

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:

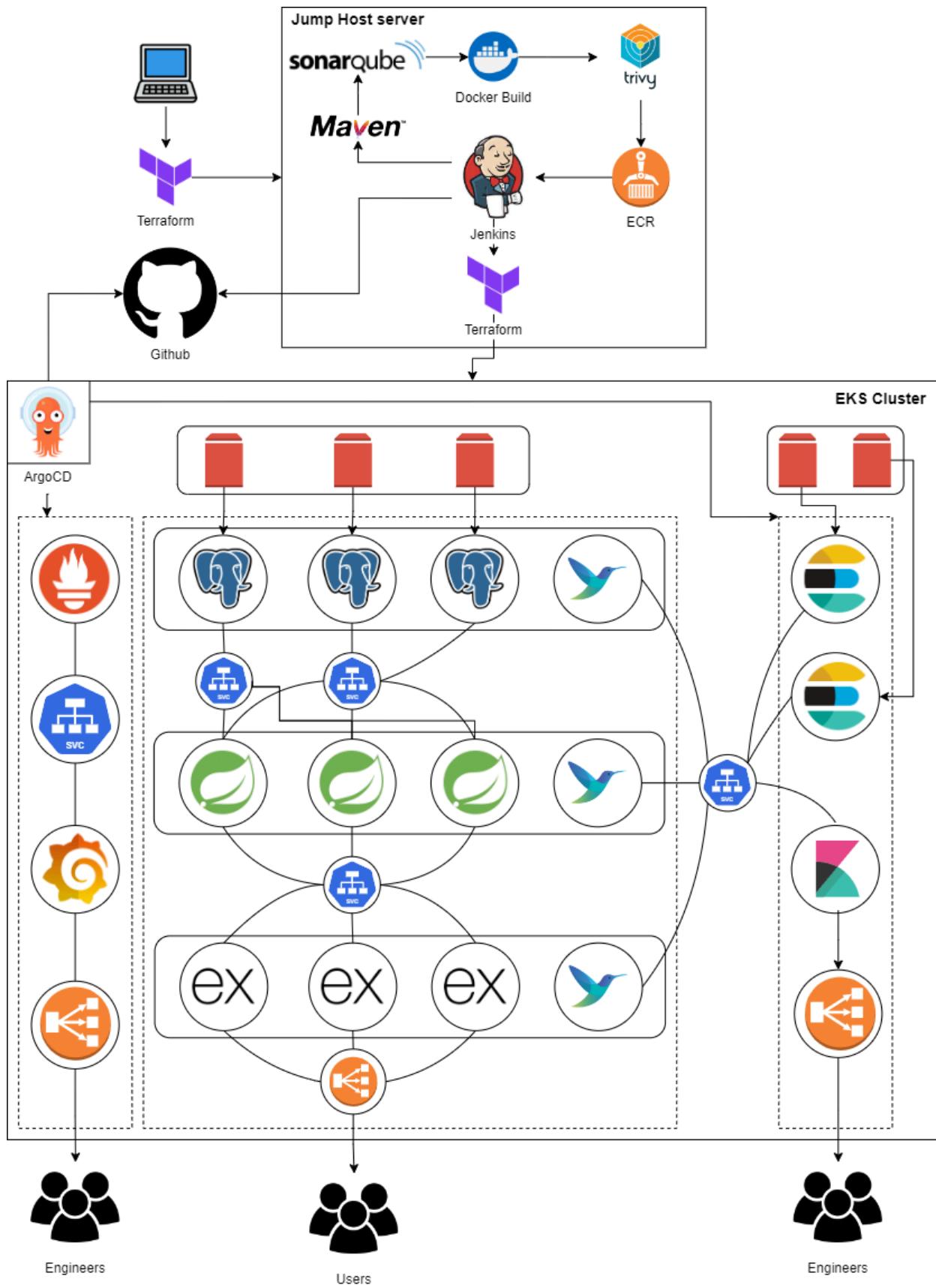
Amazon EKS facilitates scalability and elasticity by managing Kubernetes clusters efficiently. It integrates seamlessly with AWS services like IAM, VPC, enhancing security, networking, and additional monitoring capabilities.

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/blockslsicer>).

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

The screenshot shows the AWS S3 General purpose buckets page. It displays two buckets: 'eks-bucket-11072024' and 'jumphost-bucket-11072024'. Both buckets are located in 'US East (N. Virginia)' (us-east-1) and were created on July 11, 2024, at 10:01:28 (UTC+05:30) and July 12, 2024, at 06:44:06 (UTC+05:30) respectively. The 'Create bucket' button is visible in the top right corner.

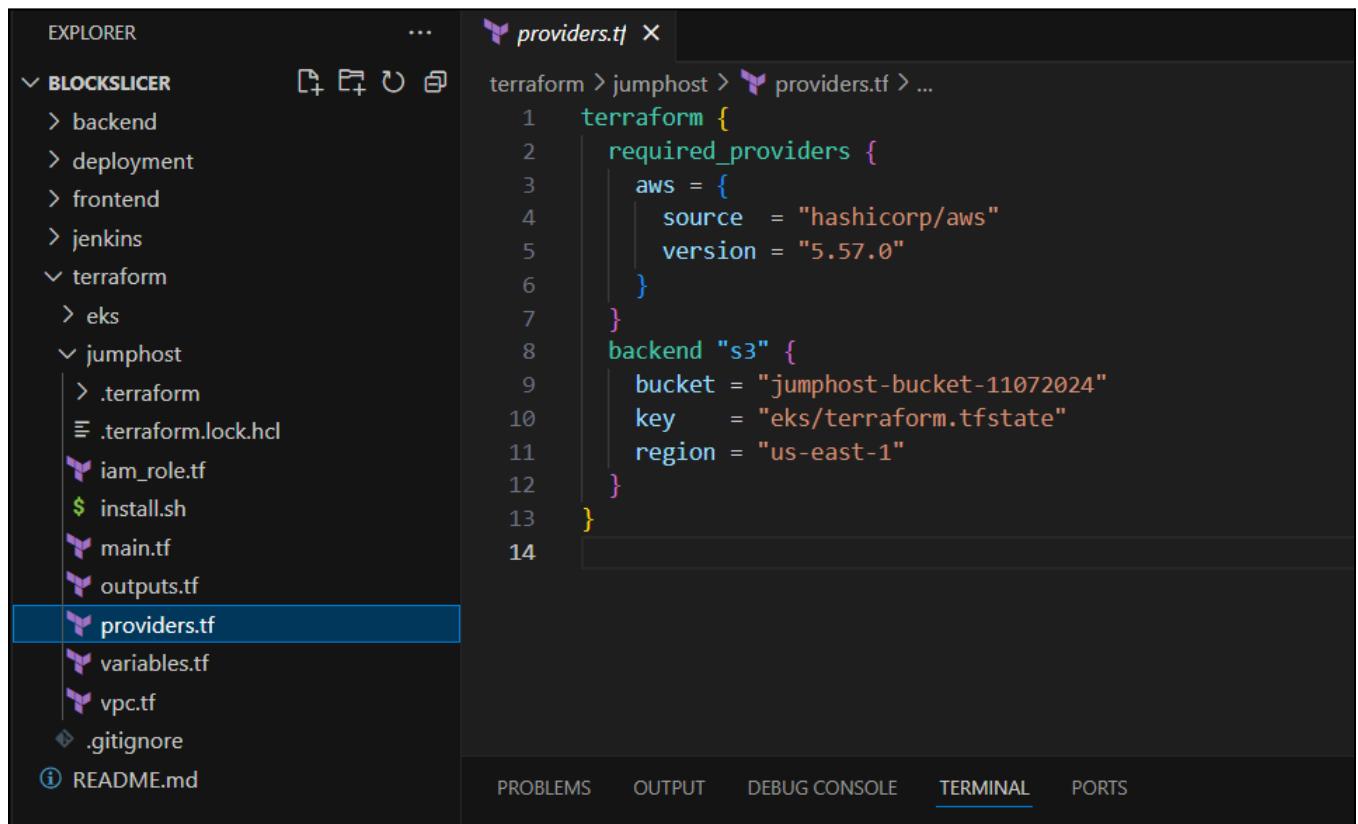
Name	AWS Region	IAM Access Analyzer	Creation date
eks-bucket-11072024	US East (N. Virginia) us-east-1	View analyzer for us-east-1	July 11, 2024, 10:01:28 (UTC+05:30)
jumphost-bucket-11072024	US East (N. Virginia) us-east-1	View analyzer for us-east-1	July 12, 2024, 06:44:06 (UTC+05:30)

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

The screenshot shows a VS Code interface with the 'BLOCKSLICER' folder open in the Explorer sidebar. The 'providers.tf' file is selected and open in the main editor area. The code defines an AWS provider with a backend S3 configuration pointing to the 'eks-bucket-11072024' bucket in 'us-east-1' region. The 'providers.tf' file is highlighted in the Explorer sidebar. The bottom of the screen shows the terminal tab with the command 'PS C:\Files and Folders\workspace\blockslsicer>'.

```
terraform > eks > providers.tf > backend "s3" > key
1  terraform {
2    required_providers {
3      aws = {
4        source  = "hashicorp/aws"
5        version = "5.57.0"
6      }
7    }
8    backend "s3" {
9      bucket = "eks-bucket-11072024"
10     key    = "eks/terraform.tfstate"
11     region = "us-east-1"
12   }
13 }
14
15 # provider "aws" {
16 #   region = local.region
17 # }
```

