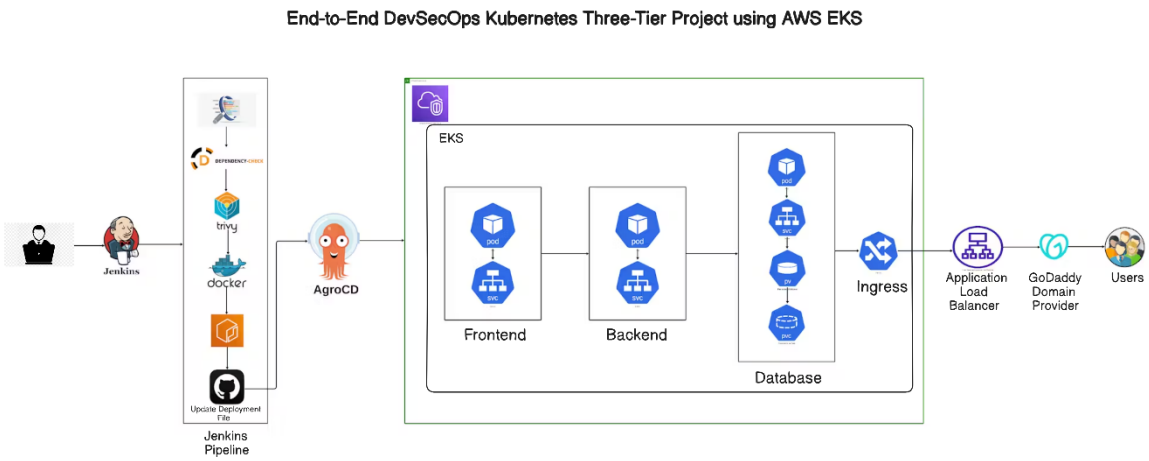
End-to-End DevSecOps Pipeline with Jenkins, ArgoCD, and EKS for Three-Tier Application Deployment



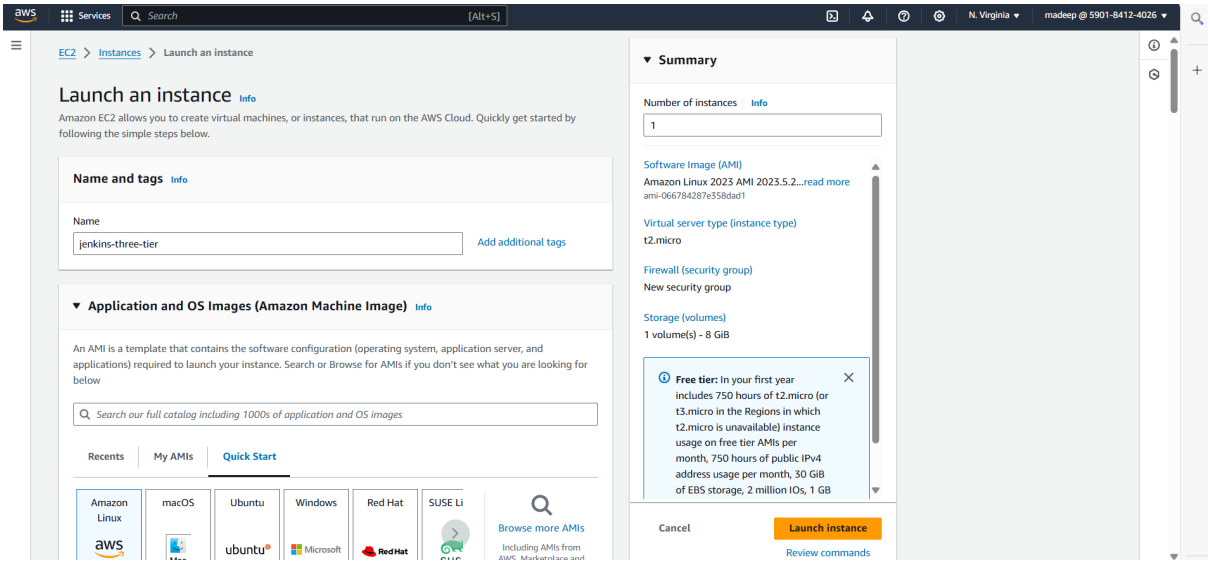
Project Overview:

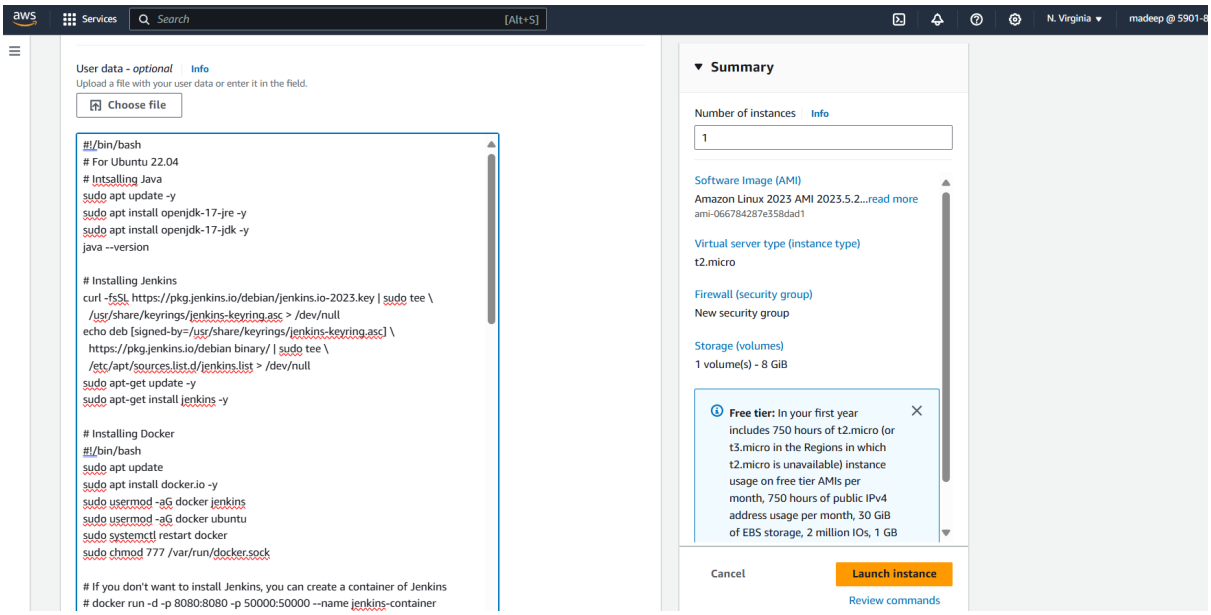
In this project, we will cover the following steps:

1. **Jenkins Server Setup and Configuration**
2. **VPC Creation**
3. **EKS Cluster and Jump Server Setup**
4. **Load Balancer Configuration for EKS**
5. **Amazon ECR Repositories**
6. **ArgoCD Installation and Configuration**
7. **SonarQube Setup for DevSecOps**
8. **Jenkins Pipeline Setup**
9. **ArgoCD Application Deployment**
10. **YAML File Configuration**
11. **EKS Cluster Monitoring**
12. **DNS Configuration for ALB**

Step-1: Jenkins Server Setup and Install Essential Tools on Jenkins Server

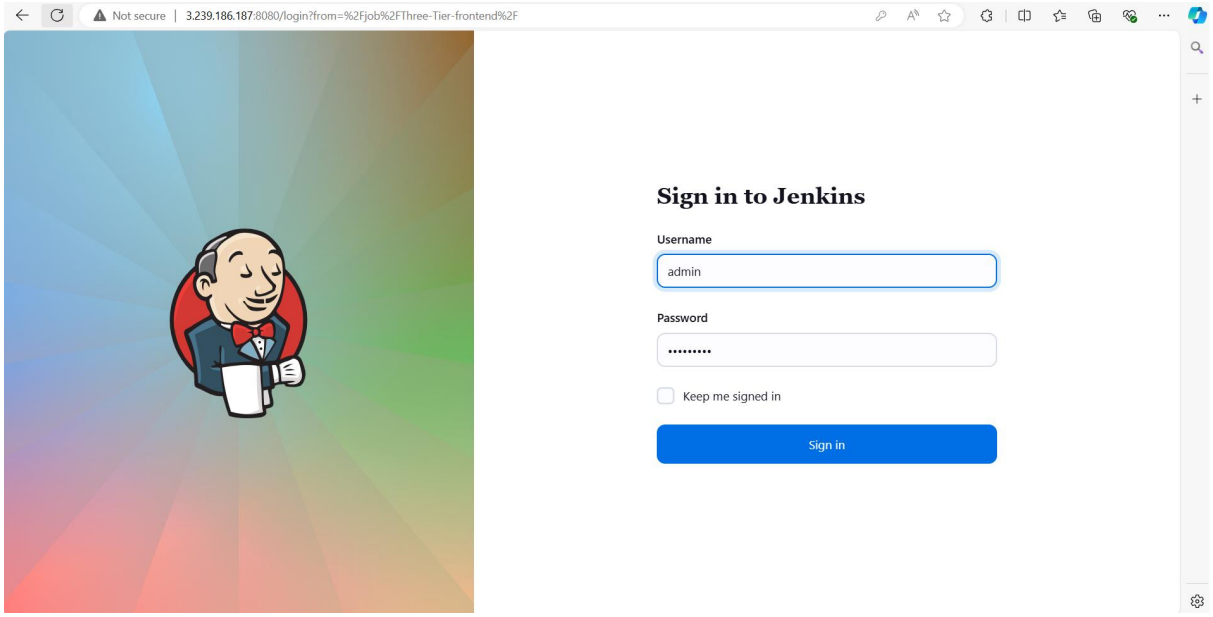
- Log in to the AWS Management Console, navigate to the EC2 dashboard, and click "Launch Instance." Choose an instance type and memory accordingly, configure security settings
- Add the user data script that is provided in git-hub to install the specific tools like java, Jenkins, AWS CLI, Trivy, Docker, SonarQube, Kubectl, eksctl and Helm and launch with your selected key pair.





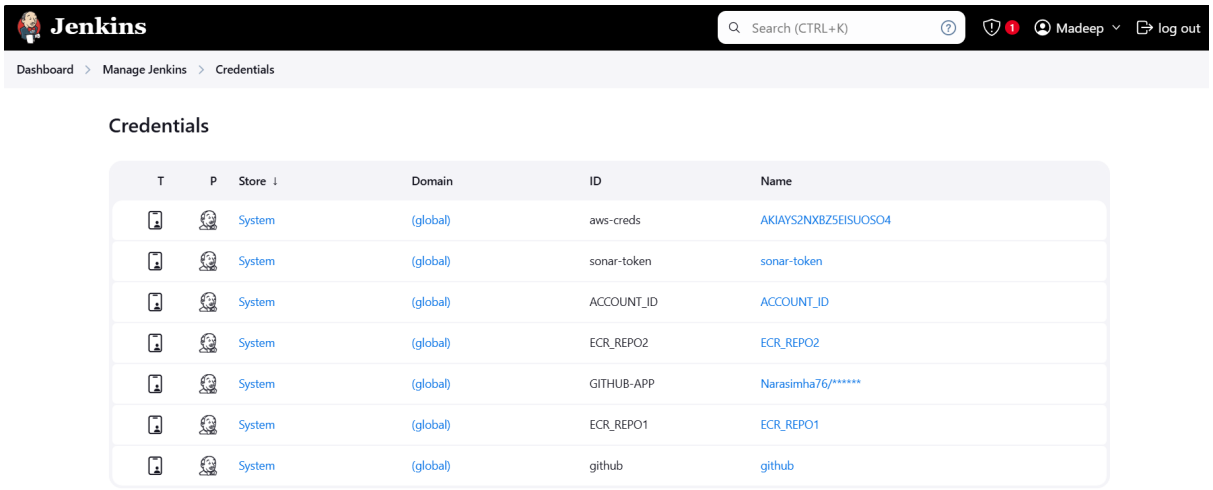
Step-2: Configure Jenkins Server

- Now, we have to configure Jenkins. So, copy the public IP of your Jenkins Server and paste it on your browser with an 8080 port.
- Install the Selected plugins.



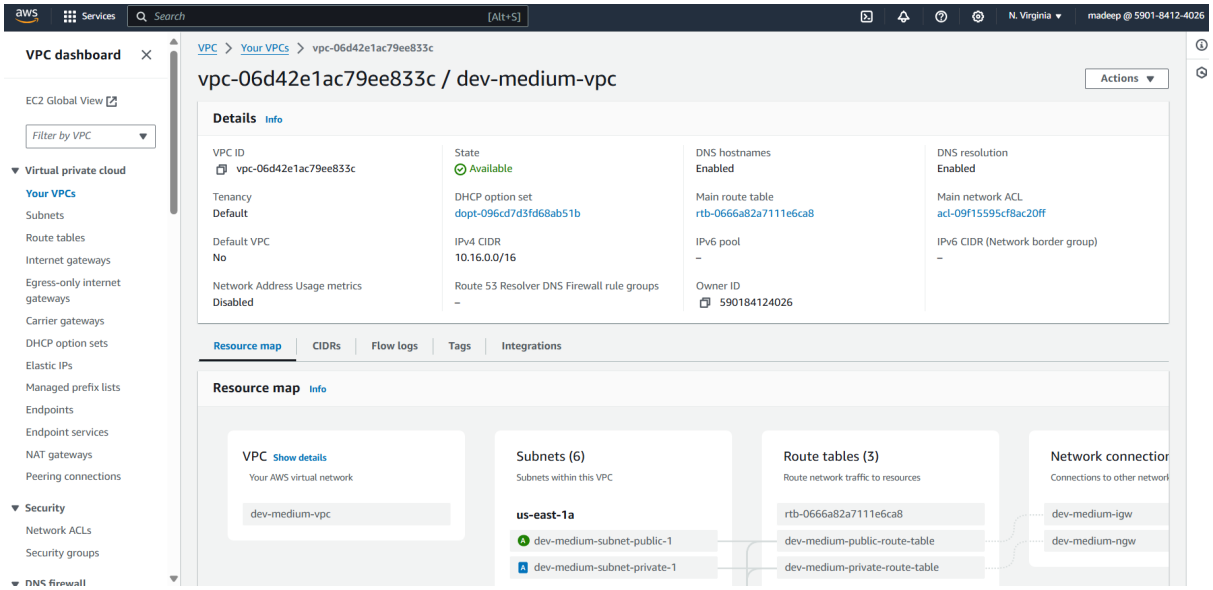
- Go to Manage Plugins in Jenkins, select Credentials, choose AWS Credentials as the Kind, add your AWS Access Key & Secret Access Key with a unique ID, and click Create.

- Add GitHub credentials by selecting Username and Personal Access Token, as repositories in industry projects are often private, and save the credentials for use in your pipelines.



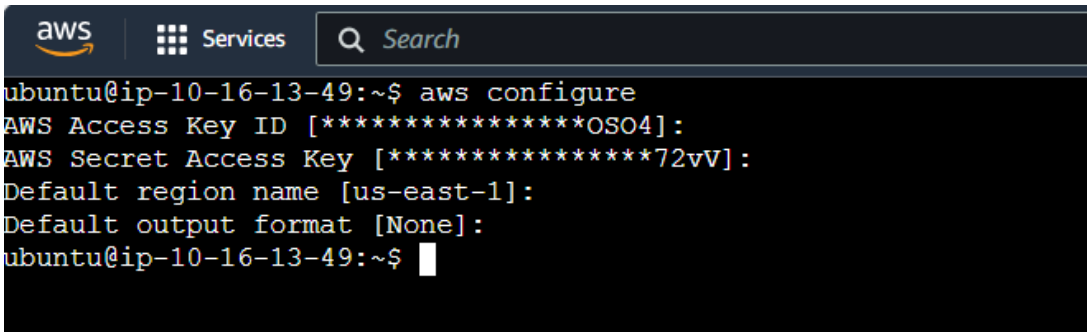
Step-3: Creating an VPC for the EKS and Jump Server

- Create a VPC with an internet gateway, a public subnet, a route table, a security group allowing access to specific ports (22, 8080, 9000, 9090, 80), and associating the route table with the subnet.



Step-4: Create jump Server an EKS Cluster using the below commands

1. Create an ec2 instance for the Jump Server using the below command:
   aws ec2 run-instances --image-id <image-id> --count 1 --instance-type <type> --key-name <keypair> --security-group-ids <sg-id>--vpc-id <vpc-id>--subnet-id <subnet-id> --region <region>
2. Create an AWS EKS cluster using the below command:
   ksctl create cluster --name <name> --region <region> --node-type <type> --nodes-min <min-num> --nodes-max <max-num>
3. Configure the aws credentials in the jump server using aws configure command:



4. After creation of the EKS cluster add the EKS cluster into the Jump server using the below command:
   aws eks update-kubeconfig --region us-east-1 --name <name of the cluster>

5. Verify the nodes that are creating along with the EKS cluster

```
ubuntu@ip-10-16-13-49:~$ kubectl get nodes
NAME                           STATUS   ROLES    AGE    VERSION
ip-10-16-162-211.ec2.internal   Ready    <none>   5d8h   v1.29.6-eks-1552ad0
ip-10-16-167-172.ec2.internal   Ready    <none>   5d8h   v1.29.6-eks-1552ad0
ubuntu@ip-10-16-13-49:~$
```

Step-5: Now, we will configure the Load Balancer on our EKS because our application will have an ingress controller.

