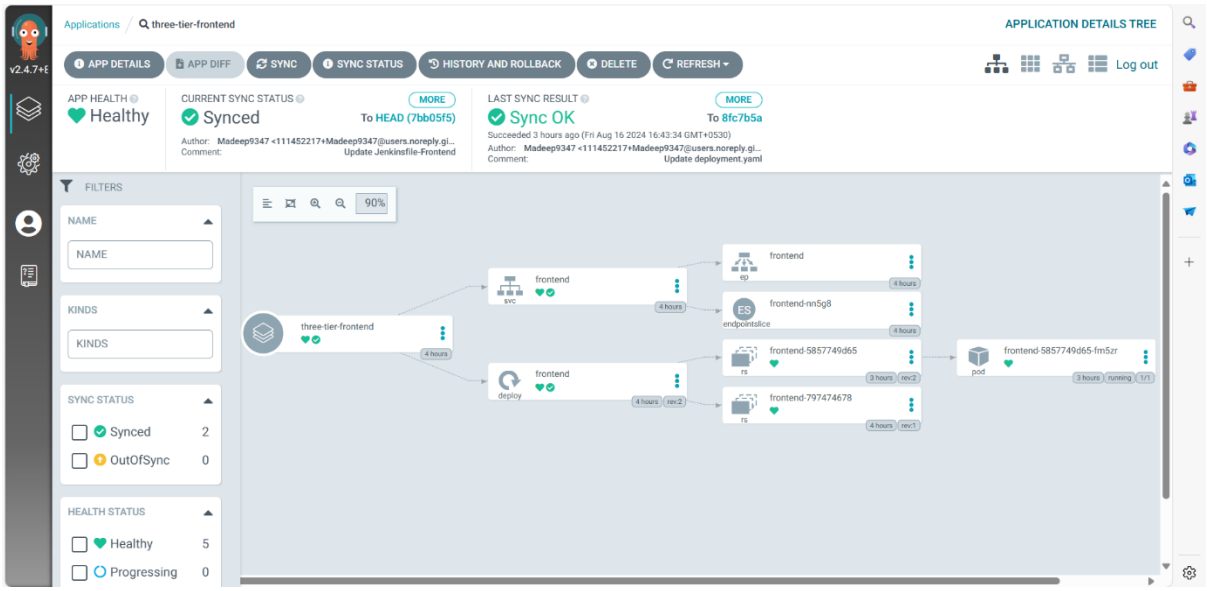
Frontend app creation:



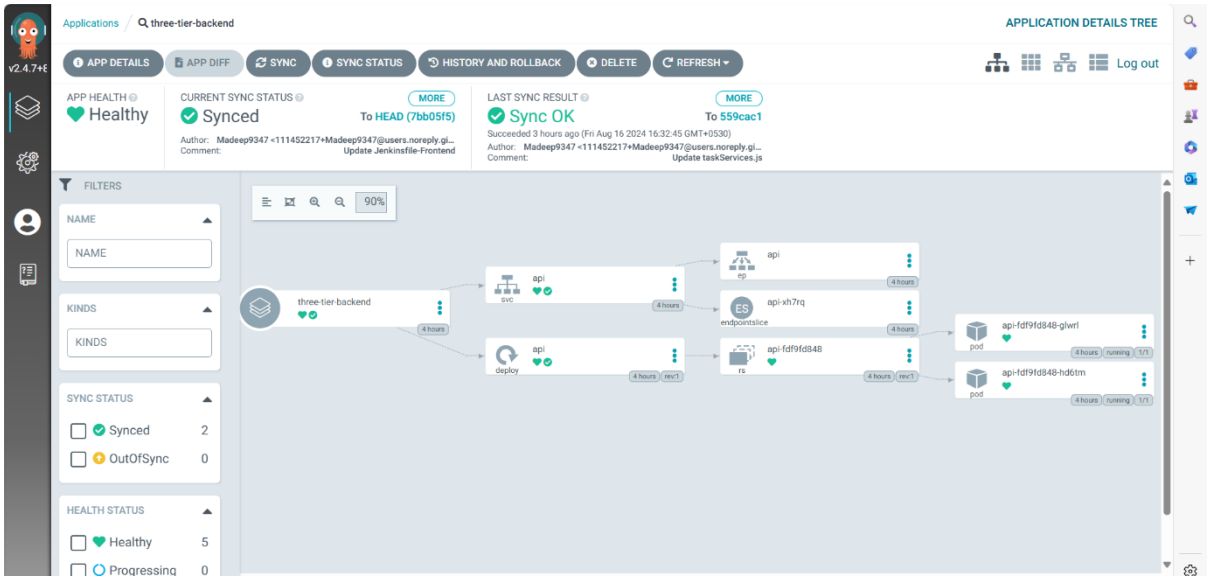


Create these apps same for the Backend, Database and Ingress.

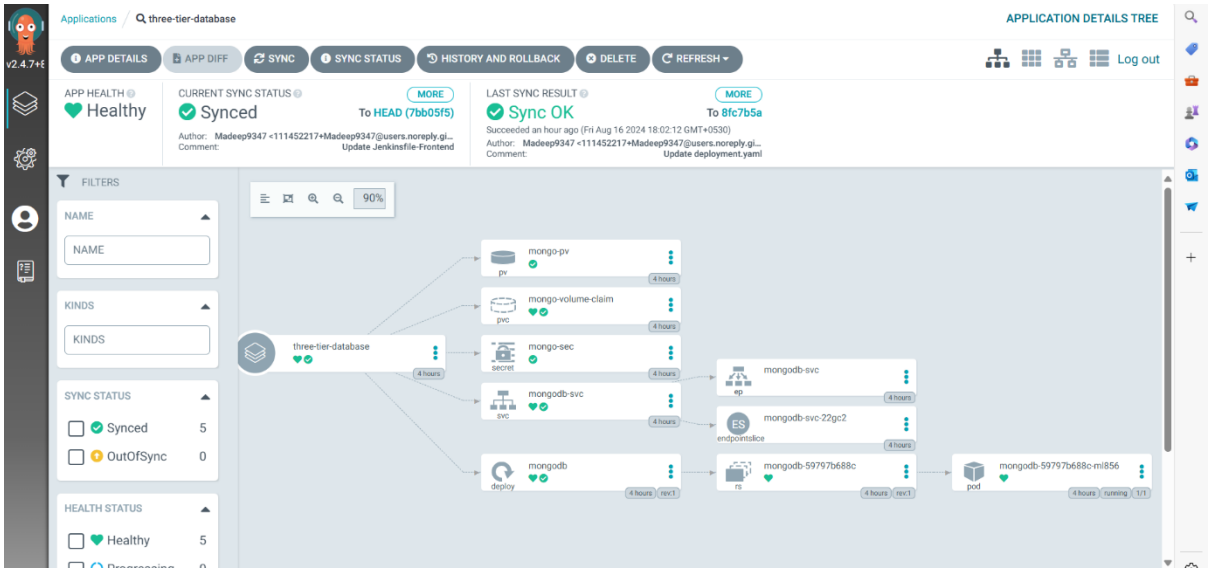
This is the Frontend Application Deployment in ArgoCD:



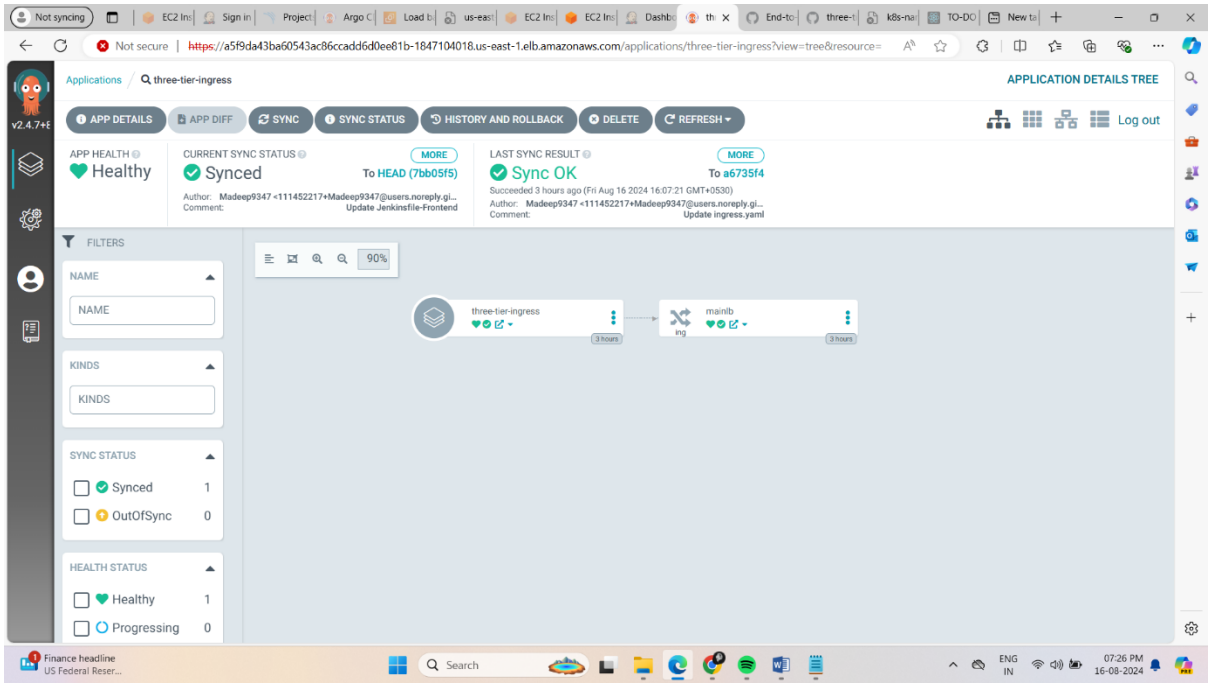
This is the Backend Application Deployment in ArgoCD:



This is the Database Application Deployment in ArgoCD:



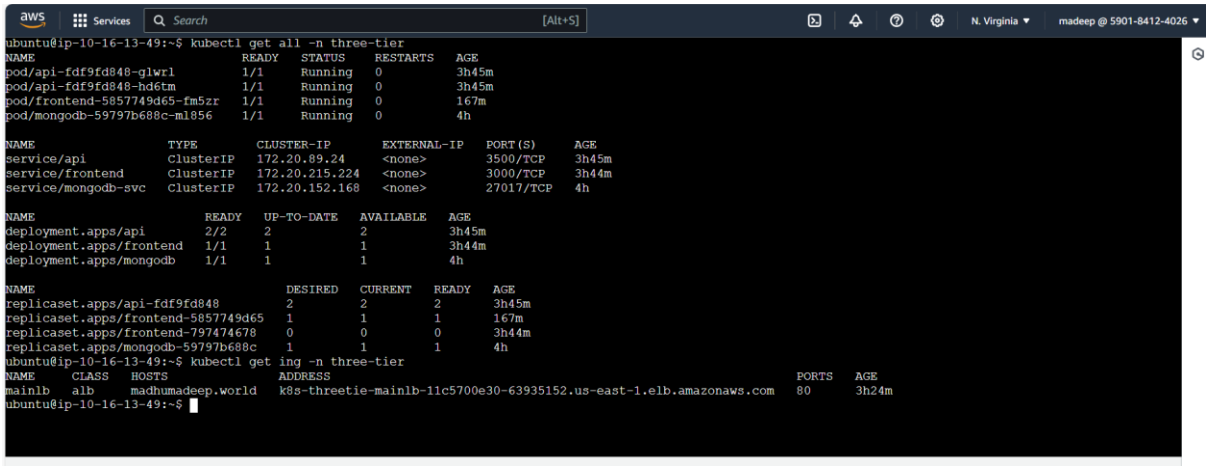
This is the Ingress Application Deployment in ArgoCD:



If you observe, we have configured the Persistent Volume & Persistent Volume Claim. So, if the pods get deleted then, the data won't be lost. The Data will be stored on the host machine.

