Deploying a Three-Tier Application on Amazon EKS Along with Prometheus, Grafana, and EFK Logging

This document outlines the deployment of a three-tier application on Amazon Elastic Kubernetes Service (EKS), incorporating robust monitoring and logging solutions. The architecture consists of:

- **Presentation Tier:** Includes frontend web servers and load balancers to interact with end-users.
- **Application Tier:** Comprises application servers or microservices for business logic and processing.
- **Data Tier:** Houses databases and storage systems for persistent data storage.

Monitoring with Prometheus and Grafana:

Prometheus, deployed as a Kubernetes Deployment and Service, collects and stores metrics from application components. Grafana, integrated with Prometheus, provides visualization and monitoring dashboards to track system health and performance metrics effectively.

Logging Support with EFK Stack:

The ElasticSearch, Fluentbit, Kibana (EFK) stack handles logging. ElasticSearch stores logs collected by Fluentbit, which is deployed as a DaemonSet to gather logs from Kubernetes pods. Kibana offers a web interface for log visualization, allowing detailed analysis and troubleshooting.

Integration with Amazon EKS:

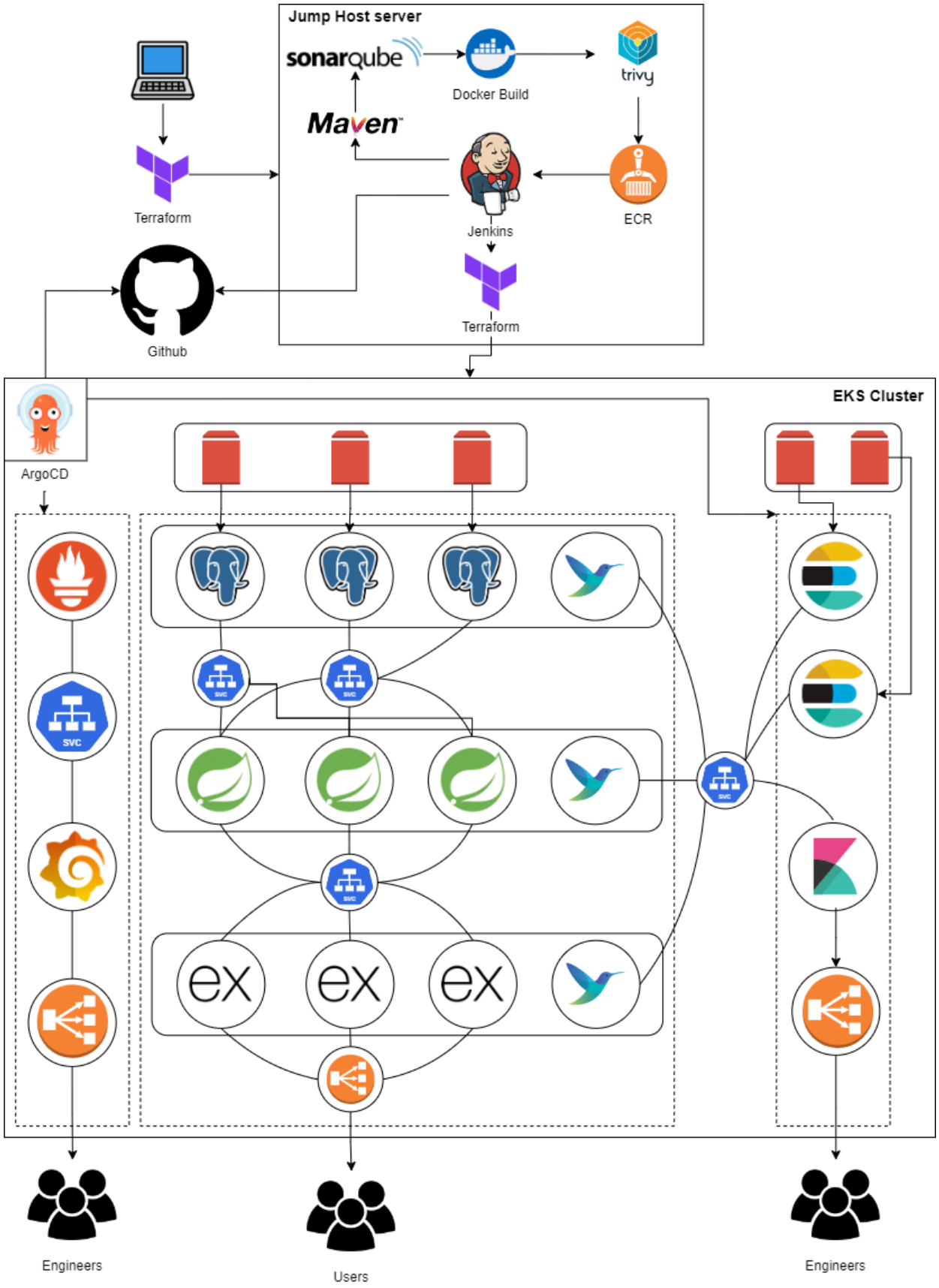
Amazon EKS facilitates scalability and elasticity by managing Kubernetes clusters efficiently. It integrates seamlessly with AWS services like IAM, VPC, enhancing security, networking, and additional monitoring capabilities.

This deployment architecture ensures scalability, resilience, and effective monitoring and logging for modern cloud-native applications on Amazon EKS.

Getting the source code:

Github link: https://github.com/saikiranpaila/blockslicer.git

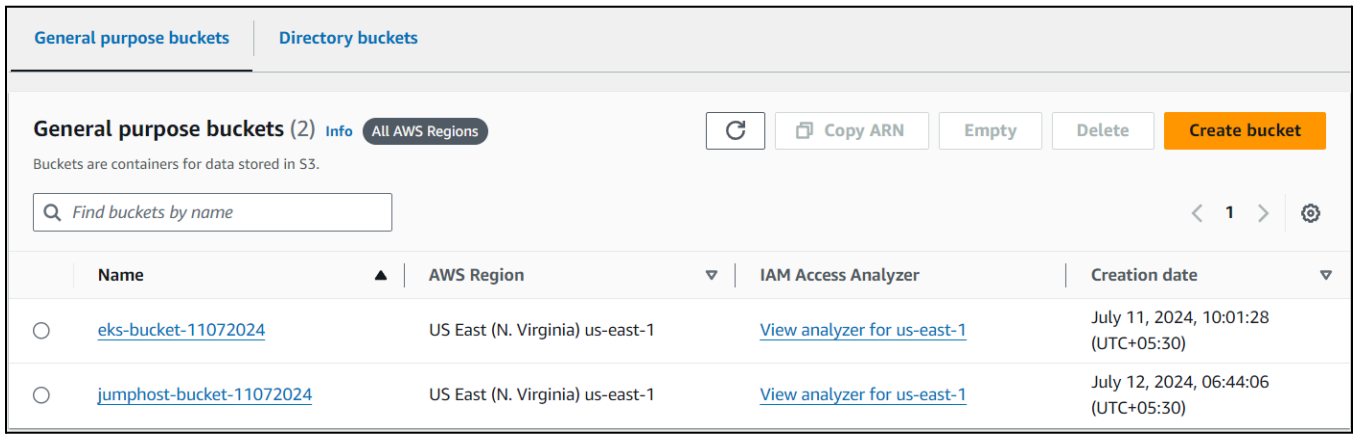
Architecture



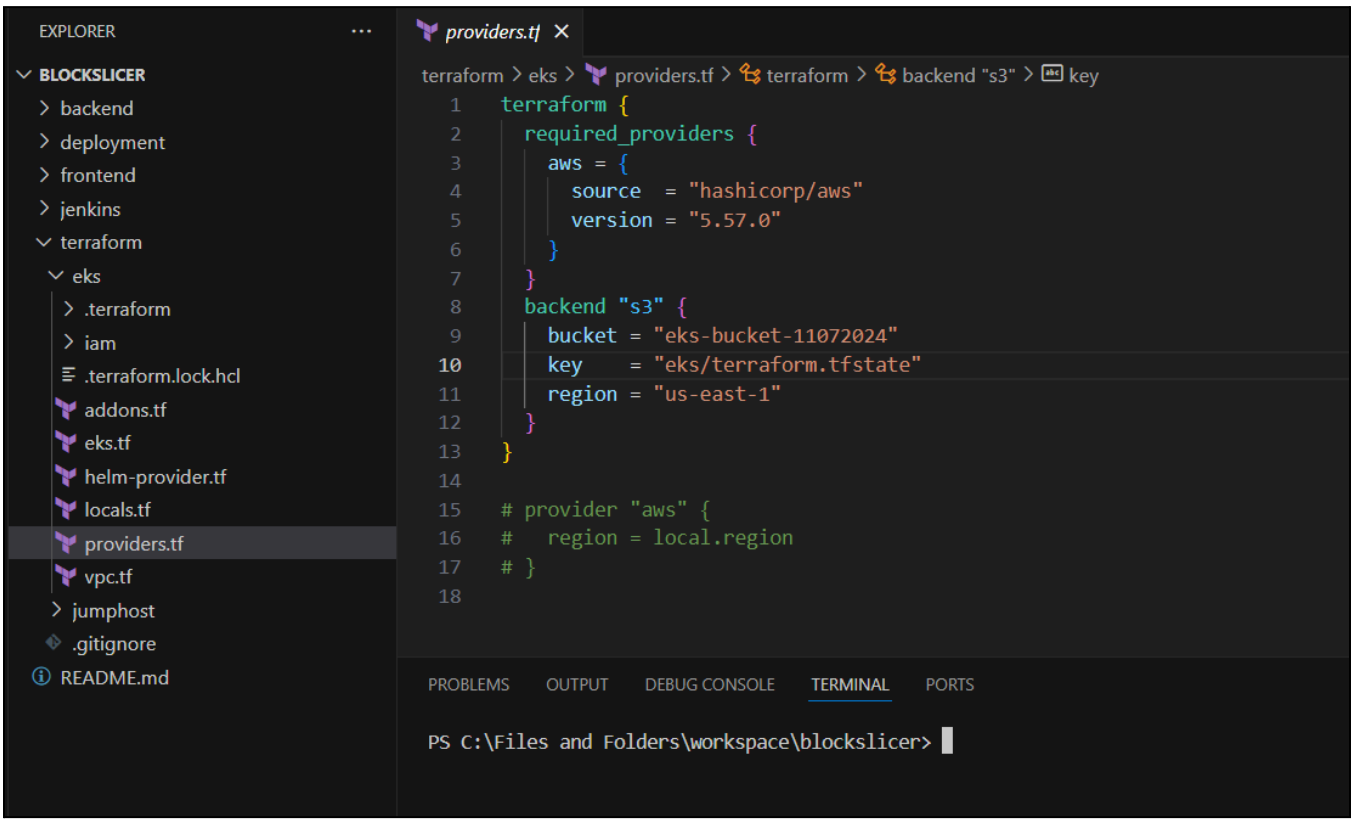
Configuring the Source Code

Step 1: Fork the github repo and clone it to your local machine.
github link (https://github.com/saikiranpaila/blockslicer).

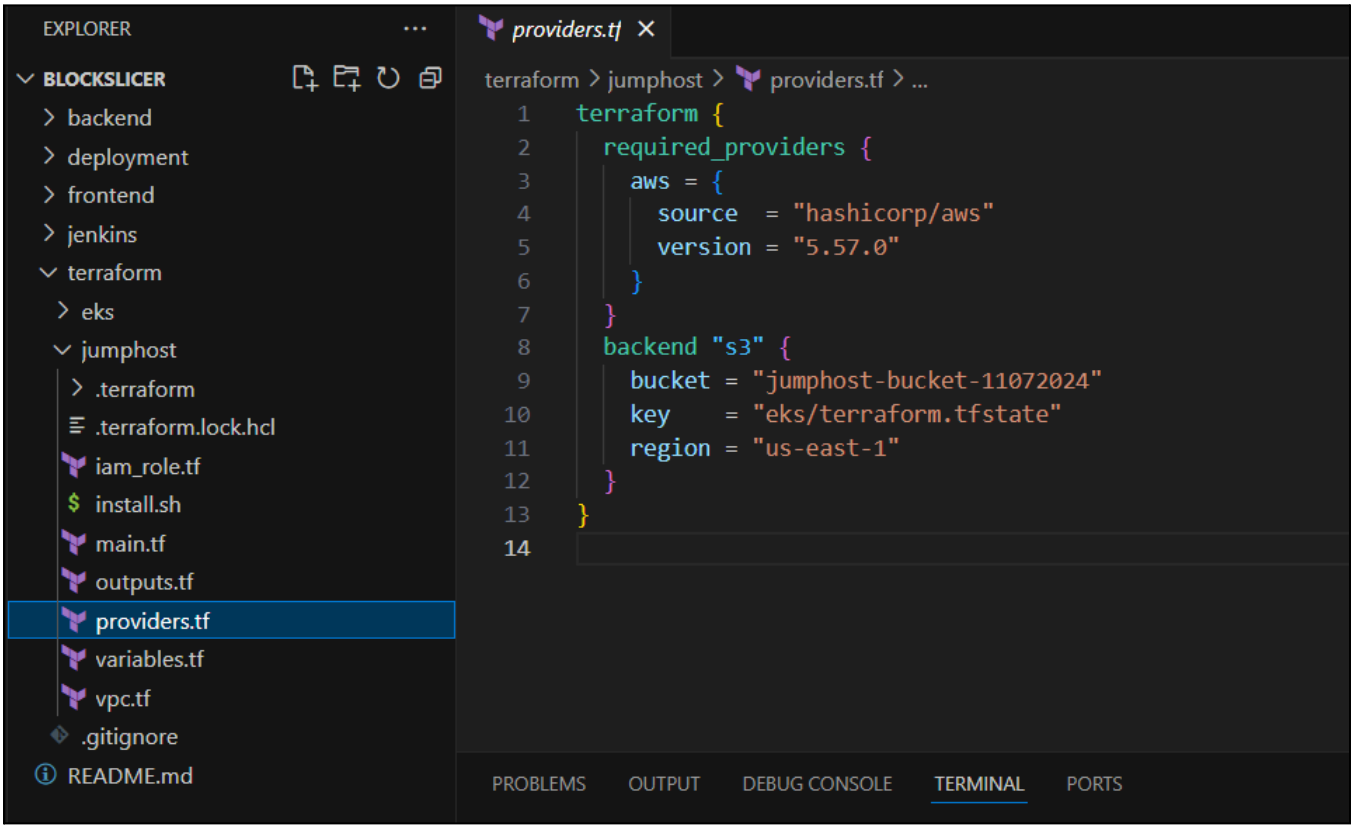
Step 2: Create two buckets in aws for saving terraform state files.



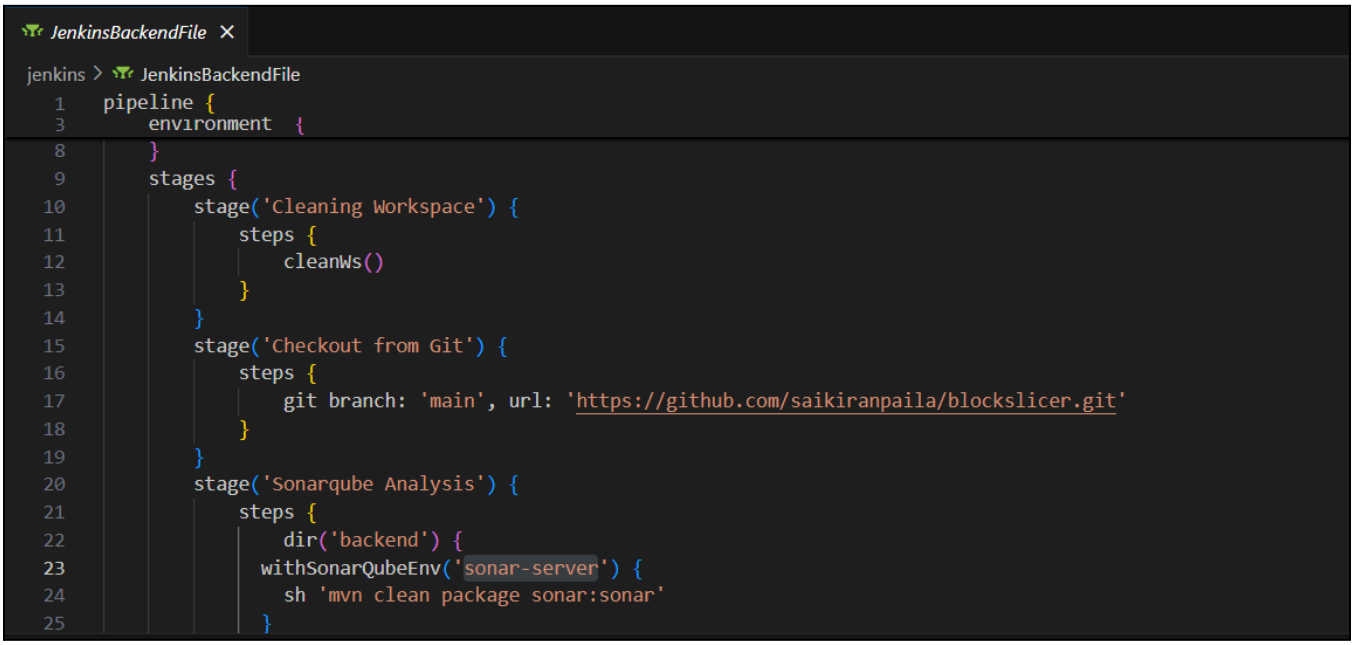
Step 3: Go ./blockslicer/terraform/jumphost/providers.tf and change the bucket name and region with the name of the bucket that you created.



Step 4: Do the same for ./blockslicer/terraform/eks/providers.tf and change the bucket name and region to the other bucket name.



Step 5: Now go to ./blockslicer/jenkins/JenkinsBackendFile and replace the git url with your github url.



Step 6: Do the same for ./blockslicer/jenkins/JenkinsFrontendFile and replace the git url with your github url.

JenkinsFrontendFile

jenkins > JenkinsFrontendFile

```
pipeline {
    stages {
        stage('Cleaning Workspace') {
            steps {
                cleanWs()
            }
        }
        stage('Checkout from Git') {
            steps {
                git branch: 'main', url: 'https://github.com/saikiranpaila/blockslicer.git'
            }
        }
        stage('Sonarqube Analysis') {
            steps {
                dir('frontend') {
                    withSonarQubeEnv('sonar-server') {
                        sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectKey=frontend -Dsonar.sources=. '''
                    }
                }
```

Step 7: Now push the changes back to the remote.