1. 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

2. Create the IAM policy using the below command:
   aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json

3. Create OIDC Provider:
   eksctl utils associate-iam-oidc-provider --region=us-east-1 --cluster=Three-Tier-K8s-EKS-Cluster –approve

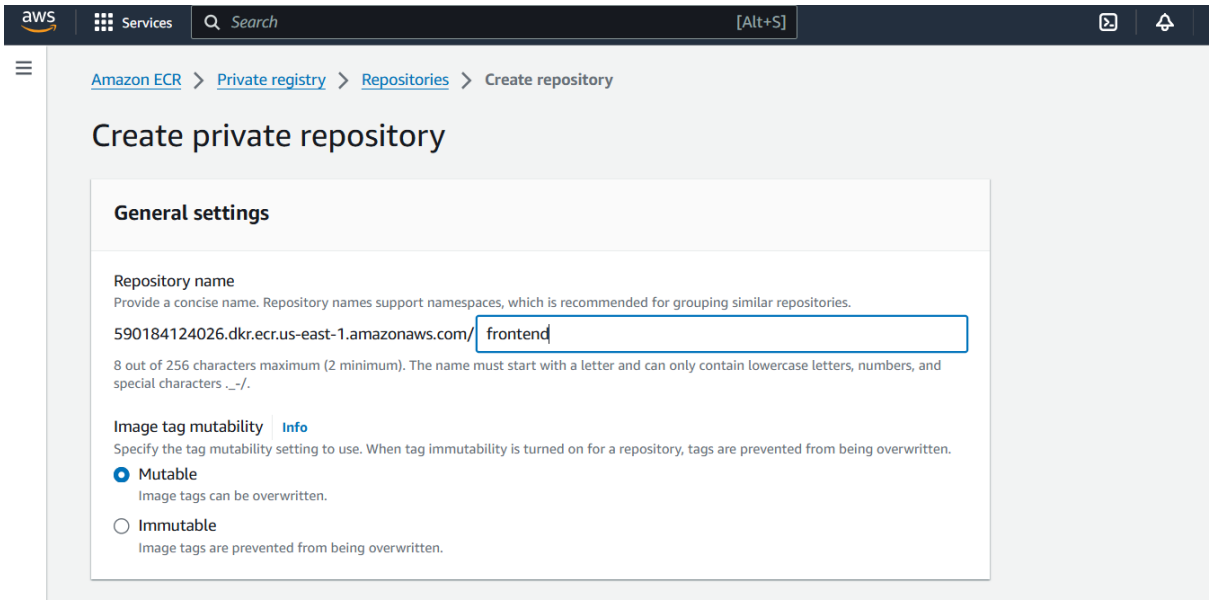
4. Create a Service Account by using below command and replace your account ID:
   eksctl create iamserviceaccount --cluster=Three-Tier-K8s-EKS-Cluster --namespace=kube-system --name=aws-load-balancer-controller --role-name AmazonEKSLoadBalancerControllerRole --attach-policy-arn=arn:aws:iam::<your_account_id>:policy/AWSLoadBalancerControllerIAMPolicy --approve --region=us-east-1

5. Run the below command to deploy the AWS Load Balancer Controller:
   helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=my-cluster --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller

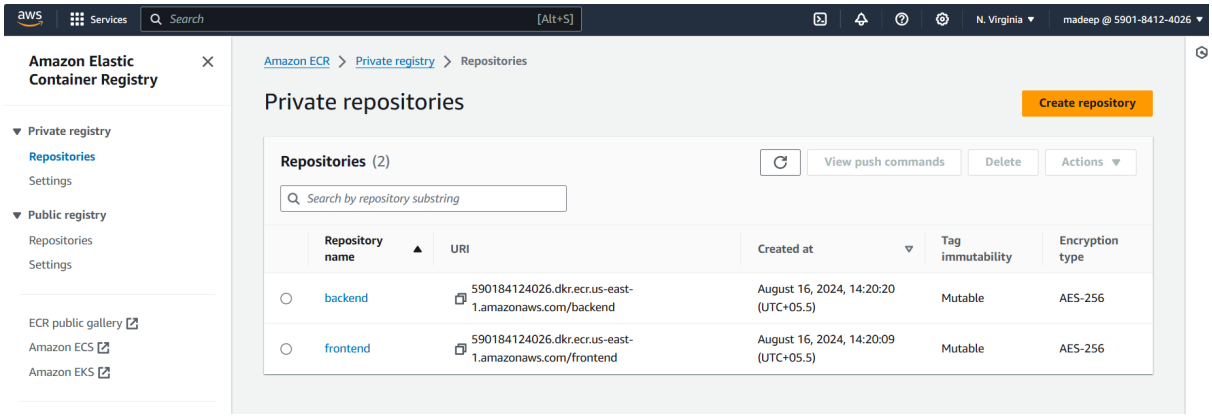
6. After 2 minutes, run the command below to check whether your pods are running or not:
   kubectl get deployment -n kube-system aws-load-balancer-controller

```
ubuntu@ip-10-16-13-49:~$ kubectl get deployment -n kube-system aws-load-balancer-controller
NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
aws-load-balancer-controller   2/2     2            2           5d9h
ubuntu@ip-10-16-13-49:~$
```

Step 6: We need to create Amazon ECR Private Repositories for both Tiers (Frontend & Backend)

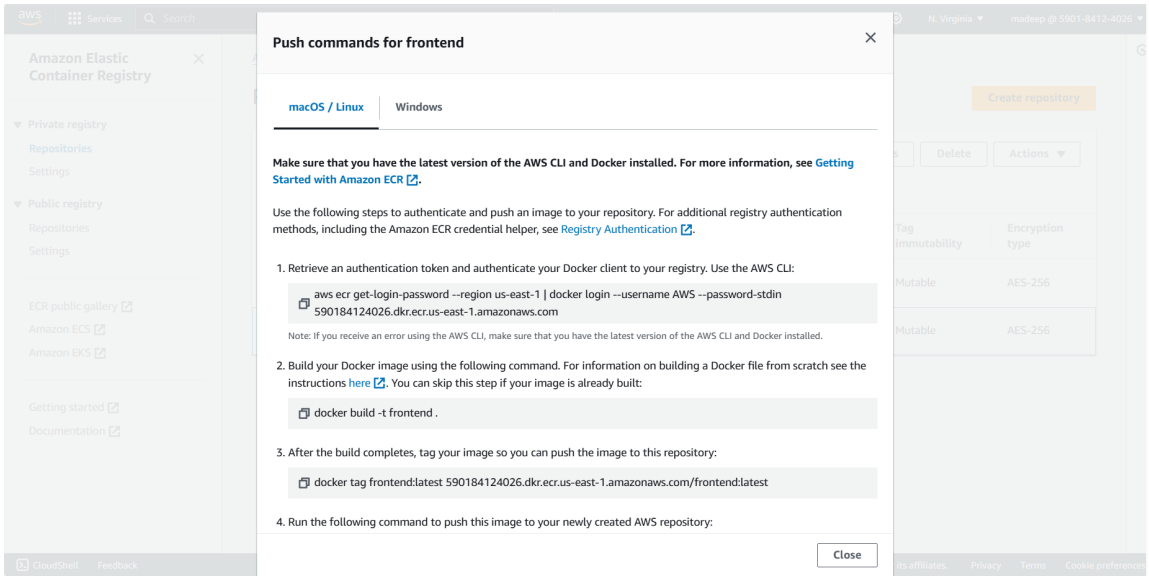


- Similarly create the Backend Repository



- Now, we need to configure ECR locally because we have to upload our images to Amazon ECR.

- Using the push commands we can configure



- Copy the 1st command for login
- Now, run the copied command on your Jenkins Server.

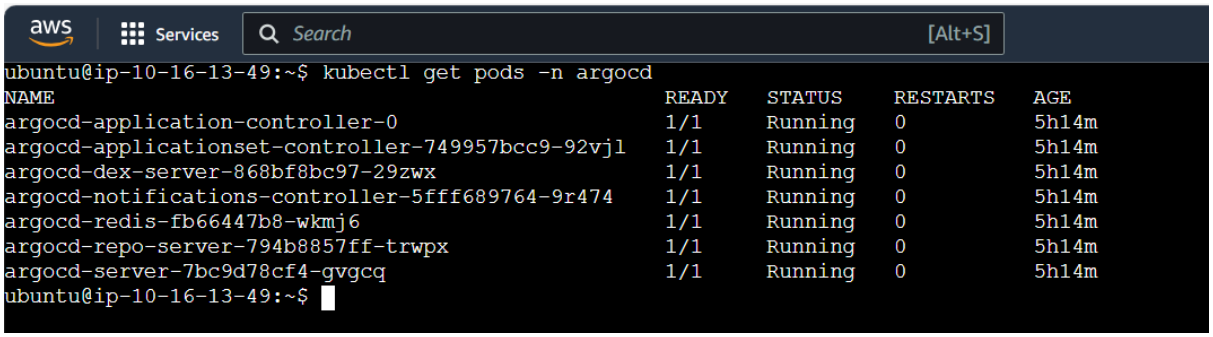
```
ubuntu@ip-10-0-25-107:~$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 590184124026.dkr.ecr.us-east-1.amazonaw
s.com
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@ip-10-0-25-107:~$
```

Step-7: Install and Configure ArgoCD

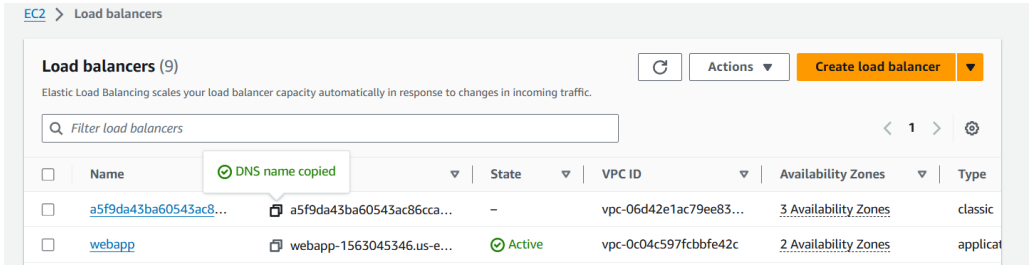
1. We will be deploying our application on a three-tier namespace. To do that, we will create a three-tier namespace on EKS.
2. Create an separate namespace for argocd using the command:
3. kubectl create namespace argoCD
4. Now, we will install argoCD
5. kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
6. these command will create all the required pods and services of the argocd server in the argocd namespace.

```
ubuntu@ip-10-16-13-49:~$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io unchanged
serviceaccount/argocd-application-controller unchanged
serviceaccount/argocd-applicationset-controller unchanged
serviceaccount/argocd-dex-server unchanged
serviceaccount/argocd-notifications-controller unchanged
serviceaccount/argocd-redis unchanged
serviceaccount/argocd-repo-server unchanged
serviceaccount/argocd-server unchanged
role.rbac.authorization.k8s.io/argocd-application-controller unchanged
role.rbac.authorization.k8s.io/argocd-applicationset-controller unchanged
role.rbac.authorization.k8s.io/argocd-dex-server unchanged
role.rbac.authorization.k8s.io/argocd-notifications-controller unchanged
role.rbac.authorization.k8s.io/argocd-server unchanged
clusterrole.rbac.authorization.k8s.io/argocd-application-controller unchanged
clusterrole.rbac.authorization.k8s.io/argocd-server unchanged
rolebinding.rbac.authorization.k8s.io/argocd-application-controller unchanged
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller unchanged
```

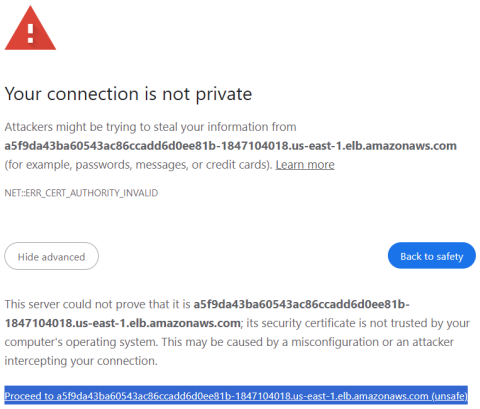
- All pods must be running, to validate run the below command:
  kubectl get pods -n argocd

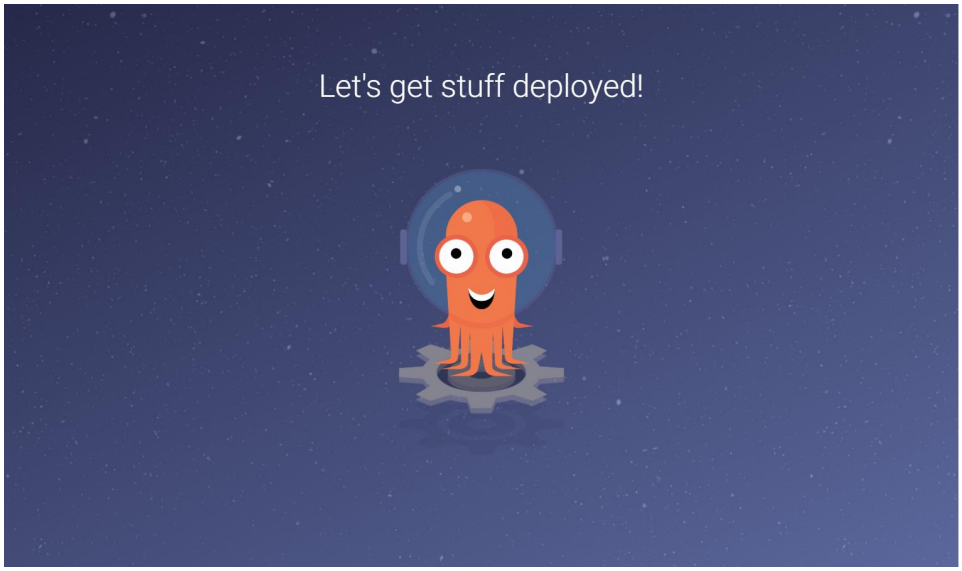


- Now, expose the argoCD server as LoadBalancer using the below command:
- kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
- You can validate whether the Load Balancer is created or not by going to the AWS Console:



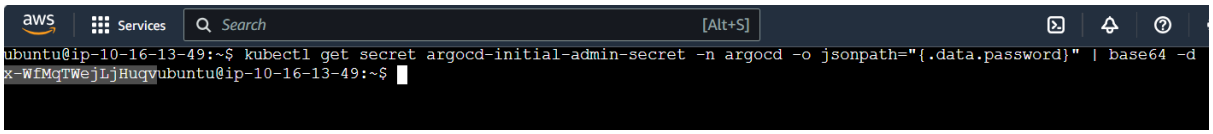
- To access the argoCD, copy the LoadBalancer DNS and hit on your favorite browser.
- You will get a warning like Your connection is not private then Click on Advanced.





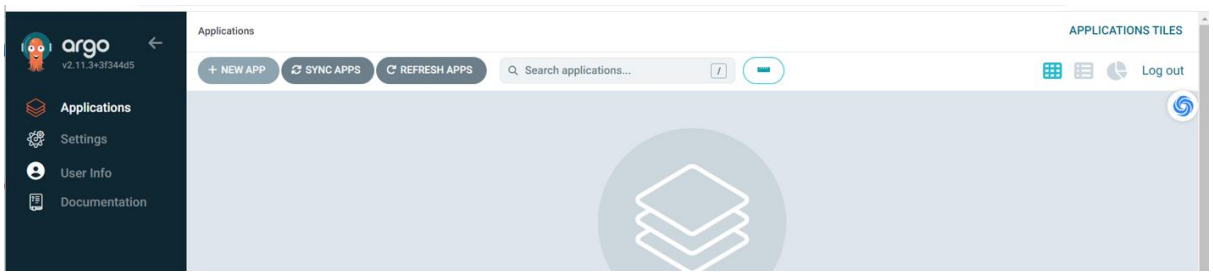
Now, we need to get the password for our argoCD server to perform the deployment using ArgoCD.

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



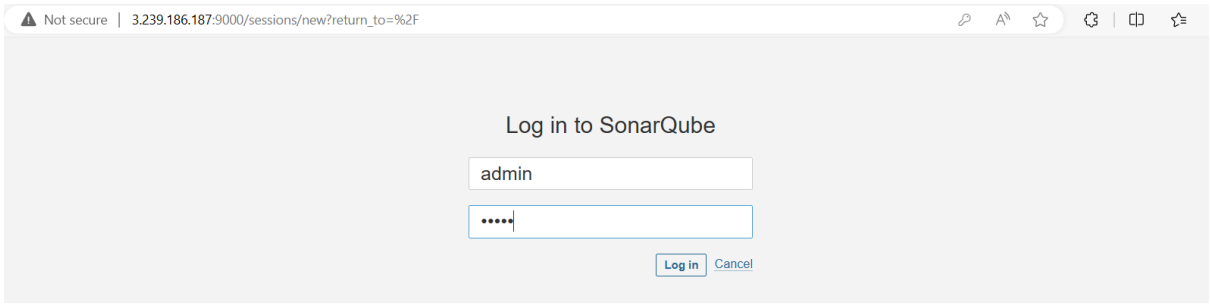
Enter the username (admin) and password in argoCD and click on SIGN IN.

Here is our ArgoCD Dashboard.



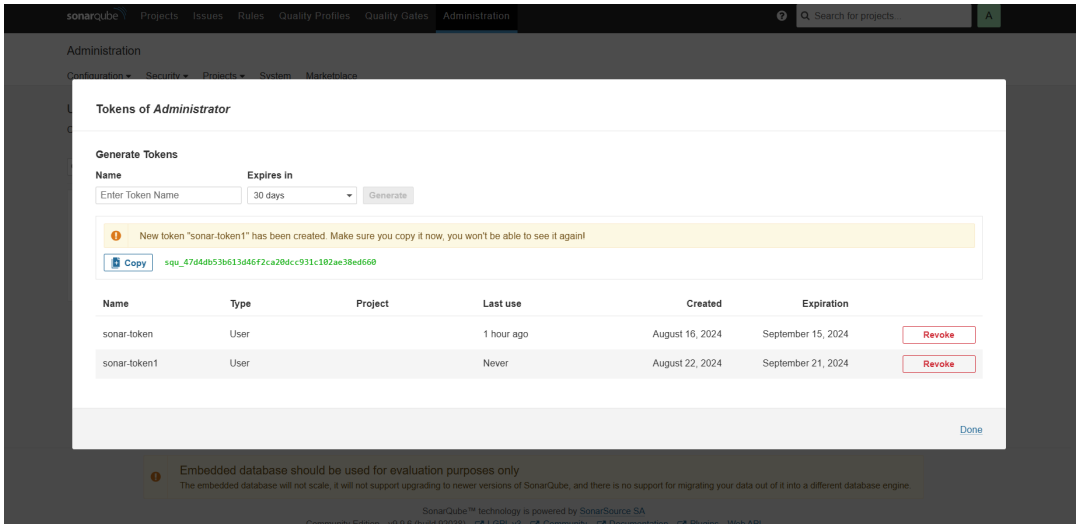
Step 8: Now, we have to configure Sonarqube for our DevSecOps Pipeline

- Previously we have created the instance using User script in the Script the Creation of Sonarqube container is declared so the Sonarqube is running in the Instance.
- To do that, copy your Jenkins Server public IP and paste it on your browser with a 9000 port



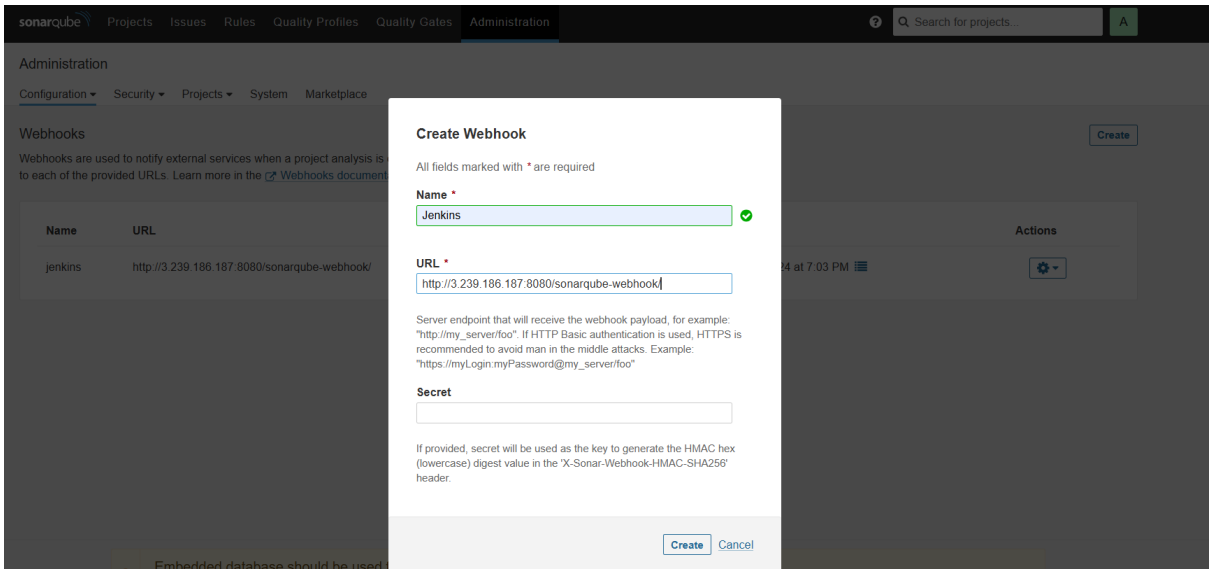
Now we need to generate the Token to access the Jenkins:

Goto: Administration > Security > users



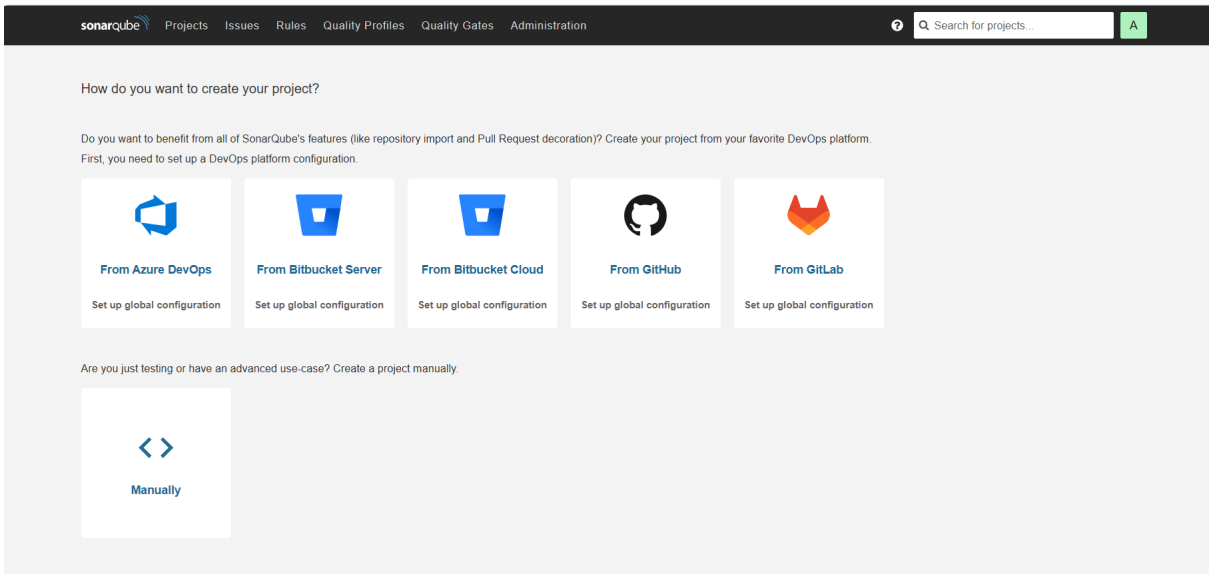
Now, we have to configure webhooks for quality checks.