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

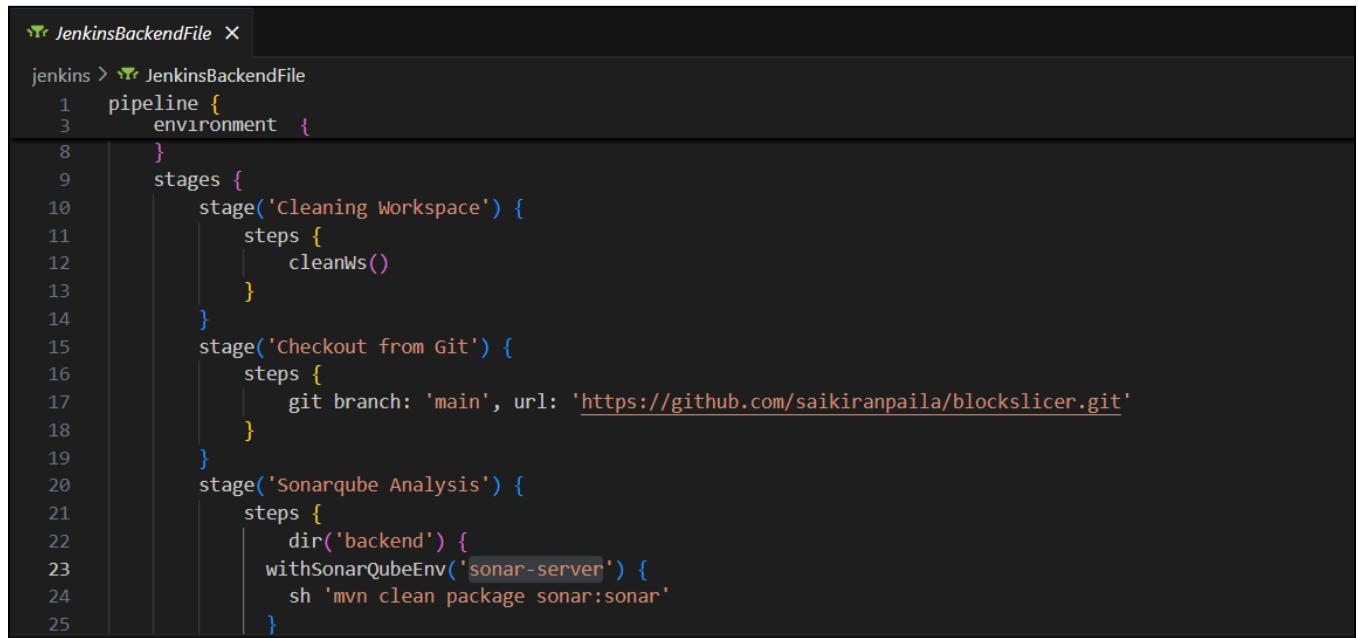


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure for 'BLOCKSLICER' with files like 'backend', 'deployment', 'frontend', 'jenkins', 'terraform' (which contains 'eks', 'jumphost', and several Terraform configuration files), and 'README.md'. The file 'providers.tf' is highlighted with a blue selection bar. The main editor area shows the content of 'providers.tf':

```
1 terraform {
2   required_providers {
3     aws = {
4       source  = "hashicorp/aws"
5       version = "5.57.0"
6     }
7   }
8   backend "s3" {
9     bucket = "jumphost-bucket-11072024"
10    key    = "eks/terraform.tfstate"
11    region = "us-east-1"
12  }
13 }
14 }
```

Below the editor, the status bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined in blue), and PORTS.

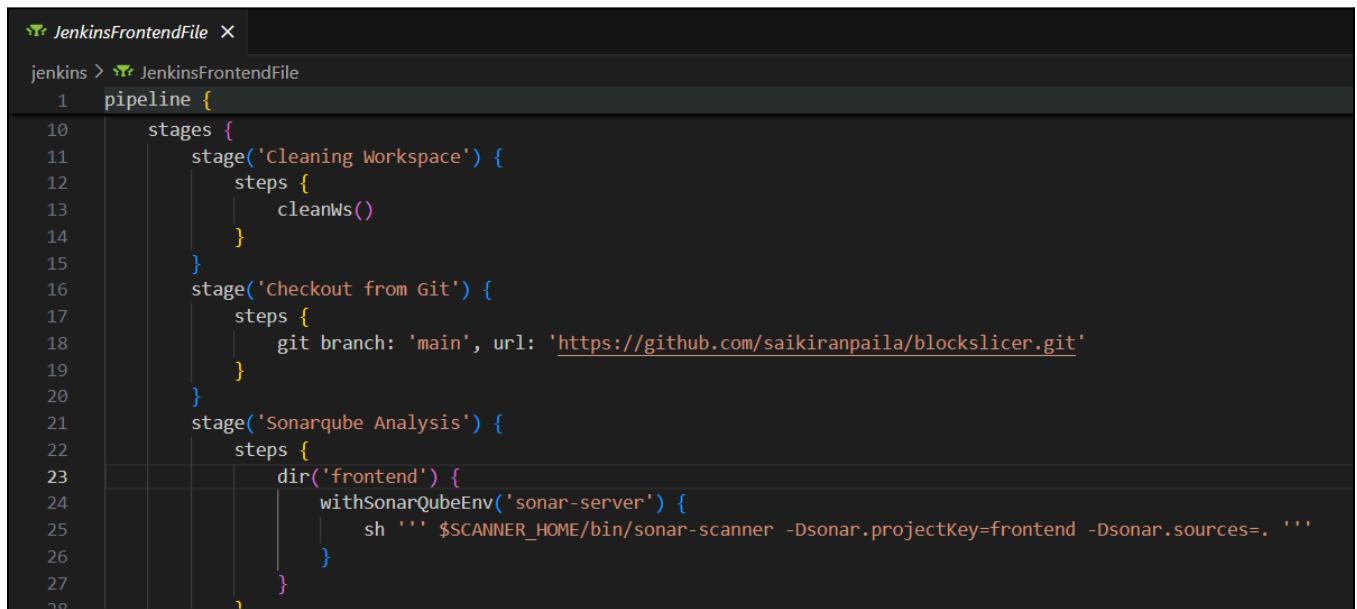
Step 5: Now go to ./blockslicer/jenkins/JenkinsBackendFile and replace the git url with your github url.



The screenshot shows the Jenkinsfile 'JenkinsBackendFile' in the Jenkins pipeline. The code defines a pipeline with stages for cleaning the workspace, checking out code from a GitHub repository, and performing SonarQube analysis.

```
1 jenkins > JenkinsBackendFile
2   pipeline {
3     environment {
4   }
5     stages {
6       stage('Cleaning Workspace') {
7         steps {
8           cleanWs()
9         }
10      }
11      stage('Checkout from Git') {
12        steps {
13          git branch: 'main', url: 'https://github.com/saikiranpaila/blockslicer.git'
14        }
15      }
16      stage('Sonarqube Analysis') {
17        steps {
18          dir('backend') {
19            withSonarQubeEnv('sonar-server') {
20              sh 'mvn clean package sonar:sonar'
21            }
22          }
23        }
24      }
25    }
26  }
27 }
```

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



```
JenkinsFrontendFile
jenkins > JenkinsFrontendFile
1 pipeline {
10     stages {
11         stage('Cleaning Workspace') {
12             steps {
13                 cleanWs()
14             }
15         }
16         stage('Checkout from Git') {
17             steps {
18                 git branch: 'main', url: 'https://github.com/saikiranpaila/blockslicer.git'
19             }
20         }
21         stage('Sonarqube Analysis') {
22             steps {
23                 dir('frontend') {
24                     withSonarQubeEnv('sonar-server') {
25                         sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectKey=frontend -Dsonar.sources=. '''
26                     }
27                 }
28             }
29         }
30     }
31 }
```

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-0b5detab38ct97713]
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.

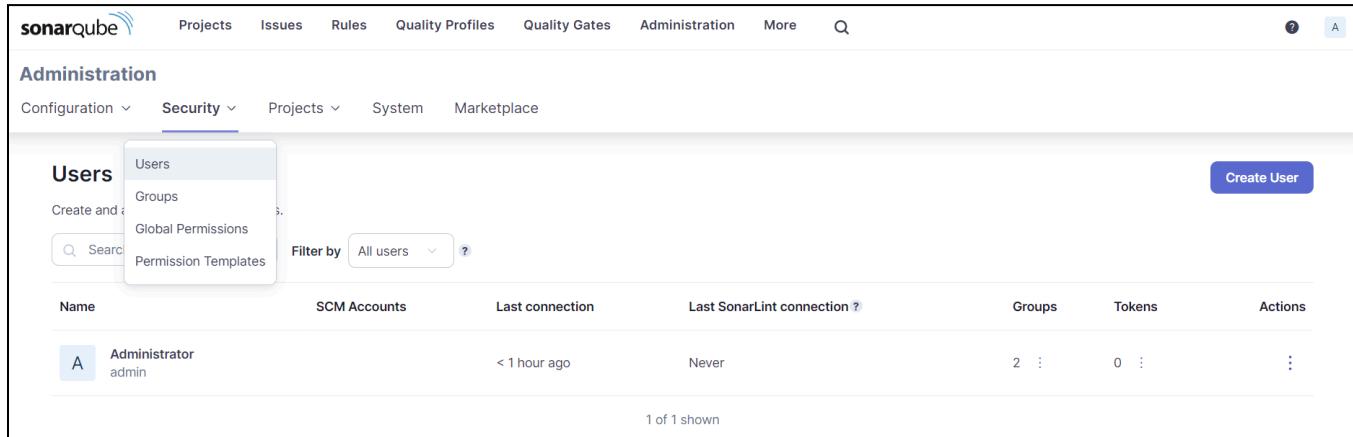
- git version
- systemctl status jenkins
- terraform version
- mvn --version
- systemctl status docker
- kubectl version
- helm version
- trivy version
- docker ps (sonarqube up and running)

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

Configuring the Jenkins Server and Sonarqube.