Deploying the Jump Host Server:

Step 1: Ensure that terraform is installed on your machine and aws cli with configured keys, by running.

- terraform version
- aws configure

Step 2: Now go to ./blockslicer/terraform/jumphost/ and run the following commands.

- terraform init
- terraform validate
- terraform init
- terraform plan
- terraform apply --auto-approve

```
aws_security_group.security-group: Creation complete after 5s [id=sg-0b57defa38cf97713]
aws_subnet.public-subnet: Still creating... [10s elapsed]
aws_subnet.public-subnet: Creation complete after 13s [id=subnet-0cb953a43e90ee3eb]
aws_route_table_association.rt-association: Creating...
aws_instance.ec2: Creating...
aws_route_table_association.rt-association: Creation complete after 1s [id=rtbassoc-031662584efa86dad]
aws_instance.ec2: Still creating... [10s elapsed]
aws_instance.ec2: Creation complete after 16s [id=i-0d9f2326b77e83d11]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

Outputs:

jumphost_public_ip = "44.201.22.48"
region = "us-east-1"
PS C:\Files and Folders\workspace\blockslicer\terraform\jumphost>
```

Step 3: Copy the “jumphost_public_ip” and ssh into the server.

Step 4: Check whether all tools are installed on the server by running the following commands.

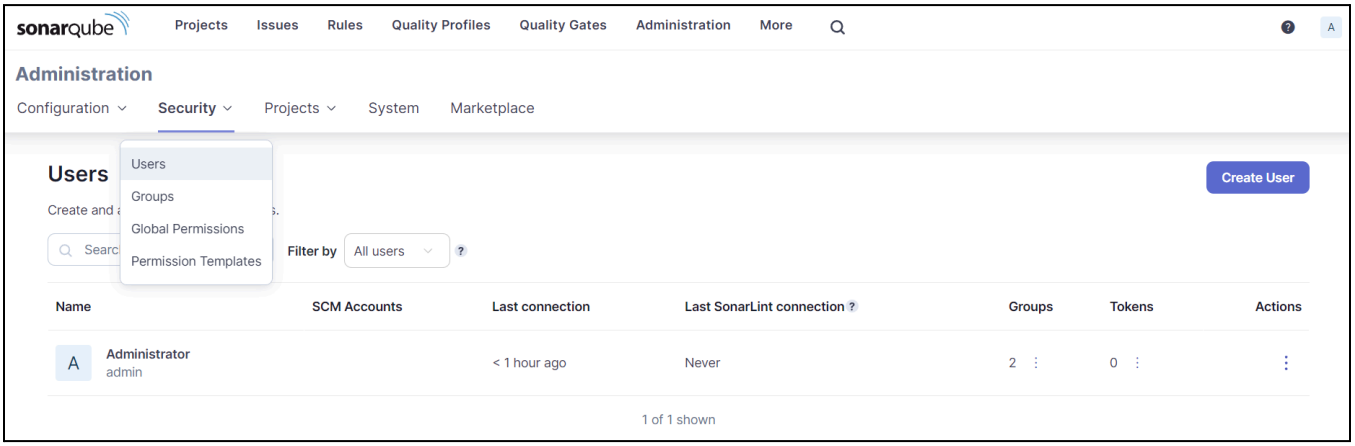
- git version
- systemctl status jenkins
- terraform version
- mvn --version
- systemctl status docker
- kubectl version
- helm version
- trivy version
- docker ps (sonarqube up and running)

Step 5: If everything is working properly then it's good to go.

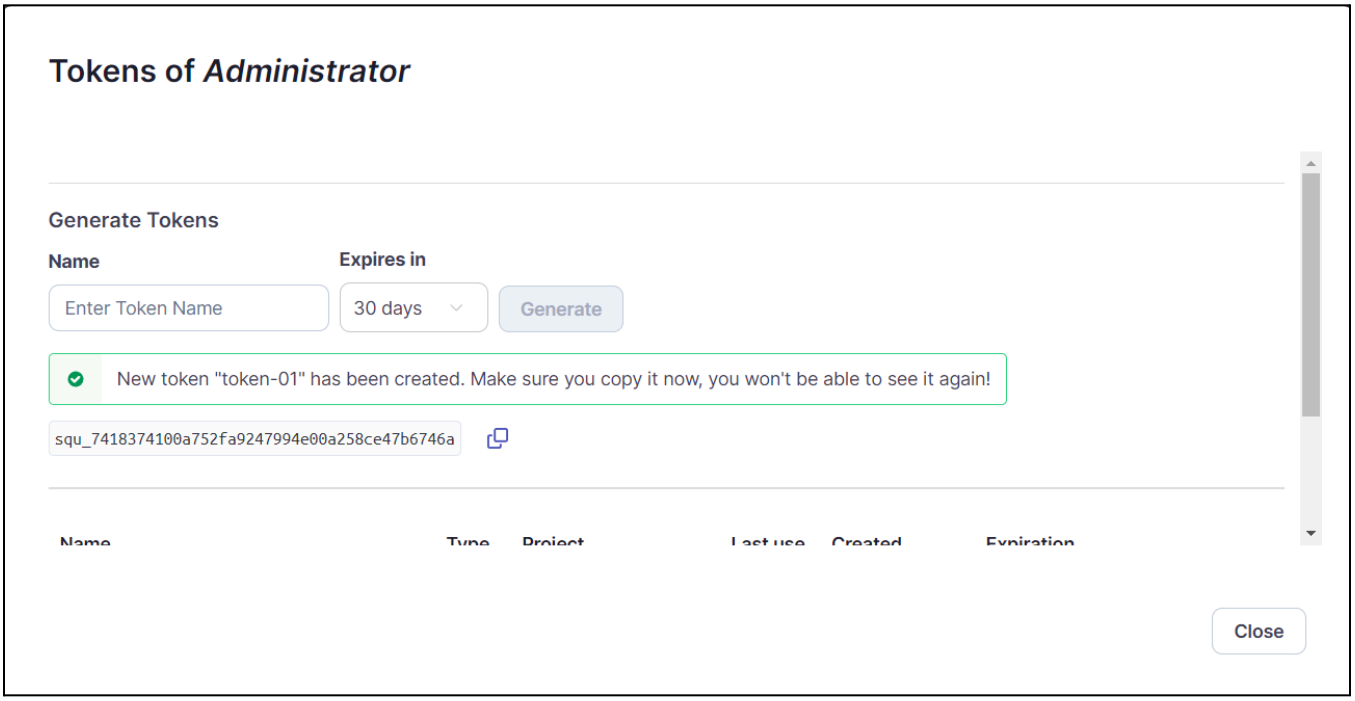
Configuring the Jenkins Server and Sonarqube.

Step 1: Take the public ip of the jump host server and hit it with port 9000 to access sonarqube. The default credentials of sonarqube are username: **admin** and password: **admin.**

Step 2: Now under Administration->security->users->tokens.



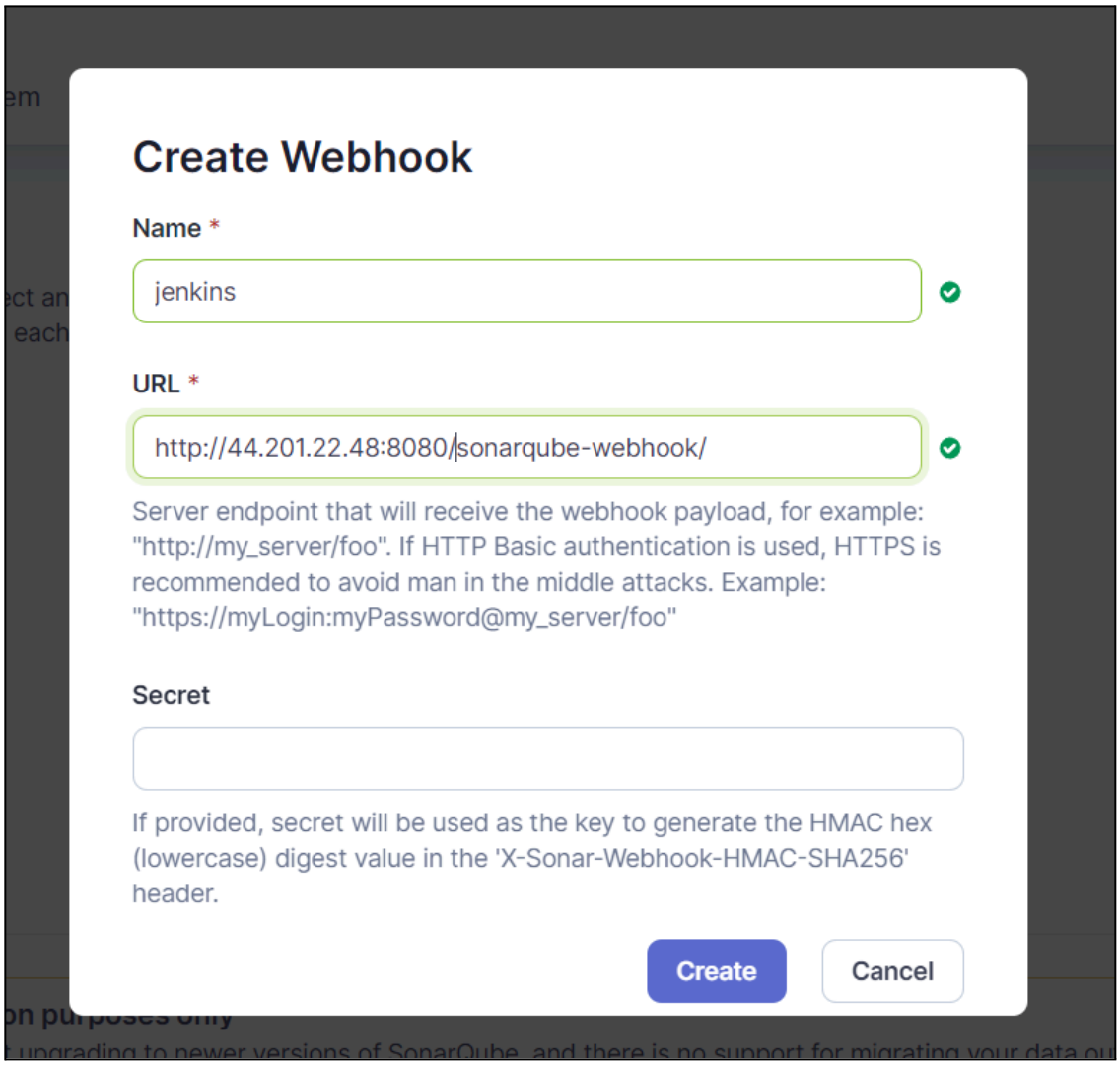
Step 3: Give a name for that token and click generate after generating copy and store the token in some place we will use it later.



Step 4: Now configure the webhook for sonarqube to return code quality status to jenkins server, by going to **configuration->webhooks,** here give some name to the webhook and

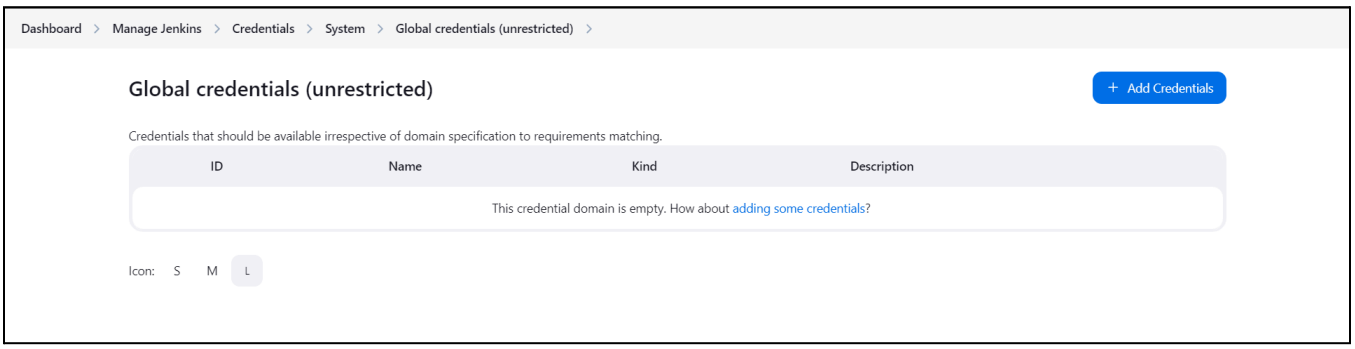
add jenkins server url in the format of

http://<public_ip_jumphost_server>:8080/sonarqube-webhook/

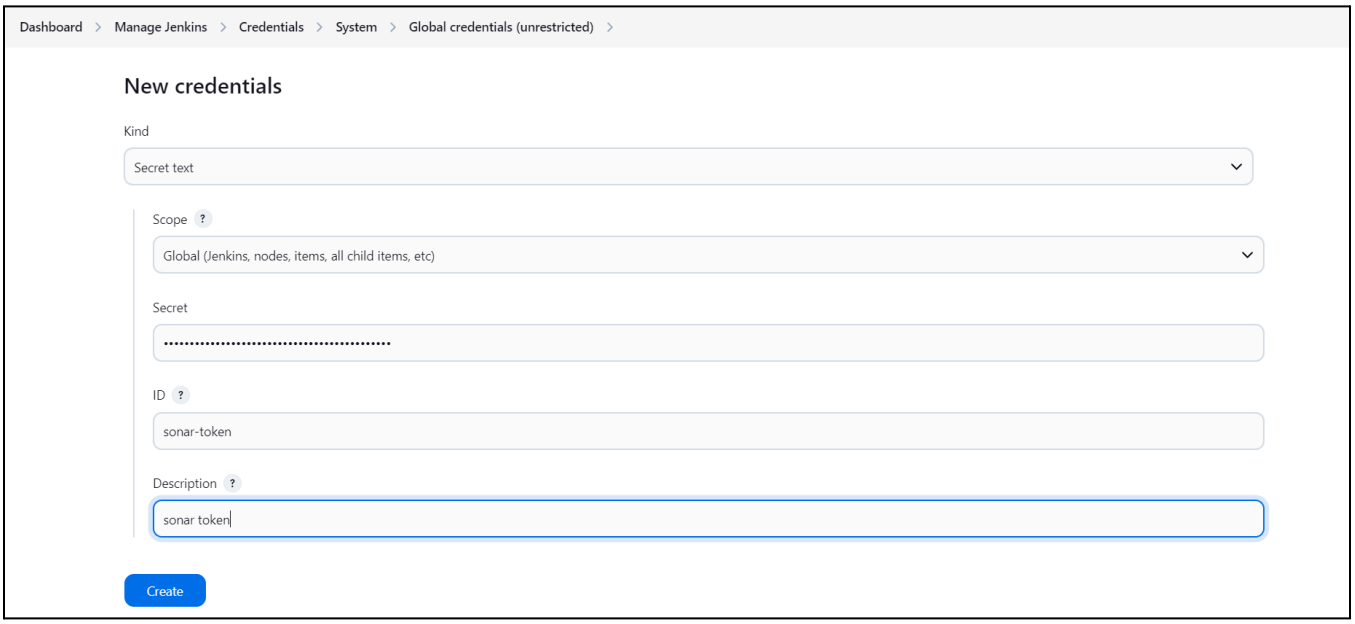


Step 5: Now hit the public ip of the jump host server with port 8080 to access jenkins dashboard.

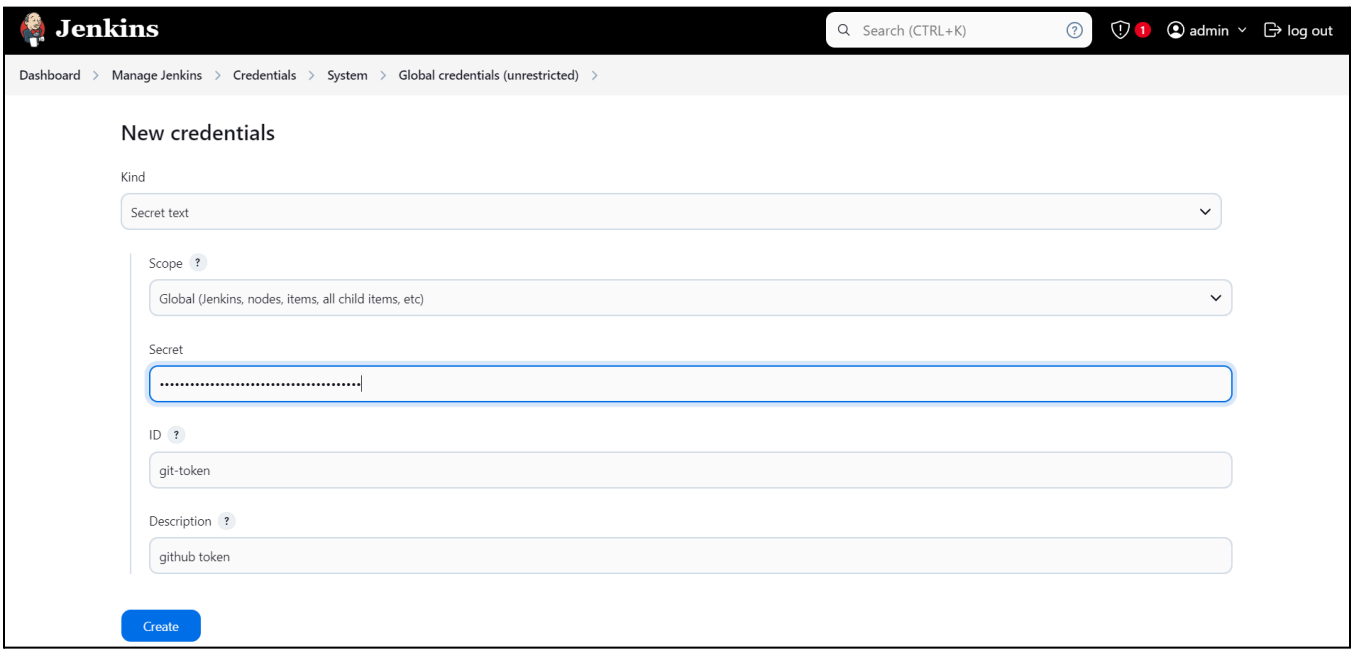
Step 6: Go to **Manage Jenkins > Credentials > System > Global Credentials** and click on **Add Credentials.**



Step 7: Select **Secret Text** paste sonarqube token in secret section and set its id to **sonar-token.**

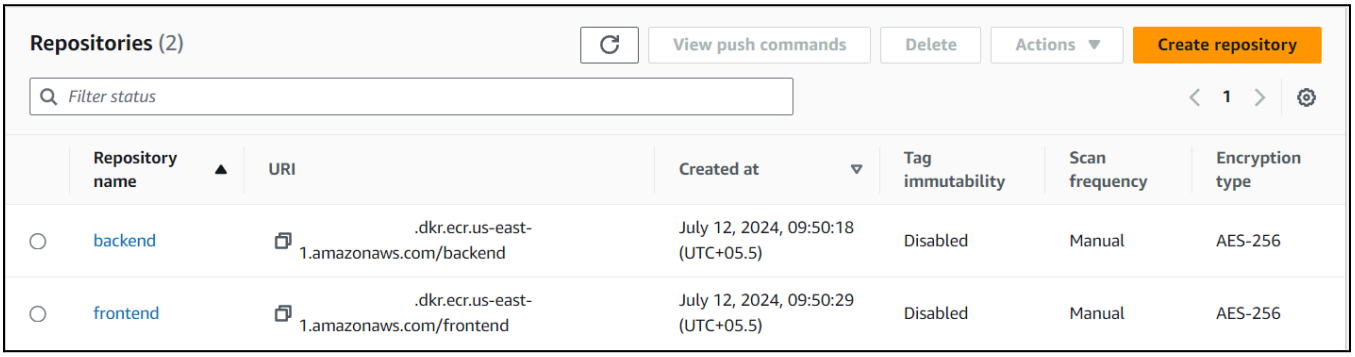


Step 8: Like previous step add github generate token (retrieved from settings > developer settings > personal access token > classic token).

