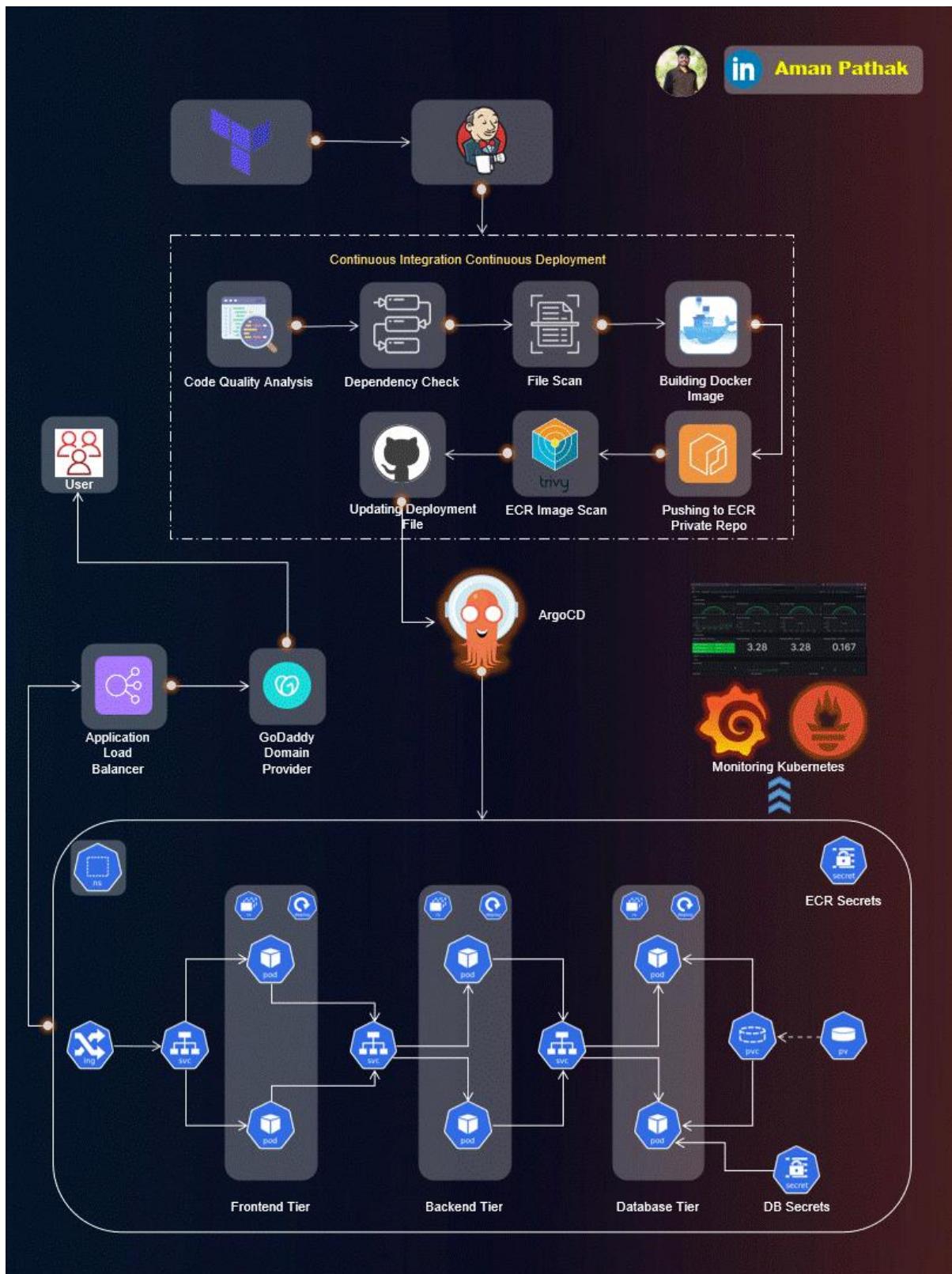


Advanced End-to-End DevSecOps Kubernetes Three-Tier Project using AWS EKS, ArgoCD, Prometheus, Grafana, and Jenkins