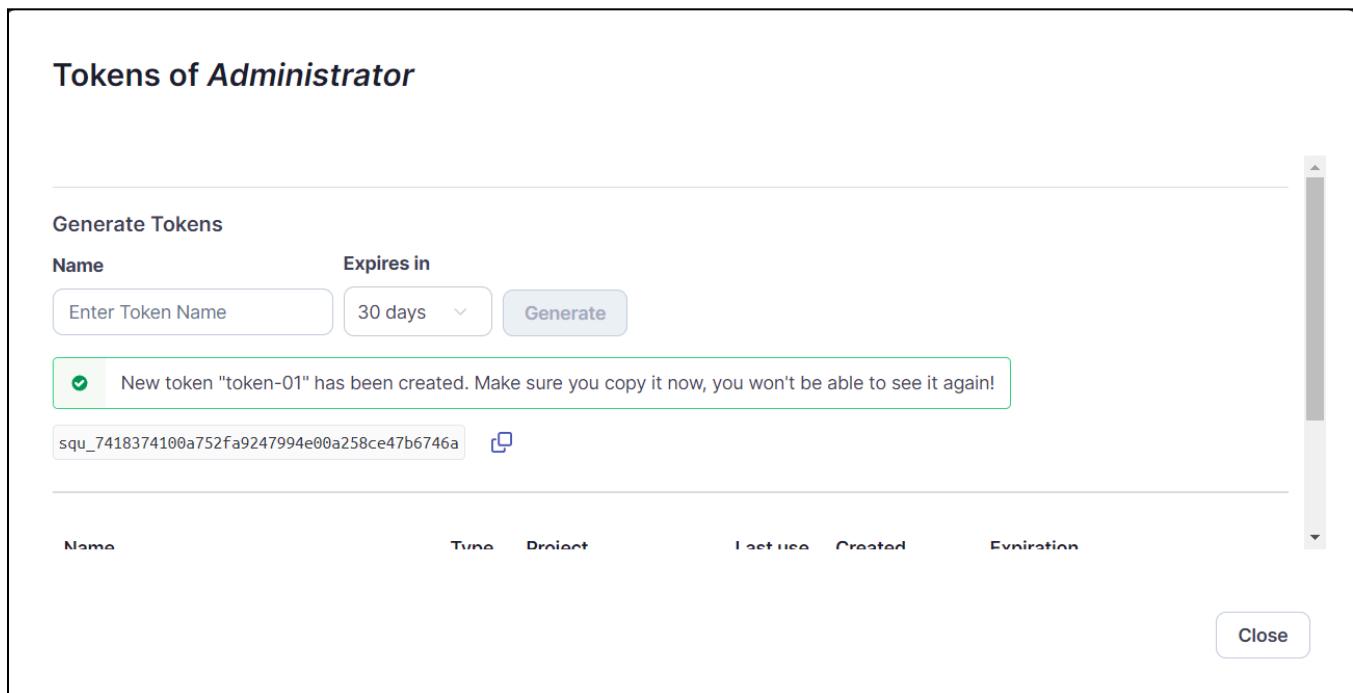
Step 1: Take the public ip of the jump host server and hit it with port 9000 to access sonarqube. The default credentials of sonarqube are username: **admin** and password: **admin**.

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



The screenshot shows the SonarQube administration interface. In the top navigation bar, 'Administration' is selected under 'Security'. On the left, a sidebar menu has 'Users' selected. The main area displays a table of users. One user, 'Administrator' (username: admin), is listed. The table columns include Name, SCM Accounts, Last connection, Last SonarLint connection, Groups, Tokens, and Actions. A 'Create User' button is located in the top right corner of the user list area. A search bar and filter dropdown are also present.

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.



The screenshot shows the 'Tokens of Administrator' dialog. It includes fields for 'Name' (with placeholder 'Enter Token Name') and 'Expires in' (set to '30 days'). A 'Generate' button is next to the expiration field. A success message box is displayed, stating 'New token "token-01" has been created. Make sure you copy it now, you won't be able to see it again!' with a checkmark icon. Below this, a token value 'squ_7418374100a752fa9247994e00a258ce47b6746a' is shown with a copy icon. At the bottom, there is a table with columns: Name, Type, Project, Last use, Created, and Expiration. A 'Close' button is in the bottom right corner.

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_jump_host_server>:8080/sonarqube-webhook/`

The screenshot shows a 'Create Webhook' dialog box. It has fields for 'Name' (jenkins) and 'URL' (http://44.201.22.48:8080/sonarqube-webhook/). A note below the URL explains that it's a server endpoint for webhook payloads. There's a 'Secret' field which is currently empty. At the bottom are 'Create' and 'Cancel' buttons.

em

Select an item from each

Create Webhook

Name *

jenkins

URL *

http://44.201.22.48:8080/sonarqube-webhook/

Server endpoint that will receive the webhook payload, for example: "http://my_server/foo". If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example: "https://myLogin:myPassword@my_server/foo"

Secret

If provided, secret will be used as the key to generate the HMAC hex (lowercase) digest value in the 'X-Sonar-Webhook-HMAC-SHA256' header.

Create Cancel

on purposes only

Upgrading to newer versions of SonarQube, and there is no support for migrating your data ou

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.

The screenshot shows the 'Global credentials (unrestricted)' page. It has a table with columns for ID, Name, Kind, and Description. A note at the bottom says 'This credential domain is empty. How about adding some credentials?'. There are icons for S, M, and L.

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
This credential domain is empty. How about adding some credentials?			

Icon: S M L

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

The screenshot shows the Jenkins 'New credentials' page under 'Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Secret text'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains a redacted token. The 'ID' field is set to 'sonar-token'. The 'Description' field is set to 'sonar token'. A blue border highlights the 'Description' field. A 'Create' button is at the bottom.

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

The screenshot shows the Jenkins 'New credentials' page under 'Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Secret text'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains a redacted GitHub token. The 'ID' field is set to 'git-token'. The 'Description' field is set to 'github token'. A blue border highlights the 'Secret' field. A 'Create' button is at the bottom.

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

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret:

ID: ACCOUNT_ID

Description: aws account id

Create

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

Repositories (2)						
	Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
<input checked="" type="radio"/>	backend	.dkr.ecr.us-east-1.amazonaws.com/backend	July 12, 2024, 09:50:18 (UTC+05.5)	Disabled	Manual	AES-256
<input checked="" type="radio"/>	frontend	.dkr.ecr.us-east-1.amazonaws.com/frontend	July 12, 2024, 09:50:29 (UTC+05.5)	Disabled	Manual	AES-256

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.

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

New credentials

Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret:

ID: ECR_REPO_02

Description: ecr front end repo

Create

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

Global credentials (unrestricted)				+ Add Credentials
ID	Name	Kind	Description	
sonar-token	sonar token	Secret text	sonar token	
git-token	github token	Secret text	github token	
ACCOUNT_ID	aws account id	Secret text	aws account id	
ECR_REPO_01	ecr backend repo	Secret text	ecr backend repo	
ECR_REPO_02	ecr front end repo	Secret text	ecr front end repo	

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

Dashboard > Manage Jenkins > Tools
SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name: sonarscanner

Install automatically:

Install from Maven Central

Version: SonarQube Scanner 6.1.0.4477

Add Installer

Add SonarQube Scanner

Save Apply

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>

Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.

▼

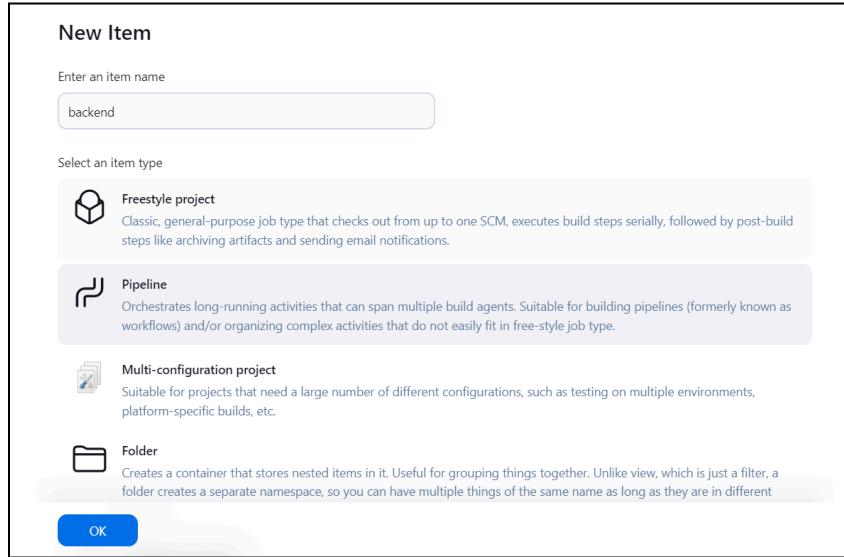
[+ Add ▾](#)

[Advanced ▾](#)