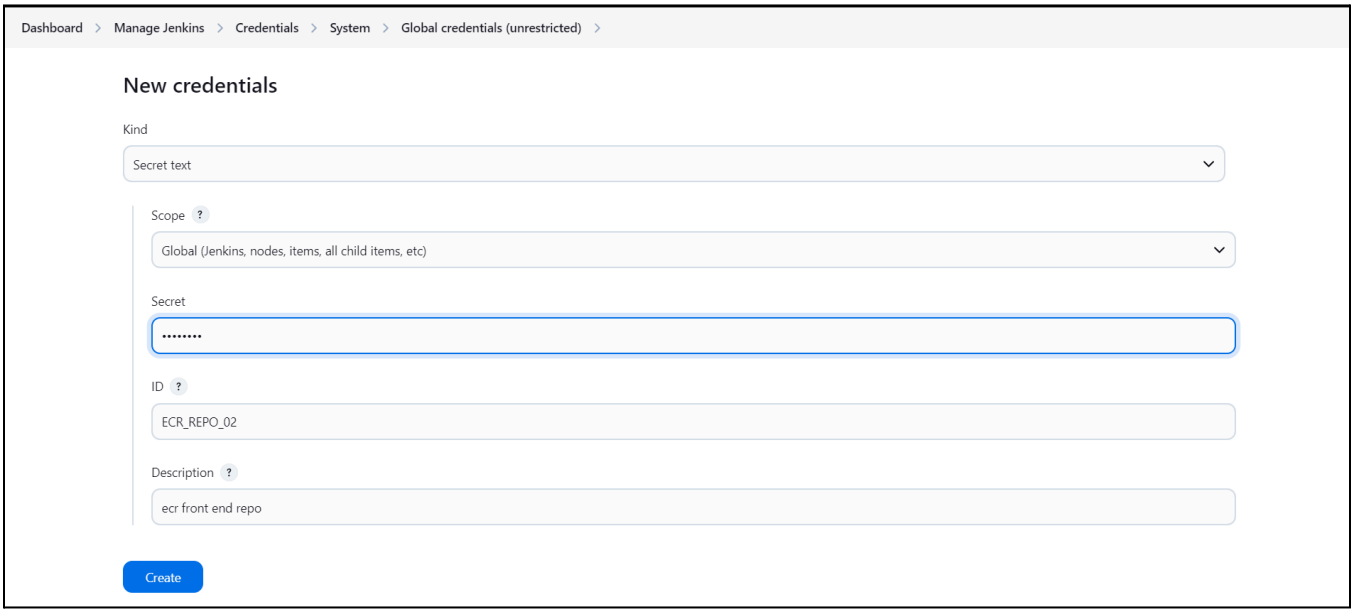


Step 9: Same step for adding AWS Account ID as secret which you can get from aws console.

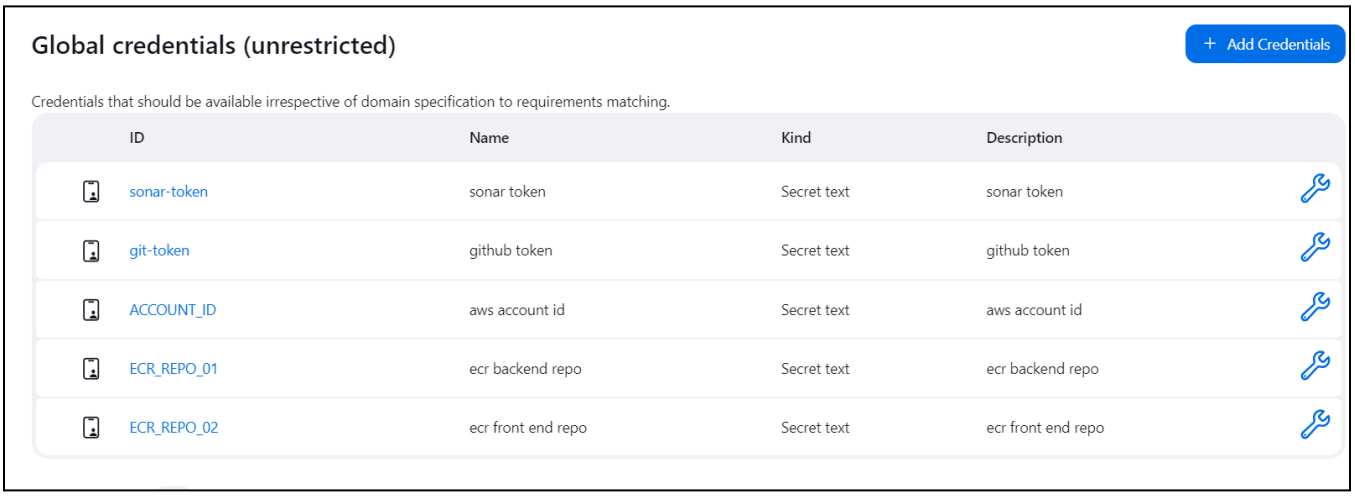
Step 10: Now head to ECR service in AWS console and create two private ECR private repositories with the name backend and frontend.



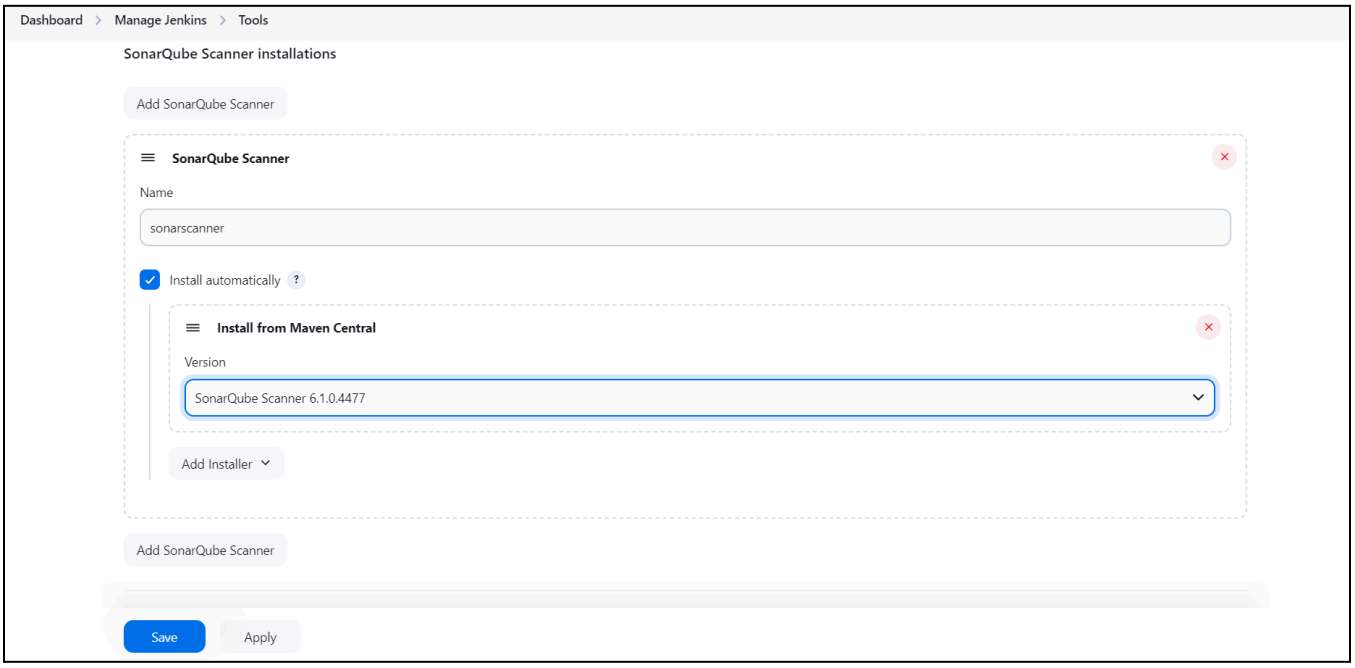
Step 11: Add these repository names as secret text in jenkins like Step 7, with **ECR_REPO_01** id for backend and **ECR_REPO_02** for frontend.



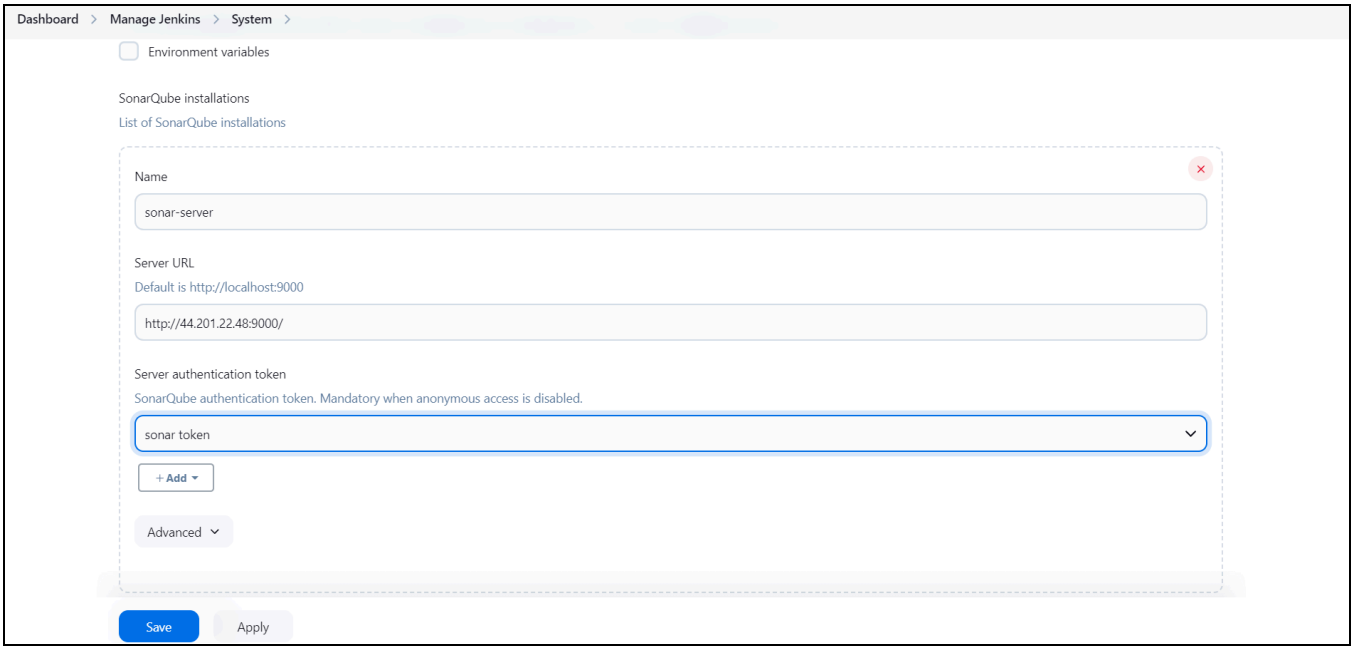
Step 12: Check whether your Global Security credentials match with the picture given below.



Step 13: Now go to Manage Jenkins > Tools > SonarQube Scanner Installation add with the name sonarscanner click apply and save.

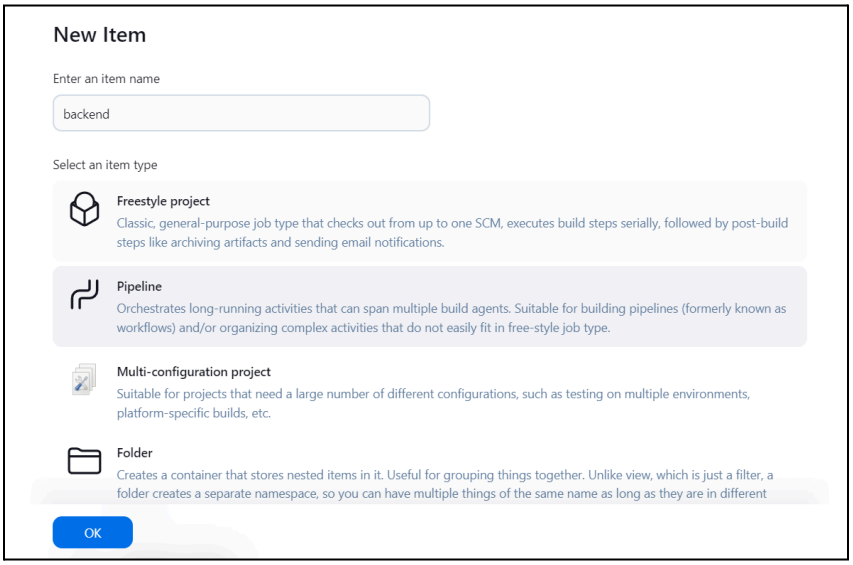


Step 14: For jenkins to communicate with sonarqube add the url of sonar to jenkins in **Manage Jenkins > System > SonarQube** Installations with the name **sonar-server** and select the **sonar-token** from the drop down for Server authentication token.

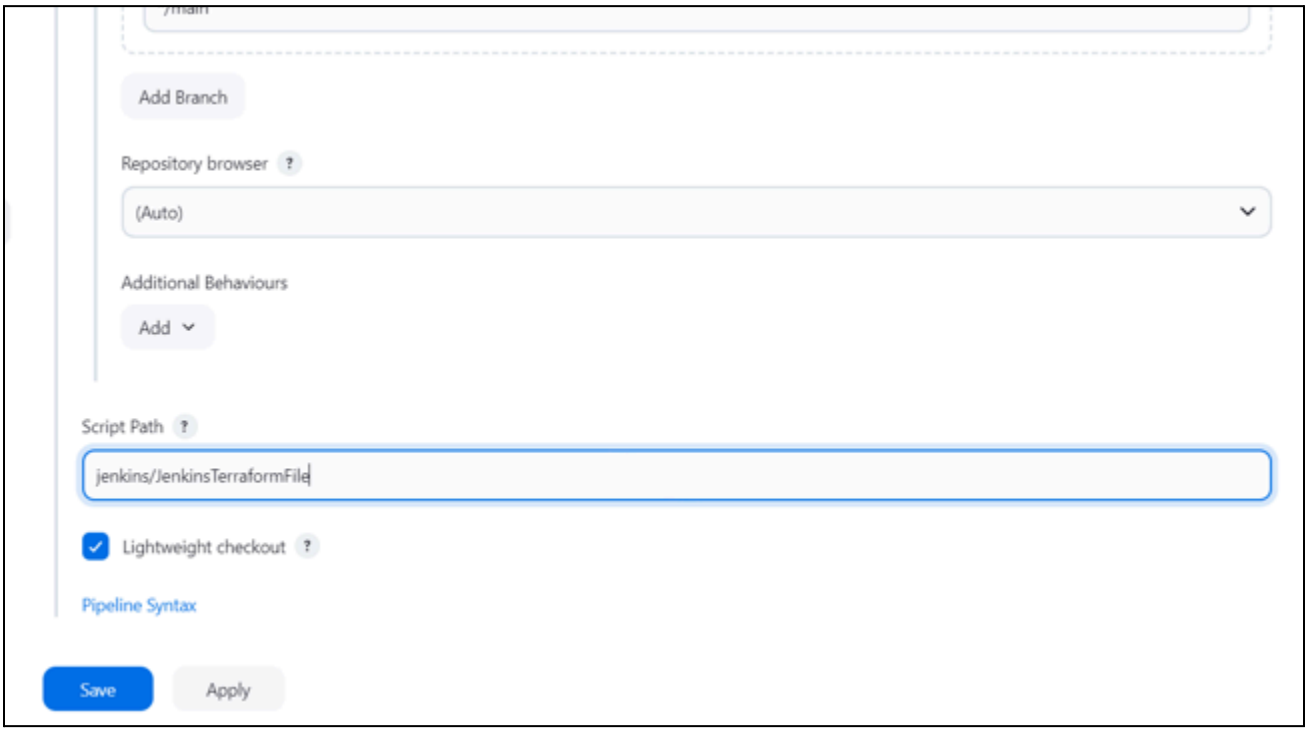

Dashboard > Manage Jenkins > System >
Environment variables
SonarQube installations
List of SonarQube installations
Name
sonar-server
Server URL
Default is http://localhost:9000
http://44.201.22.48:9000/
Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.
sonar token
+ Add
Advanced
Save
Apply

Jenkins Pipeline for EKS, backend and frontend.