PHASE1: Infrastructure setup

PHASE2: Jenkins and Argocd setup

PHASE3: Monitoring setup

Phase1: Infrastructure Setup

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

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

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

Step2: Access Jenkins:

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

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

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

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

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

The screenshot shows the Jenkins 'Plugins' page under 'Manage Jenkins'. The search bar at the top contains 'pipeline: AWS'. Below the search bar, there are two listed plugins:

- AWS Credentials**: Version 231.v08a_59f17d742, Released 4 mo 8 days ago. Description: Allows storing Amazon IAM credentials within the Jenkins Credentials API. Store Amazon IAM access keys (AWSAccessKeyId and AWSSecretKey) within the Jenkins Credentials API. Also support IAM Roles and IAM MFA Token.
- Pipeline: AWS Steps**: Version 1.45, Released 4 mo 8 days ago. Description: This plugin adds Jenkins pipeline steps to interact with the AWS API.

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

The screenshot shows the Jenkins 'New credentials' page under 'Manage Jenkins > Credentials > Global credentials (unrestricted)'. A new AWS Credential is being created with the following details:

- Kind:** AWS Credentials
- Scope:** Global (Jenkins, nodes, items, all child items, etc)
- ID:** aws-cred
- Description:** (empty)
- Access Key ID:** (input field)

A 'Create' button is visible at the bottom left.

Step4: Install terraform in Jenkins

Goto to tools → Terraform installations → install from the local

The screenshot shows the Jenkins 'Terraform installations' page under 'Manage Jenkins > Tools'. A new Terraform installation is being configured with the following details:

- Name:** terraform
- Install directory:** /usr/bin/terraform
- Warning:** /usr/bin/terraform is not a directory on the Jenkins controller (but perhaps it exists on some agents)
- Install automatically:** (checkbox)

Buttons for 'Save' and 'Apply' are at the bottom.

Step 5: Create Jenkins job

Create pipeline Jenkins job and use the jenkinsfile

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

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

```

1 terraform {
2   required_version = ">= 1.9.3"
3   required_providers {
4     aws = {
5       source  = "hashicorp/aws"
6       version = ">= 5.49.0"
7     }
8   }
9   backend "s3" {
10    bucket = "my-eks-bucket-terraform"
11    region = "us-east-1"
12    key    = "eks/terraform.tfstate"
13    dynamodb_table = "dynamodb-state-lock"
14    encrypt  = true
15  }
16 }
17
18 provider "aws" {
19   region = var.aws_region
20 }

```

	Preparing	Git Pulling	Init	Validate	Action
Avg	375ms	1s	8s	1s	4min 25s
Aug 16 12:46	358ms	270ms	22s	3s	17min 38s
Aug 16 12:45	366ms	261ms	10s	3s	4s
#2 Aug 16 12:44	354ms	3s	573ms failed	111ms failed	110ms failed
#1 Aug 16 12:39	425ms	713ms	592ms failed	131ms failed	91ms failed

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

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

Step 7: configure the aws credentials in the jump server

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

Step 8: Update the Kubeconfig file

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

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

```
kubectl get nodes
```

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

Download the policy for the LoadBalancer prerequisite.

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

Step 10: Create the IAM policy using the below command

```
aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document
file://iam\_policy.json
```

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

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

Check the load balancer is created or not

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

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

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

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

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

Step 13: Create namespace for Argocd

```
kubectl create namespace argocd
```

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

```
kubectl get all -n argocd
```

```
kubectl get svc -n argocd
```