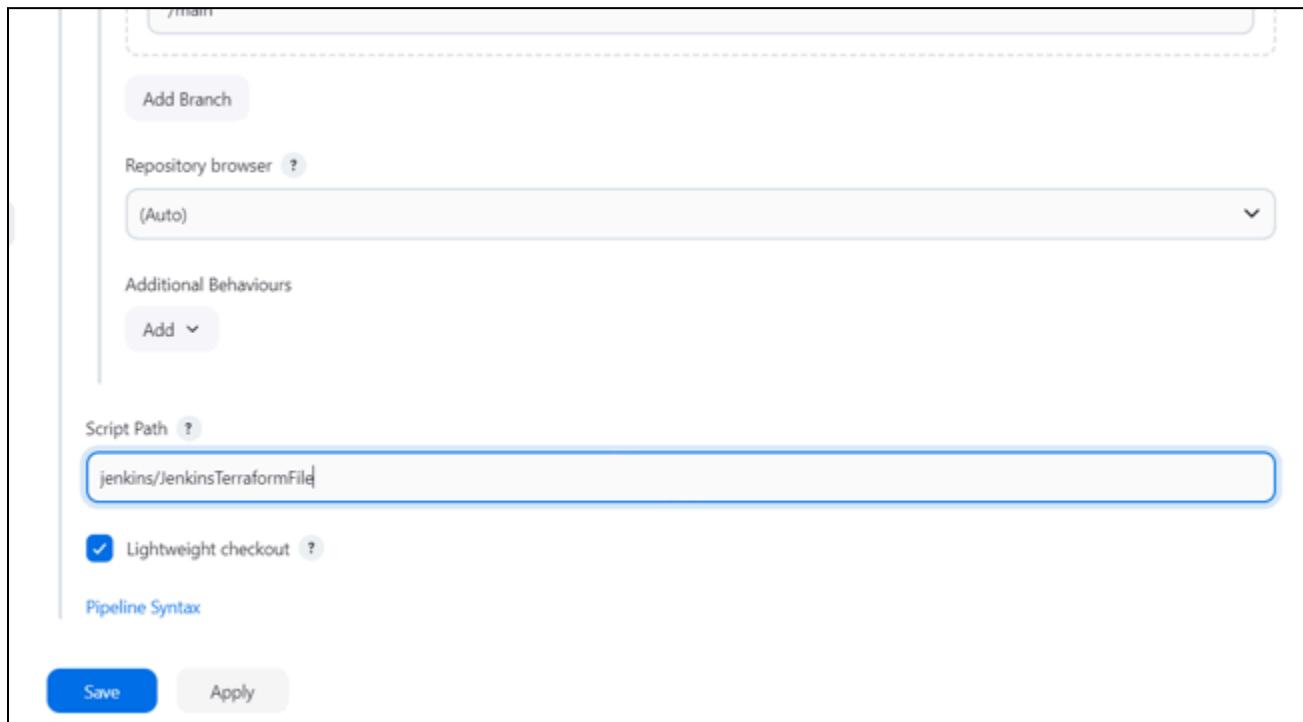
[Save](#) [Apply](#)

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.

All	+						
S	W	Name ↓	Last Success	Last Failure	Last Duration		
...	...	backend	N/A	N/A	N/A	...	▶
...	...	EKS	N/A	N/A	N/A	...	▶
...	...	frontend	N/A	N/A	N/A	...	▶

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

The screenshot shows the Jenkins dashboard with the following details:

- Left sidebar:** Includes links for New Item, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, and My Views. A "Build Queue" section indicates "No builds in the queue".
- Top header:** Shows the Jenkins logo, search bar ("Search (CTRL+K)"), user info ("admin"), and log out link.
- Main content area:** A table showing pipeline status:

All	+						
S	W	Name ↓	Last Success	Last Failure	Last Duration		
...	...	backend	N/A	N/A	N/A	...	▶
✓	...	EKS	4 min 22 sec #4	11 min #3	38 sec	...	▶
...	...	frontend	N/A	N/A	N/A	...	▶
- Bottom right:** REST API and Jenkins 2.468 links.

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

Jenkins

Search (CTRL+K)

admin log out

Dashboard >

+ New Item

Build History All +

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

Build Queue No builds in the queue.

Build Executor Status 0/2

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀️	backend	2 min 41 sec #1	N/A	57 sec
✓	☁️	EKS	6 min 31 sec #4	13 min #3	38 sec
✓	☀️	frontend	2 min 20 sec #1	N/A	1 min 17 sec

Icon: S M L

REST API Jenkins 2.468

The Jenkins dashboard displays a summary of recent builds. The 'backend' project has a success status with a duration of 2 min 41 sec. The 'EKS' project has a failure status with a duration of 6 min 31 sec. The 'frontend' project has a success status with a duration of 2 min 20 sec. There are no builds in the queue or active executors.

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

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration More Q

Create Project

My Favorites All

Filters

Quality Gate

Passed 2

Failed 0

Reliability

A 2

B 0

C 0

D 0

E 0

Security

A 0

Issues Search for projects... Perspective Overall Status Sort by Name 2 project(s) Passed

frontend PUBLIC Last analysis: 17 minutes ago - 2.2k Lines of Code - JavaScript, CSS, ...

Security 0 Reliability 2 Maintainability 25 Hotspots Reviewed 0.0% Coverage 0.0% Duplications 0.0%

game PUBLIC Last analysis: 17 minutes ago - 815 Lines of Code - Java, XML

Security 0 Reliability 13 Maintainability 22 Hotspots Reviewed 0.0% Coverage 0.0% Duplications 12.6%

2 of 2 shown

The SonarQube interface shows two projects: 'frontend' and 'game'. Both projects have passed quality gates. The 'frontend' project has 2.2k lines of code, while the 'game' project has 815 lines of code. The 'frontend' project has 25 maintainability issues, while the 'game' project has 22 maintainability issues. Coverage is 0.0% for both projects.

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.

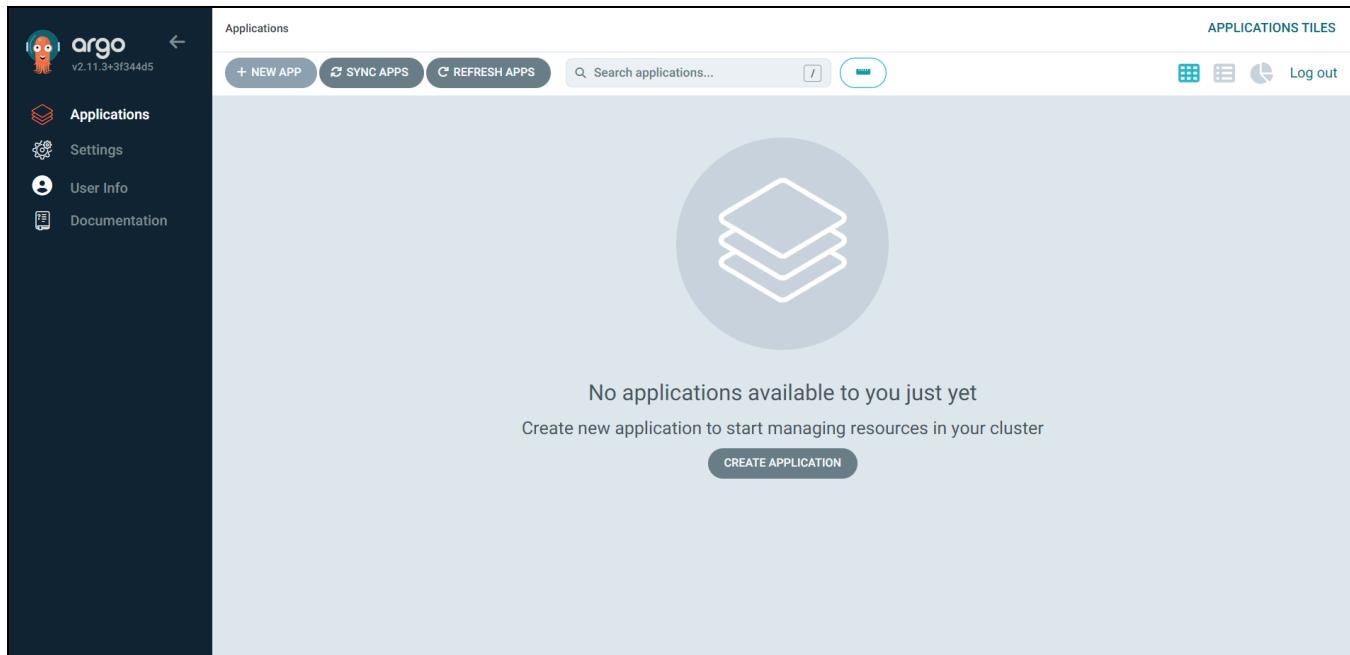
The screenshot shows the AWS EC2 Load Balancers page. It displays a table with one row of data. The columns are: Name, State, VPC ID, Availability Zones, and Type. The Name column contains the value 'a80fcf14de7ed49f69c6f7...', which is highlighted with a green border and the text 'DNS name copied' above it. The State column shows a dash (-). The VPC ID column shows 'vpc-0b3aca76c2909c6f4'. The Availability Zones column shows '2 Availability Zones'. The Type column shows 'classic'. At the top right of the table, there is a yellow button labeled 'Create load balancer'.

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

The screenshot shows the ArgoCD login interface. On the left, there is a large orange cartoon octopus character with a gear, set against a dark background with stars. To the right of the octopus, the word 'argo' is written in a stylized orange font. Below this, there are two input fields: 'Username' and 'Password', both currently empty. Below the password field is a blue 'SIGN IN' button. At the bottom right of the screen, the word 'argo' is repeated in a smaller font.

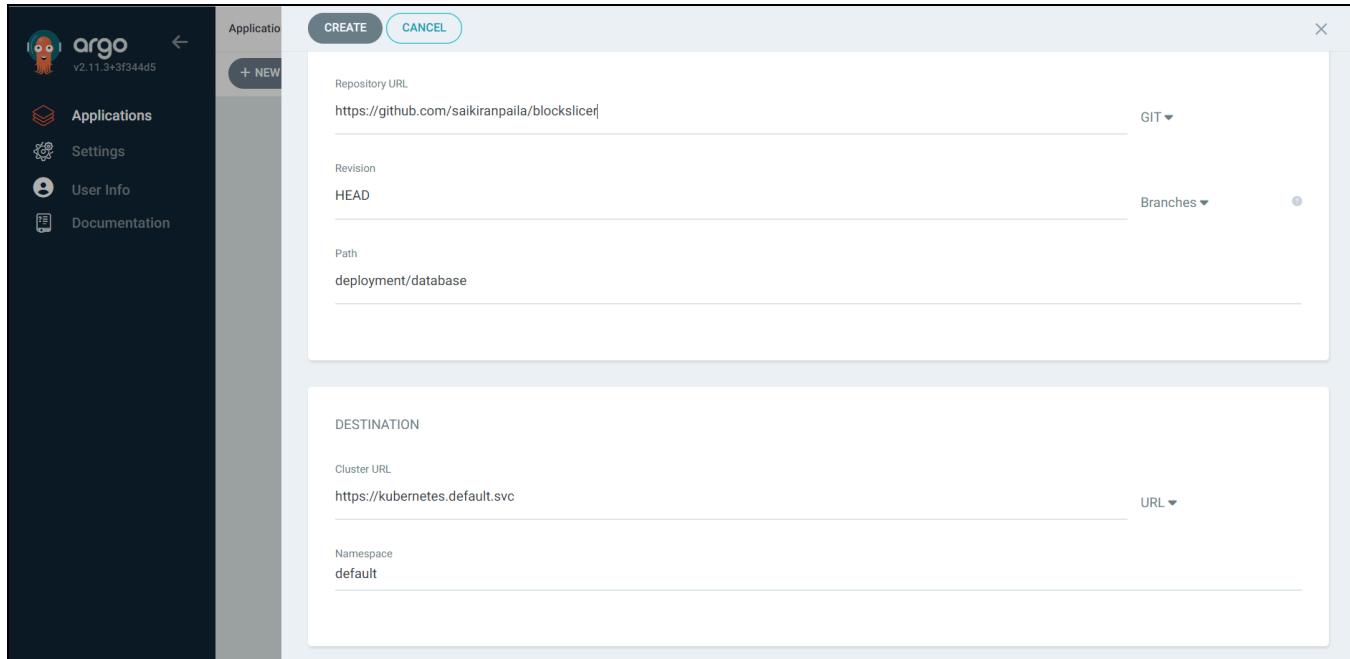
Step 7: Default username for ArgoCD is **admin** and password can be retrieved by running **kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d** on the jumphost server.