Step 1: On jenkins dashboard click on **New Item** give it a name as **EKS,** select pipeline script and click on **OK.**

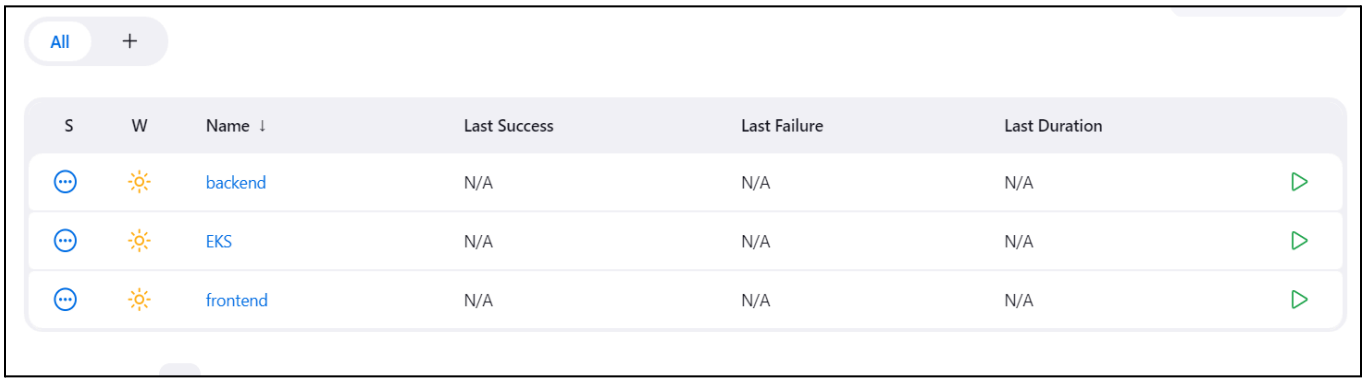


Step 2: Now Scroll down and select **Pipeline script from SCM,** paste the github url of the project and change branch to **main** and for script path field give jenkins/JenkinsTerraformFile.

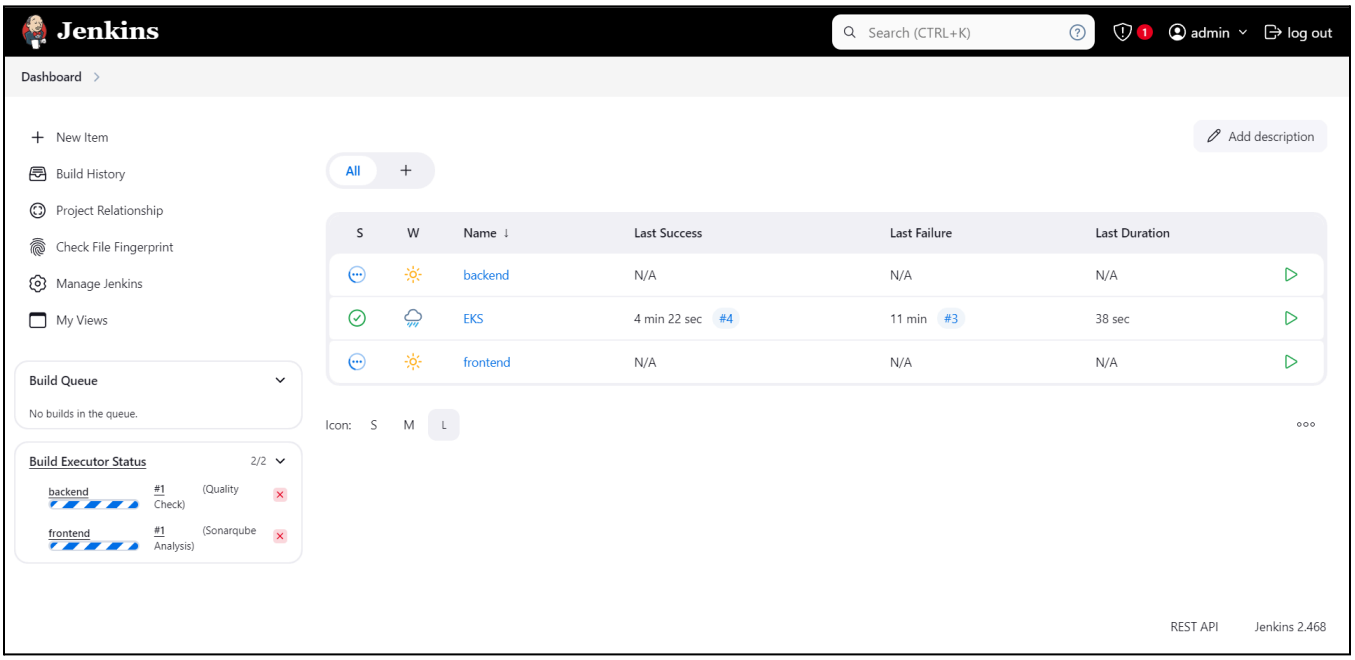


Step 3: Click on Apply and Save, Start building the EKS infrastructure by clicking on build with parameters and the parameter set to **'apply'. (It takes around 23-25 minutes to build meanwhile we will also configure the build pipeline for frontend and backend)**

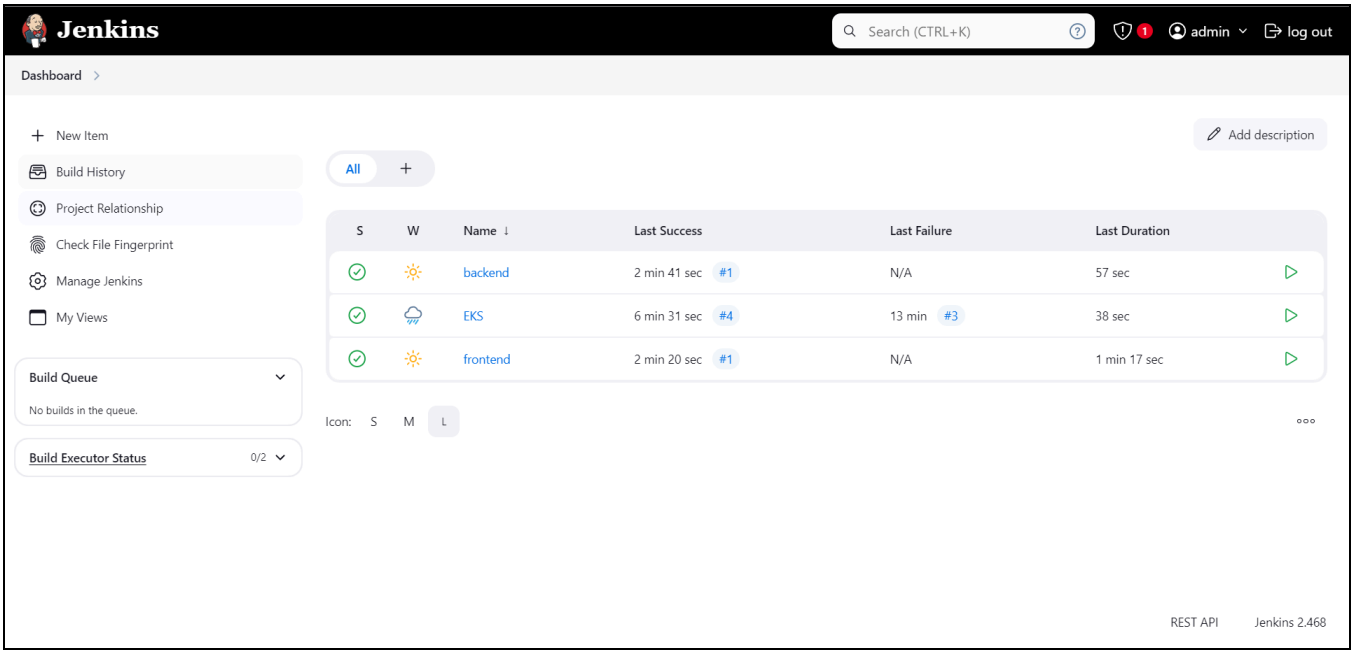
Step 4: Following Step 1 and Step 2 for creating the pipeline for backend and frontend with the pipeline name set to **backend** and **frontend** respectively and for script path **jenkins/JenkinsBackendFile** and **jenkins/JenkinsFrontendFile** respectively.



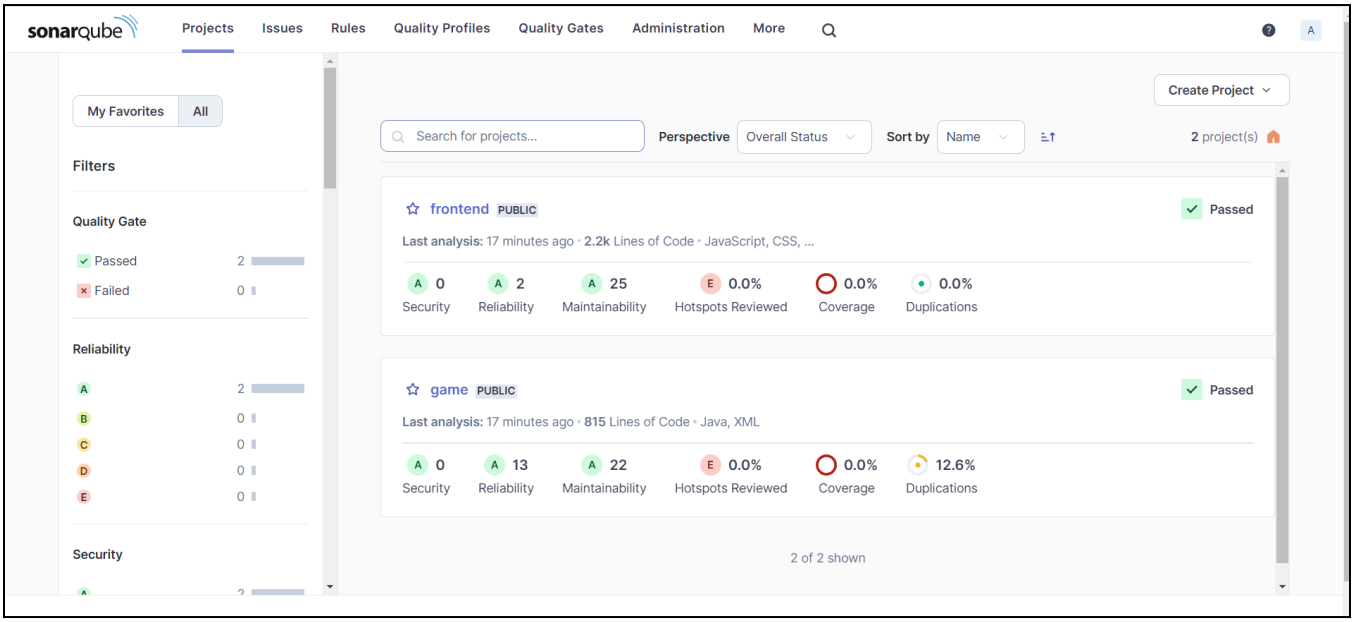
Step 5: After EKS build gets completed start building frontend and backend jobs too.



Step 6: Wait for all the builds to be completed.



Step 7: Check the code analysis by going to the sonarqube projects section.



Continuous Deployment with ArgoCD

Step 1: SSH into the jump host server and verify that you are able to access the kubernetes server.

Step 2: Install ArgoCD by running the following commands.

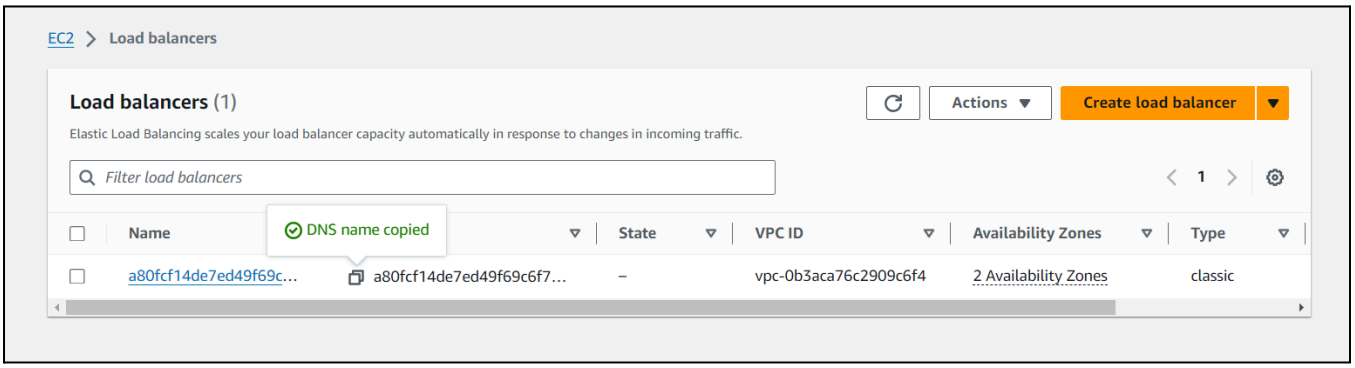
- kubectl create namespace argocd
- kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/mainfests/install.yaml

Step 3: Change service of argocd to LoadBalancer by running the following command.

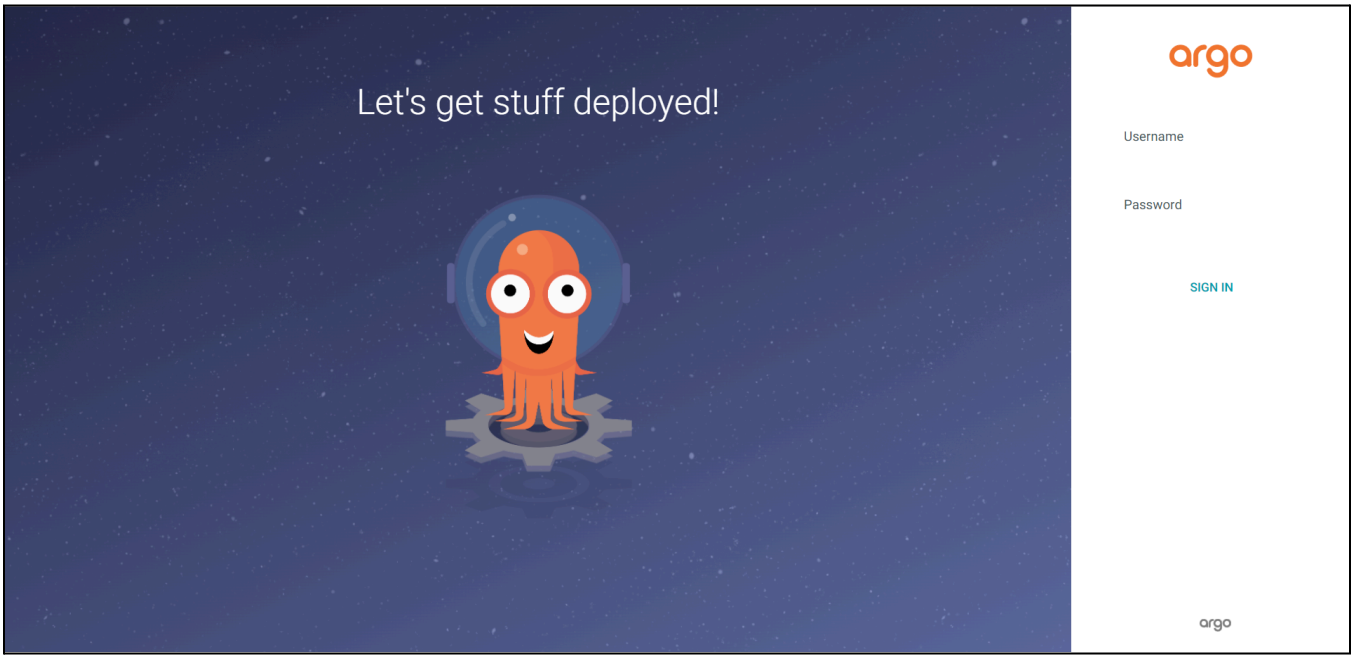
- kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

Step 4: Go to **AWS EC2 console**, in the **Load Balancer** Section wait for the load balancer to be provisioned.

Step 5: Once it is provisioned, copy the **load balancer DNS** and hit it.

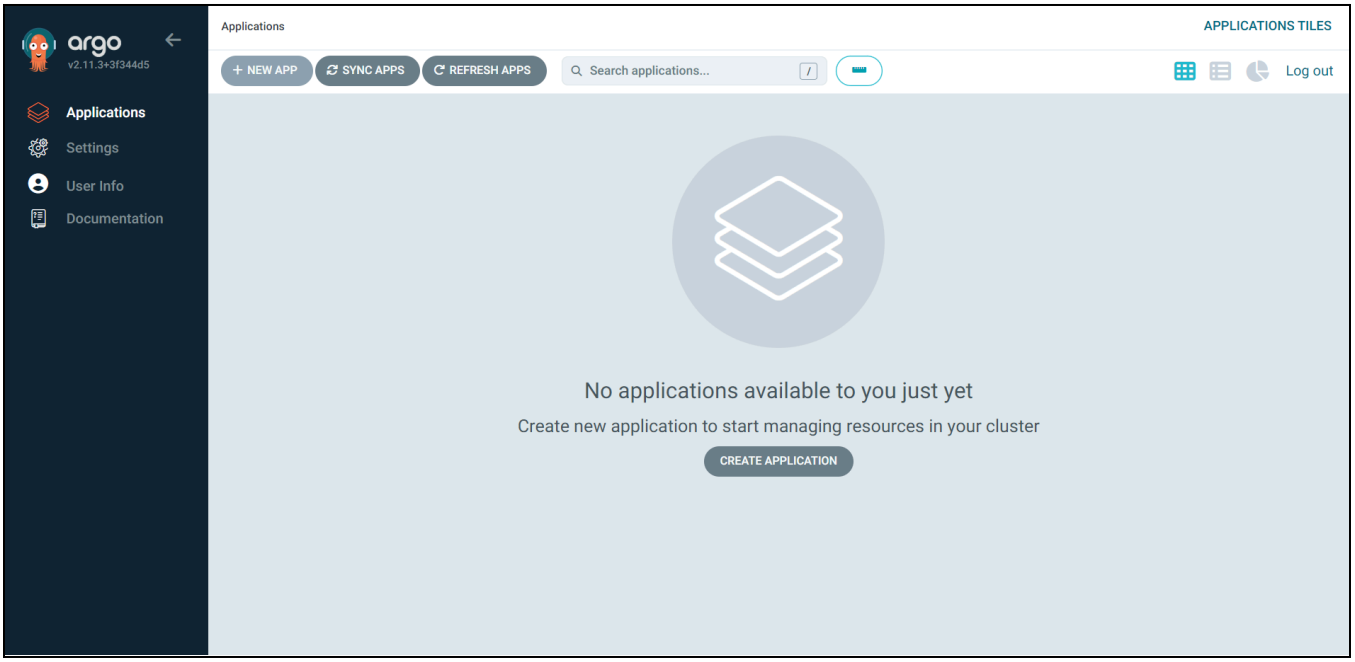


Step 6: Once you hit the DNS you will be landed on the ArgoCD login page.