Step 14:To ensure all the pods and the service are running are not check in the eks cluster using command:

Kubectl get all –n three-tier



Step 15: Make sure to modify the deployment.yml files in frontend,backend.

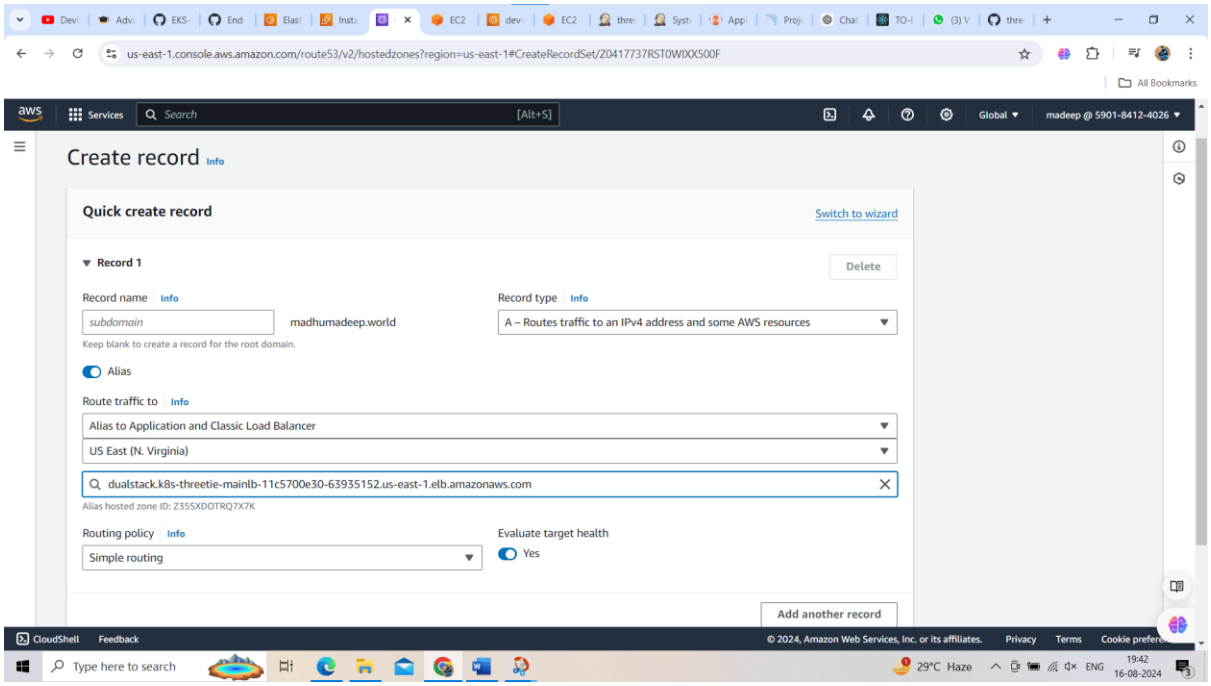
```
spec:
  imagePullSecrets:
  - name: ecr-registry-secret
  containers:
  - name: api
    image: 590184124026.dkr.ecr.us-east-1.amazonaws.com/backend:4
    imagePullPolicy: Always
    env:
      - name: MONGO_CONN_STR
        value: mongodb://mongodb-svc:27017/todo?directConnection=true
      - name: MONGO_USERNAME
```

```
    containers:
    - name: frontend
      image: 590184124026.dkr.ecr.us-east-1.amazonaws.com/frontend:1
      imagePullPolicy: Always
      env:
        - name: REACT_APP_BACKEND_URL
          value: "http://madhumadeep.world/api/tasks"
      ports:
      - containerPort: 3000
```

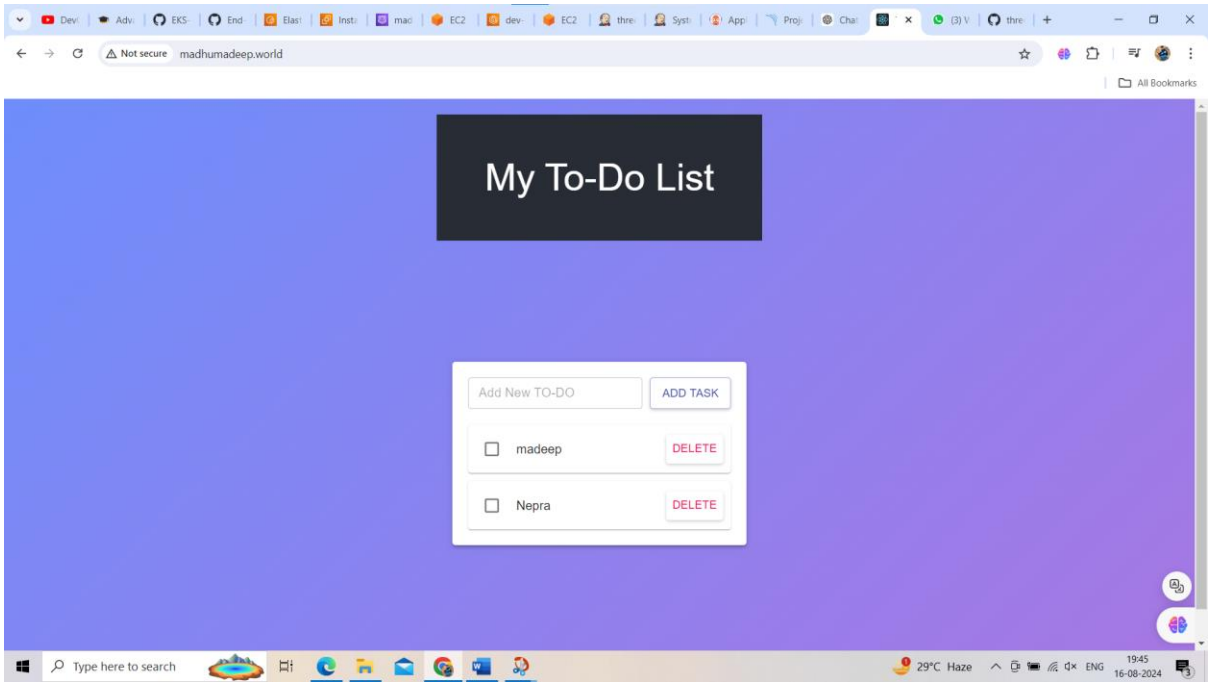
And also edit the domain name in evn

```
    env:
      - name: REACT_APP_BACKEND_URL
        value: "http://madhumadeep.world/api/tasks"
```