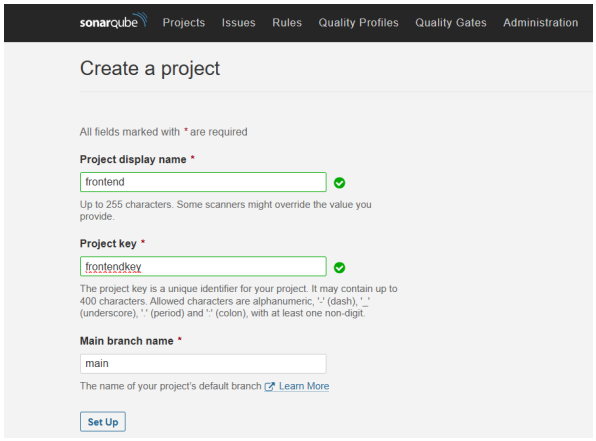
Click on Administration then, Configuration and select Webhooks



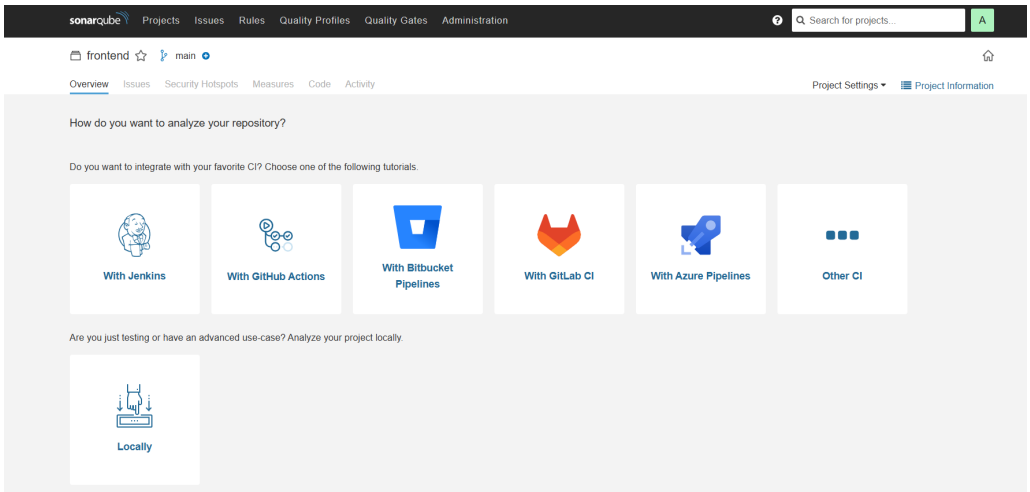
Now, we have to create a Project for frontend code.

Click on Manually

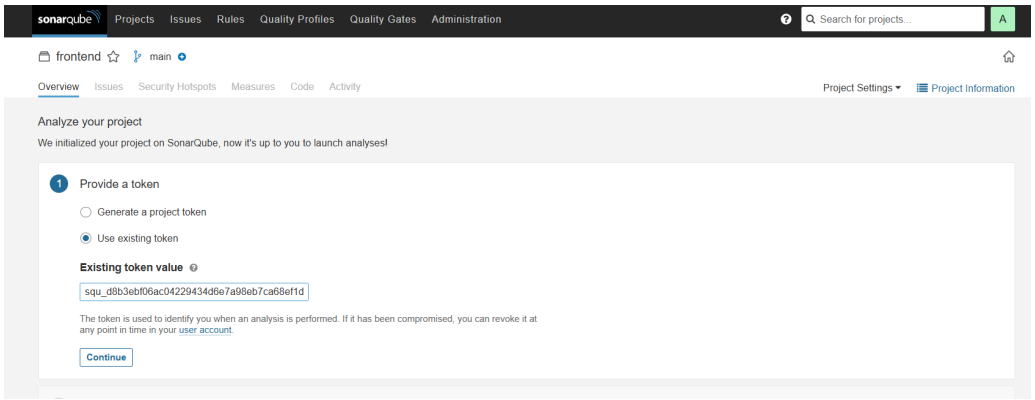




Click on Locally.



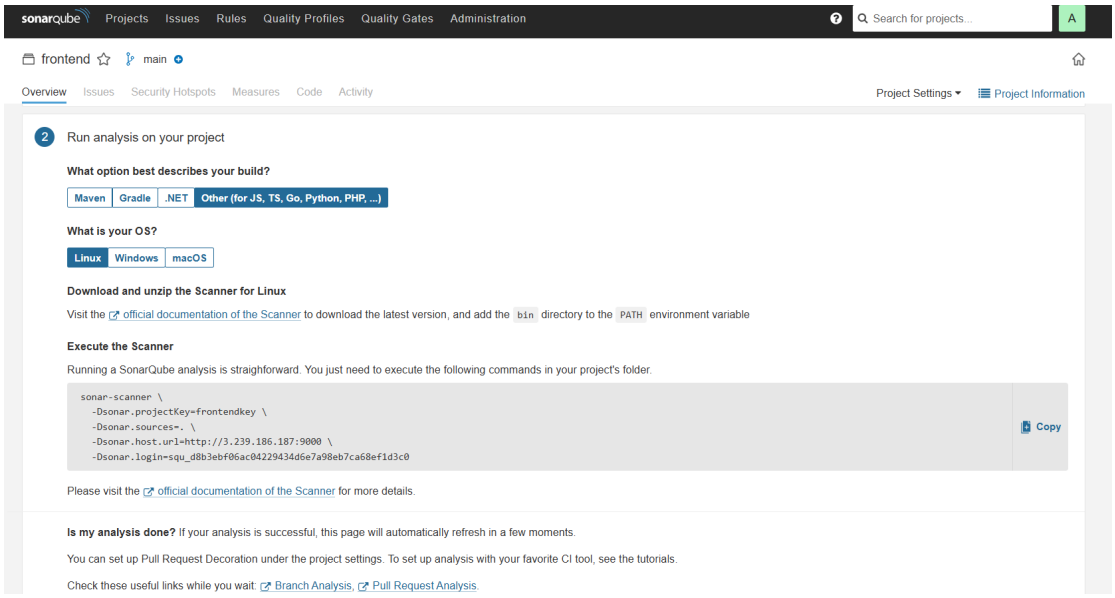
Select the Use existing token and click on Continue.



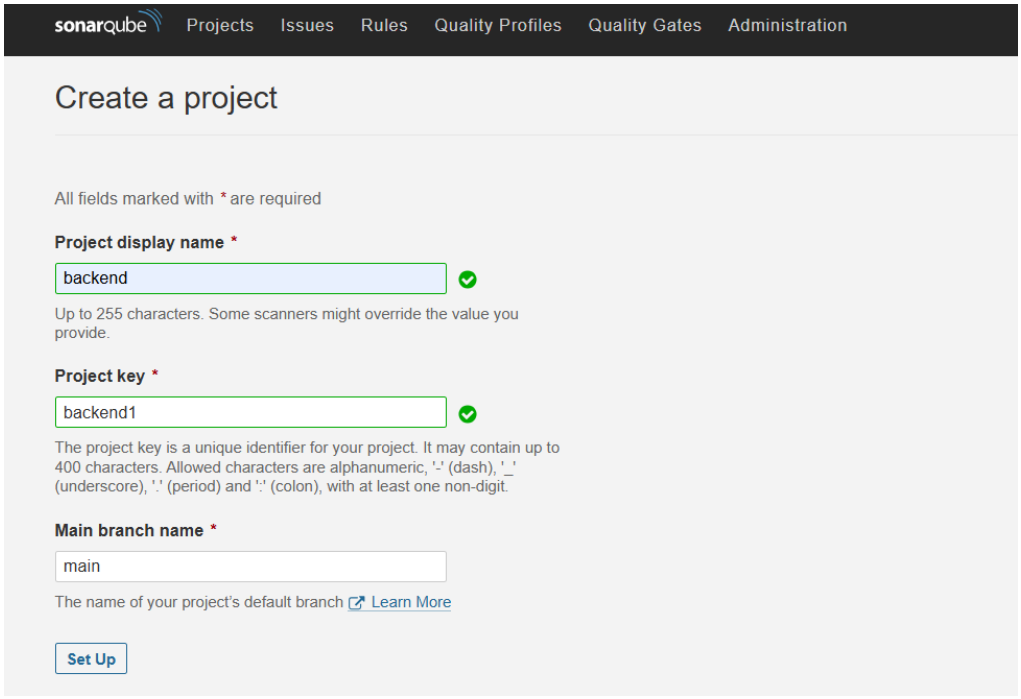
Select Other and Linux as OS.

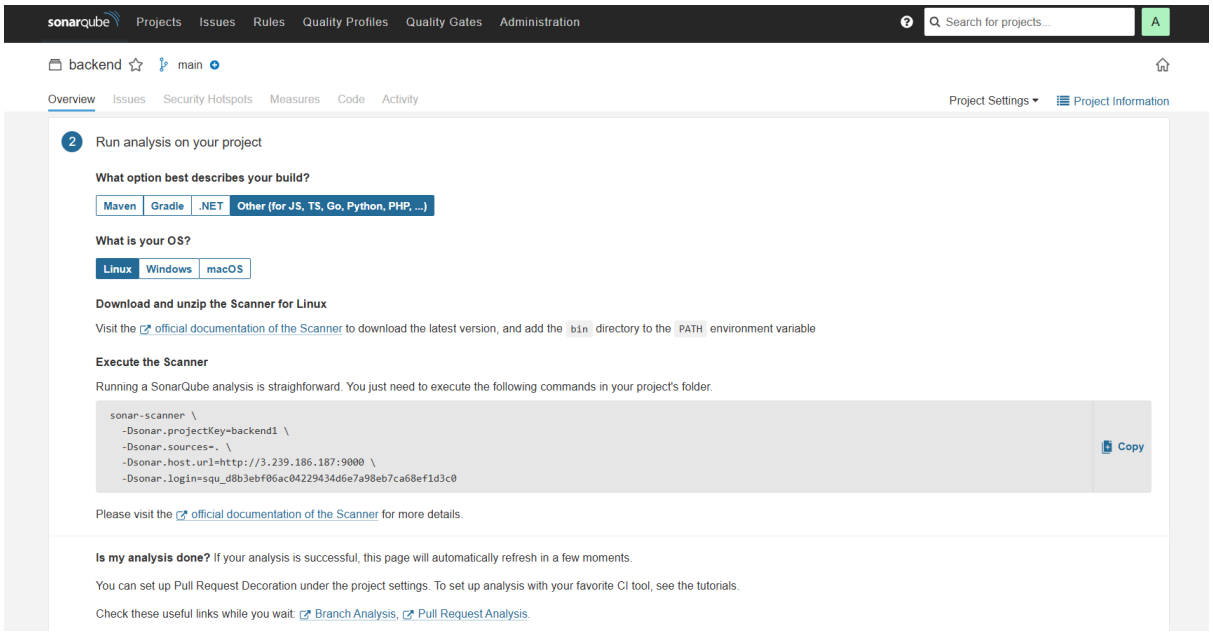
After performing the above steps, you will get the command which you can see in the below snippet.

Now, use the command in the Jenkins Frontend Pipeline where Code Quality Analysis will be performed.

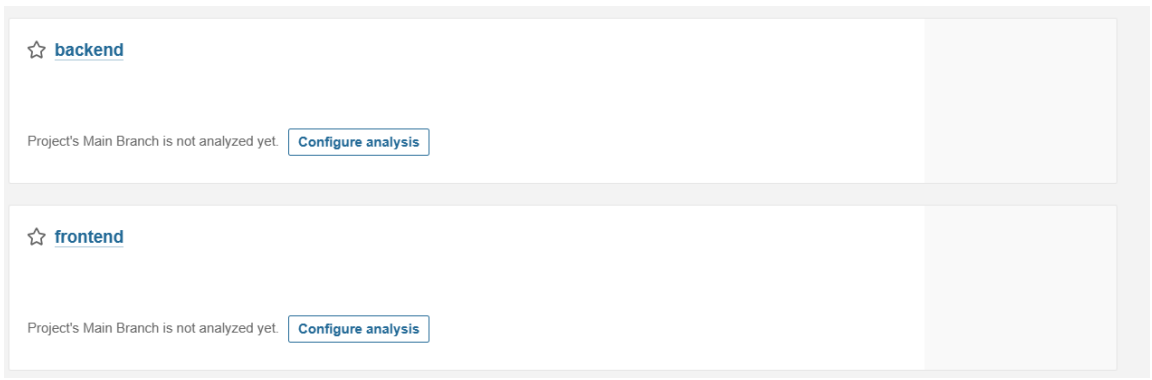


Now, we have to create a Project for backend code.





Now the Frontend and the Backend Project is Created in the Sonarqube.



Now, we have to store the sonar credentials in the Jenkins Credentials.

- Go to Dashboard -> Manage Jenkins -> Credentials
- Select the kind as Secret text paste your token in Secret and keep other things as it is.
- Click on Create

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	aws-creds	AKIAYS2NXBZ5EISUOSO4
		System	(global)	sonar-token	sonar-token
		System	(global)	ACCOUNT_ID	ACCOUNT_ID

- Now create the Credentials of GitHub Personal access Token (Not password) and Also create the ACCOUNT ID in secret and also create the secret name for the Frontend Repo and Backend Repo in the Jenkins Credentials.

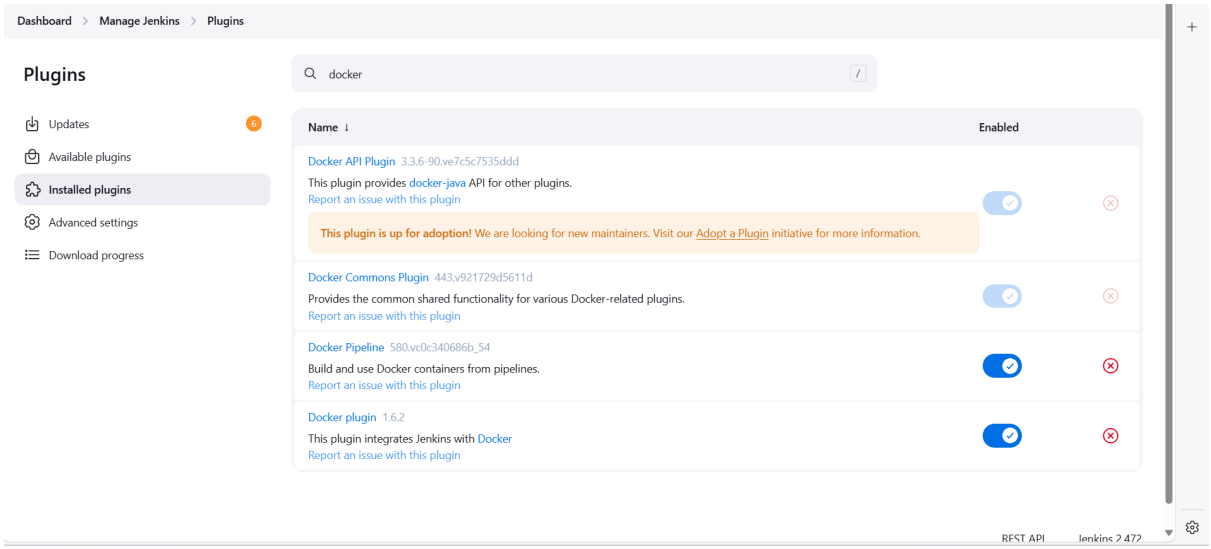
Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	aws-creds	AKIAYS2NXBZ5EISUOSO4
		System	(global)	sonar-token	sonar-token
		System	(global)	ACCOUNT_ID	ACCOUNT_ID
		System	(global)	ECR_REPO2	ECR_REPO2
		System	(global)	GITHUB-APP	Narasimha76/******
		System	(global)	ECR_REPO1	ECR_REPO1
		System	(global)	github	github