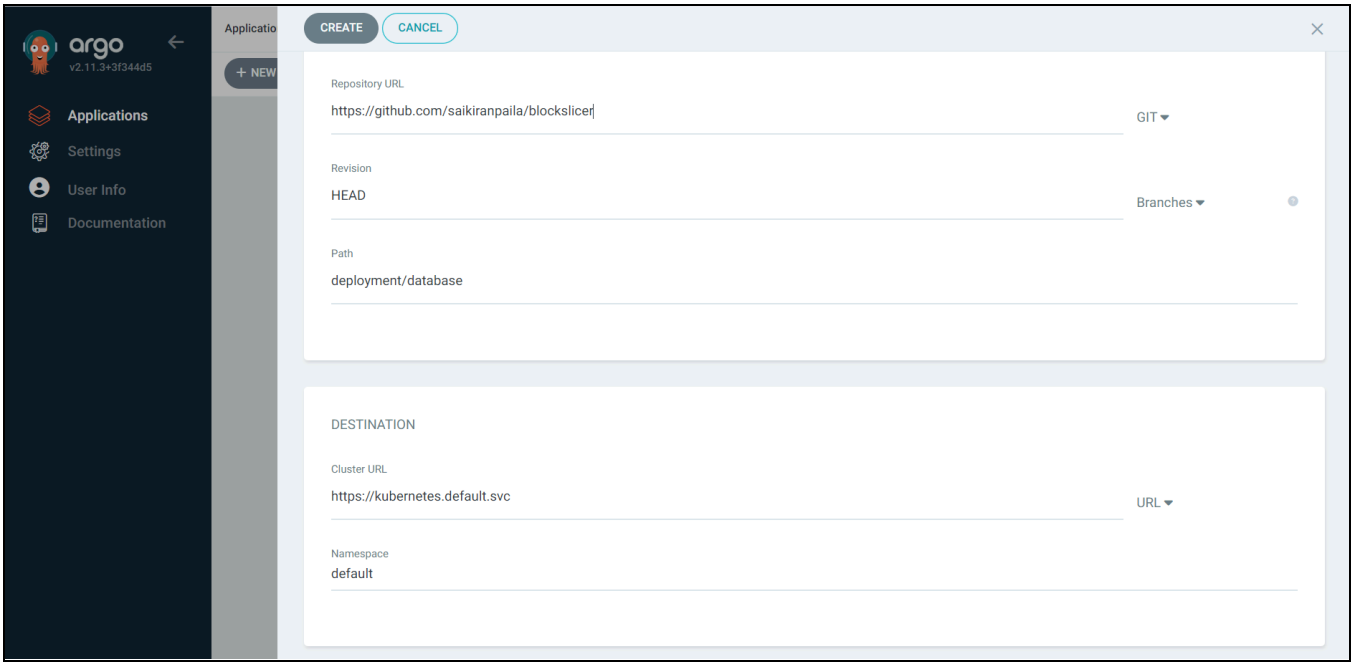
Step 7: Default username for AgroCD is **admin** and password can be retrieved by running **kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d** on the jumphost server.

Step 8: Login to the ArgoCD using these credentials and click on **NEW APP.**



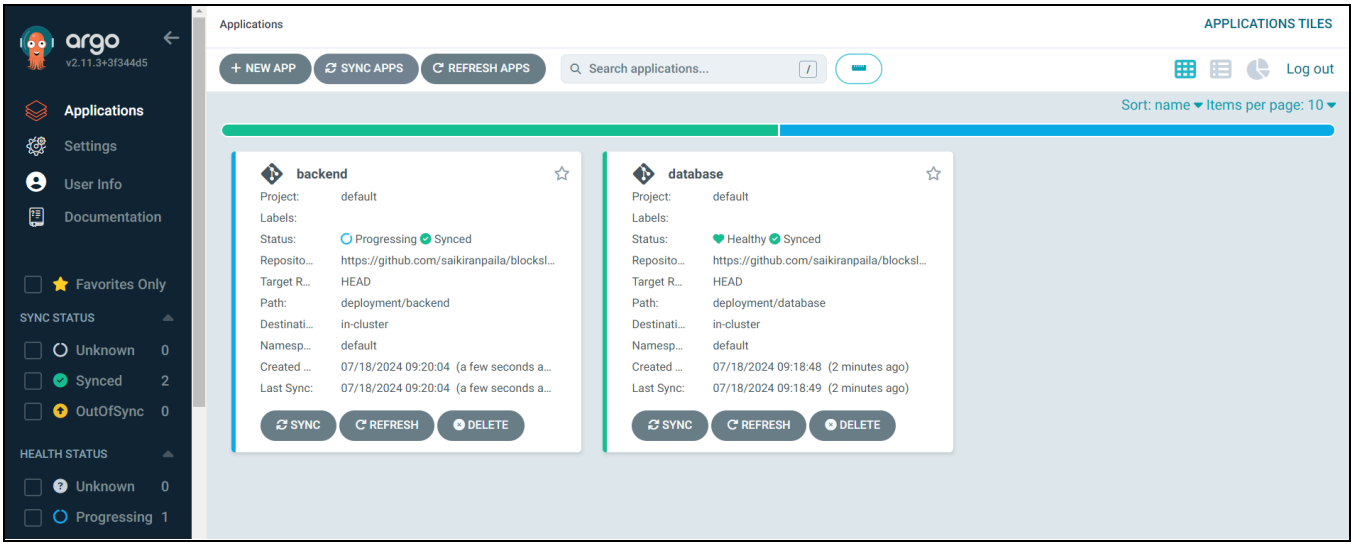
Step 9: First we will deploy **database** on **default** namespace, by following these steps,

1. Set the **Application Name** to the **database.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Under Source Section
   - 4.1. Set **Repository Url** to **<your github url>**
   - 4.2. Set Path to **deployment/database**
5. Under the Destination Section.
   - 5.1. Set cluster url to https://kubernetes.default.svc
   - 5.2. Namespace to **default**
6. Click on **create.**



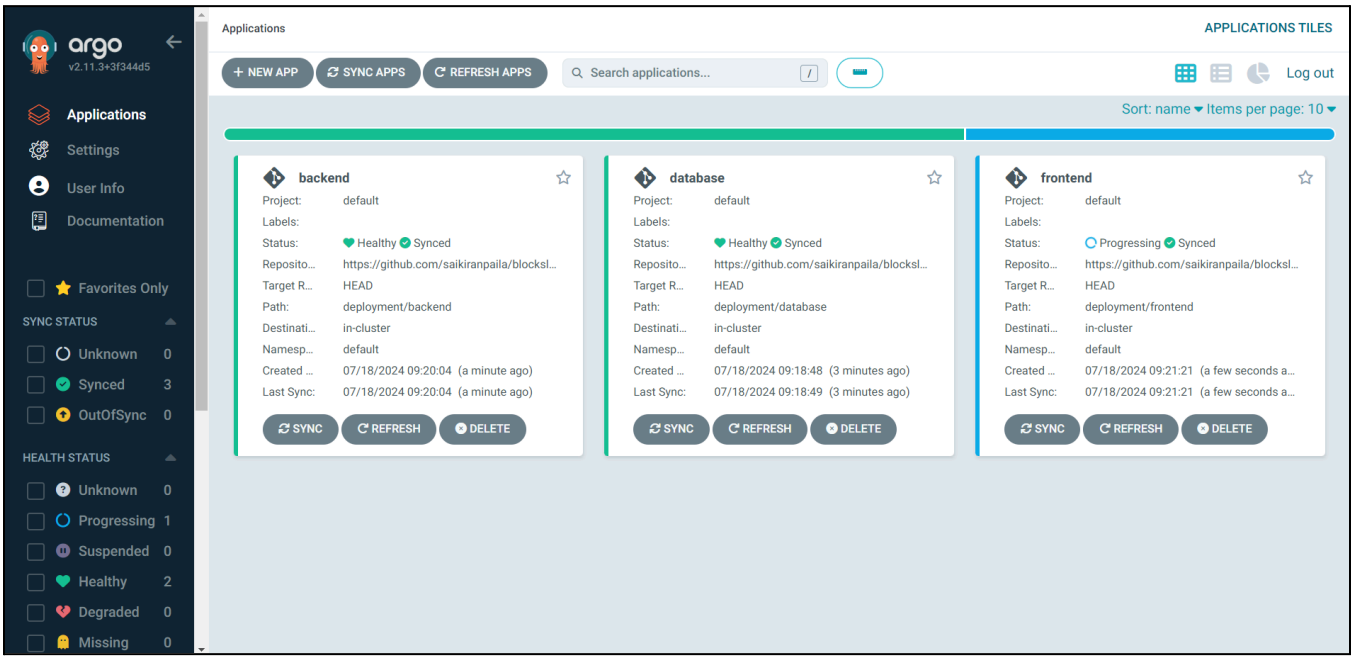
Step 10: Do the same for **backend** on **default** namespace, by following these steps,

1. Set the **Application Name** to the **backend.**
2. Set **Project Name** to **default.**
3. Select **Sync Policy** to **Automatic.**
4. Under Source Section
   - 4.1. Set **Repository Url** to **<your github url>**
   - 4.2. Set Path to **deployment/backend**
5. Under the Destination Section.
   - 5.1. Set cluster url to https://kubernetes.default.svc
   - 5.2. Namespace to **default**
6. Click on **create.**



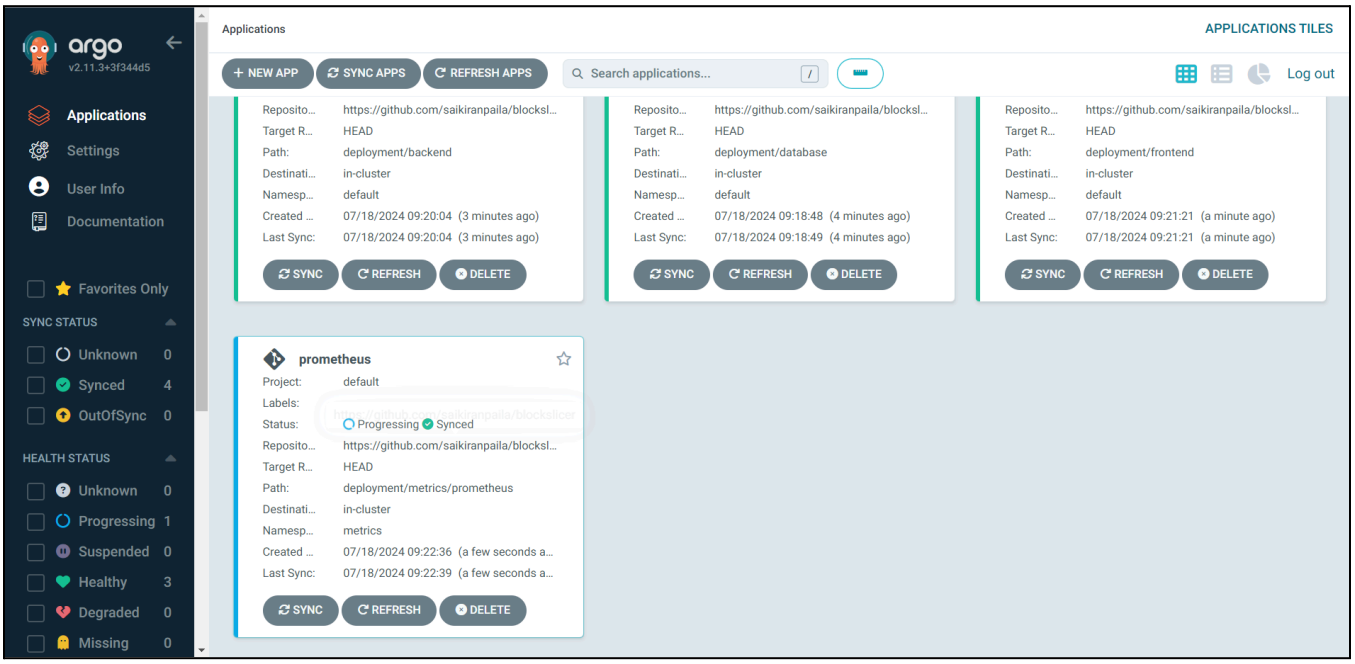
Step 11: Do the same for **frontend** on **default** namespace, by following these steps,

1. Set the **Application Name** to the **frontend.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Under Source Section
    a. Set **Repository Url** to **<your github url>**
    b. Set Path to **deployment/frontend**
5. Under the Destination Section.
    a. Set cluster url to https://kubernetes.default.svc
    b. Namespace to **default**
6. Click on **create.**



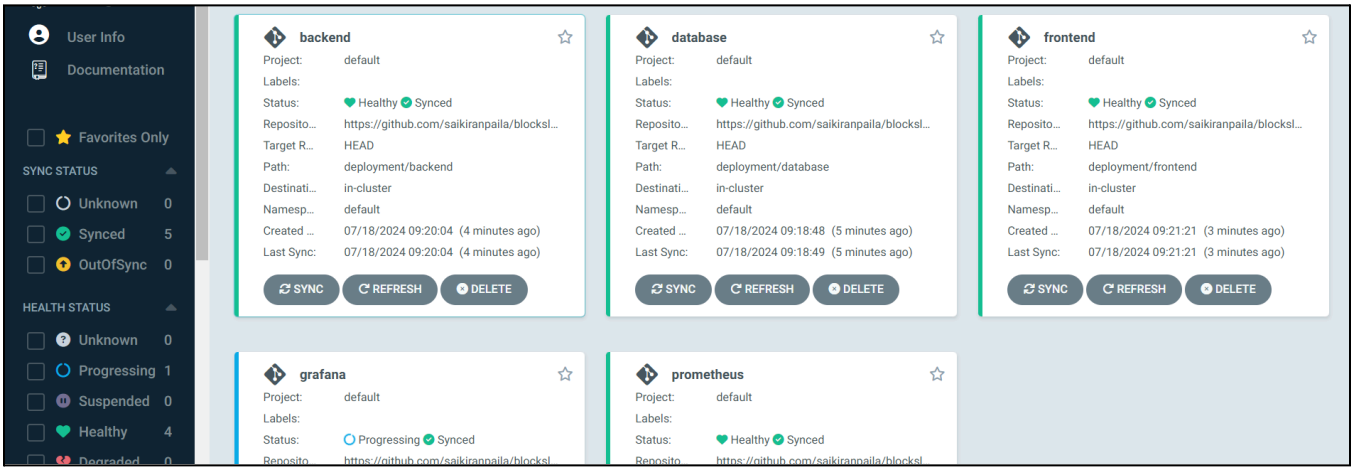
Step 12: For prometheus,

1. Set the **Application Name** to the **prometheus.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Mark **AUTO-CREATE NAMESPACE**
5. Under Source Section
    a. Set **Repository Url** to **<your github url>**
    b. Set Path to **deployment/prometheus**
6. Under the Destination Section.
    a. Set cluster url to https://kubernetes.default.svc
    b. Namespace to **metrics**
7. Click on **create.**



Step 13: For grafana,

1. Set the **Application Name** to the **grafana.**
2. Set **Project Name** to **default.**
3. Select **Sync Policy** to **Automatic.**
4. Mark **AUTO-CREATE NAMESPACE**
5. Under Source Section
   - 5.1. Set **Repository Url** to **<your github url>**
   - 5.2. Set Path to **deployment/grafana**
6. Under the Destination Section.
   - 6.1. Set cluster url to https://kubernetes.default.svc
   - 6.2. Namespace to **metrics**
7. Click on **create.**



Step 14: For ElasticSearch,

1. Set the **Application Name** to the **elastic.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Mark **AUTO-CREATE NAMESPACE**
5. Under Source Section
    5.1. Set **Repository Url** to **<your github url>**
    5.2. Set Path to **deployment/elastic**
6. Under the Destination Section.
    6.1. Set cluster url to https://kubernetes.default.svc
    6.2. Namespace to **logging**
7. Click on **create.**

Step 15: For Fluent-bit,

1. Set the **Application Name** to the **fluent-bit.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Mark **AUTO-CREATE NAMESPACE**
5. Under Source Section
    5.1. Set **Repository Url** to **<your github url>**
    5.2. Set Path to **deployment/fluent-bit**
6. Under the Destination Section.
    6.1. Set cluster url to https://kubernetes.default.svc
    6.2. Namespace to **logging**
7. Click on **create.**

Step 16: For Kibana,

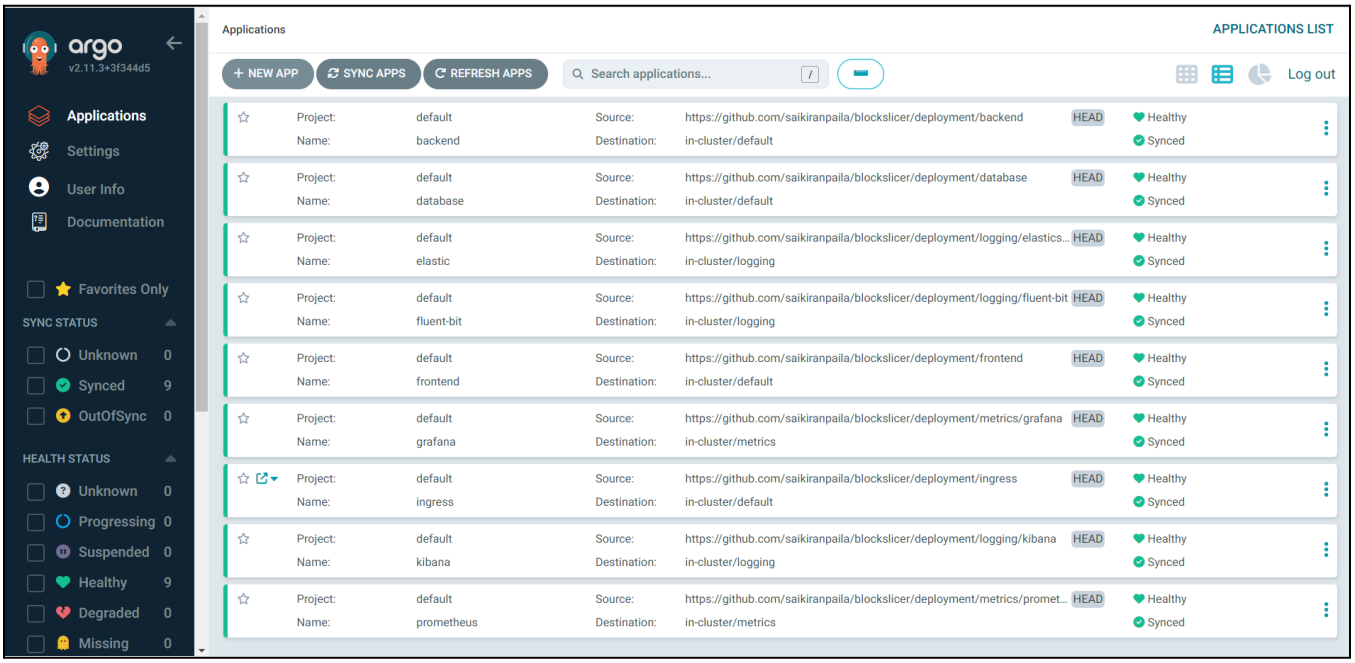
1. Set the **Application Name** to the **kibana.**
2. Set **Project Name** to **default**.
3. Select **Sync Policy** to **Automatic.**
4. Mark **AUTO-CREATE NAMESPACE**
5. Under Source Section
    5.1. Set **Repository Url** to **<your github url>**
    5.2. Set Path to **deployment/kibana**
6. Under the Destination Section.
    6.1. Set cluster url to https://kubernetes.default.svc