Step 16: Create the DNS record for ingress-controller load balancer



Step 17: Wait for 2-3 minutes and Access the Application using the Domain name



PHASE 3 : Monitoring the EKS cluster

We will achieve the monitoring using Helm

Step1 :Add the prometheus repo by using the below command

helm repo add stable https://charts.helm.sh/stable

Step 2: Install the Prometheus

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus prometheus-community/prometheus

```
ubuntu@ip-10-16-13-49:~$ kubectl get all
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/grafana-56475dbb85-fgjdp                         1/1     Running   0          60s
pod/prometheus-alertmanager-0                        1/1     Running   0          65s
pod/prometheus-kube-state-metrics-79c867c577-thxvx   1/1     Running   0          65s
pod/prometheus-prometheus-node-exporter-2sjfr        1/1     Running   0          65s
pod/prometheus-prometheus-node-exporter-5blv5        1/1     Running   0          65s
pod/prometheus-prometheus-pushgateway-57c548bd6f-r8pqd   1/1     Running   0          65s
pod/prometheus-server-65b79f9866-4xp5h               2/2     Running   0          65s

NAME                                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/grafana                               ClusterIP   172.20.131.173   <none>        80/TCP     61s
service/kubernetes                            ClusterIP   172.20.0.1       <none>        443/TCP    27h
service/prometheus-alertmanager               ClusterIP   172.20.246.8     <none>        9093/TCP   65s
service/prometheus-alertmanager-headless      ClusterIP   None             <none>        9093/TCP   65s
service/prometheus-kube-state-metrics         ClusterIP   172.20.24.217    <none>        8080/TCP   65s
service/prometheus-prometheus-node-exporter   ClusterIP   172.20.35.64     <none>        9100/TCP   65s
service/prometheus-prometheus-pushgateway     ClusterIP   172.20.179.123   <none>        9091/TCP   65s
service/prometheus-server                     ClusterIP   172.20.130.81    <none>        80/TCP     65s

NAME                                                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/prometheus-prometheus-node-exporter   2         2         2       2            2           kubernetes.io/os=linux   65s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana                       1/1     1            1           61s
deployment.apps/prometheus-kube-state-metrics 1/1     1            1           65s
```

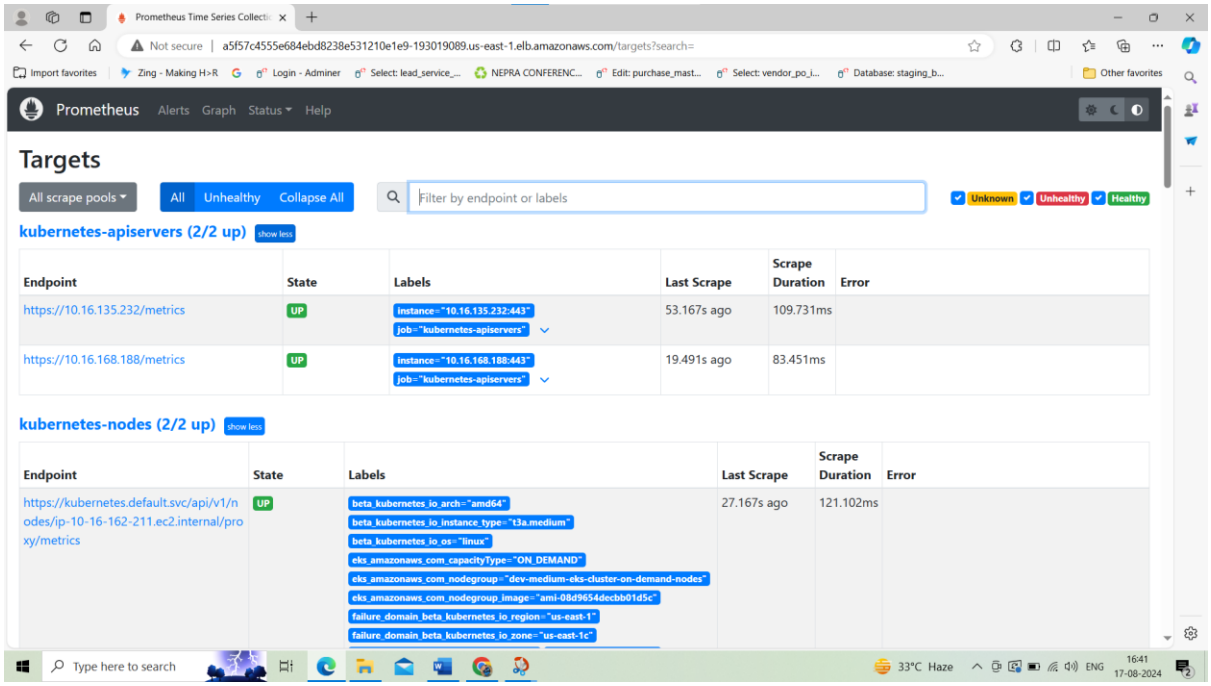
Step 3: Now, we need to access our Prometheus and Grafana consoles from outside of the cluster.