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**.

Application	Project	Status	Last Sync
backend	default	Synced	07/18/2024 09:20:04
database	default	Synced	07/18/2024 09:18:49

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.

The screenshot shows the Argo UI interface. On the left is a sidebar with navigation links: Applications, Settings, User Info, Documentation, SYNC STATUS, and HEALTH STATUS. The main area displays three application cards:

- backend**: Project: default, Labels: None, Status: Healthy Synced, Repository: https://github.com/saikiranpaila/blocksl..., Target R...: HEAD, Path: deployment/backend, Destination: in-cluster, Namespace: default, Created ...: 07/18/2024 09:20:04 (a minute ago), Last Sync: 07/18/2024 09:20:04 (a minute ago). Buttons: SYNC, REFRESH, DELETE.
- database**: Project: default, Labels: None, Status: Healthy Synced, Repository: https://github.com/saikiranpaila/blocksl..., Target R...: HEAD, Path: deployment/database, Destination: in-cluster, Namespace: default, Created ...: 07/18/2024 09:18:48 (3 minutes ago), Last Sync: 07/18/2024 09:18:49 (3 minutes ago). Buttons: SYNC, REFRESH, DELETE.
- frontend**: Project: default, Labels: None, Status: Progressing Synced, Repository: https://github.com/saikiranpaila/blocksl..., Target R...: HEAD, Path: deployment/frontend, Destination: in-cluster, Namespace: default, Created ...: 07/18/2024 09:21:21 (a few seconds ago), Last Sync: 07/18/2024 09:21:21 (a few seconds ago). Buttons: SYNC, REFRESH, DELETE.

At the top right, there are application tiles and a log out button. The bottom right corner shows "Sort: name ▾ Items per page: 10 ▾".

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.

Applications

APPLICATIONS TILES

prometheus

- Project: default
- Labels:
- Status: Progressing Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/metrics/prometheus
- Destination: in-cluster
- Namespace: metrics
- Created: 07/18/2024 09:22:36 (a few seconds ago)
- Last Sync: 07/18/2024 09:22:39 (a few seconds ago)

database

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/database
- Destination: in-cluster
- Namespace: default
- Created: 07/18/2024 09:18:48 (5 minutes ago)
- Last Sync: 07/18/2024 09:18:49 (5 minutes ago)

frontend

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/frontend
- Destination: in-cluster
- Namespace: default
- Created: 07/18/2024 09:21:21 (3 minutes ago)
- Last Sync: 07/18/2024 09:21:21 (3 minutes ago)

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.

backend

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/backend
- Destination: in-cluster
- Namespace: default
- Created: 07/18/2024 09:20:04 (4 minutes ago)
- Last Sync: 07/18/2024 09:20:04 (4 minutes ago)

database

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/database
- Destination: in-cluster
- Namespace: default
- Created: 07/18/2024 09:18:48 (5 minutes ago)
- Last Sync: 07/18/2024 09:18:49 (5 minutes ago)

frontend

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/frontend
- Destination: in-cluster
- Namespace: default
- Created: 07/18/2024 09:21:21 (3 minutes ago)
- Last Sync: 07/18/2024 09:21:21 (3 minutes ago)

grafana

- Project: default
- Labels:
- Status: Progressing Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/grafana
- Destination: in-cluster
- Namespace: metrics
- Created: 07/18/2024 09:22:36 (a few seconds ago)
- Last Sync: 07/18/2024 09:22:39 (a few seconds ago)

prometheus

- Project: default
- Labels:
- Status: Healthy Synced
- Repository: https://github.com/saikiranpala/blocksl...
- Target Ref: HEAD
- Path: deployment/metrics/prometheus
- Destination: in-cluster
- Namespace: metrics
- Created: 07/18/2024 09:22:36 (a few seconds ago)
- Last Sync: 07/18/2024 09:22:39 (a few seconds ago)

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.

Project	Name	Source	Destination	HEAD	Synced
default	backend	https://github.com/saikiranpaila/blockslizer/deployment/backend	in-cluster/default	HEAD	Healthy Synced
default	database	https://github.com/saikiranpaila/blockslizer/deployment/database	in-cluster/default	HEAD	Healthy Synced
default	elastic	https://github.com/saikiranpaila/blockslizer/deployment/logging/elastics...	in-cluster/logging	HEAD	Healthy Synced
default	fluent-bit	https://github.com/saikiranpaila/blockslizer/deployment/logging/fluent-bit	in-cluster/logging	HEAD	Healthy Synced
default	frontend	https://github.com/saikiranpaila/blockslizer/deployment/frontend	in-cluster/default	HEAD	Healthy Synced
default	grafana	https://github.com/saikiranpaila/blockslizer/deployment/metrics/grafana	in-cluster/metrics	HEAD	Healthy Synced
default	ingress	https://github.com/saikiranpaila/blockslizer/deployment/ingress	in-cluster/default	HEAD	Healthy Synced
default	kibana	https://github.com/saikiranpaila/blockslizer/deployment/logging/kibana	in-cluster/logging	HEAD	Healthy Synced
default	prometheus	https://github.com/saikiranpaila/blockslizer/deployment/metrics/promet...	in-cluster/metrics	HEAD	Healthy Synced

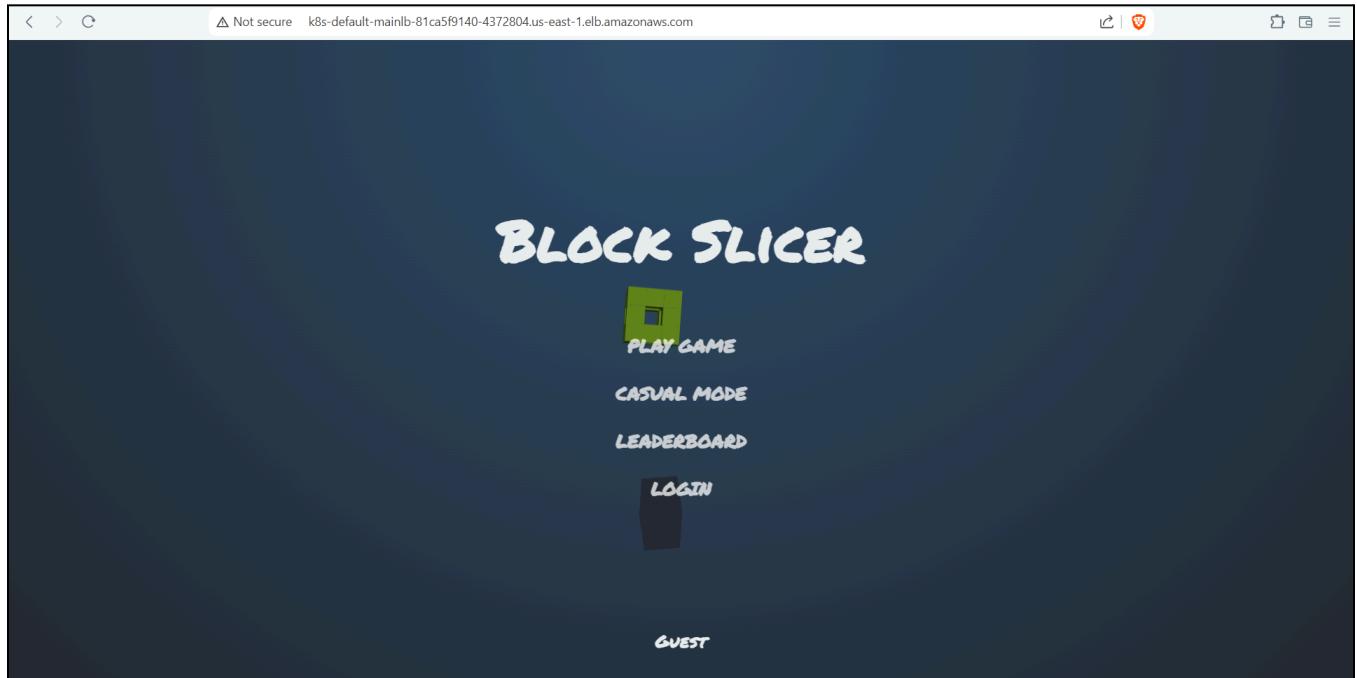
Step 18: If all the apps are in sync, you have successfully deployed the apps.