6.2. Namespace to **logging**

7. Click on **create.**

Step 17: For Ingress,

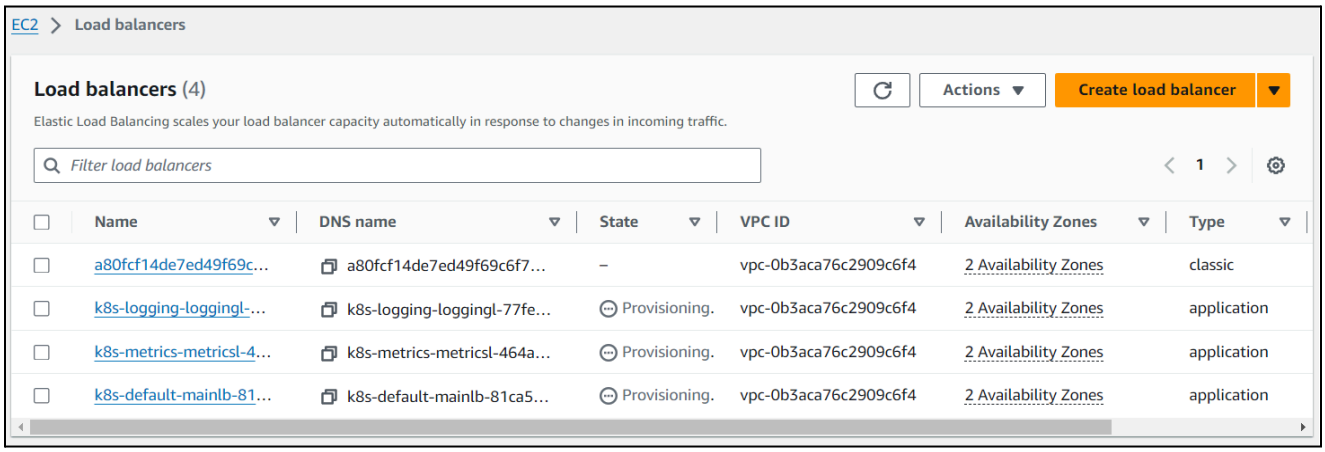
1. Set the **Application Name** to the **ingress.**
2. Set **Project Name** to **default.**
3. Select **Sync Policy** to **Automatic.**
4. Under Source Section
   4.1. Set **Repository Url** to **<your github url>**
   4.2. Set Path to **deployment/ingress**
5. Under the Destination Section.
   5.1. Set cluster url to https://kubernetes.default.svc
   5.2. Namespace to **default**
6. Click on **create.**



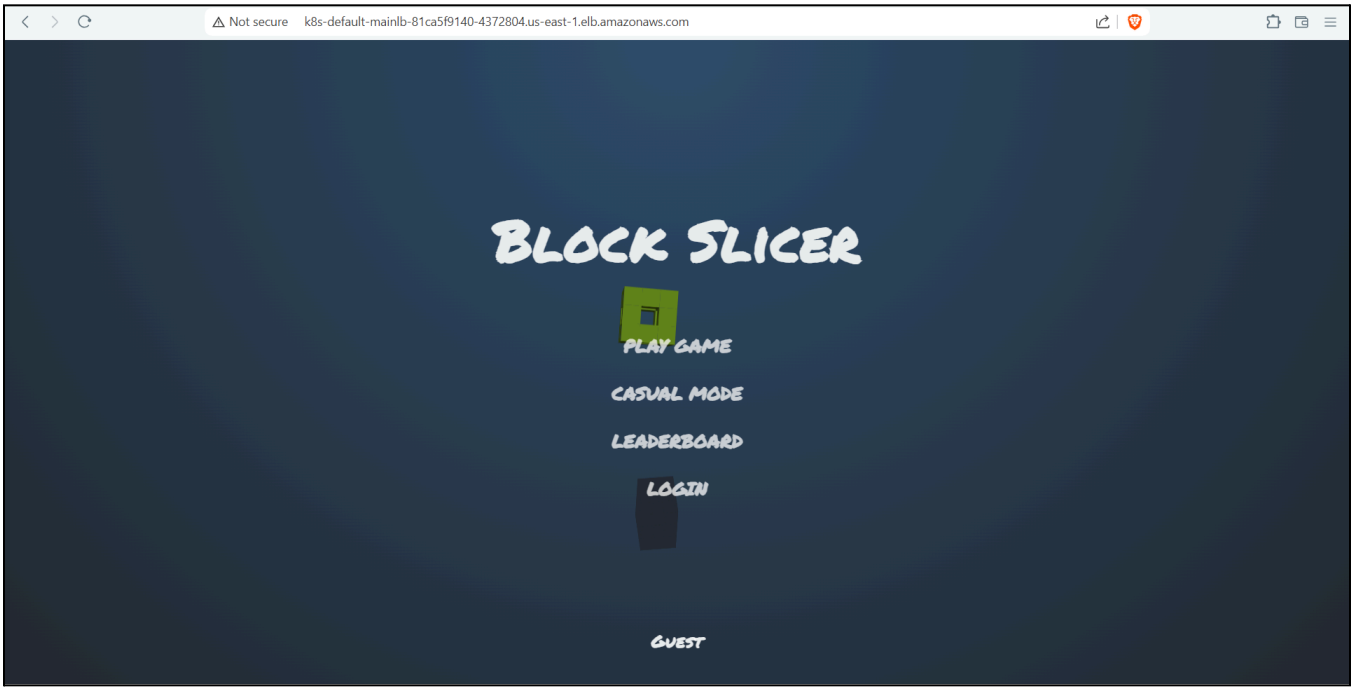
Step 18: If all the apps are in sync, you have successfully deployed the apps.

Access the applications (Main app, metrics app, logging app)

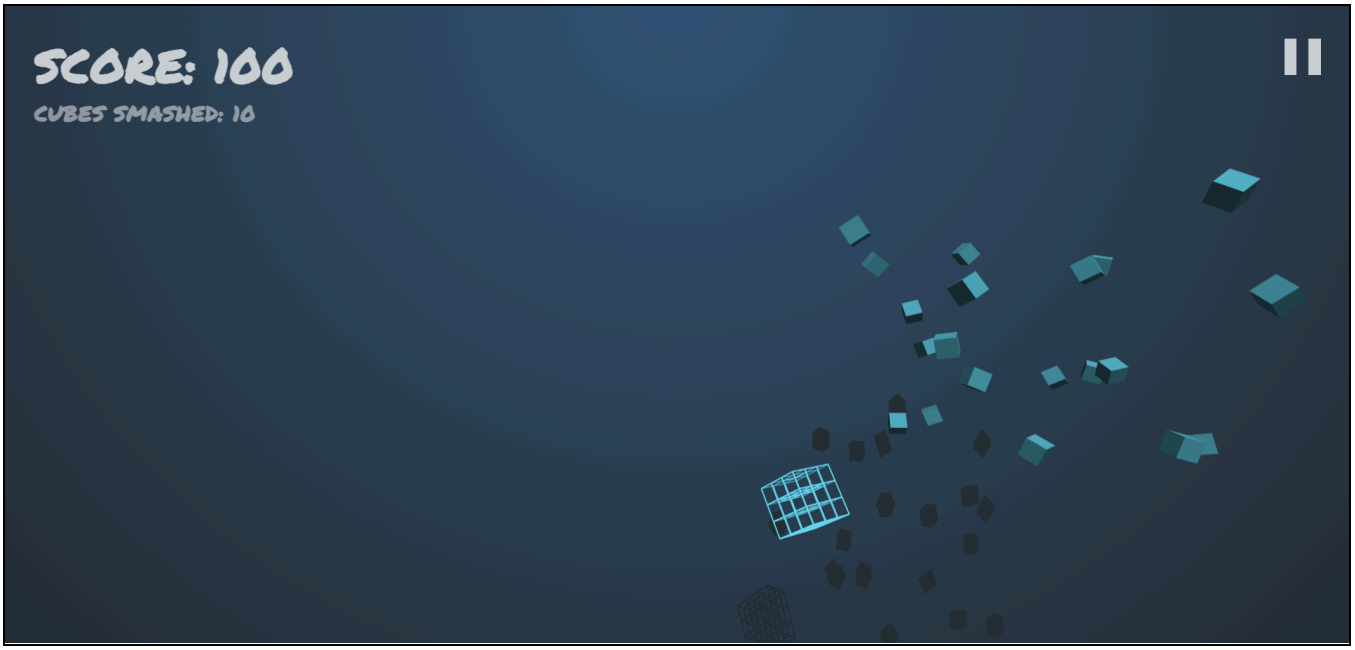
Step 1: Once you have deployed ingress in ArgoCD, you can find three Load Balancers in the AWS console.



Step 2: Each loadbalancer is prefixed for what it is meant for, once all the LoadBalancers are provisioned copy the DNS of **MAIN APP** hit it. You will be landed on a Web Game called **Block Slicer.** Here the play is very simple; you have to just slice through the blocks. Person who has sliced the highest blocks will be on the top of the leaderboard.



Step 3: To check whether the app is working, click on login and then click on signup to sign up an account, and login with the same credentials. Now play the game for a few seconds.



Step 4: Now check on the leaderboard that your account exists. Parallel other players can also create their own accounts and beat the leaderboard.

Step 5: To check whether the details are reflected in the databases, run the following commands,

- kubectl exec -it master-0 -- /bin/bash
- psql -U postgres
- \c game
- select * from gamersscore;

This will return you all the gamers' scores.

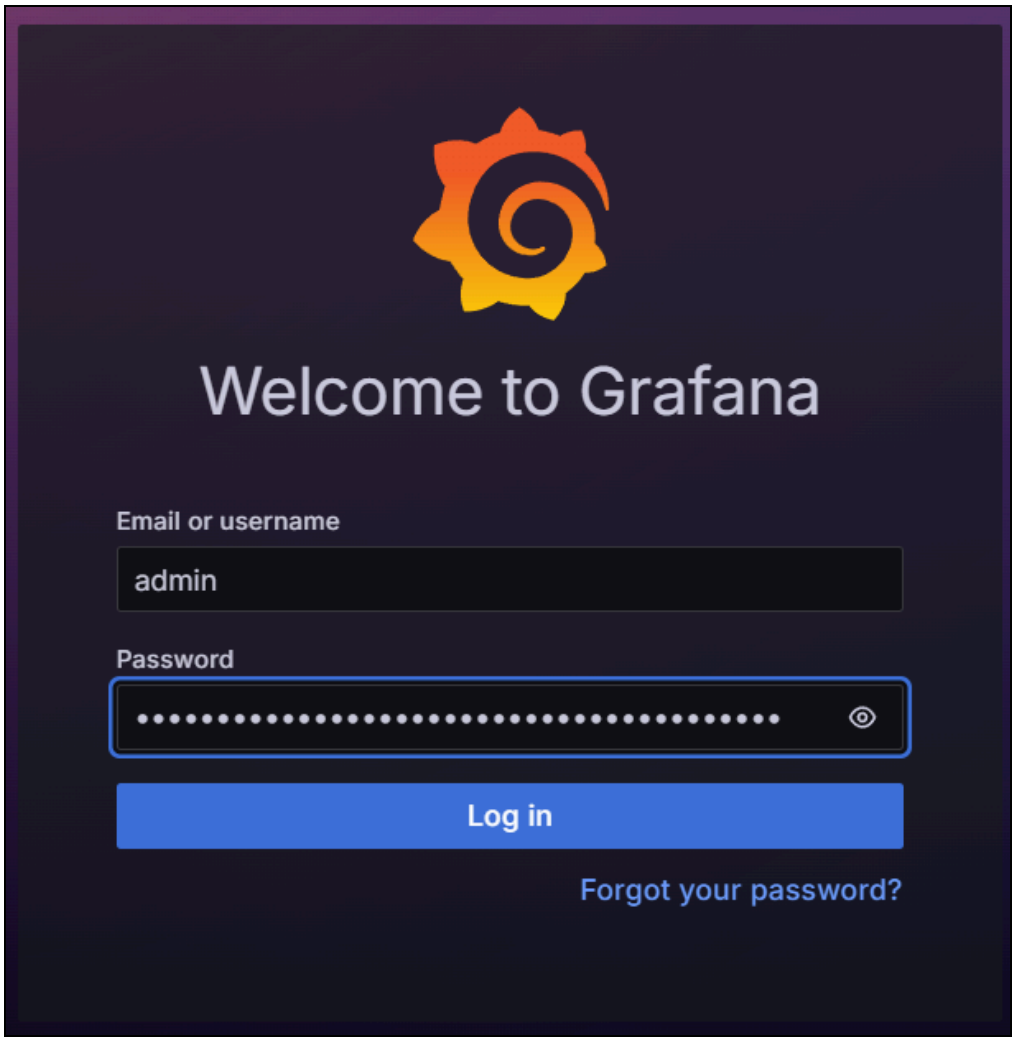
```
backend-deployment-5579bcc578-8pr6h   1/1     Running   0          15m
backend-deployment-5579bcc578-pj6p9   1/1     Running   0          15m
frontend-deployment-c598f5cc9-8fd2t   1/1     Running   0          14m
frontend-deployment-c598f5cc9-v9bvv   1/1     Running   0          14m
master-0                              1/1     Running   0          17m
replica-0                             1/1     Running   0          17m
replica-1                             1/1     Running   0          16m
[root@ip-10-0-1-53 ec2-user]# kubectl exec -it master-0 -- /bin/bash
root@master-0:/# psql -U postgres
psql (16.3 (Debian 16.3-1.pgdg120+1))
Type "help" for help.

postgres=# \c game
You are now connected to database "game" as user "postgres".
game=# \l
                                                    List of databases
   Name    |  Owner   | Encoding | Locale Provider |  Collate   |   Ctype    | ICU Locale | ICU
-----------+----------+----------+-----------------+------------+------------+------------+---
 game      | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |
 postgres  | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |
 template0 | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |
           |          |          |                 |            |            |            |
 template1 | postgres | UTF8     | libc            | en_US.utf8 | en_US.utf8 |            |
           |          |          |                 |            |            |            |
(4 rows)

game=# \d
              List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | gamercreadentials | table | postgres
 public | gamersscore       | table | postgres
(2 rows)

game=# select * from gamersscore;
 userid | blocksliced | score
--------+-------------+-------
 kiran  |          10 |   100
(1 row)
```

Step 6: To check whether the replication is also taking place, run the following commands,

- kubectl exec -it replica-0 -- /bin/bash
- psql -U postgres
- \c game
- select * from gamersscore;

This will return you all the gamers' scores which are also on the master database.

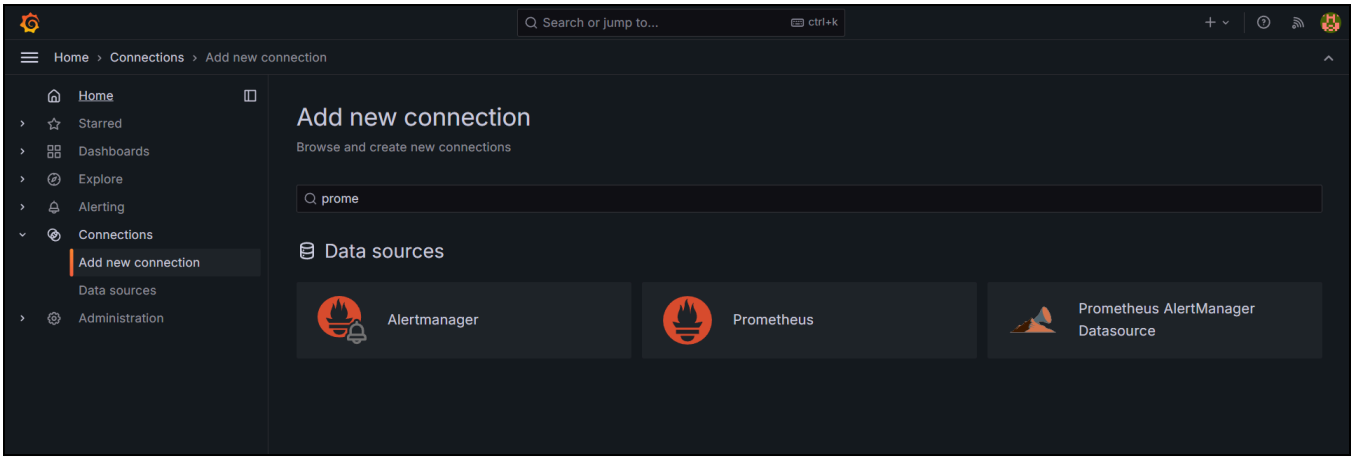
Step 7: Now, to check the metrics of the kubernetes cluster, copy the metrics Load Balancer URL and hit it. You will be prompted to grafana login page.