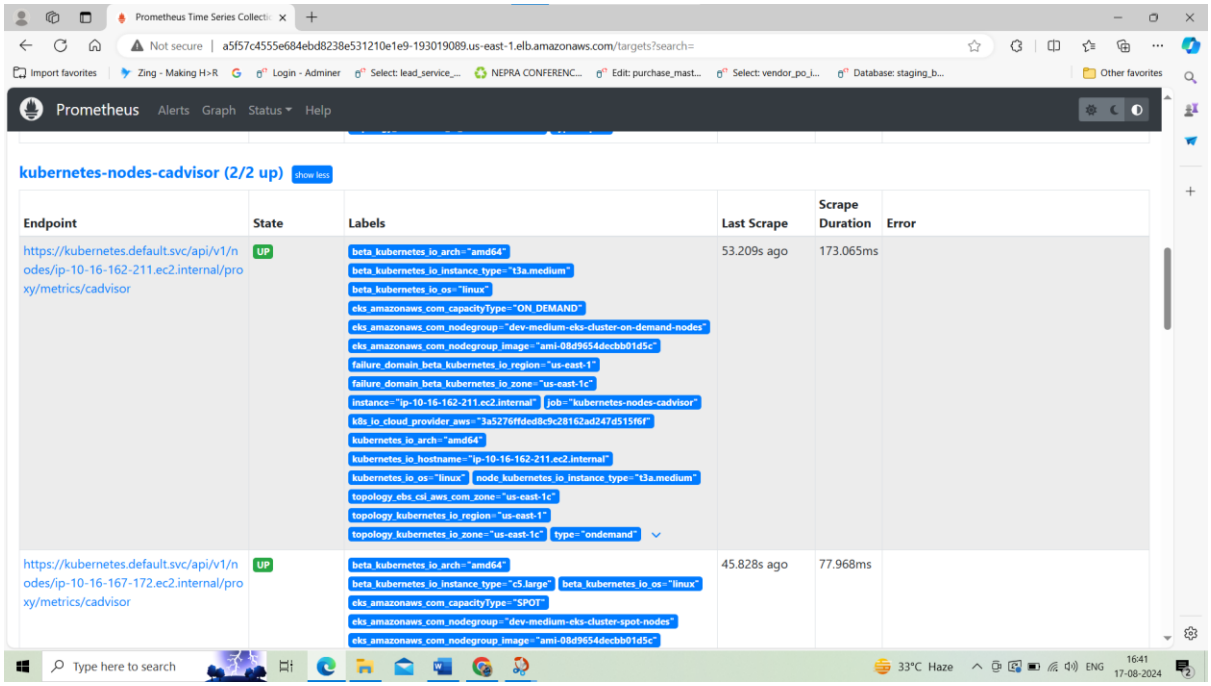
For that, we need to change the Service type from ClusterType to **LoadBalancer**

Edit the **stable-kube-prometheus-sta-prometheus** service

kubectl edit svc Prometheus-server

```
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana                               ClusterIP   172.20.131.173   <none>        80/TCP     14m
kubernetes                            ClusterIP   172.20.0.1       <none>        443/TCP    27h
prometheus-alertmanager               ClusterIP   172.20.246.8     <none>        9093/TCP   14m
prometheus-alertmanager-headless      ClusterIP   None             <none>        9093/TCP   14m
prometheus-kube-state-metrics         ClusterIP   172.20.24.217    <none>        8080/TCP   14m
prometheus-prometheus-node-exporter   ClusterIP   172.20.35.64     <none>        9100/TCP   14m
prometheus-prometheus-pushgateway     ClusterIP   172.20.179.123   <none>        9091/TCP   14m
prometheus-server                     ClusterIP   172.20.130.81    <none>        80/TCP     14m
ubuntu@ip-10-16-13-49:~$ kubectl edit svc prometheus-server
service/prometheus-server edited
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)          AGE
grafana                               ClusterIP      172.20.131.173   <none>                                                                   80/TCP           16m
kubernetes                            ClusterIP      172.20.0.1       <none>                                                                   443/TCP          27h
prometheus-alertmanager               ClusterIP      172.20.246.8     <none>                                                                   9093/TCP         16m
prometheus-alertmanager-headless      ClusterIP      None             <none>                                                                   9093/TCP         16m
prometheus-kube-state-metrics         ClusterIP      172.20.24.217    <none>                                                                   8080/TCP         16m
prometheus-prometheus-node-exporter   ClusterIP      172.20.35.64     <none>                                                                   9100/TCP         16m
prometheus-prometheus-pushgateway     ClusterIP      172.20.179.123   <none>                                                                   9091/TCP         16m
prometheus-server                     LoadBalancer   172.20.130.81    a5f57c4555e684ebd8238e531210e1e9-193019089.us-east-1.elb.amazonaws.com   80:31696/TCP     16m
ubuntu@ip-10-16-13-49:~$
```





Step 4: Install Grafana

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

Step 5: Edit the stable-grafana service

Kubectl edit svc grafana

```
  selector:
    app.kubernetes.io/instance: grafana
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
```

```
ubuntu@ip-10-16-13-49:~$ kubectl edit svc grafana
service/grafana edited
ubuntu@ip-10-16-13-49:~$ kubectl get svc grafana
NAME      TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)        AGE
grafana   LoadBalancer   172.20.131.173   aea4138f4fd5b42798b4f2c8634e0c6a-1845028998.us-east-1.elb.amazonaws.com   80:30170/TCP   31m
ubuntu@ip-10-16-13-49:~$
```

Step 6: Get the password for grafana from secrets

```
ubuntu@ip-10-16-13-49:~$ kubectl get secrets
NAME                                  TYPE                 DATA   AGE
grafana                               Opaque               3      36m
sh.helm.release.v1.grafana.v1         helm.sh/release.v1   1      36m
sh.helm.release.v1.prometheus.v1      helm.sh/release.v1   1      36m
```