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

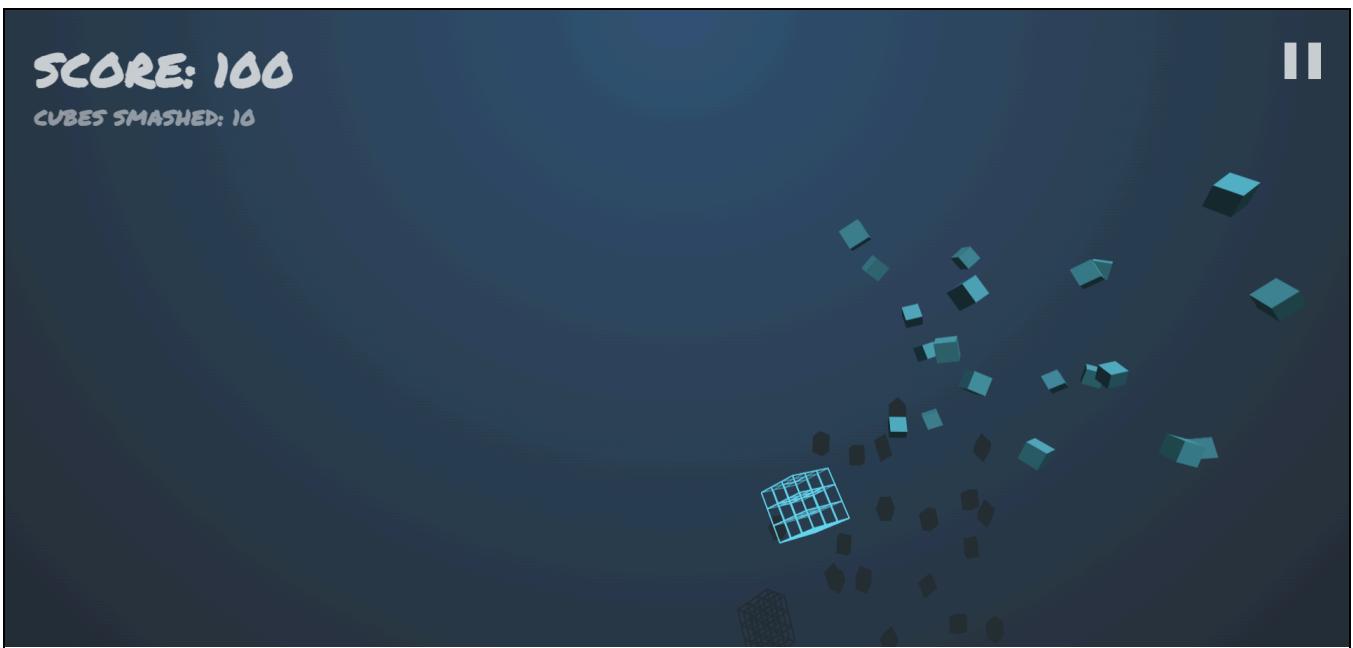
Step 1: Once you have deployed ingress in ArgoCD, you can find three Load Balancers in the AWS console.

Load balancers (4)						
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.						
<input type="text"/> Filter load balancers						
	Name	DNS name	State	VPC ID	Availability Zones	Type
<input type="checkbox"/>	a80fcf14de7ed49f69c...	a80fcf14de7ed49f69c6f7...	-	vpc-0b3aca76c2909c6f4	2 Availability Zones	classic
<input type="checkbox"/>	k8s-logging-loggingl...	k8s-logging-loggingl-77fe...	Provisioning.	vpc-0b3aca76c2909c6f4	2 Availability Zones	application
<input type="checkbox"/>	k8s-metrics-metricsl-4...	k8s-metrics-metricsl-464a...	Provisioning.	vpc-0b3aca76c2909c6f4	2 Availability Zones	application
<input type="checkbox"/>	k8s-default-mainlb-81...	k8s-default-mainlb-81ca5...	Provisioning.	vpc-0b3aca76c2909c6f4	2 Availability Zones	application

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.

RANK	NAME	SCORE	BLOCKS SLICED
1	KIRAN	120	10
2	SHIVAM	100	10
3	RAJESH	90	9
4	RAJU	90	9
5	RAVI)	70	7
6	RAVI	70	7
7	SURESH	20	2

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-55/bcc5/8-8pr6h  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 | gamerscredentials | table | postgres
public | gamersscore    | table | postgres
(2 rows)