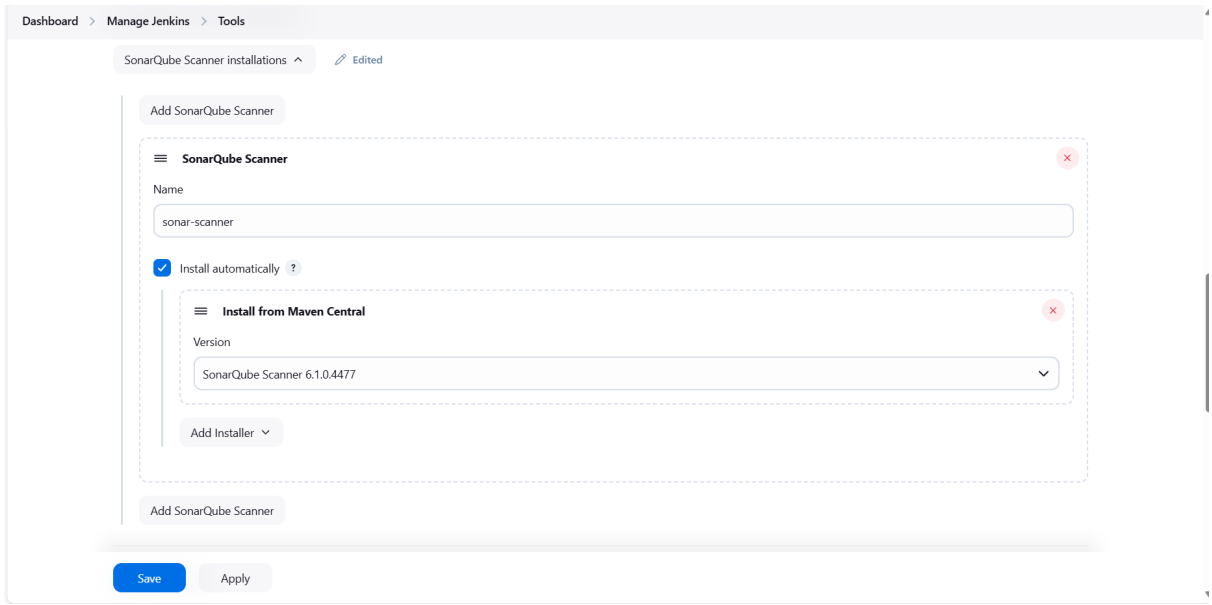
Step-9: Install the required plugins and configure the plugins to deploy our Three-Tier Application

Install the following plugins by going to Dashboard -> Manage Jenkins -> Plugins -> Available Plugins

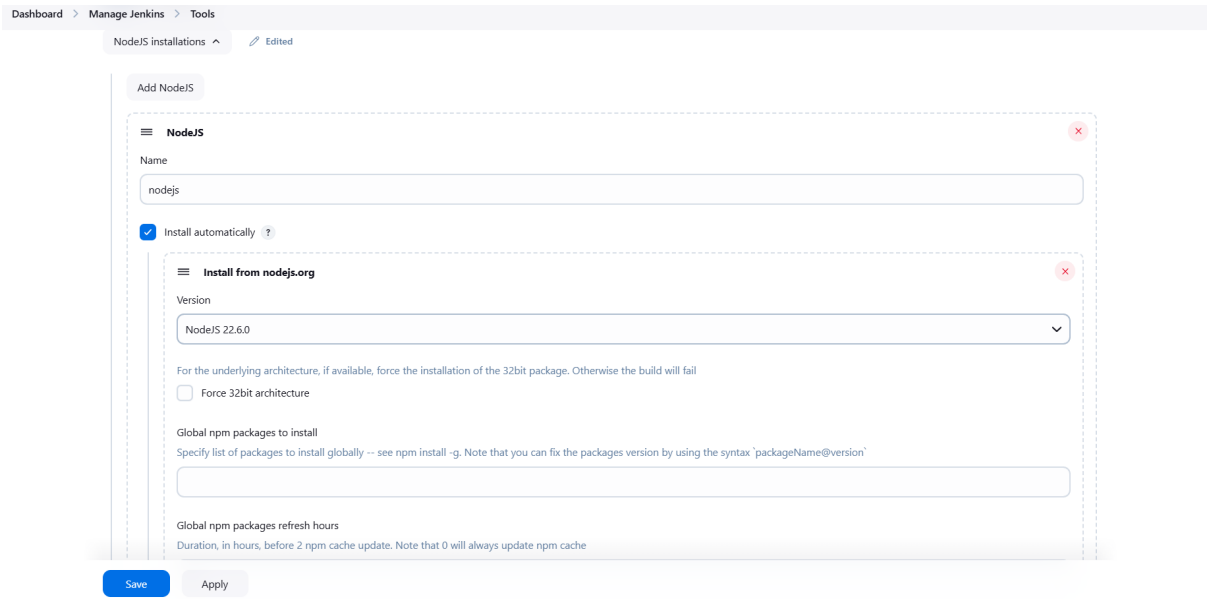
1. Docker
2. Docker Commons
3. Docker Pipeline
4. Docker API
5. docker-build-step
6. Eclipse Temurin installer
7. NodeJS
8. OWASP Dependency-Check
9. SonarQube Scanner



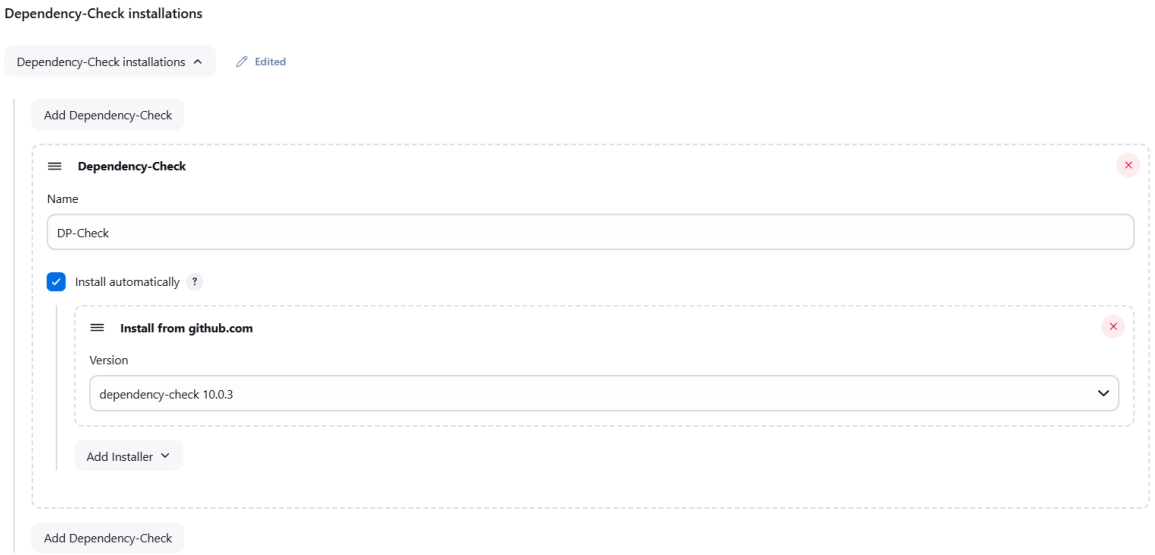
- Now, we have to configure the installed plugins.
- Go to Dashboard -> Manage Jenkins -> Tools
- Now, we will configure the sonarqube-scanner
- Search for the sonarqube scanner and provide the configuration like the below snippet.



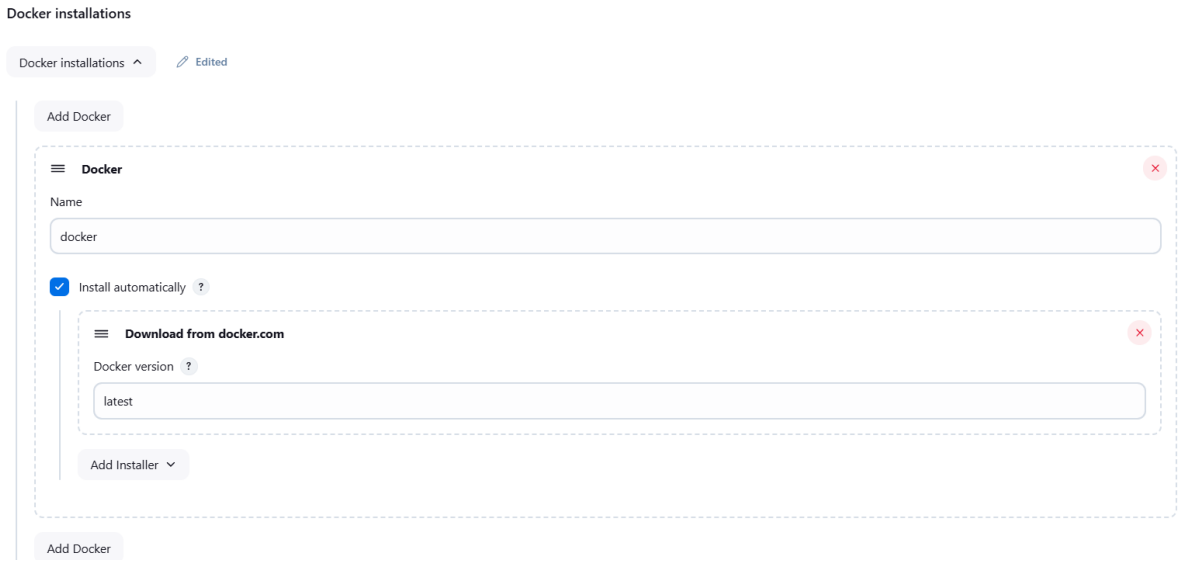
- Now, we will configure nodejs
- Search for node and provide the configuration like the below snippet.



- Now, we will configure the OWASP Dependency check
- Search for Dependency-Check and provide the configuration like the below snippet.

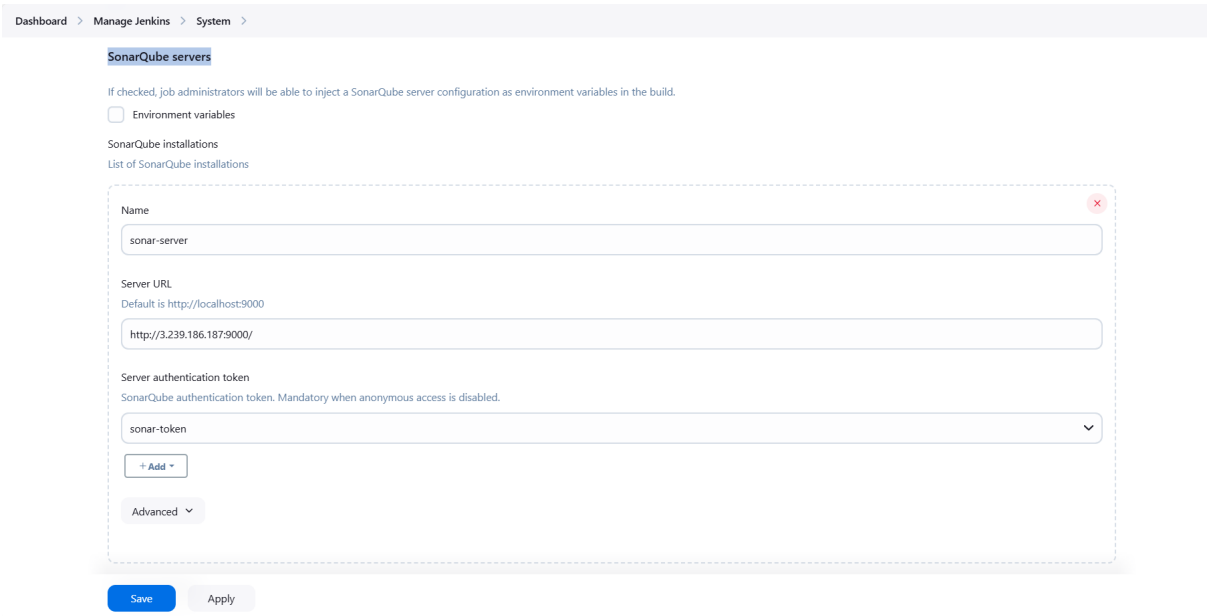


- Now, we will configure the docker
- Search for docker and provide the configuration like the below snippet.



Now, we have to set the path for Sonarqube in Jenkins

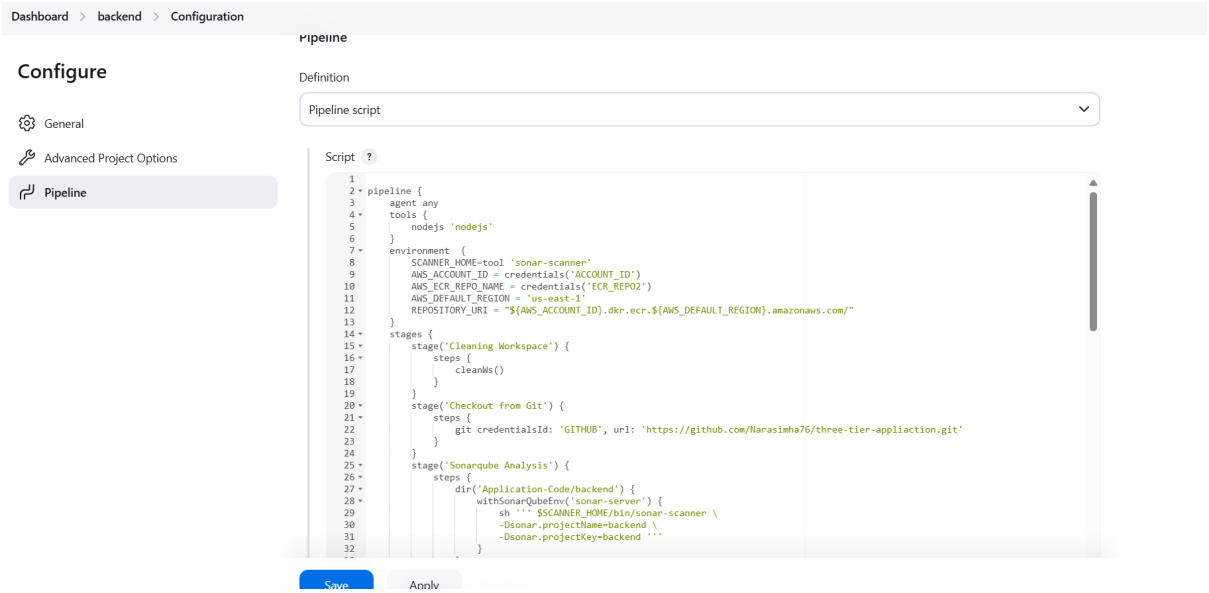
- Go to Dashboard -> Manage Jenkins -> System
- Search for SonarQube installations
- Provide the name as it is, then in the Server URL copy the sonarqube public IP (same as Jenkins) with port 9000 select the sonar token that we have added recently, and click on Apply & Save.



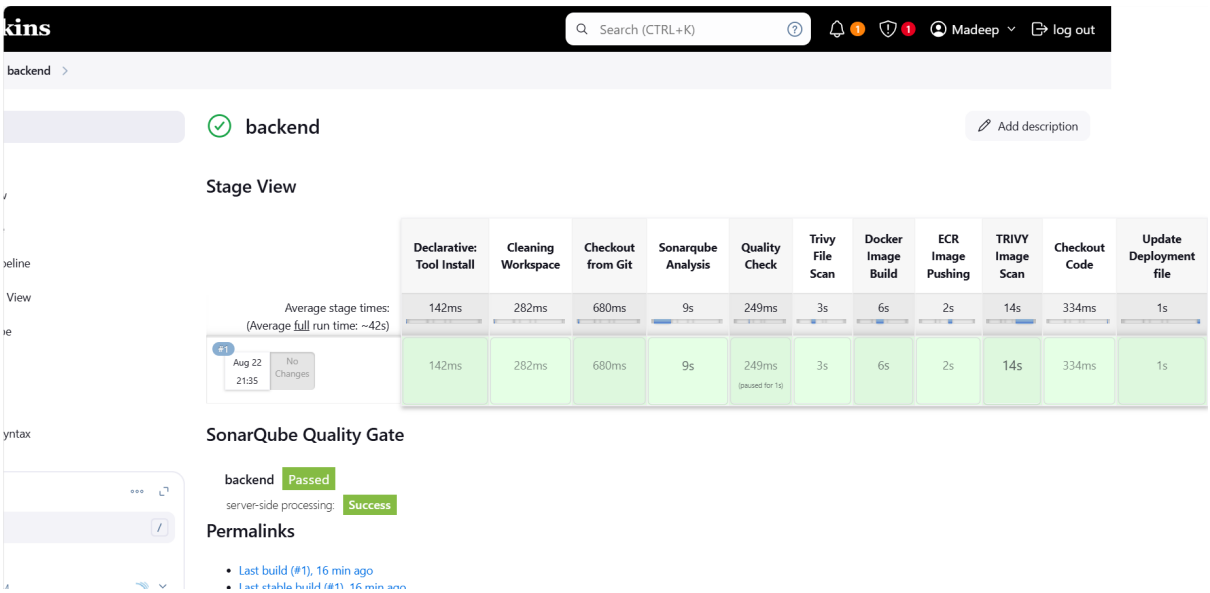
Step-10: Now create our Jenkins Pipeline to deploy our Backend Code.

Create an Pipeline for the Backend Application

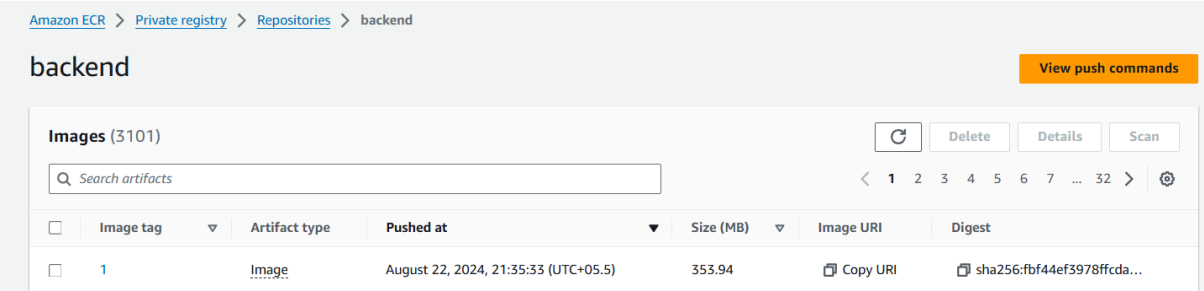
- This is the Jenkins file to deploy the Backend Code on EKS.
- Copy and paste it into the Jenkins Pipeline Script https://github.com/Narasimha76/three-tier-appliaction/blob/main/Jenkins-Pipeline-Code/Jenkinsfile-Backend
- Note: Do the changes in the Pipeline according to your project.



- Click Apply & Save.
- Now, click on the build.