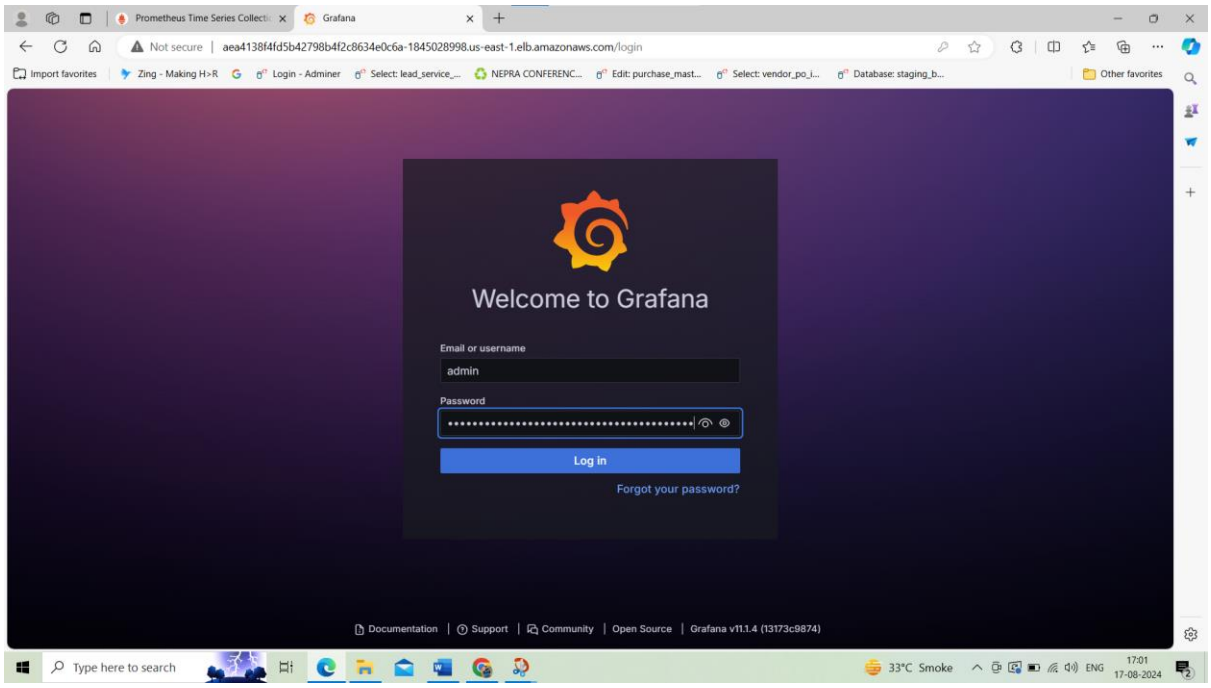
Kubectl edit secret grafana

```
apiVersion: v1
data:
  admin-password: MERPWXdGUUhNRmVXdXdkYlA3TUM4dkZjR3FZNm82V1N1cFBmY21qZA==
  admin-user: YWRtaW4=
```

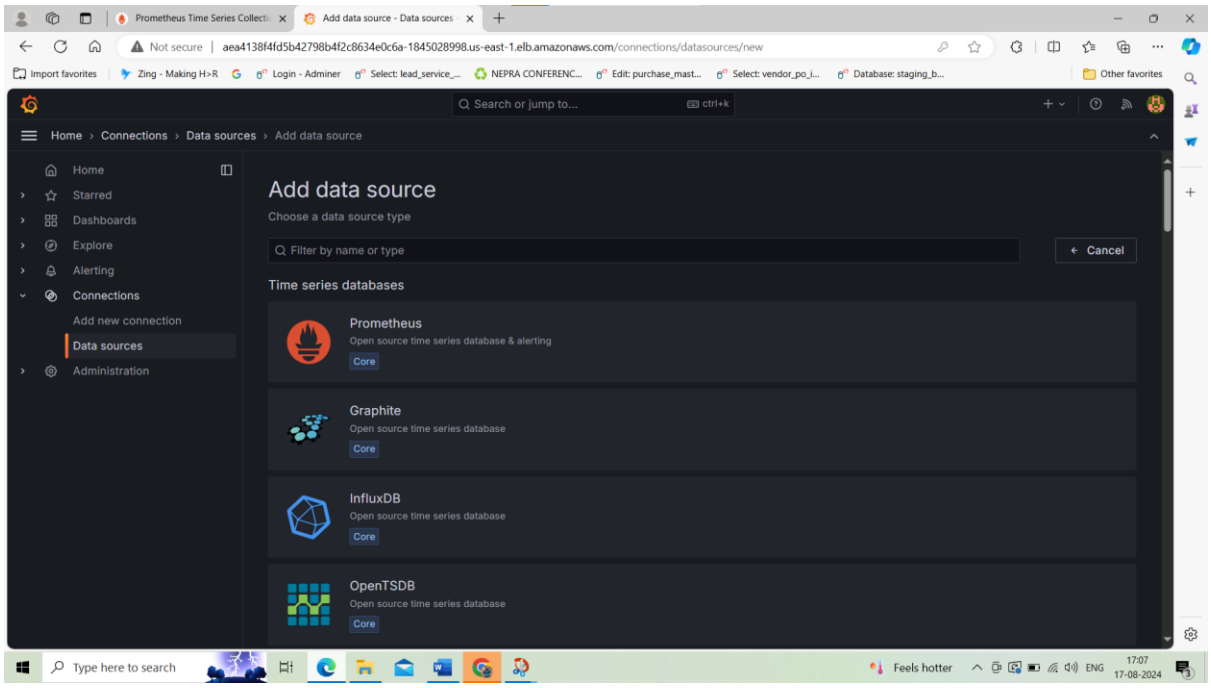
Copy the password and decode with base64

```
ubuntu@ip-10-16-13-49:~$ echo MERPWXdGUUhNRmVXdXdkYlA3TUM4dkZjR3FZNm82V1N1cFBmY21qZA== | base64 --decode
0DOYwFQHMFeWuwdbP7MC8vFcGqY6o6WSupPfcmjdubuntu@ip-10-16-13-49:~$
```