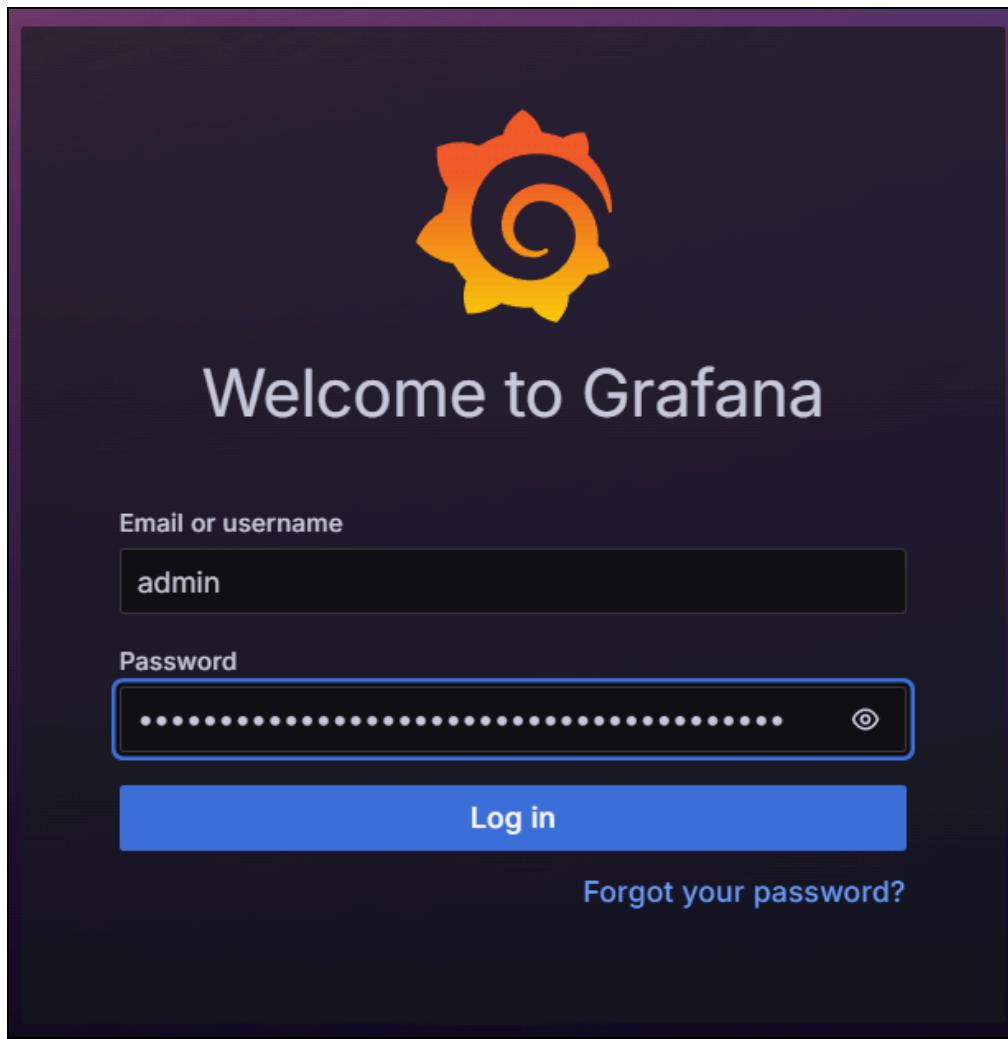
game=# select * from gamersscore;
 userid | blockssliced | score
-----+-----+-----+
 kirran |          10 |     100
(1 row)
```

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.

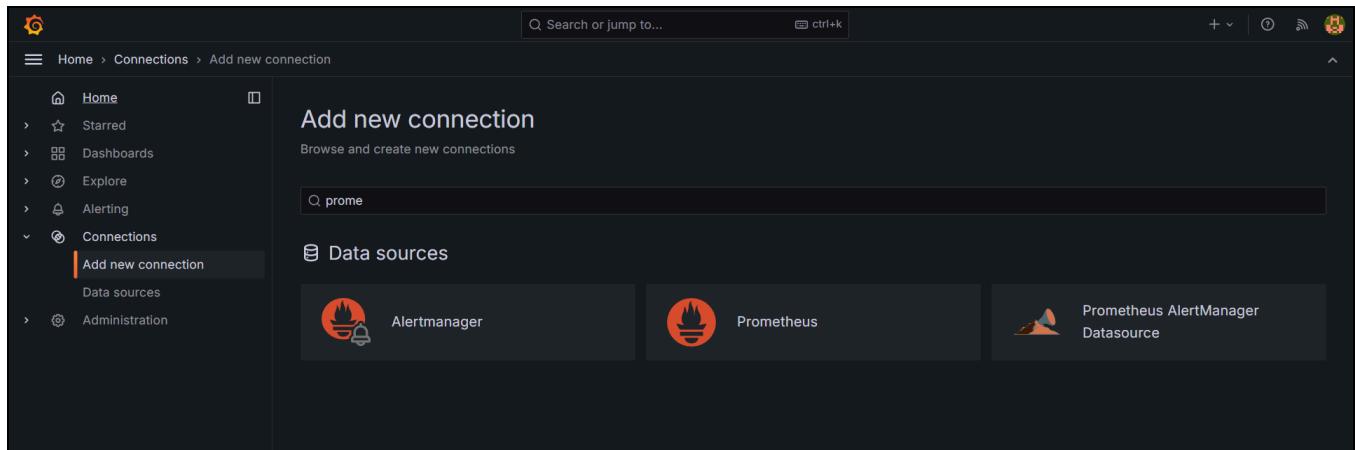
Step 7: Now, to check the metrics of the kubernetes cluster, copy the metrics Load Balancer URL and hit it. You will be prompted to grafana login page.



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>.

The screenshot shows the 'Prometheus' data source configuration in Grafana. At the top right, there are 'From' (Grafana Labs), 'Signature' (Core), and a blue 'Add new data source' button. Below the header, there's a brief description of Prometheus as an open-source time series database & alerting system, with a 'Learn more' link. A navigation bar includes 'Overview' (which is underlined) and 'Version history'. The main content area is titled 'Prometheus Data Source - Native Plugin' and contains a note about Grafana's built-in support for Prometheus. It also provides a link to the official documentation: <http://docs.grafana.org/datasources/prometheus/>.

The screenshot shows the 'Connection' configuration page for the Prometheus data source. It features a single input field labeled 'Prometheus server URL *' with a placeholder value of 'http://prometheus-server'. To the right of the input field is a small ⓘ icon.

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

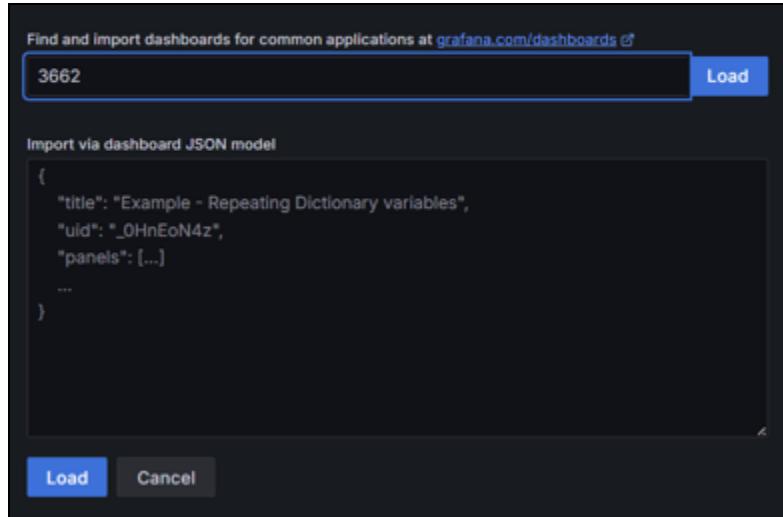
The screenshot shows the 'Dashboards' section in Grafana. At the top right, there is a 'New' button with a dropdown menu containing 'New dashboard', 'New folder', and 'Import'. Below the button, there's a search bar and filters for 'Search for dashboards and folders', 'Filter by tag', and 'Starred'. On the far right, there are icons for 'Sort' and other dashboard management options. A small orange decorative element is visible at the bottom center.

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

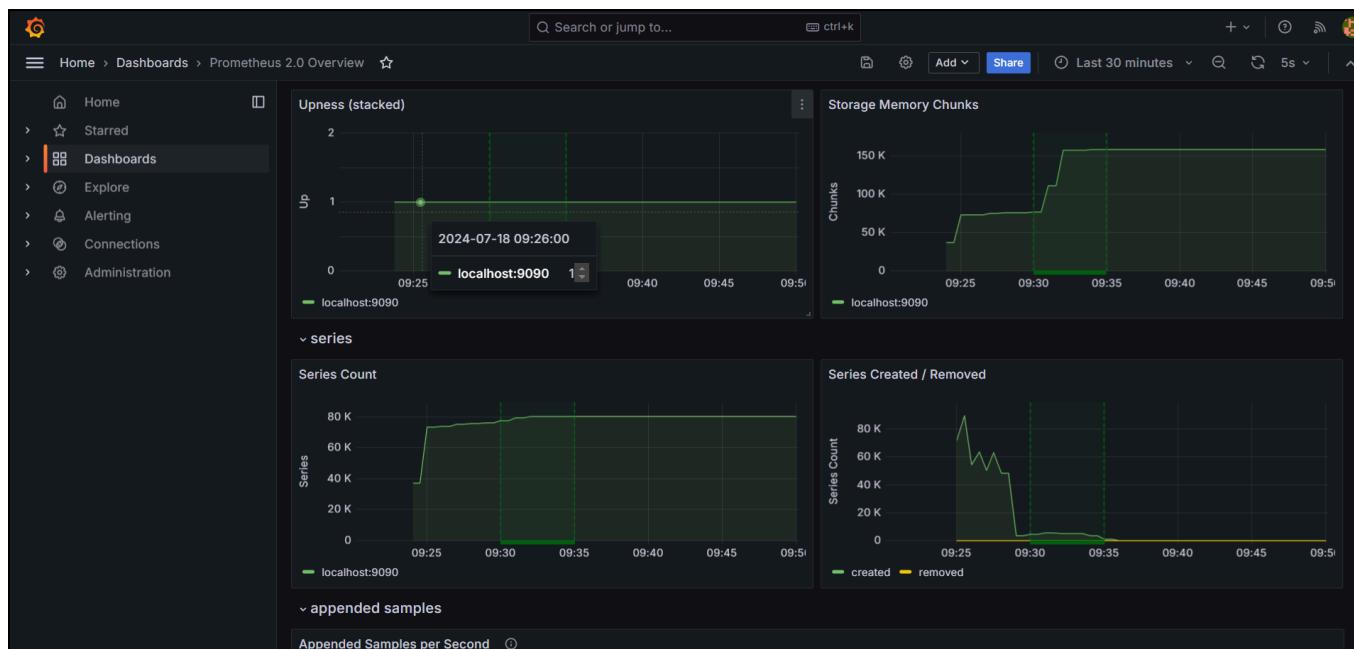
Dashboard	ID
k8s-addons-prometheus.json	19105
k8s-addons-trivy-operator.json	16337
k8s-system-api-server.json	15761
k8s-system-coredns.json	15762
k8s-views-global.json	15757
k8s-views-namespaces.json	15758
k8s-views-nodes.json	15759

k8s-views-pods.json	15760
----------------------------	--------------

Above are some of the example of kubernetes compatible dashboards.

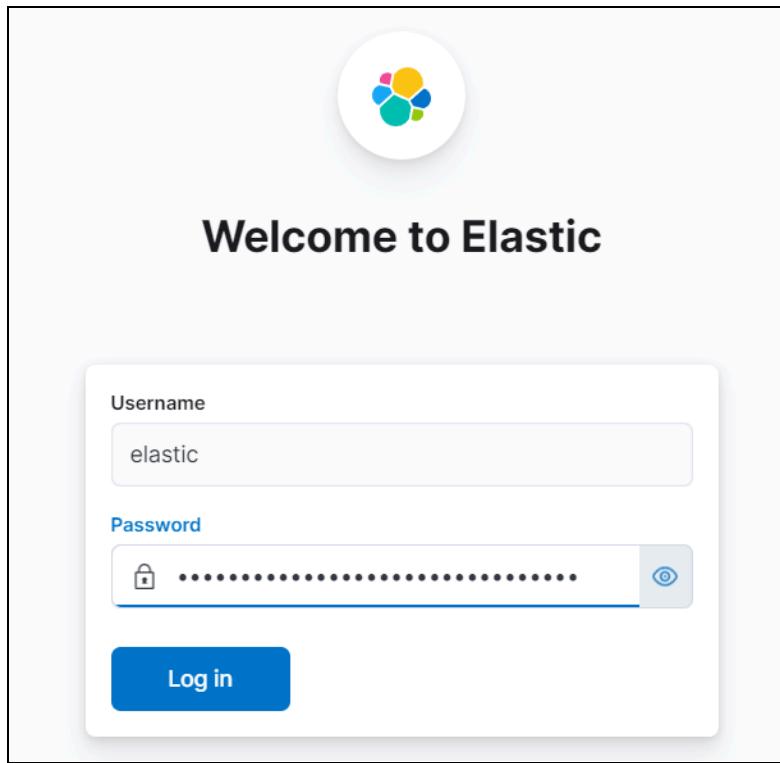


Step 13: On server metrics select prometheus and click on Import.



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=logging 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.

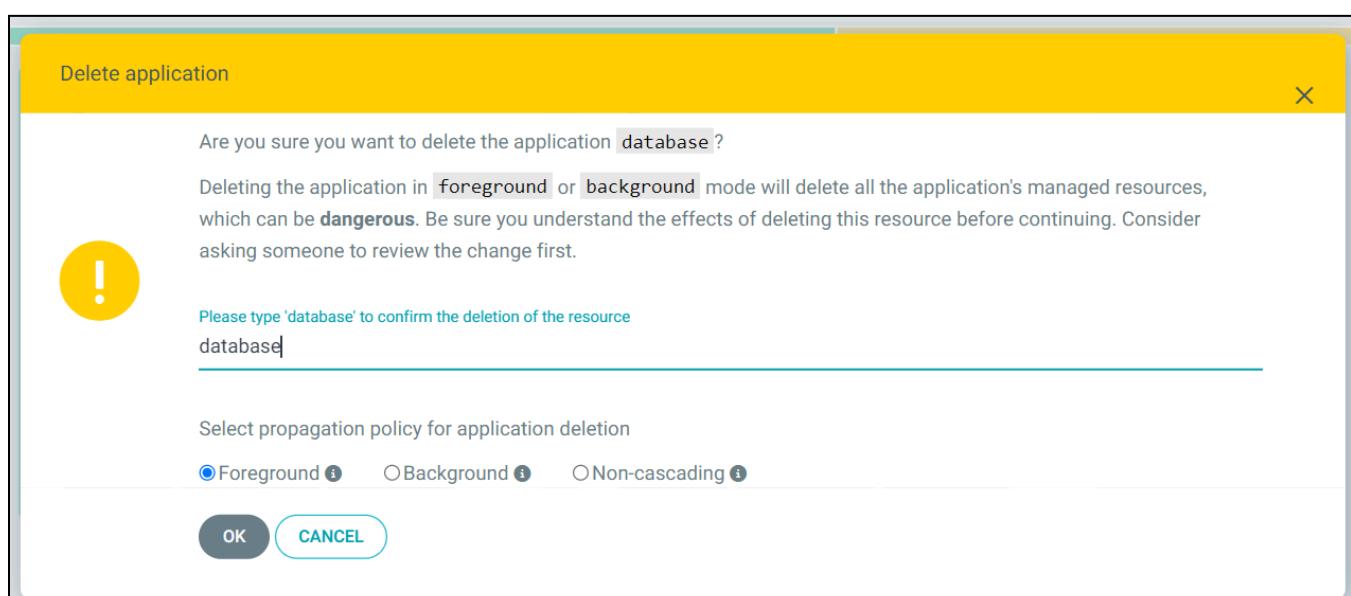