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

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

Step 15: Now access the Argocd with loadbalancer dns name

Default username: admin

Initial admin password is stored in

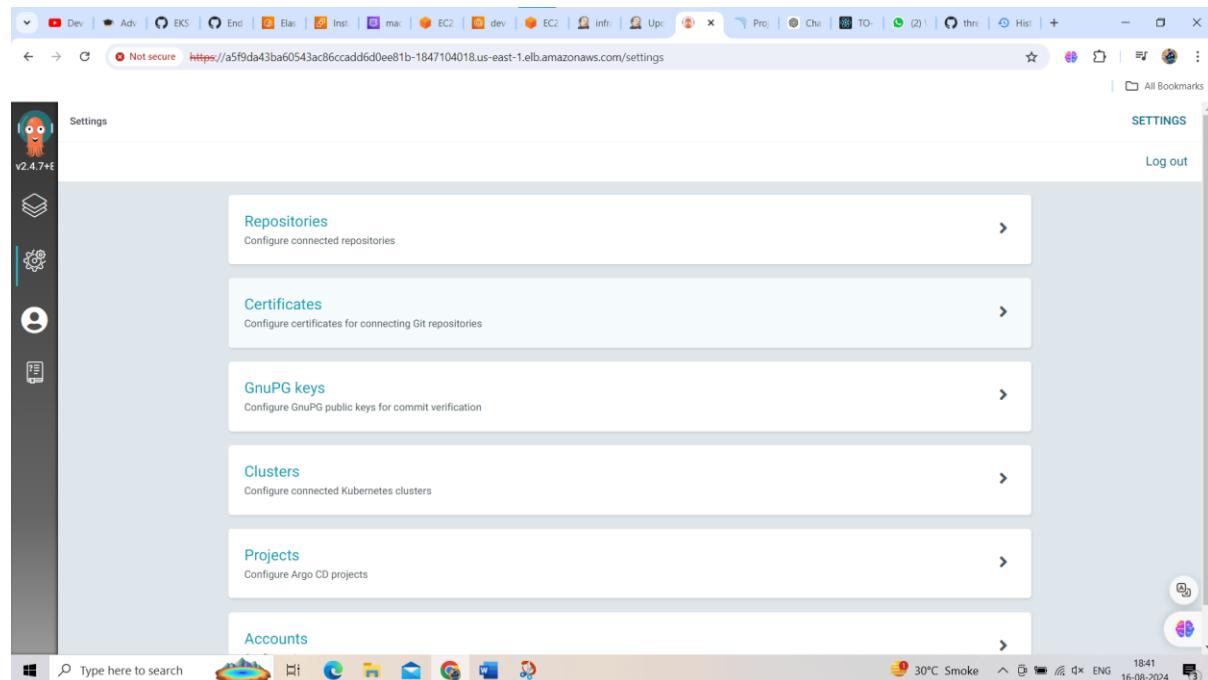
```
kubectl edit secret argocd-initial-admin-secret -n argocd
```

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

get the password and decode with base64

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

Copy the password and login to the argocd



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

Step 1: Sonarqube setup

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

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

Access SonarQube:

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

The default username and password are both admin.

SonarQube Projects Issues Rules Quality Profiles Quality Gates Administration

My Favorites All

Search for projects... A

Create Project +

Last analysis: 3 hours ago

Last analysis: 4 hours ago

2 of 2 shown

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Community Edition - v9.9.6 (build 92038) - [LGPL v3](#) - [Community](#) - [Documentation](#) - [Plugins - Web API](#)

Step2: Create the Token

Goto administration → security → users→ generate token

Name	Type	Project	Last use	Created	Expiration
sonar-token	User		3 hours ago	August 16, 2024	September 15, 2024

Copy the token and add this token in the Jenkins credentials

New credentials

Kind: Secret text

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

Secret:

ID: sonar-token

Description:

Create

Step3: create the projects for frontend and backend

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects...

Create a project

All fields marked with * are required

Project display name *

frontend-app

Up to 255 characters. Some scanners might override the value you provide.

Project key *

frontend-app

The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '-' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Main branch name *

main

The name of your project's default branch [Learn More](#)

Set Up

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Community Edition - v9.9.6 (build 92039) - [GPL v3](#) - [Community](#) - [Documentation](#) - [Plugins - Web API](#)

Click on setup and select manually and use the generated token

sonarcube Projects Issues Rules Quality Profiles Quality Gates Administration

Not secure 3.239.186.187:9000/dashboard?id=frontend-app&selectedTutorial=local

Search for projects.