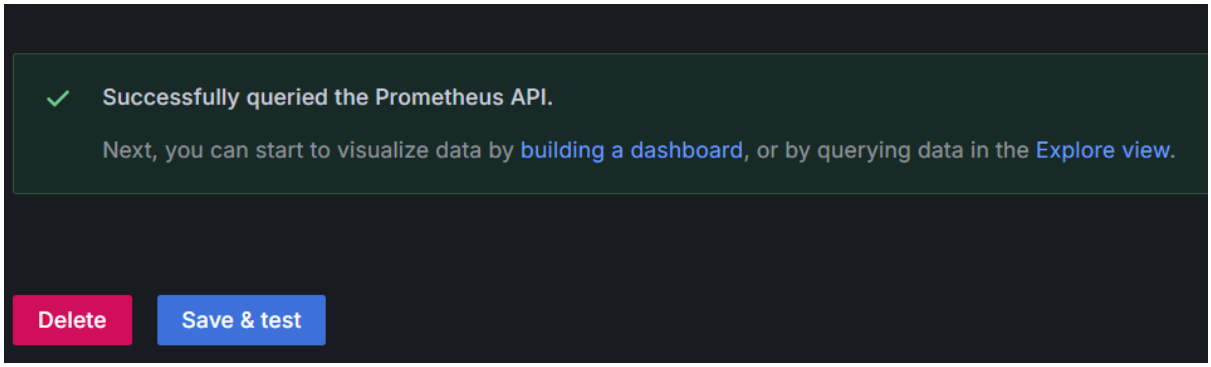
Step7: Now access the grafana using loadbalancer



Step 8: select the data source as prometheus

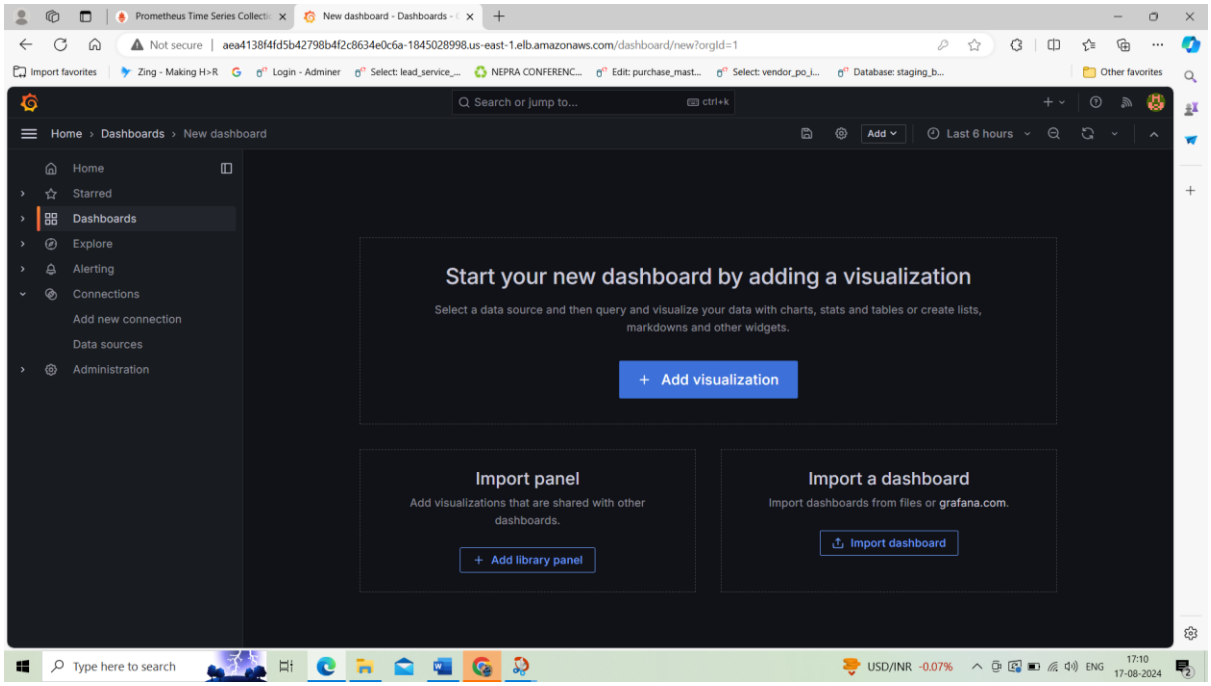


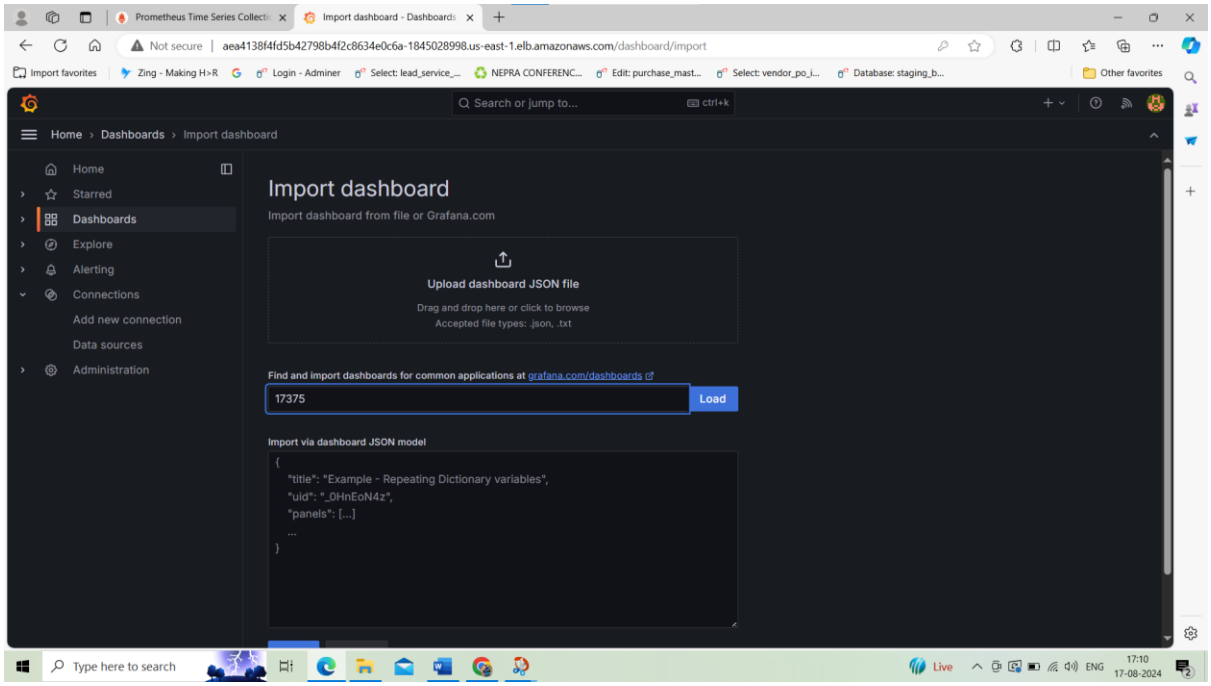
Enter your loadbalancer domain name and test