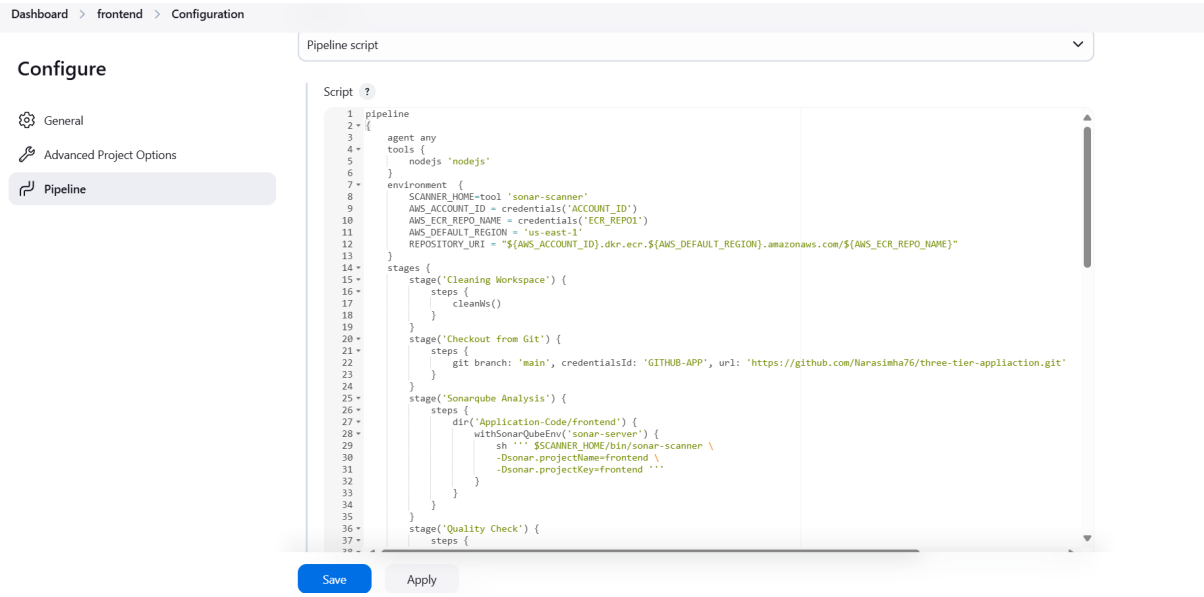


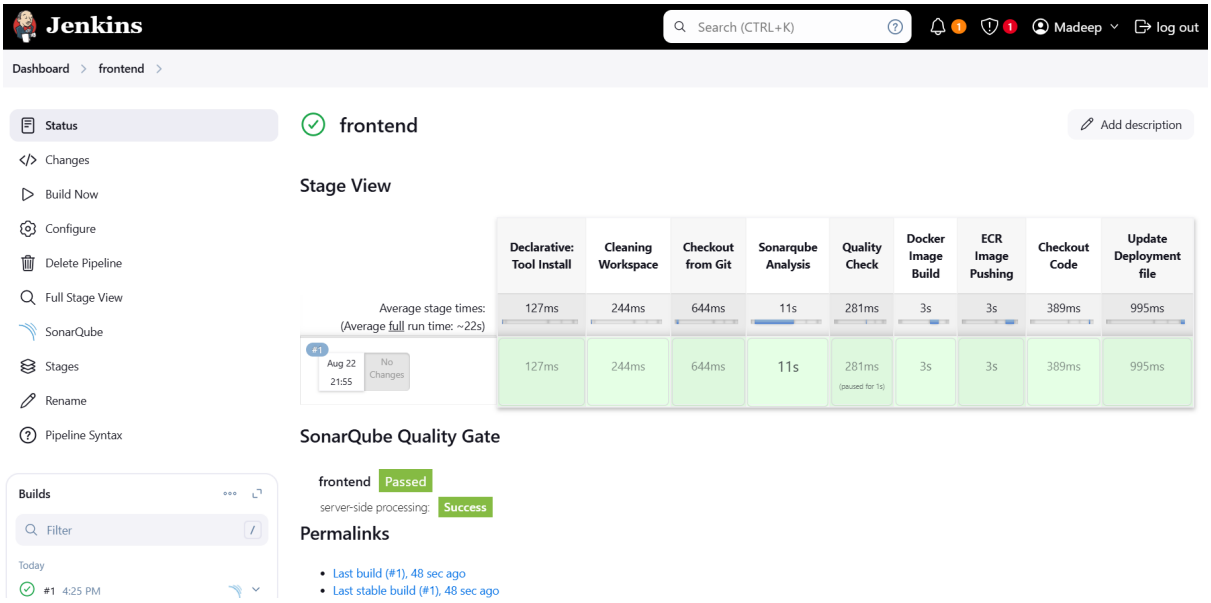
- You can check the ECR repository for the new image in the backend repository



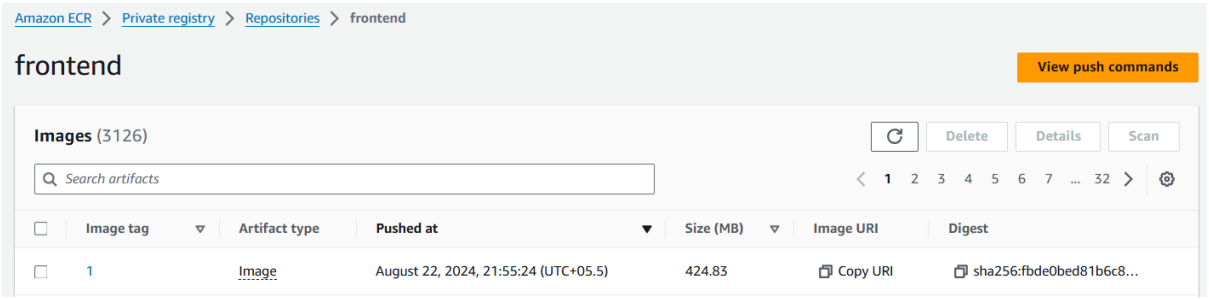
Now Create another pipeline for the Frontend Application

- This is the Jenkins file to deploy the Frontend Code on EKS.
- Copy and paste it into the Jenkins Pipeline Script https://github.com/Narasimha76/three-tier-appliaction/blob/main/Jenkins-Pipeline-Code/Jenkinsfile-Frontend
- Note: Do the changes in the Pipeline according to your project.



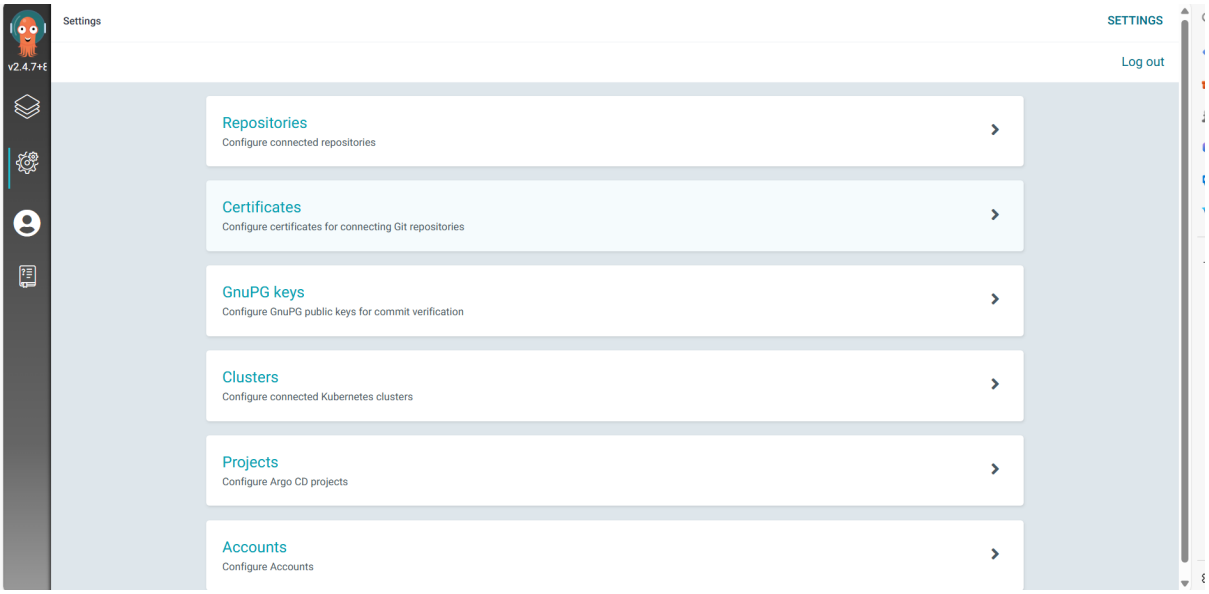


- You can see all the stages are executed successfully in the pipeline script
- You can check the ECR repository for the new image in the frontend repository

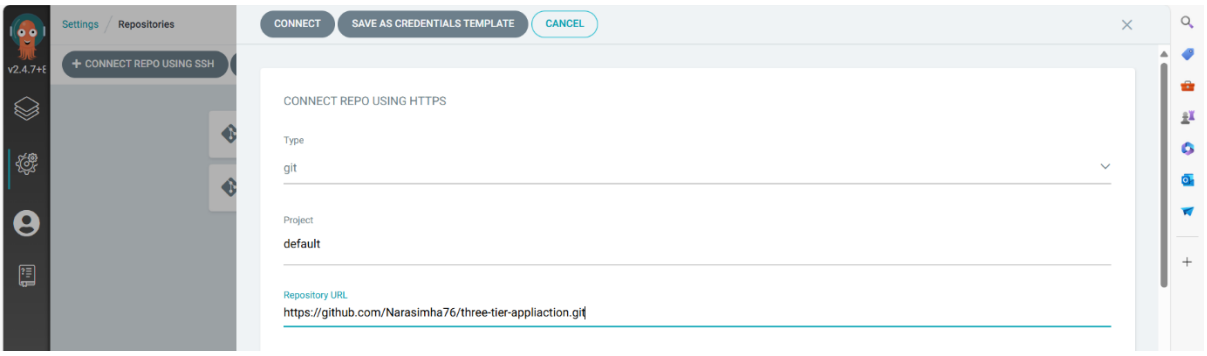


Step-11: We will deploy our Three-Tier Application using ArgoCD.

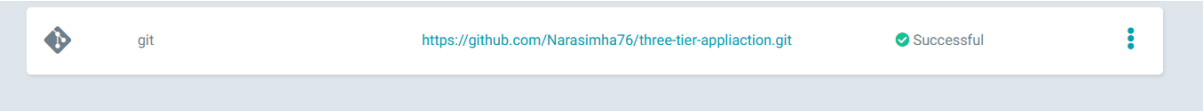
- As our repository is private. So, we need to configure the Private Repository in ArgoCD.
- Click on Settings and select Repositories.
- Add your github repository where the Kubernetes Manifest Files are present.



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

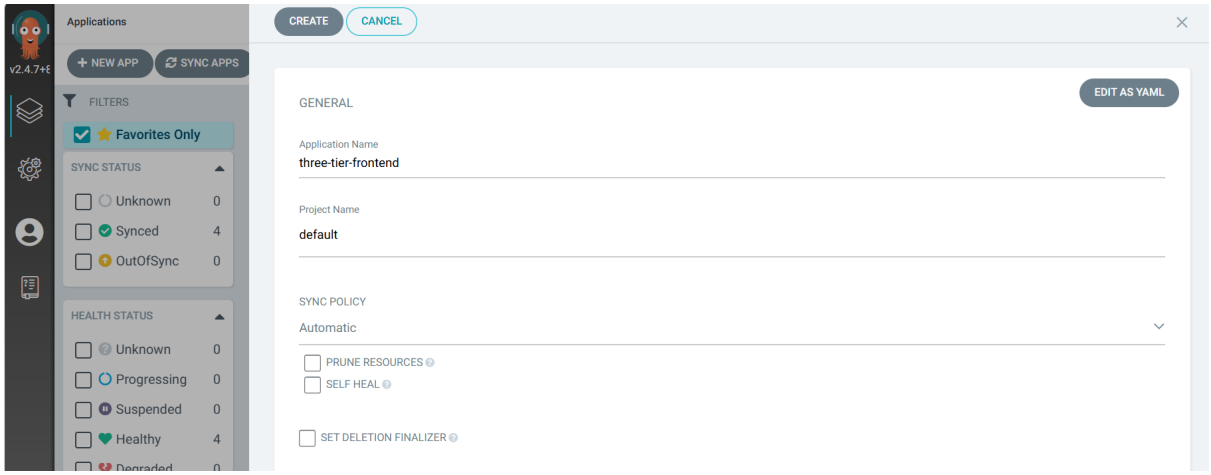


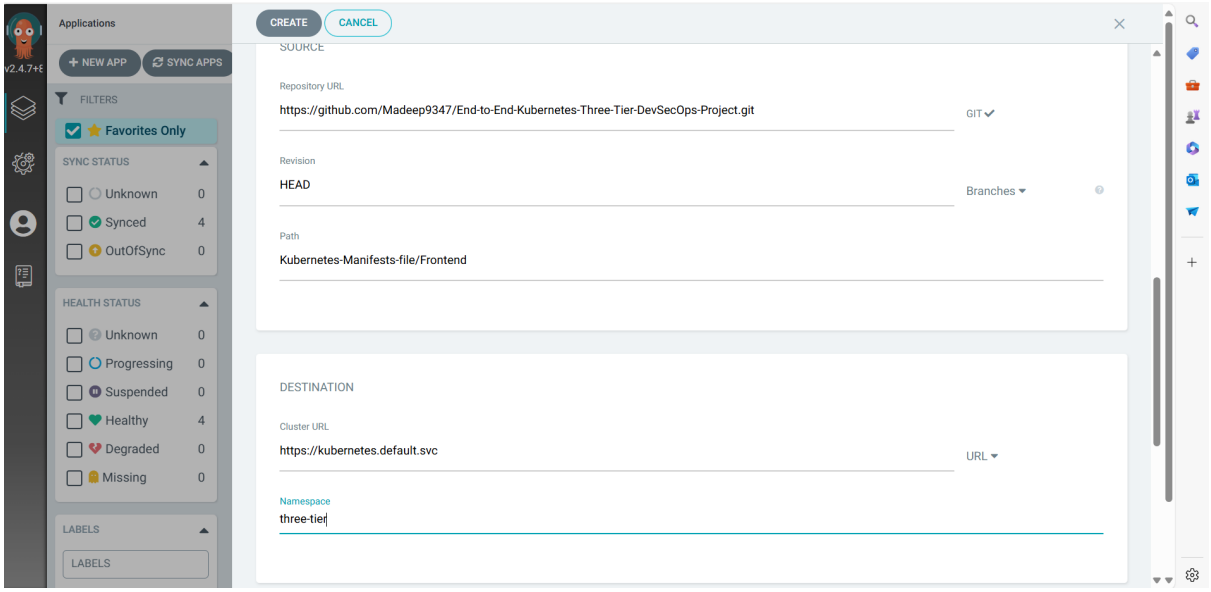
- And Click on connect then the repo will be connected to the argocd:



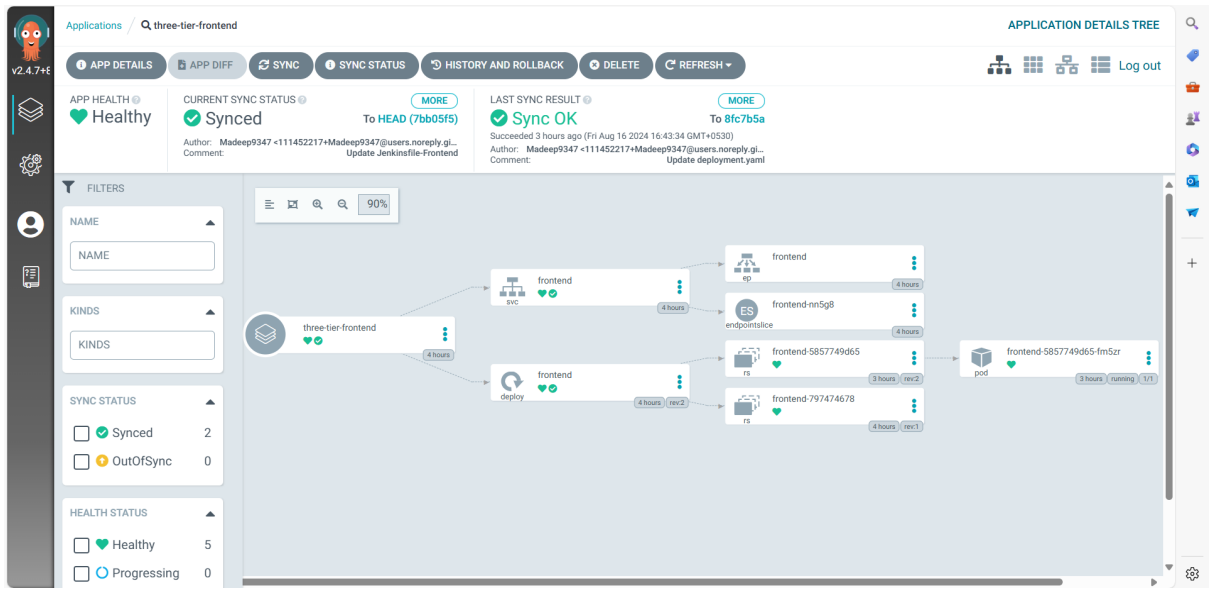
- Now create the separate apps for Frontend, Backend, Database and the ingress using manifest files:

- Frontend app creation:

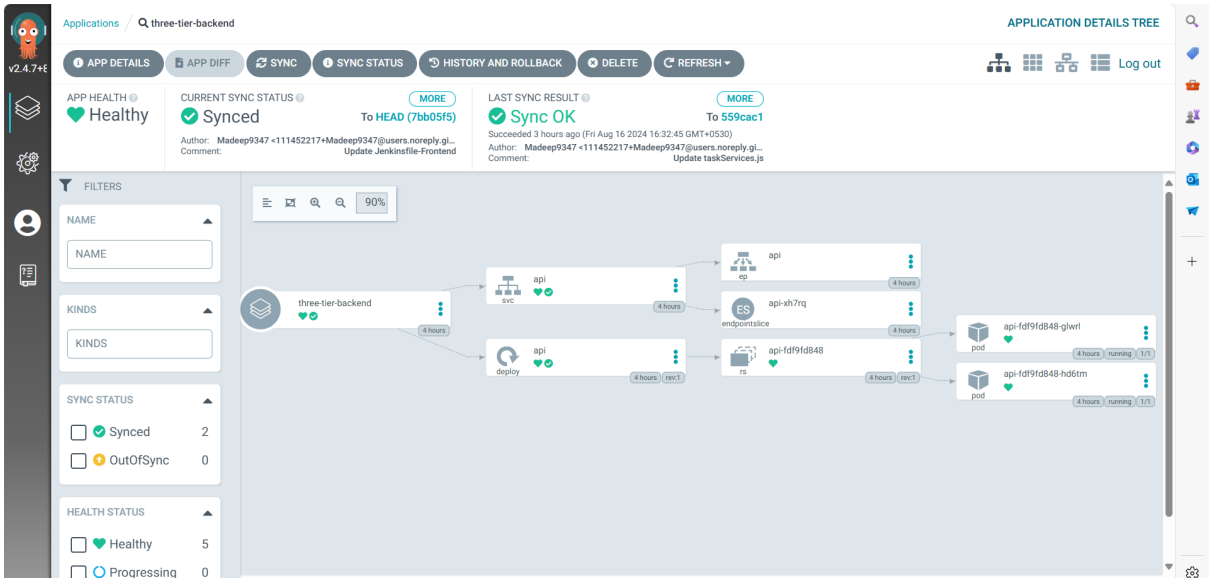




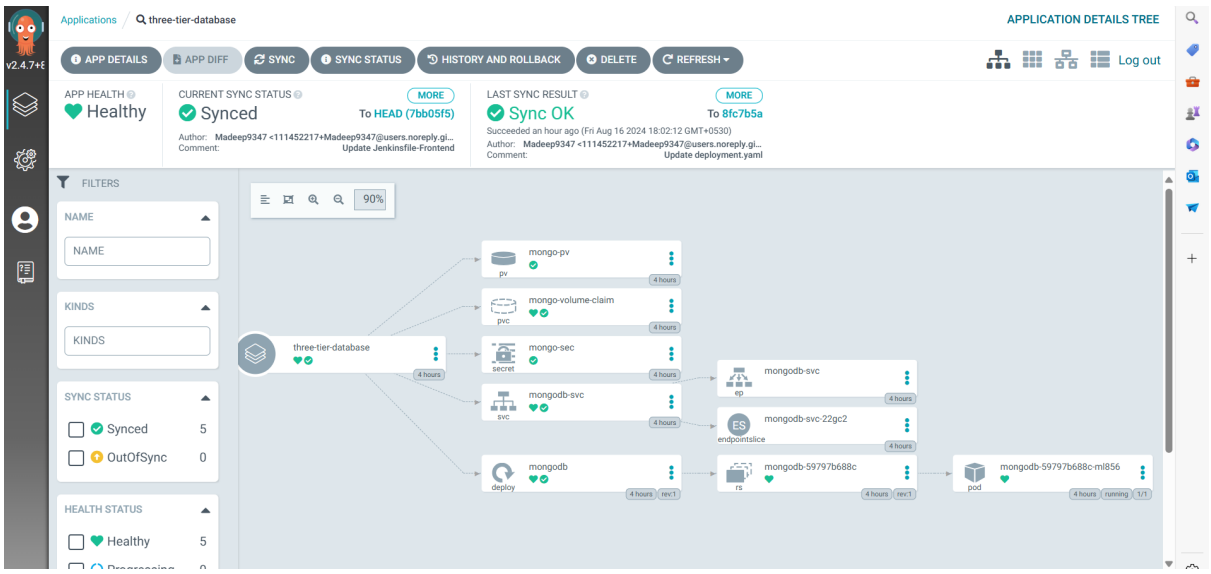
- This is the Frontend Application Deployment in ArgoCD:



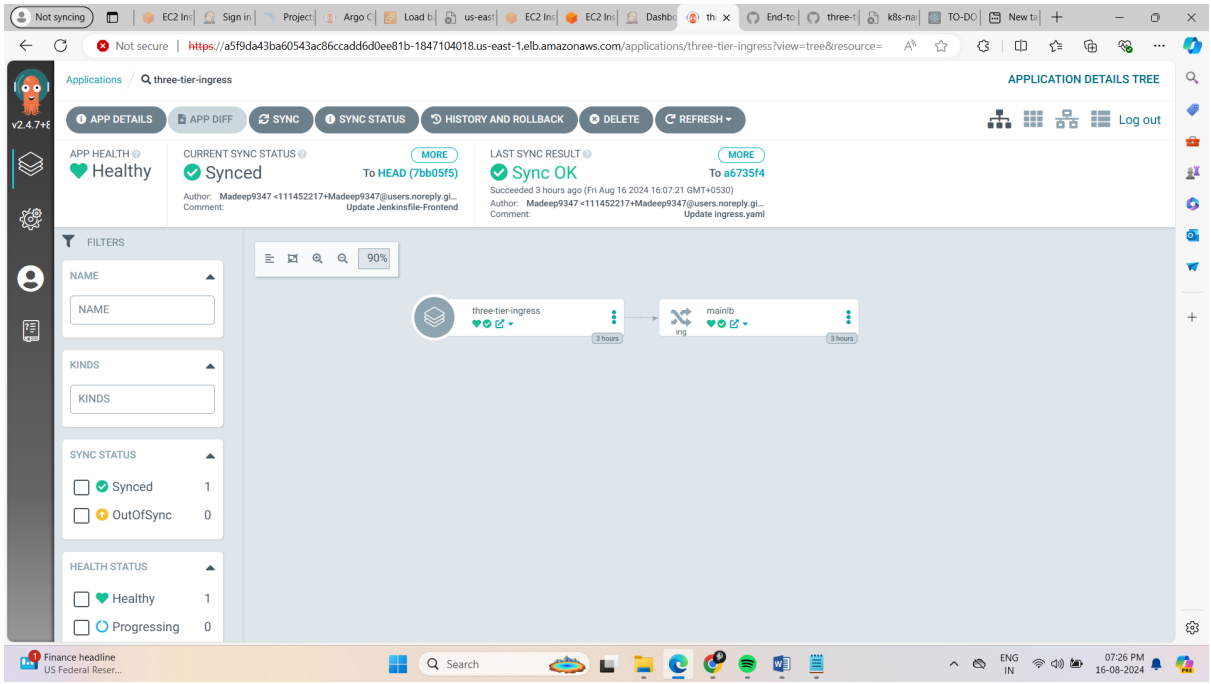
- Create these apps same for the Backend, Database and Ingress.
- This is the Backend Application Deployment in ArgoCD:



- This is the Database Application Deployment in ArgoCD:

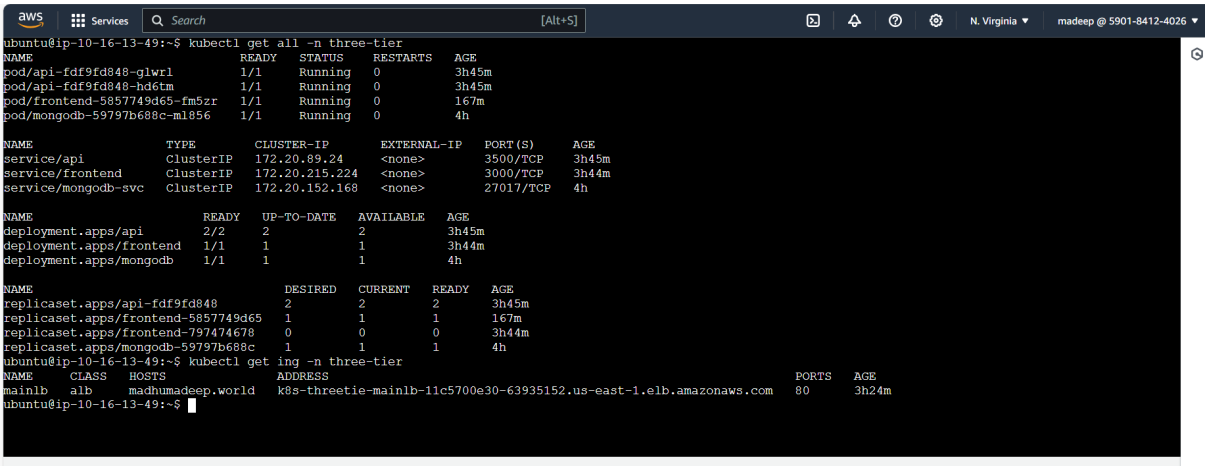


- This is the Ingress Application Deployment in ArgoCD:



- 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.

- 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-12: Make sure to modify the deployment.yml files in frontend, backend.

1. In frontend deployment file edit the file with your Domain name

```
23        spec:
24          imagePullSecrets:
25          - name: ecr-registry-secret
26          containers:
27          - name: frontend
28            image: 590184124026.dkr.ecr.us-east-1.amazonaws.com/frontend:1
29            imagePullPolicy: Always
30            env:
31              - name: REACT_APP_BACKEND_URL
32                value: "http://narasimha.madhumadeep.world/api/tasks"
33            ports:
34            - containerPort: 3000
```

2. Edit Backend deployment file

```
23        spec:
24          imagePullSecrets:
25          - name: ecr-registry-secret
26          containers:
27          - name: api
28            image: 590184124026.dkr.ecr.us-east-1.amazonaws.com/backend:1
29            imagePullPolicy: Always
30            env:
31              - name: MONGO_CONN_STR
32                value: mongodb://mongodb-svc:27017/todo?directConnection=true
33              - name: MONGO_USERNAME
34                valueFrom:
35                  secretKeyRef:
36                    name: mongo-sec
37                    key: username
38              - name: MONGO_PASSWORD
39                valueFrom:
40                  secretKeyRef:
41                    name: mongo-sec
42                    key: password
43            ports:
44            - containerPort: 3500
```

Step-13: We will set up the Monitoring for our EKS Cluster. We can monitor the Cluster Specifications and other necessary things.

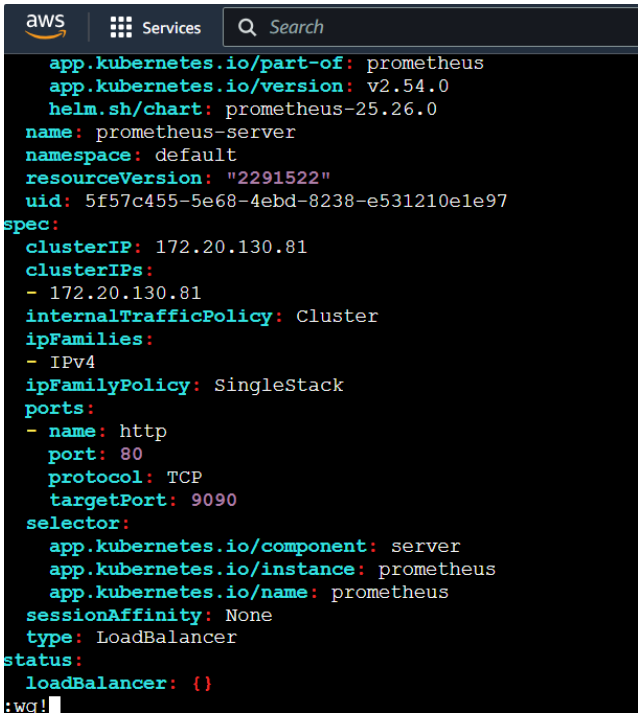
We will achieve the monitoring using Helm

1. Add the prometheus repo by using the below command:
   helm repo add stable https://charts.helm.sh/stable
2. Install the Prometheus:
   helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
3. helm install prometheus prometheus-community/prometheus
4. helm repo add grafana https://grafana.github.io/helm-charts
5. helm repo update
6. helm install grafana grafana/grafana
7. Now, check the service by the below command:
   kubectl get svc

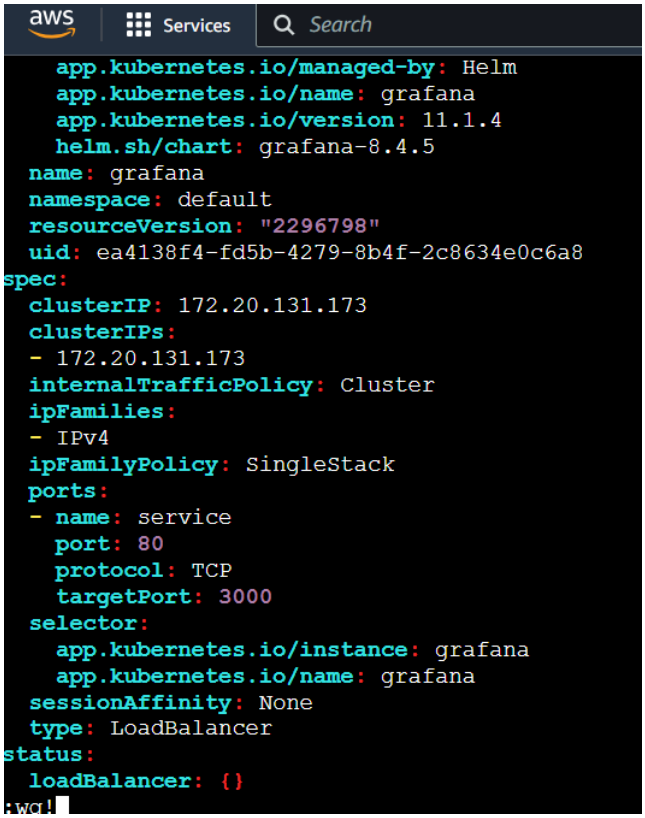
```
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana                               ClusterIP   172.20.131.173   <none>        80/TCP     5d6h
kubernetes                            ClusterIP   172.20.0.1       <none>        443/TCP    6d9h
prometheus-alertmanager               ClusterIP   172.20.246.8     <none>        9093/TCP   5d6h
prometheus-alertmanager-headless      ClusterIP   None             <none>        9093/TCP   5d6h
prometheus-kube-state-metrics         ClusterIP   172.20.24.217    <none>        8080/TCP   5d6h
prometheus-prometheus-node-exporter   ClusterIP   172.20.35.64     <none>        9100/TCP   5d6h
prometheus-prometheus-pushgateway     ClusterIP   172.20.179.123   <none>        9091/TCP   5d6h
prometheus-server                     ClusterIP   172.20.130.81    <none>        80/TCP     5d6h
ubuntu@ip-10-16-13-49:~$
```

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

1. For that, we need to change the Service type from ClusterType to LoadBalancer
2. Edit the stable-kube-prometheus-server service
3. kubectl edit svc stable-kube-prometheus-sta-prometheus



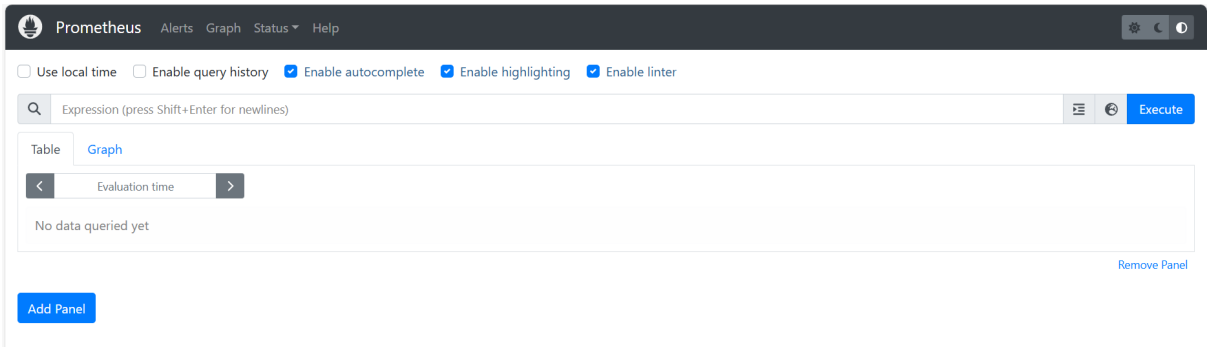
- Edit the stable-grafana service
- kubectl edit svc stable-grafana

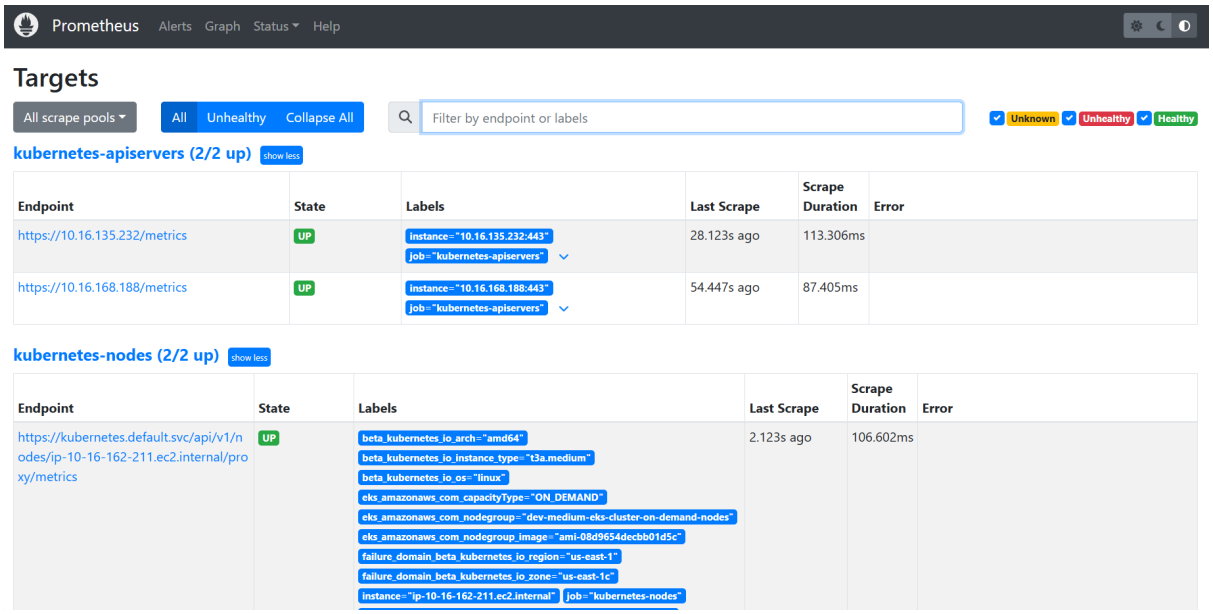


- Now, if you list again the service then, you will see the LoadBalancers DNS names

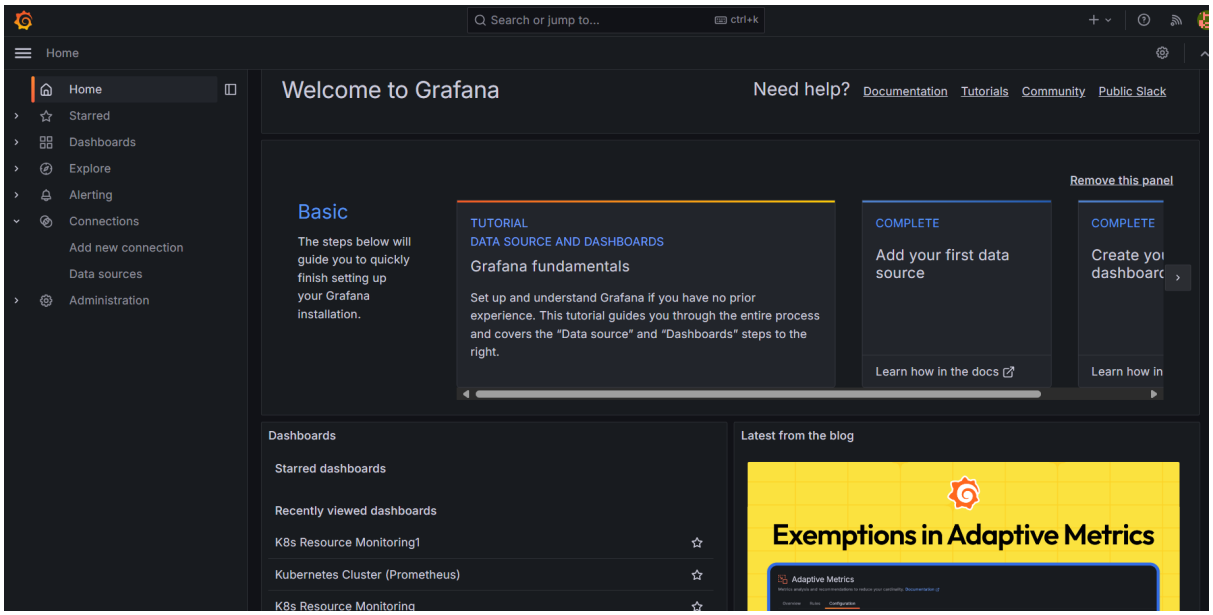
```
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME                                TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)
  AGE
grafana                             LoadBalancer   172.20.131.173   aea4138f4fd5b42798b4f2c8634e0c6a-1462404586.us-east-1.elb.amazonaws.com   80:30119/TCP
  5d6h
kubernetes                          ClusterIP      172.20.0.1       <none>                                                                    443/TCP
  6d9h
prometheus-alertmanager             ClusterIP      172.20.246.8     <none>                                                                    9093/TCP
  5d6h
prometheus-alertmanager-headless    ClusterIP      None             <none>                                                                    9093/TCP
  5d6h
prometheus-kube-state-metrics       ClusterIP      172.20.24.217    <none>                                                                    8080/TCP
  5d6h
prometheus-prometheus-node-exporter ClusterIP      172.20.35.64     <none>                                                                    9100/TCP
  5d6h
prometheus-prometheus-pushgateway   ClusterIP      172.20.179.123   <none>                                                                    9091/TCP
  5d6h
prometheus-server                   LoadBalancer   172.20.130.81    a5f57c4555e684ebd8238e531210e1e9-2124946716.us-east-1.elb.amazonaws.com   80:32333/TCP
  5d6h
```