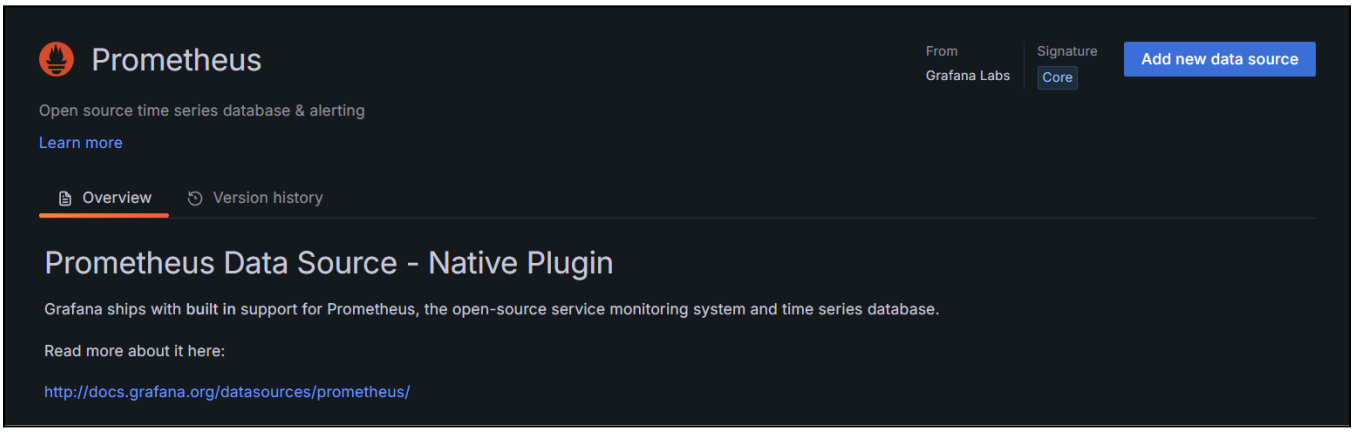
Step 8: Default username for Grafana is admin and password can be retrieved by running the following command on jumphost server,

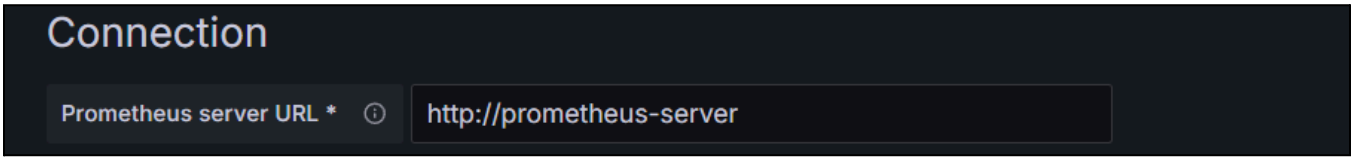
- **kubectl get secret --namespace metrics grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo**

Step 9: Login with these credentials and on left sidebar go to **connections > Add new connection** search for **prometheus.**

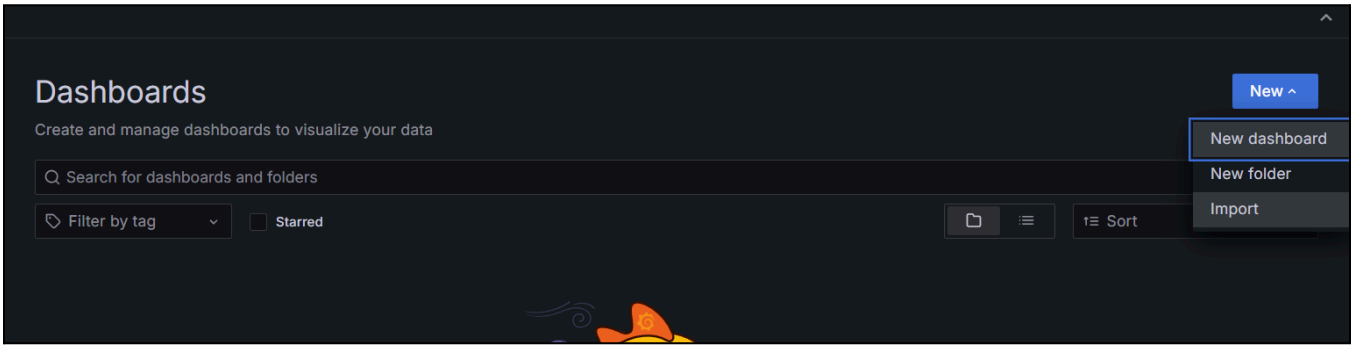


Step 10: Click on **Prometheus** and on Add new data source, this will ask you to provide a prometheus server url connection add the url as http://prometheus-server.





Step 11: Head to the **Dashboards** section and click on **New** and select **Import.**

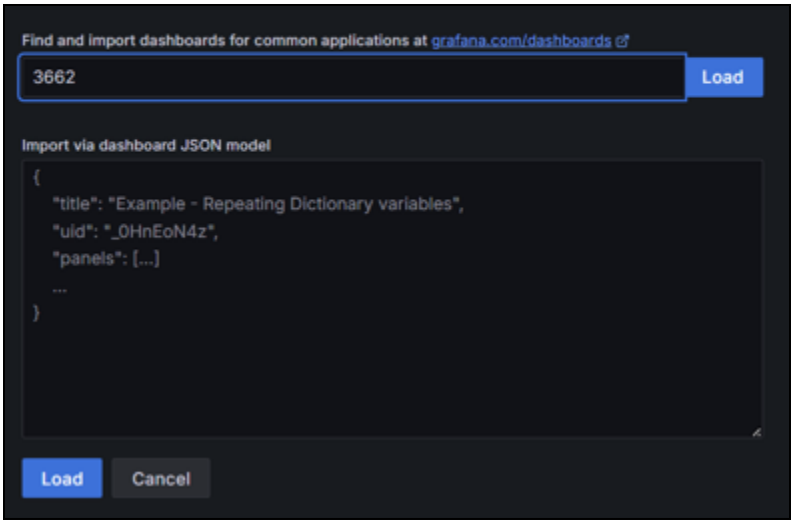


Step 12: Now give some dashboard id that are compatible with the kubernetes cluster,

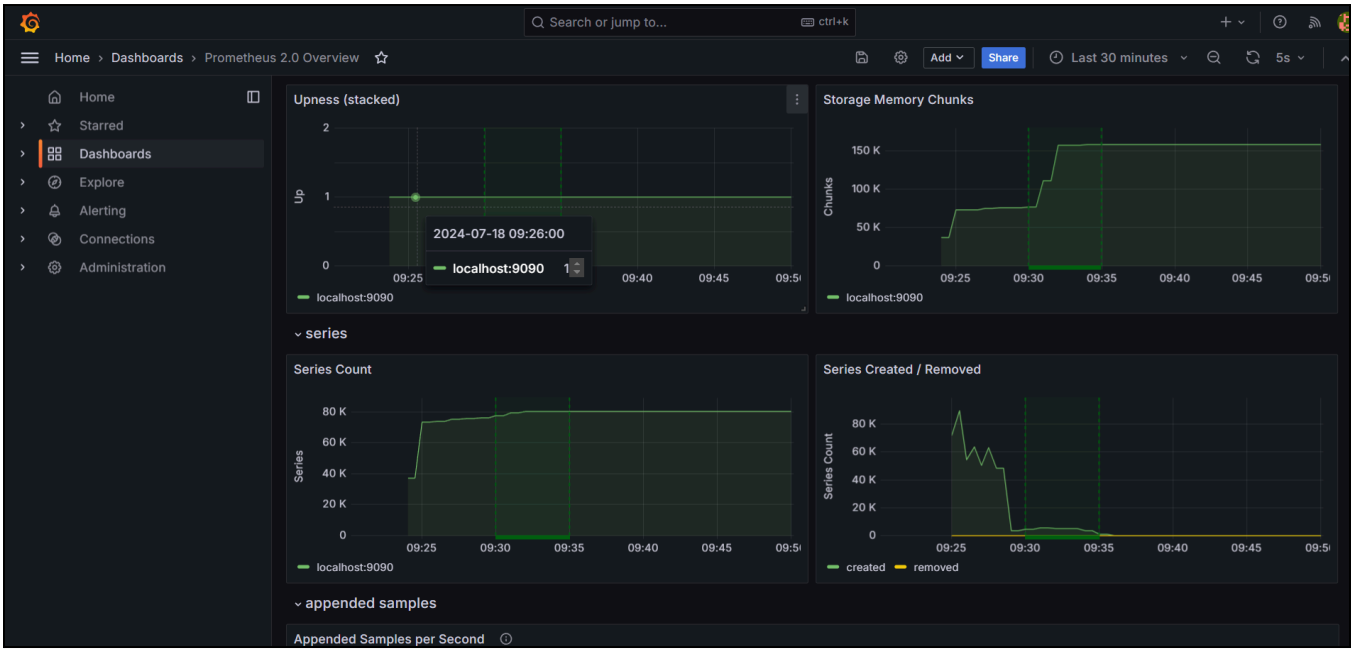
Dashboard	ID
k8s-addons-prometheus.json	19105
k8s-addons-trivy-operator.json	16337
k8s-system-api-server.json	15761
k8s-system-coredns.json	15762
k8s-views-global.json	15757
k8s-views-namespaces.json	15758
k8s-views-nodes.json	15759

k8s-views-pods.json	15760

Above are some of the example of kubernetes compatible dashboards.

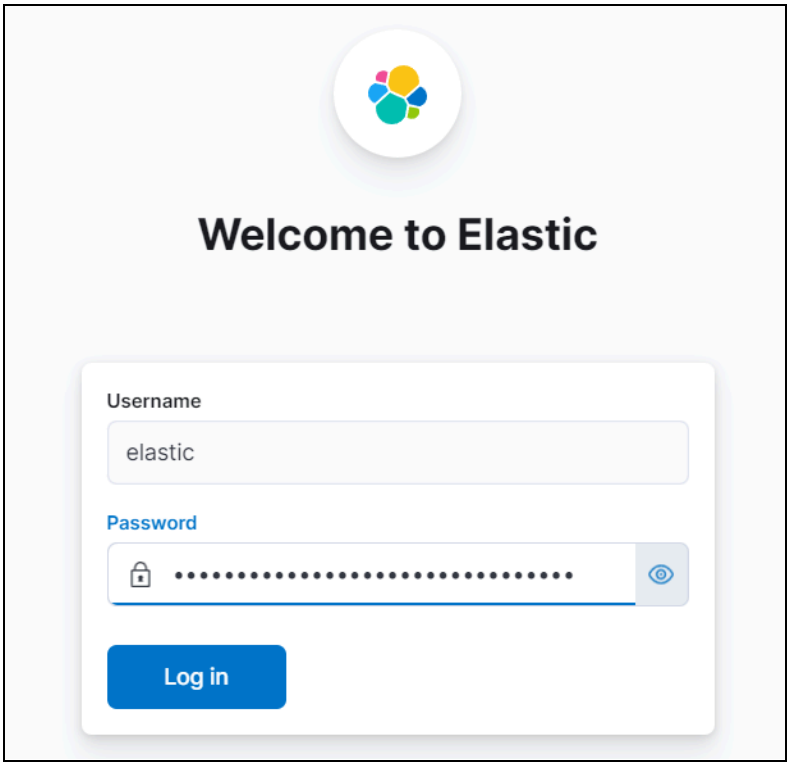


Step 13: On server metrics select prometheus and click on Import.



This should give you a page like the above showing some metrics of kubernetes cluster and some kube-system pods individual cpu and memory usage.

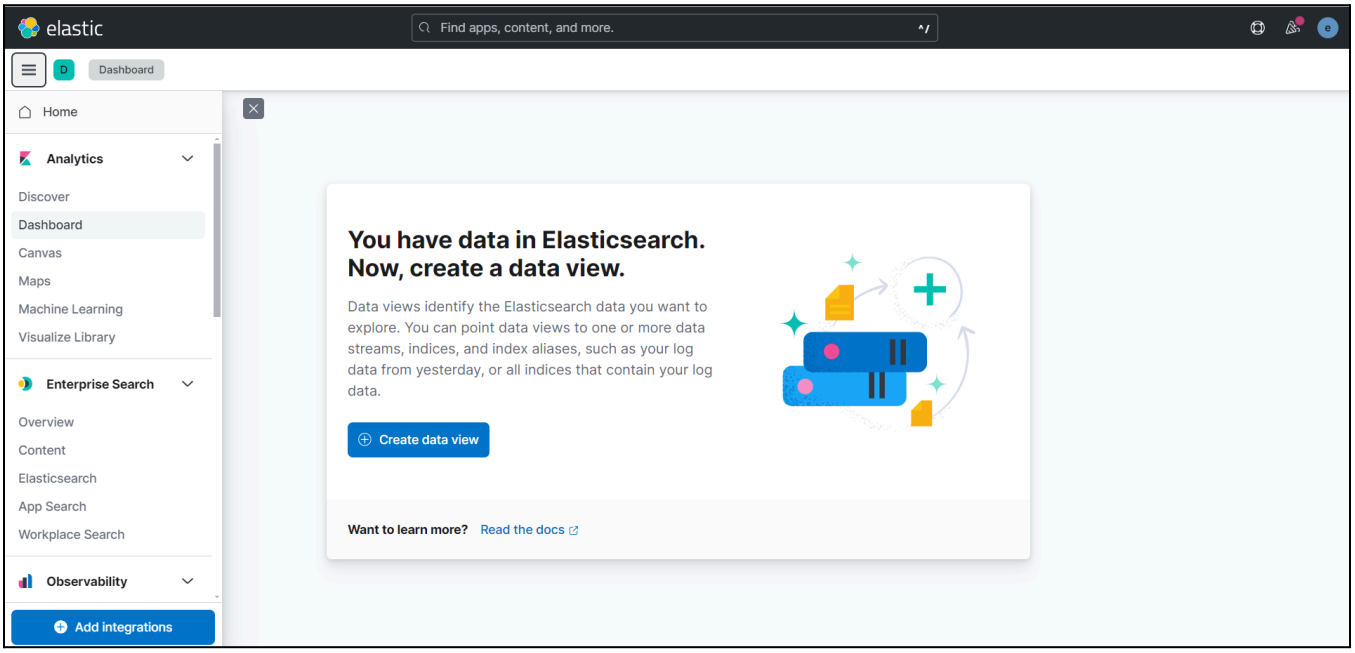
Step 14: Now to check the logs of the pods, copy the logging Load Balancer URL and hit it. This should bring you to the screen of the Elastic Login Page.



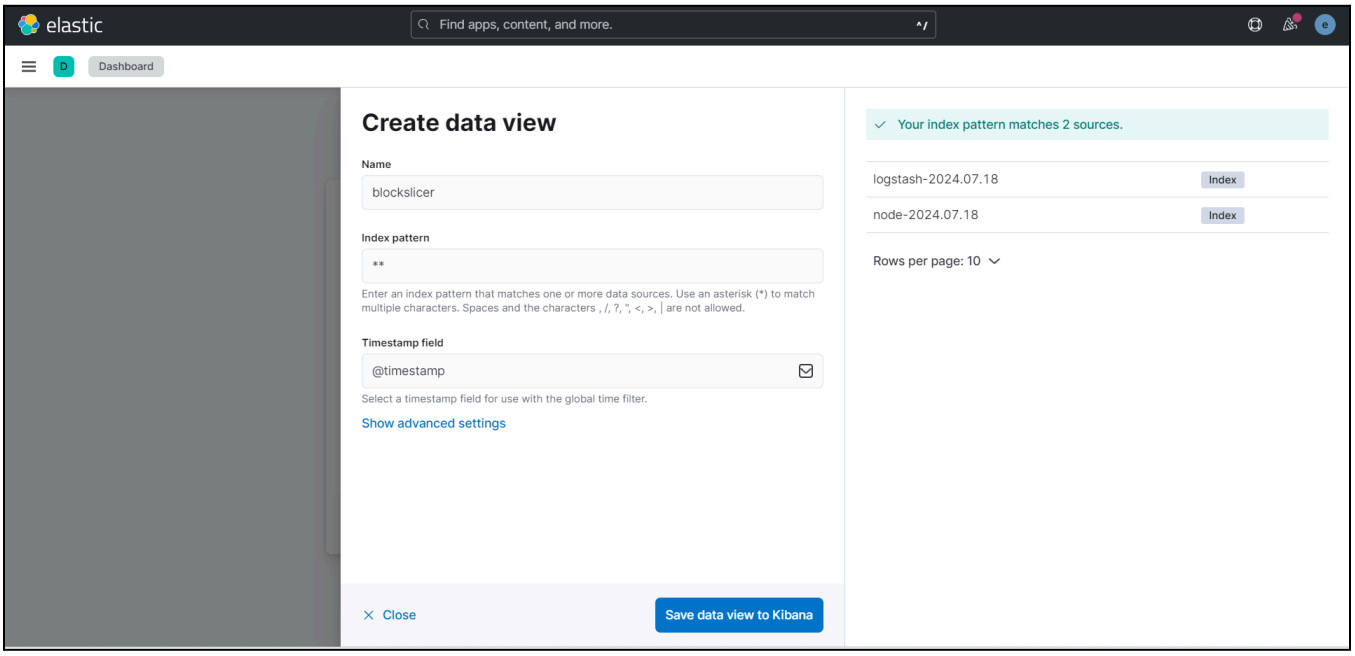
Step 15: To get the user name and password of elastic run the following command on the jumphost server.