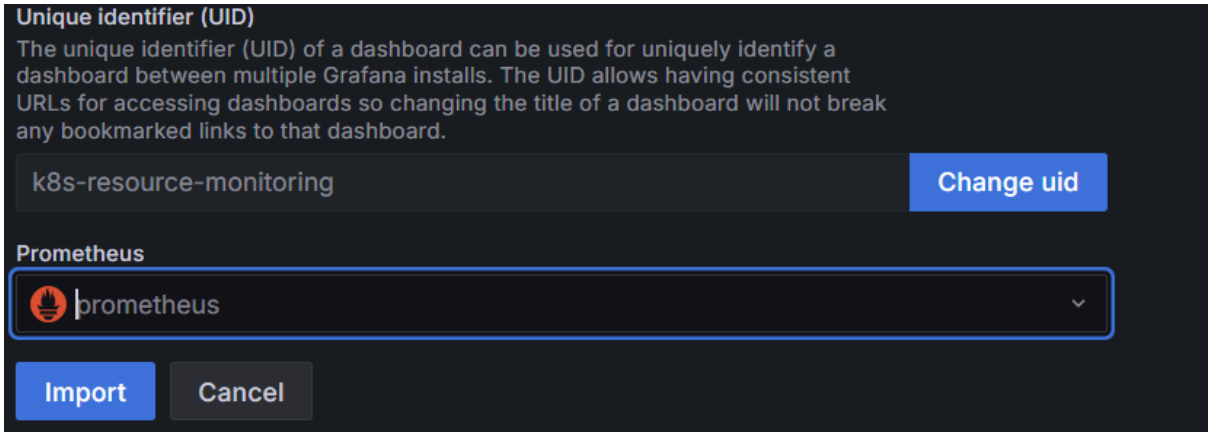


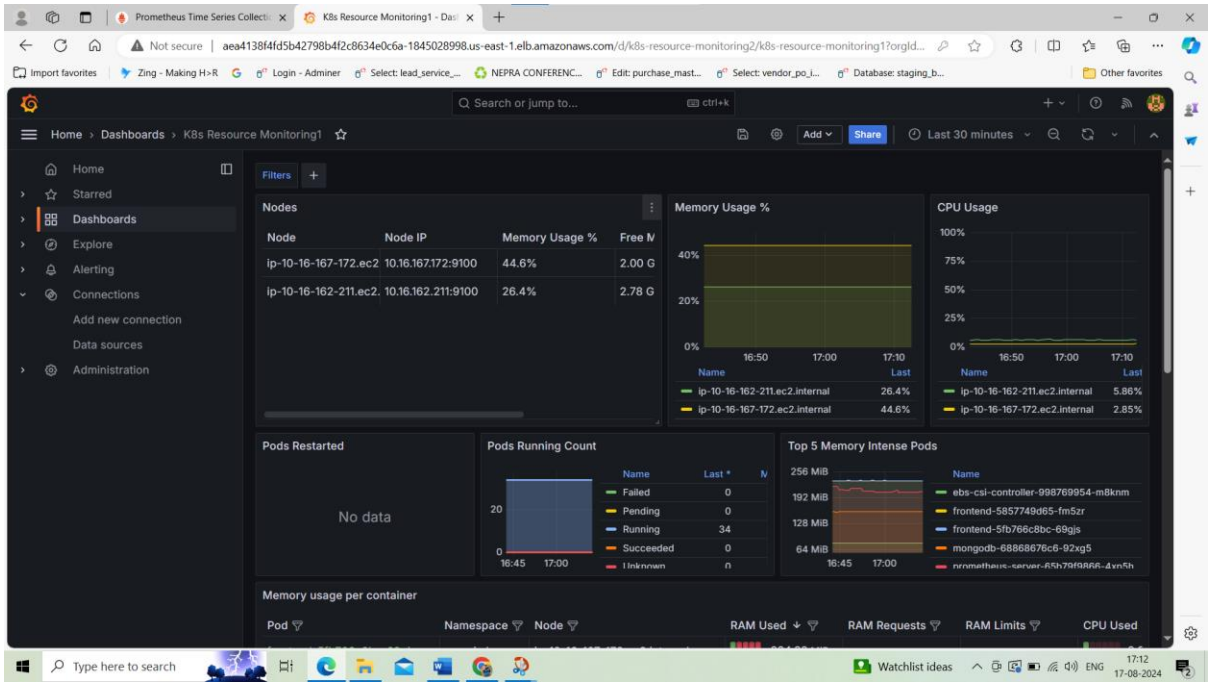
Step 9: Create the Dashboard

Import the dashboard




Import dashboard
Import dashboard from file or Grafana.com
Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt
Find and import dashboards for common applications at grafana.com/dashboards
17375
Load
Import via dashboard JSON model
{
"title": "Example - Repeating Dictionary variables",
"uid": "_0HnEoN4z",
"panels": [...]
...
}


Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.
k8s-resource-monitoring
Change uid
Prometheus
prometheus
Import
Cancel


Home › Dashboards › K8s Resource Monitoring1
Nodes
Node
Node IP
Memory Usage %
ip-10-16-167-172.ec2
10.16.167.172:9100
44.6%
ip-10-16-162-211.ec2.
10.16.162.211:9100
26.4%
Memory Usage %
CPU Usage
Pods Restarted
No data
Pods Running Count
Failed 0
Pending 0
Running 34
Succeeded 0
Top 5 Memory Intense Pods
ebs-csi-controller-998769954-m8knm
frontend-5857749d65-fm5zr
frontend-5fb766c8bc-69gjs
mongodb-68868676c6-92xg5
Memory usage per container
Pod
Namespace
Node
RAM Used
RAM Requests
RAM Limits
CPU Used