- Now, access your Prometheus Dashboard using command:
- Paste the <Prometheus-LB-DNS>:9090 in your favorite browser and you will see like this
- Click on Status and select Target, You will see a lot of Targets

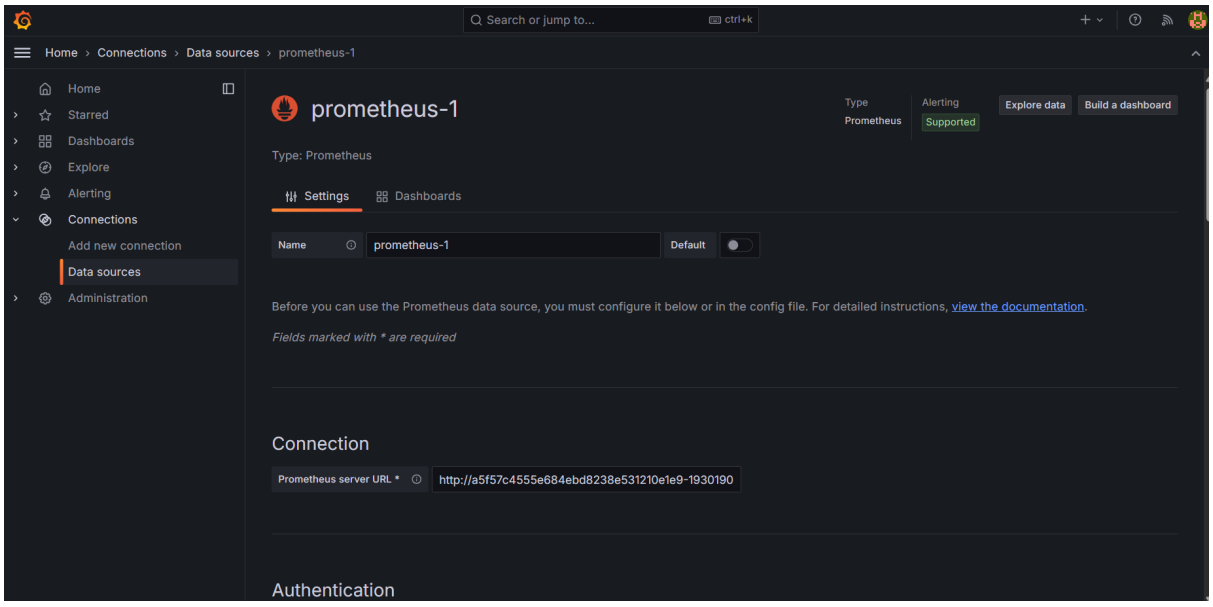


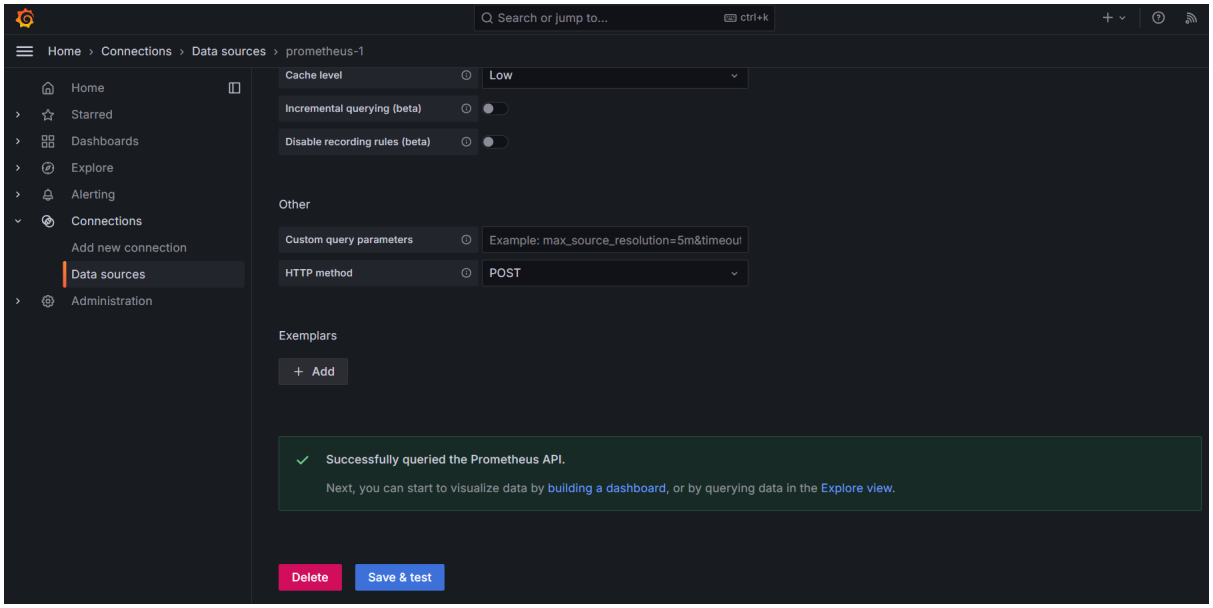


- Now, access your Grafana Dashboard
- Copy the ALB DNS of Grafana and paste it into your favorite browser.
- The username will be admin and the password will be prom-operator for your Grafana.



- Now, click on Data Source
- Select Prometheus
- In the Connection, paste your <Prometheus-LB-DNS>:9090

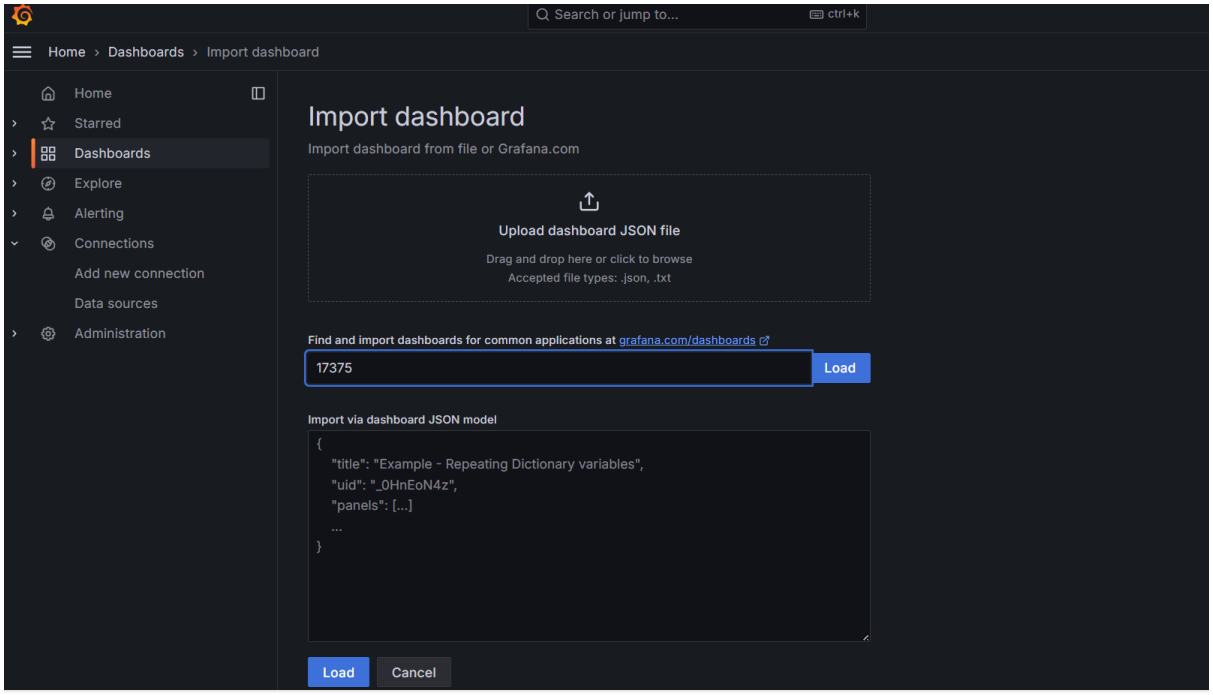


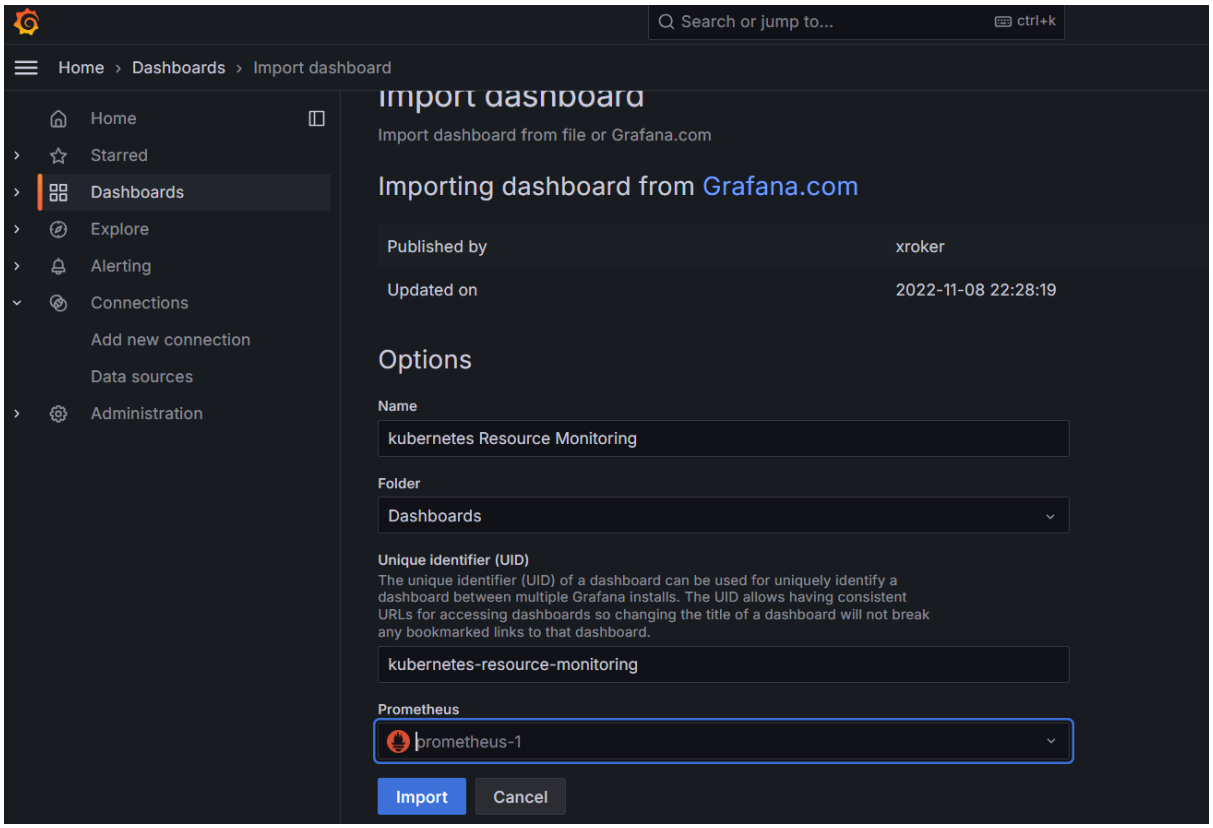


- If the URL is correct, then you will see a green notification
- Click on Save & test.

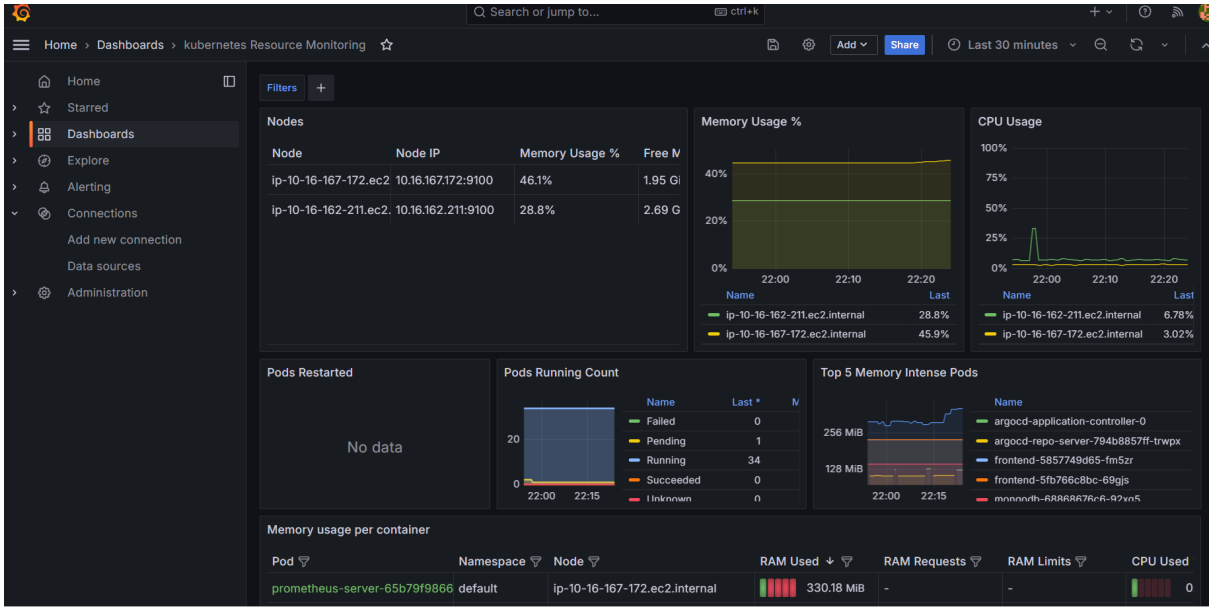
Now, we will create a dashboard to visualize our Kubernetes Cluster Logs

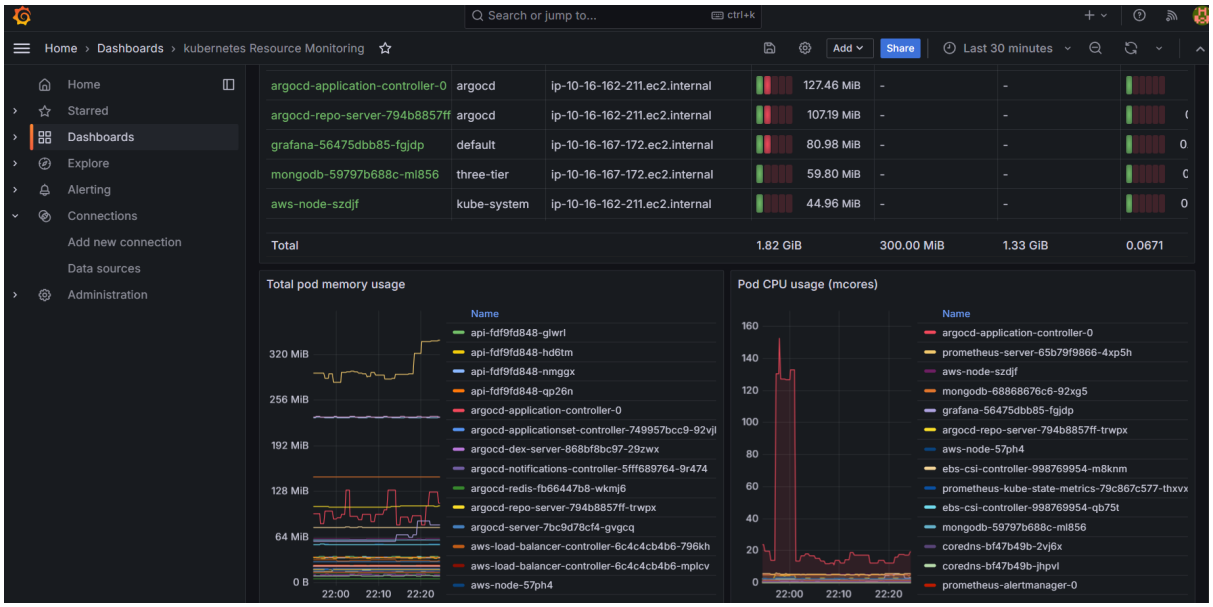
1. Click on Dashboard.
2. Once you click on Dashboard. You will see a lot of Kubernetes components monitoring.
3. Let’s try to import a type of Kubernetes Dashboard.
4. Click on New and select Import
5. Provide 17375 ID and click on Load
6. Note: 17315 is a unique ID from grafana which is used to Monitor and Visualize Kubernetes Data.


Search or jump to...
ctrl+k
Home › Dashboards › Import dashboard
Home
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
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": [...]
...
}
Load
Cancel


Search or jump to...
ctrl+k
Home › Dashboards › Import dashboard
Home
Starred
Dashboards
Explore
Alerting
Connections
Add new connection
Data sources
Administration
Import dashboard
Import dashboard from file or Grafana.com
Importing dashboard from Grafana.com
Published by
xroker
Updated on
2022-11-08 22:28:19
Options
Name
kubernetes Resource Monitoring
Folder
Dashboards
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.
kubernetes-resource-monitoring
Prometheus
prometheus-1
Import
Cancel

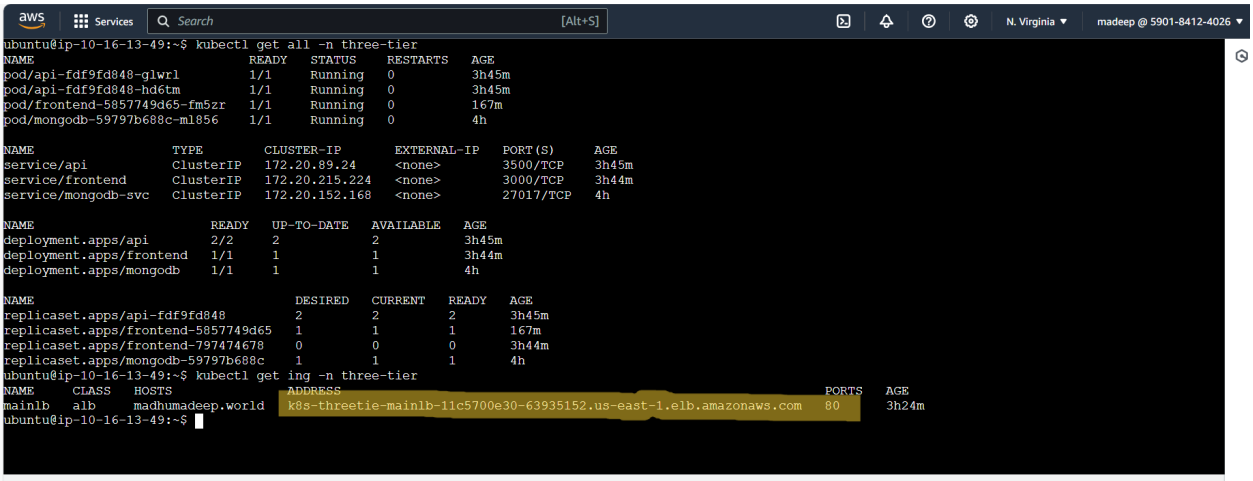
- You can view your Kubernetes Cluster Data.