- **kubectl get secrets --namespace=logging elasticsearch-master-credentials -ojsonpath='{.data.username}' | base64 -d**
- **kubectl get secrets --namespace=loggin elasticsearch-master-credentials -ojsonpath='{.data.password}' | base64 -d**

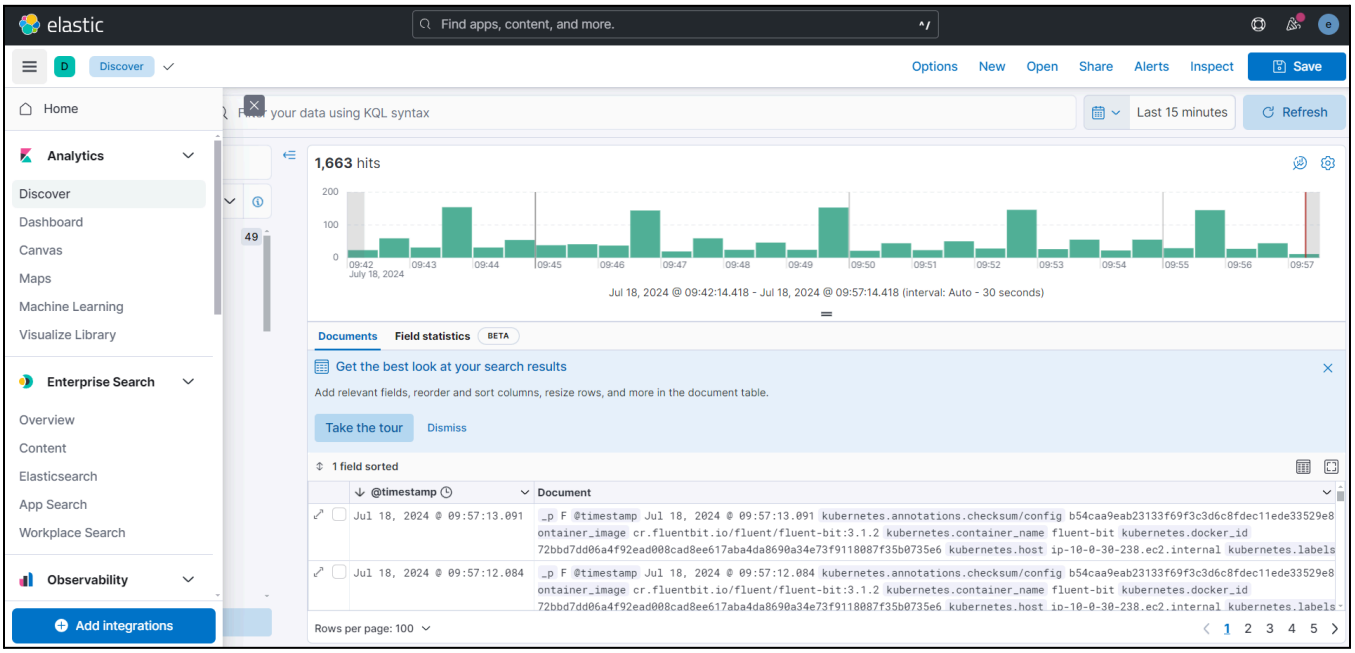
Step 16: Login with these credentials now you will be redirect to the Home page of **Kibana.**



Step 17: On the left side menu click on **Dashboard** and click on **create data view.** That will prompt you to a data view creation page, give some name for data view, set the index pattern to ** if you want to gather all the logs and according to the **timestamp order.** Click on **Save data view to kibana.**

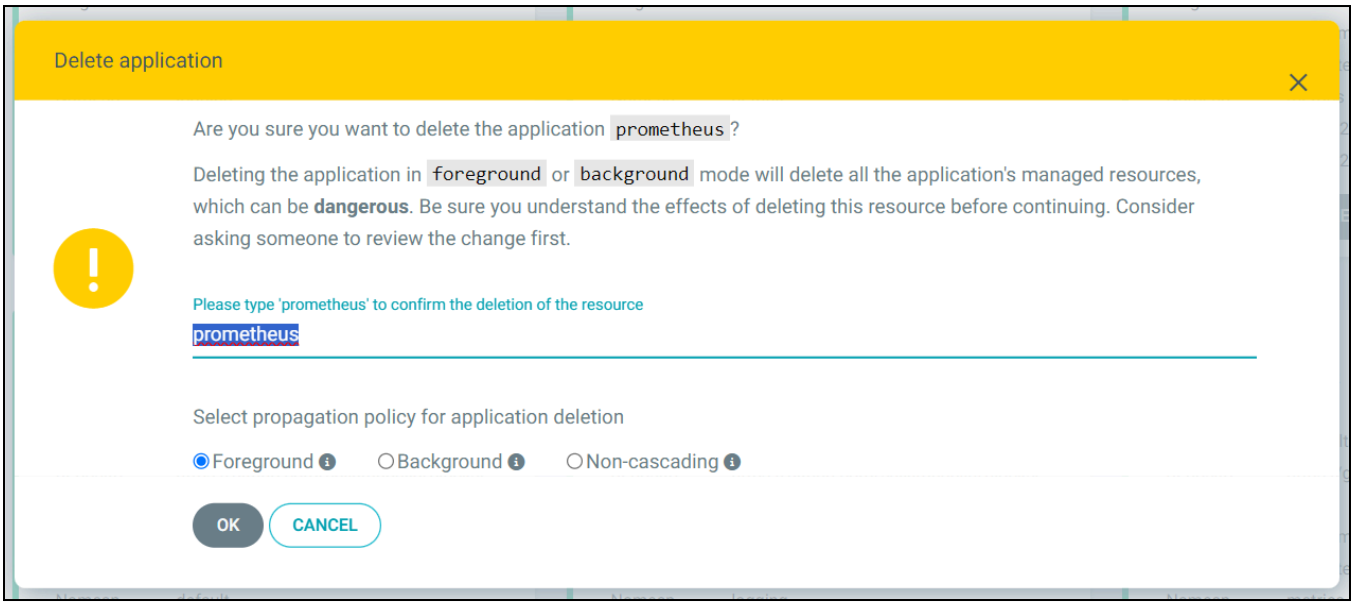


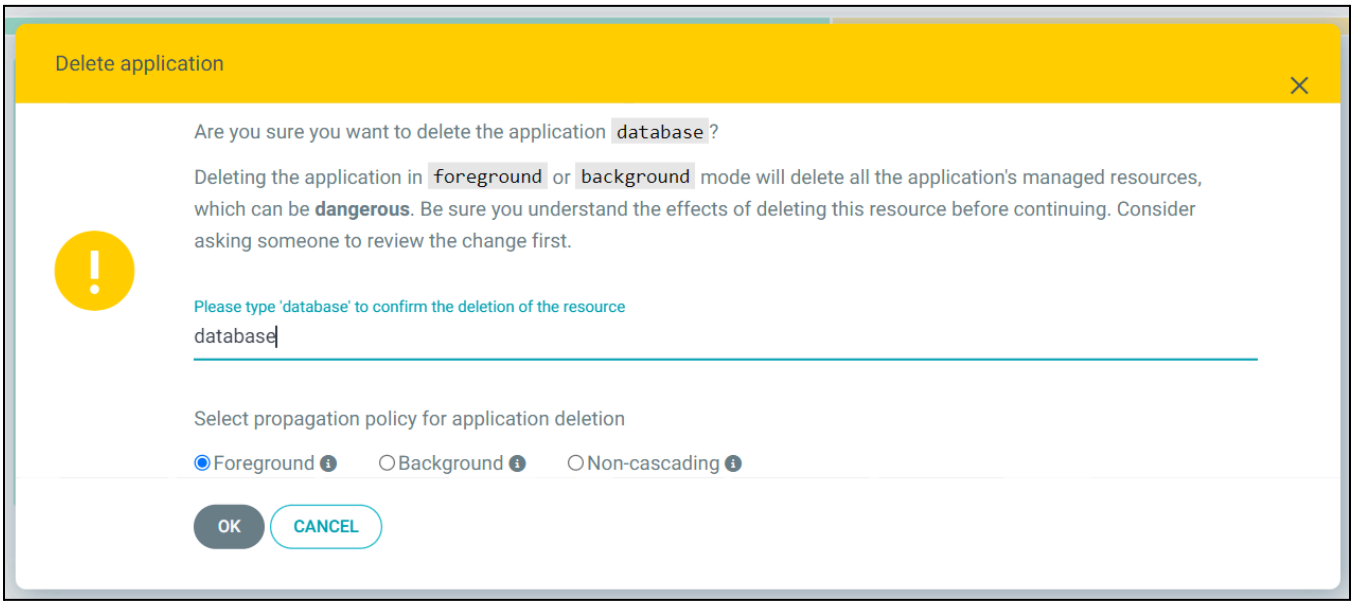
Step 18: Monitor your pod logs here, you can also filter logs by running KQL scripts.



Performing Cleanup.

Step 1: In order to ensure that our terraform destroys all the resources correctly then it is necessary to delete the things in the right order. Go to ArgoCD dashboard and start deleting the Apps on after the other in this order,
(Ingress > Kibana > Fluent-bit > Elastic >Grafana > Prometheus> FrontEnd > Backend > Database)

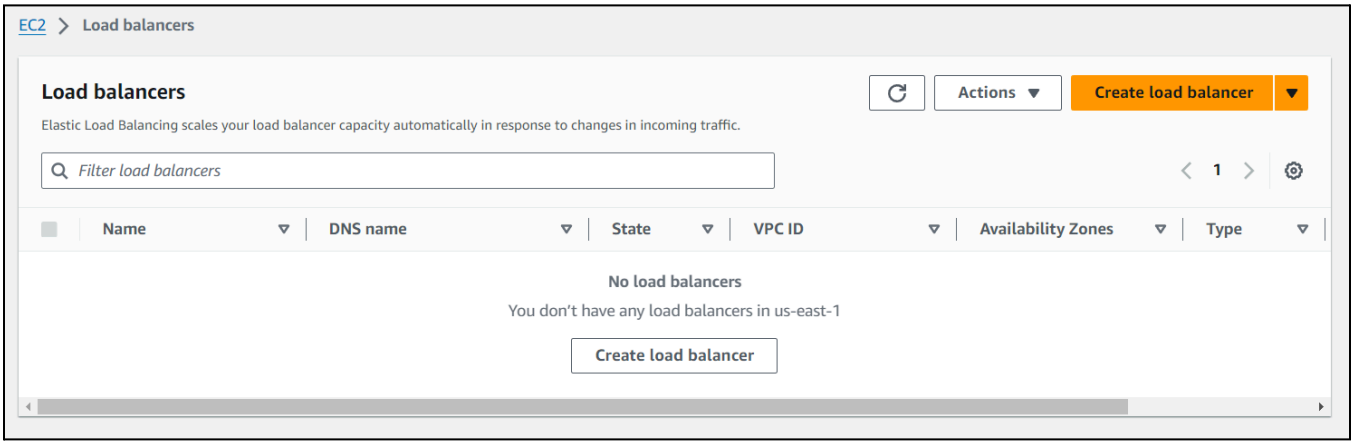




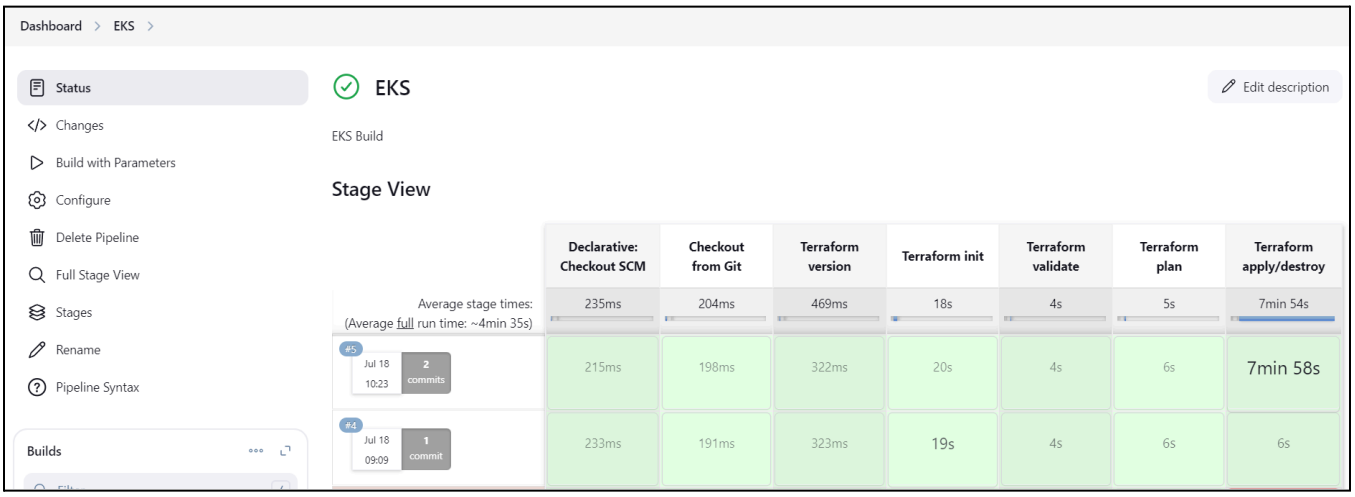
Step 2: After Deleting all the apps from ArgoCD, Now it's time to delete ArgoCD by simply running the command.

- **kubectl delete namespace argocd**

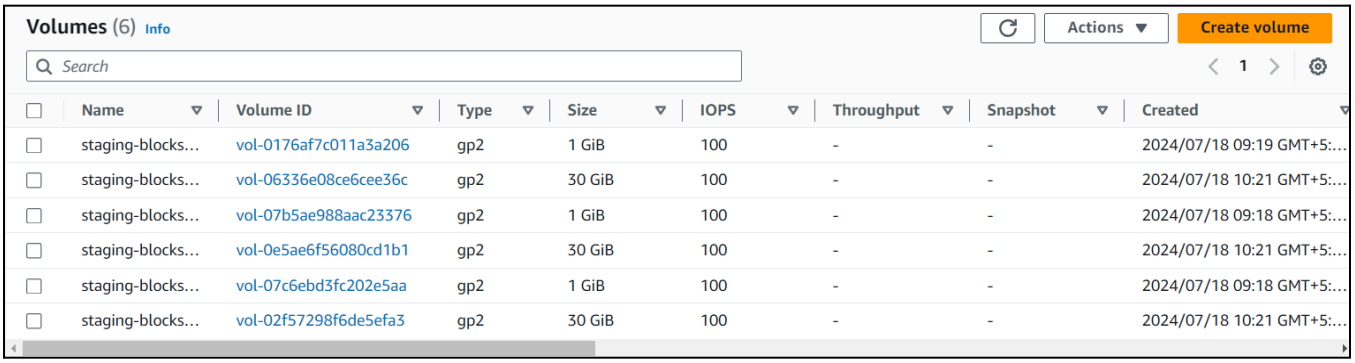
Step 3: Now ensure that no Load Balancer exists in your aws console.

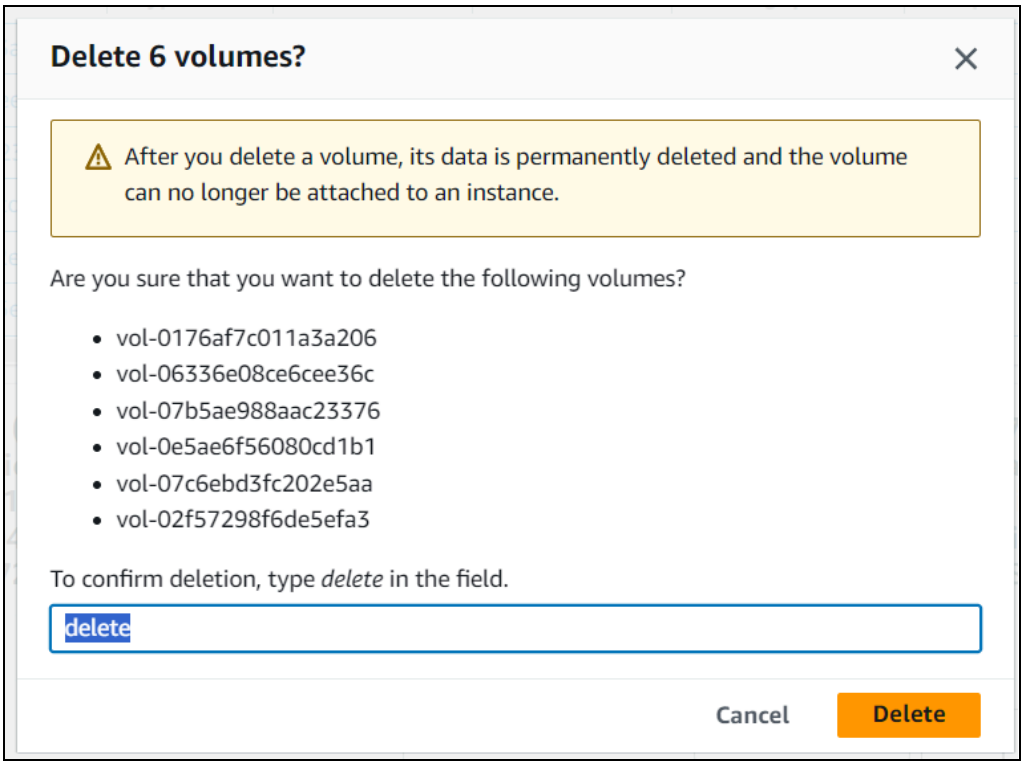


Step 4: Now login to jenkins and run EKS pipeline build by setting the parameter to destroy. That should take about 8 minutes to destroy.

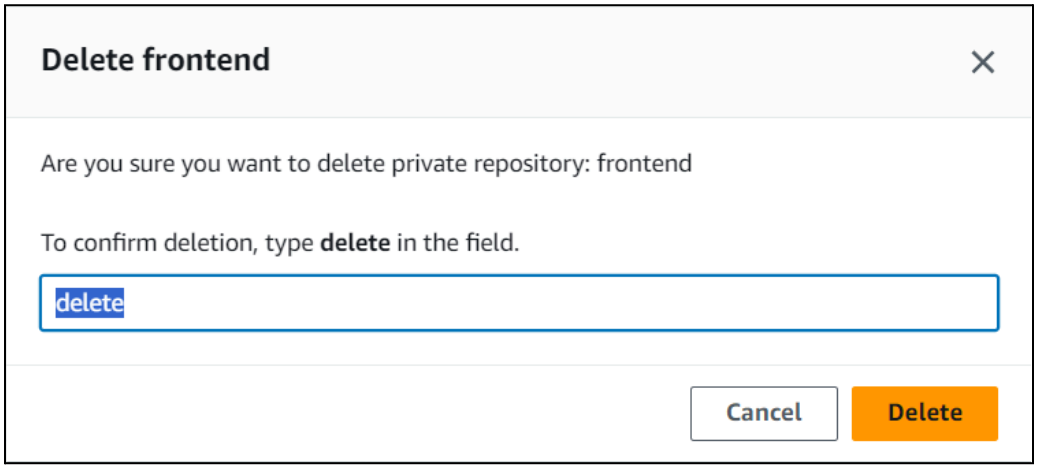


Step 5: Now head to AWS EC2 console and under volumes section delete all the volumes.





Step 6: Now in the AWS ECR console under the private repositories section delete the frontend and backend repositories.



Step 7: Deleting the S3 buckets.