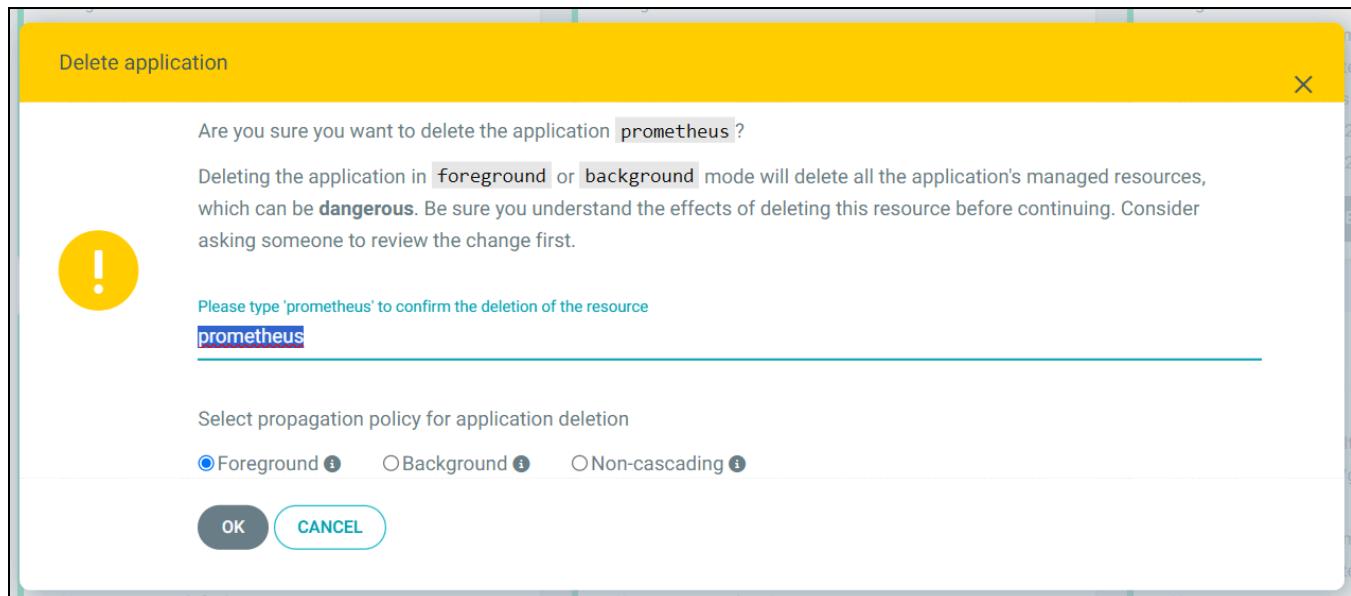
Step 17: On the left side menu click on Dashboard and click on create data view. That will prompt you to a data view creation page, give some name for data view, set the index pattern to ** if you want to gather all the logs and according to the timestamp order. Click on Save data view to kibana.

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 one 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.

The screenshot shows the AWS EC2 Load balancers console. At the top, there is a header with the EC2 logo, a breadcrumb trail ('EC2 > Load balancers'), and a search bar labeled 'Filter load balancers'. Below the header is a table with columns: Name, DNS name, State, VPC ID, Availability Zones, and Type. A message 'No load balancers' is displayed, followed by the text 'You don't have any load balancers in us-east-1'. At the bottom of the page is a large orange button labeled 'Create load balancer'.

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.

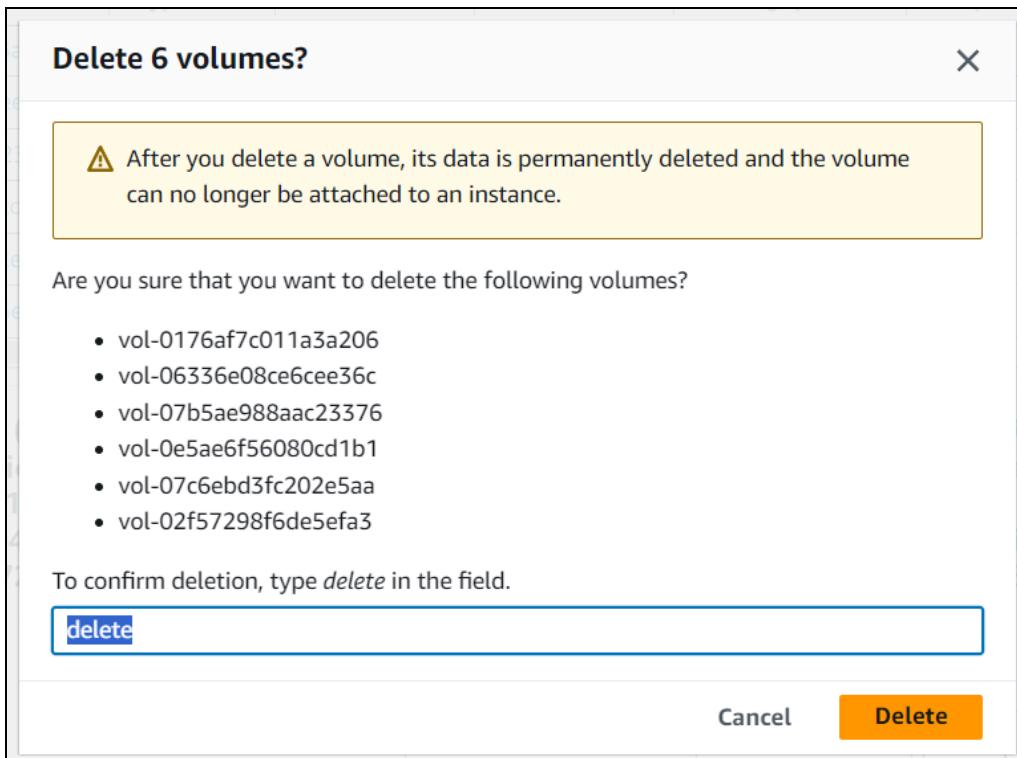
The screenshot shows the Jenkins Pipeline Stage View for an EKS pipeline. On the left, there is a sidebar with options: Status (selected), Changes, Build with Parameters, Configure, Delete Pipeline, Full Stage View, Stages, Rename, and Pipeline Syntax. The main area shows a table titled 'Stage View' with the following data:

	Declarative: Checkout SCM	Checkout from Git	Terraform version	Terraform init	Terraform validate	Terraform plan	Terraform apply/destroy
#5 Jul 18 10:23	235ms	204ms	469ms	18s	4s	5s	7min 54s
#4 Jul 18 09:09	215ms	198ms	322ms	20s	4s	6s	7min 58s
	233ms	191ms	323ms	19s	4s	6s	6s

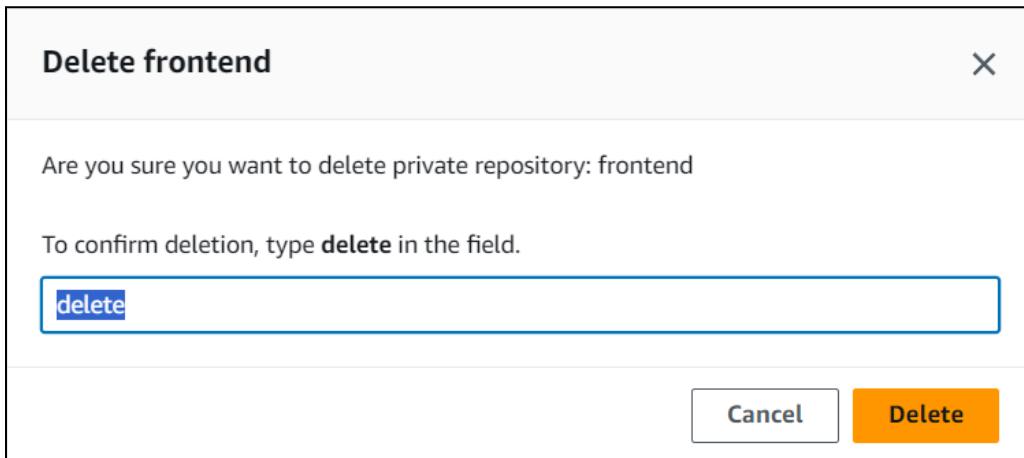
Step 5: Now head to AWS EC2 console and under volumes section delete all the volumes.

The screenshot shows the AWS EC2 Volumes console. At the top, there is a header with the EC2 logo, a breadcrumb trail ('EC2 > Volumes'), and a search bar labeled 'Search'. Below the header is a table with columns: Name, Volume ID, Type, Size, IOPS, Throughput, Snapshot, and Created. The table contains the following data:

Name	Volume ID	Type	Size	IOPS	Throughput	Snapshot	Created
staging-blocks...	vol-0176af7c011a3a206	gp2	1 GiB	100	-	-	2024/07/18 09:19 GMT+5...
staging-blocks...	vol-06336e08ce6cee36c	gp2	30 GiB	100	-	-	2024/07/18 10:21 GMT+5...
staging-blocks...	vol-07b5ae988aac23376	gp2	1 GiB	100	-	-	2024/07/18 09:18 GMT+5...
staging-blocks...	vol-0e5ae6f56080cd1b1	gp2	30 GiB	100	-	-	2024/07/18 10:21 GMT+5...
staging-blocks...	vol-07c6ebd3fc202e5aa	gp2	1 GiB	100	-	-	2024/07/18 09:18 GMT+5...
staging-blocks...	vol-02f57298f6de5efa3	gp2	30 GiB	100	-	-	2024/07/18 10:21 GMT+5...



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.