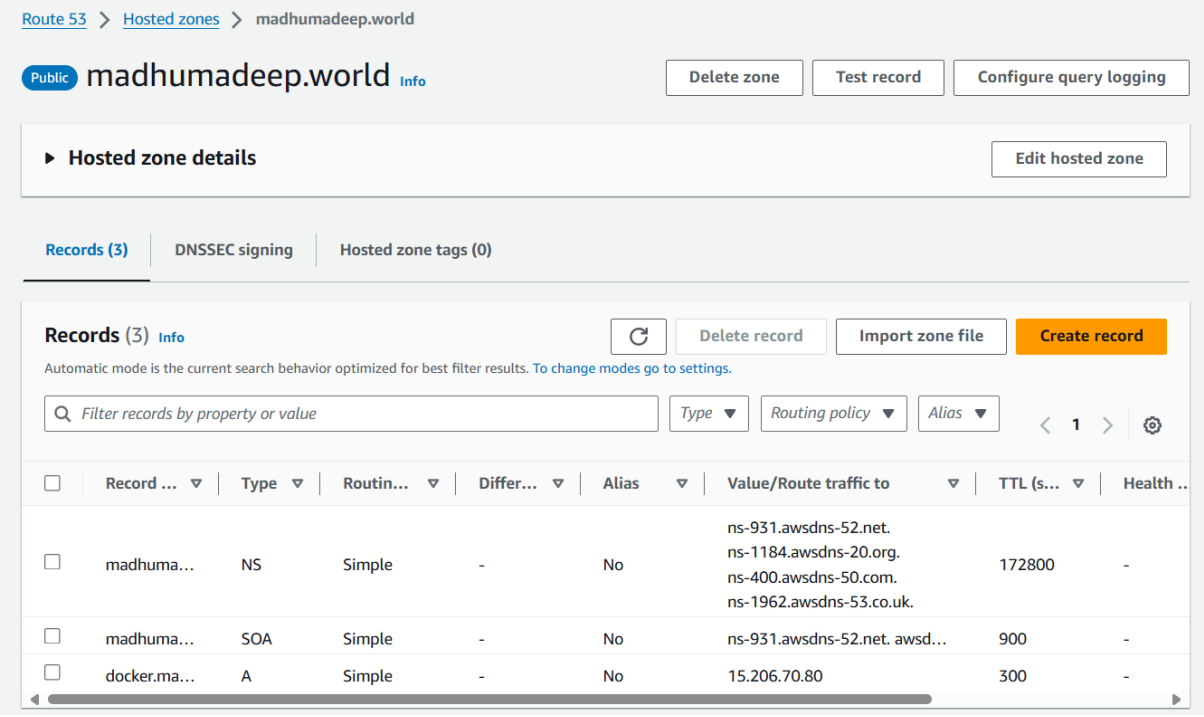


Step-14: Create the DNS record for the ALB

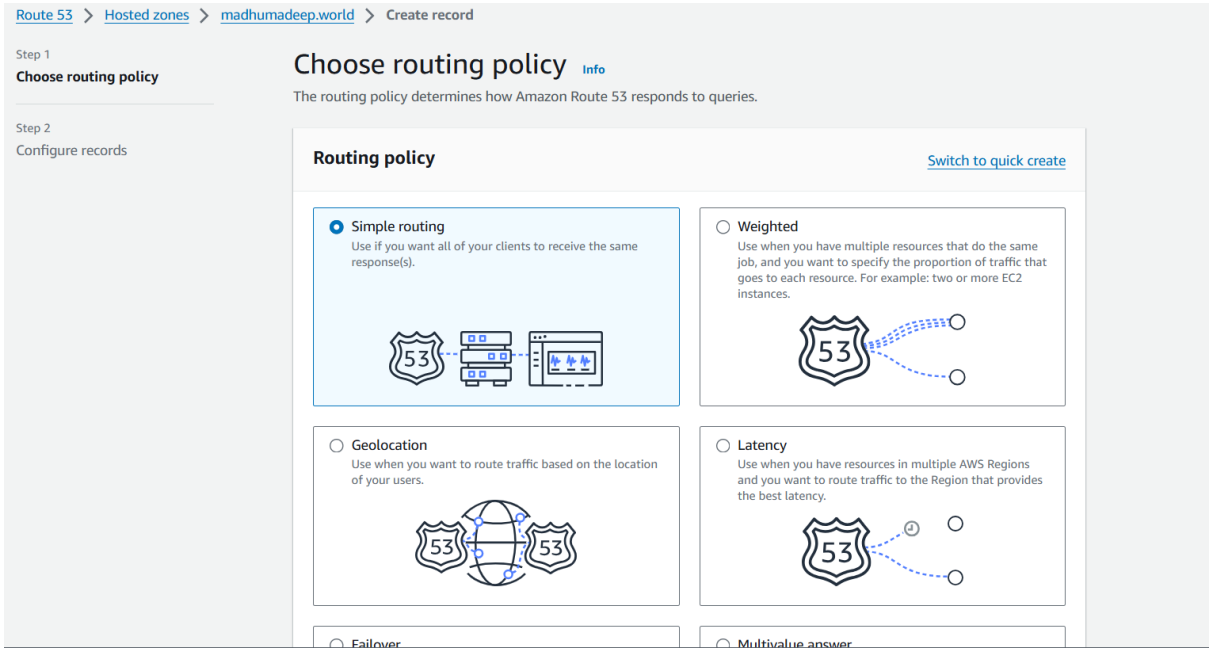
- Once your Ingress application is deployed. It will create an Application Load Balancer
- You can check out the load balancer named with k8s-three.

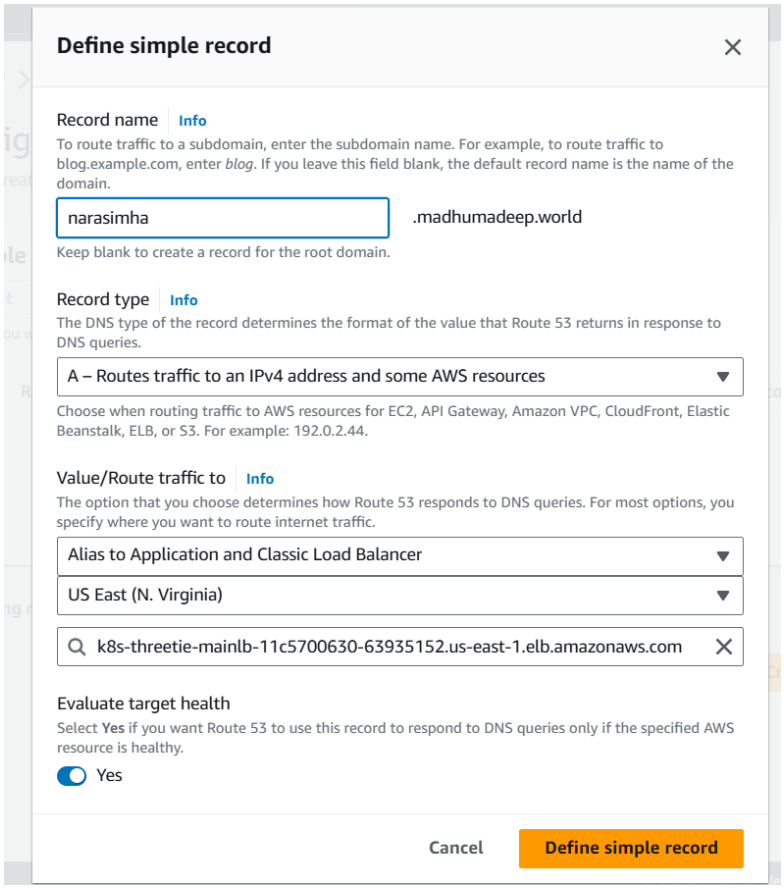


- Now, Copy the ALB-DNS and go to your Domain Provider in my case GODaddy is the domain provider.



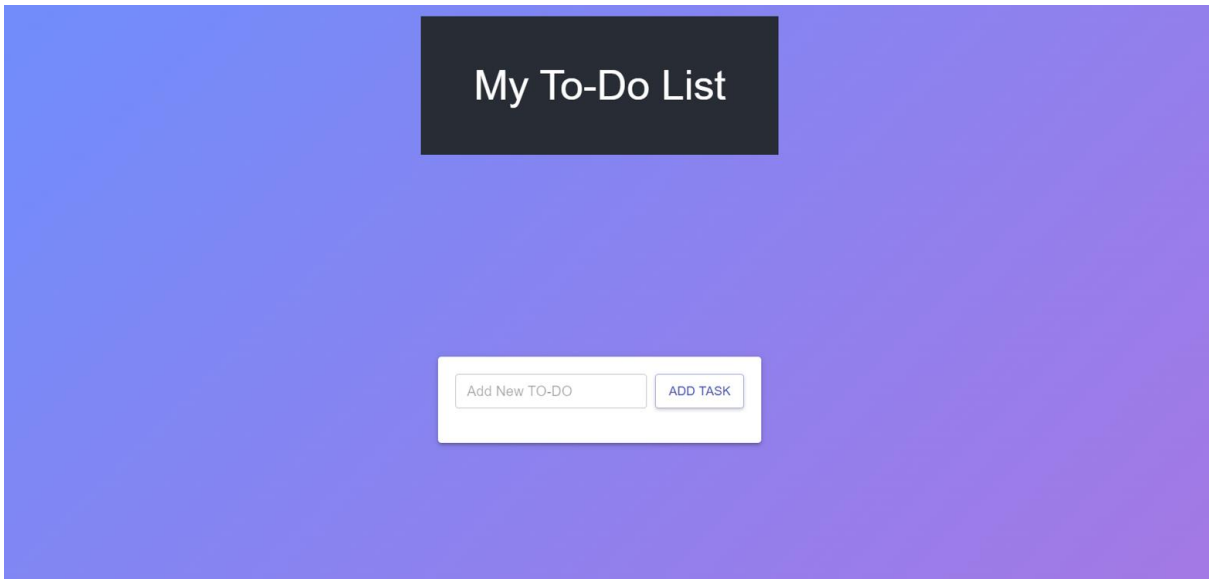
- Choose Simple Routing



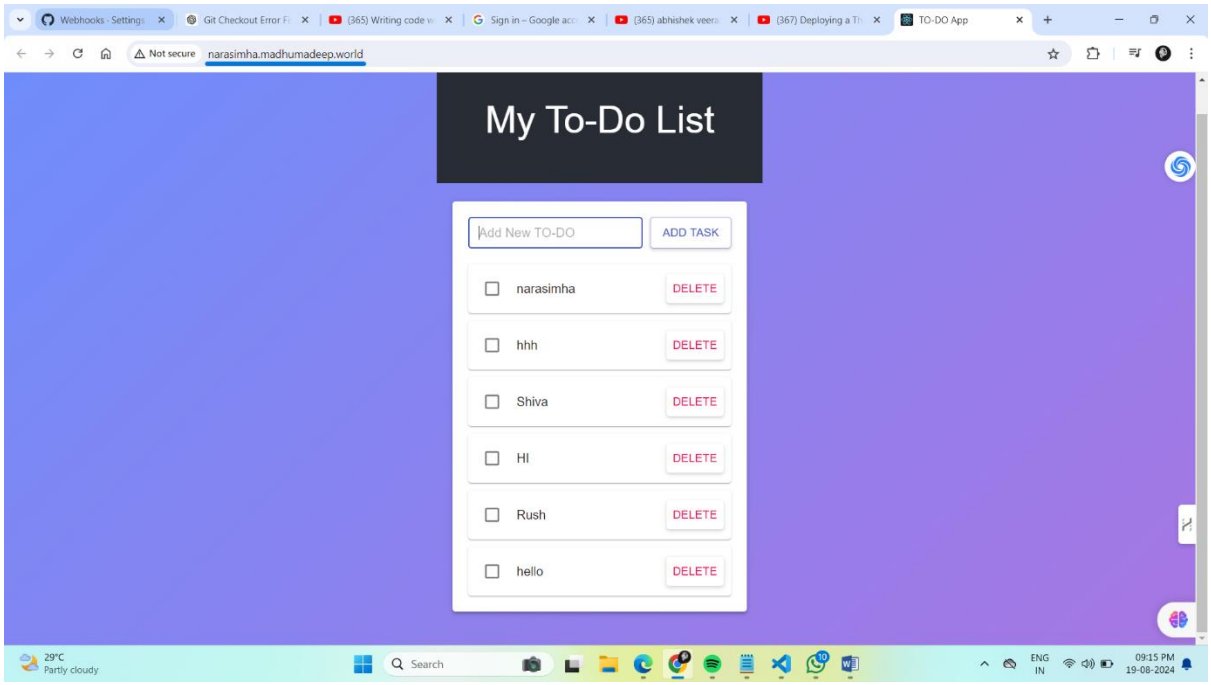


- Click on Define a simple record.

- Now, hit your subdomain after 2 to 3 minutes in your browser to see the Application.

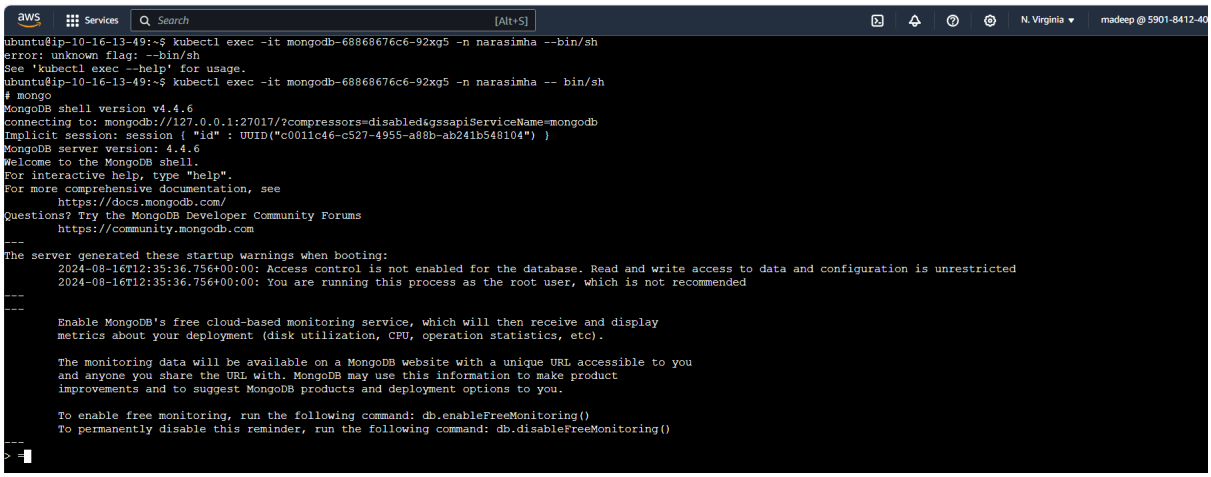


- Now test the application by adding some data

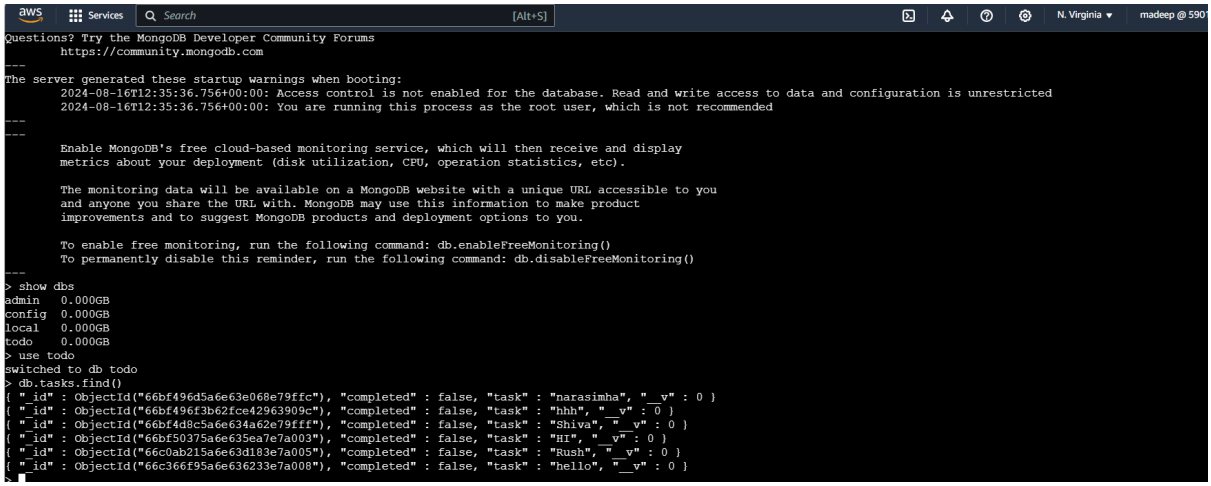


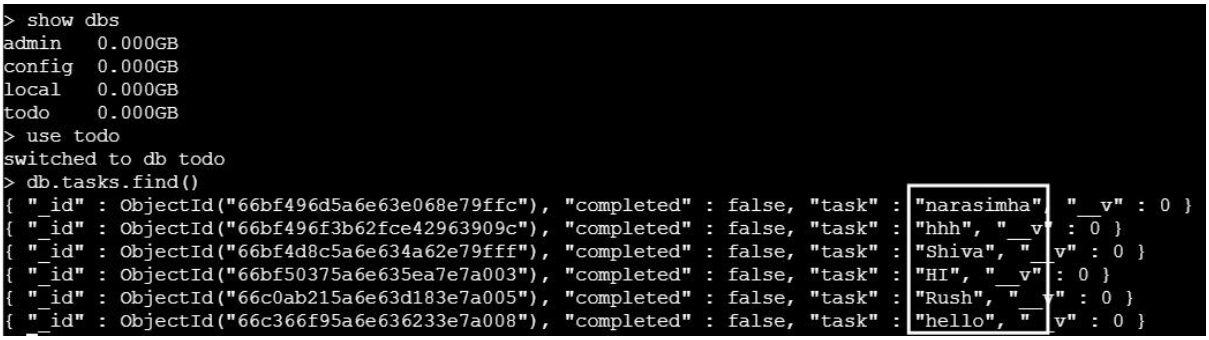
We can check the data is stored in the database (mongodb container) using commands:

1. Using the command “exec –it <mongobd-container-name> -n <namespace> -- bin/sh



2. Now we are in the Mongdb database
3. Now use the command “show dbs” it will give’s the list of databases that you are created
4. And select the db using the command “use<db-name>” in my case db name is todo so we use the command as “use todo”
5. Then it will shows the data that is stored in the database





6. You can see the data that we have entered in the application.

GitHub Repo : https://github.com/Narasimha76/three-tier-appliaction.git