Project Settings Project Information

Overview Issues Security Hotspots Measures Code Activity

Analyze your project
We initialized your project on SonarQube, now it's up to you to launch analyses!

1 Provide a token
 Generate a project token
 Use existing token
Existing token value

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your user account.

Continue

2 Run analysis on your project

Embedded database should be used for evaluation purposes only
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Community Edition - v9.6 (build 92038) - [GPL v3](#) - [Community](#) - [Documentation](#) - [Plugins - Web API](#)

30°C Haze 18:58 16-08-2024

sonarcube Projects Issues Rules Quality Profiles Quality Gates Administration

Not secure 3.239.186.187:9000/dashboard?id=frontend-app&selectedTutorial=local

Search for projects.

Project Settings Project Information

Overview Issues Security Hotspots Measures Code Activity

Analyze your project
We initialized your project on SonarQube, now it's up to you to launch analyses!

1 Provide a token
✓ squ_bb3d3b529730321dabd22b5f80c865aa7ab80777

2 Run analysis on your project
What option best describes your build?
 Maven Gradle .NET Other (for JS, TS, Go, Python, PHP, ...)

What is your OS?
 Linux Windows macOS

Download and unzip the Scanner for Linux
Visit the [official documentation](#) of the Scanner to download the latest version, and add the `bin` directory to the `PATH` environment variable

Execute the Scanner
Running a SonarQube analysis is straightforward. You just need to execute the following commands in your project's folder.

```
sonar-scanner \
-Dsonar.projectKey=frontend-app \
-Dsonar.sources= \
-Dsonar.host.url=http://3.239.186.187:9000 \
-Dsonar.login=squ_bb3d3b529730321dabd22b5f80c865aa7ab80777
```

Copy

30°C Haze 18:59 16-08-2024

Step 5: Create the ECR repositories

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

Here create the 2 repositories for frontend and backend

The screenshot shows the AWS ECR console with the URL us-east-1.console.aws.amazon.com/ecr/private-registry/repositories?region=us-east-1. The left sidebar shows navigation options for Private registry (Repositories, Settings) and Public registry (ECR public gallery, Amazon ECS, Amazon EKS). The main area displays 'Private repositories' with two entries:

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
backend	590184124026.dkr.ecr.us-east-1.amazonaws.com/backend	August 16, 2024, 14:20:20 (UTC+05:5)	Disabled	Manual	AES-256
frontend	590184124026.dkr.ecr.us-east-1.amazonaws.com/frontend	August 16, 2024, 14:20:09 (UTC+05:5)	Disabled	Manual	AES-256

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

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

Final all the credentials looks like this

The screenshot shows the Jenkins Manage Jenkins > Credentials page with the URL [Not secure 3.239.186.187:8080/manage/credentials/](http://3.239.186.187:8080/manage/credentials/). The page lists the following credentials:

T	P	Store	Domain	ID	Name
		System	(global)	aws-creds	AKIAY52NXBZ5EISUOSO4
		System	(global)	sonar-token	sonar-token
		System	(global)	ACCOUNT_ID	ACCOUNT_ID
		System	(global)	ECR_REPO2	ECR_REPO2
		System	(global)	GITHUB-APP	Madeep9347/*****
		System	(global)	ECR_REPO1	ECR_REPO1
		System	(global)	github	github

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

Step8: Setup the tools in Jenkins

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name: sonar-scanner

Install automatically:

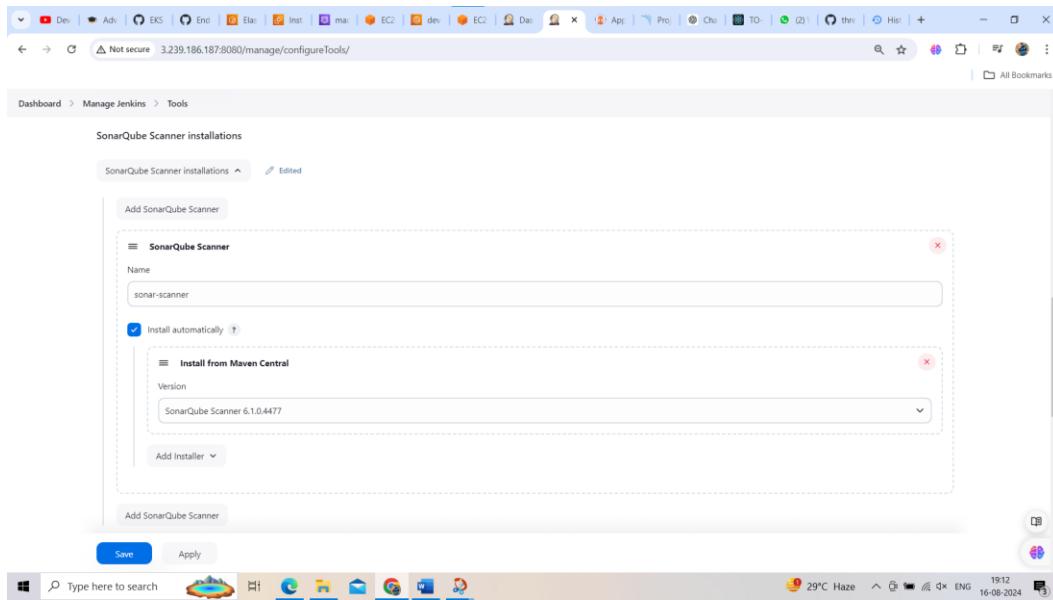
Install from Maven Central

Version: SonarQube Scanner 6.1.0.4477

Add Installer

Add SonarQube Scanner

Save Apply



NodeJS installations

Add NodeJS

NodeJS

Name: nodejs

Install automatically:

Install from nodejs.org

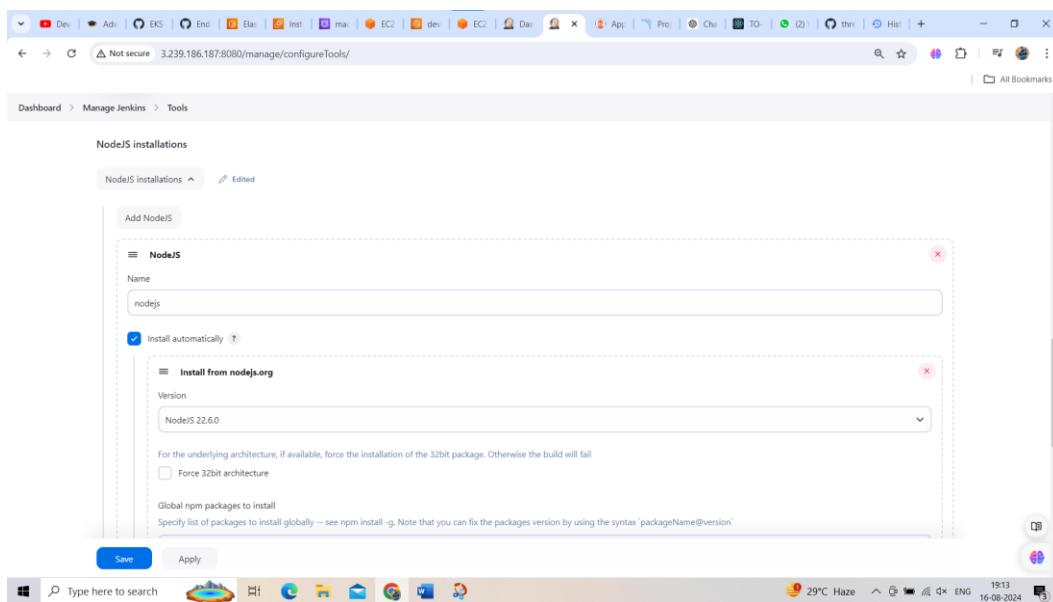
Version: NodeJS 22.6.0

Force 32bit architecture:

Global npm packages to install:

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax 'packageName@version'

Save Apply



Dependency-Check installations

Add Dependency-Check

Dependency-Check

Name: DP-Check

Install automatically:

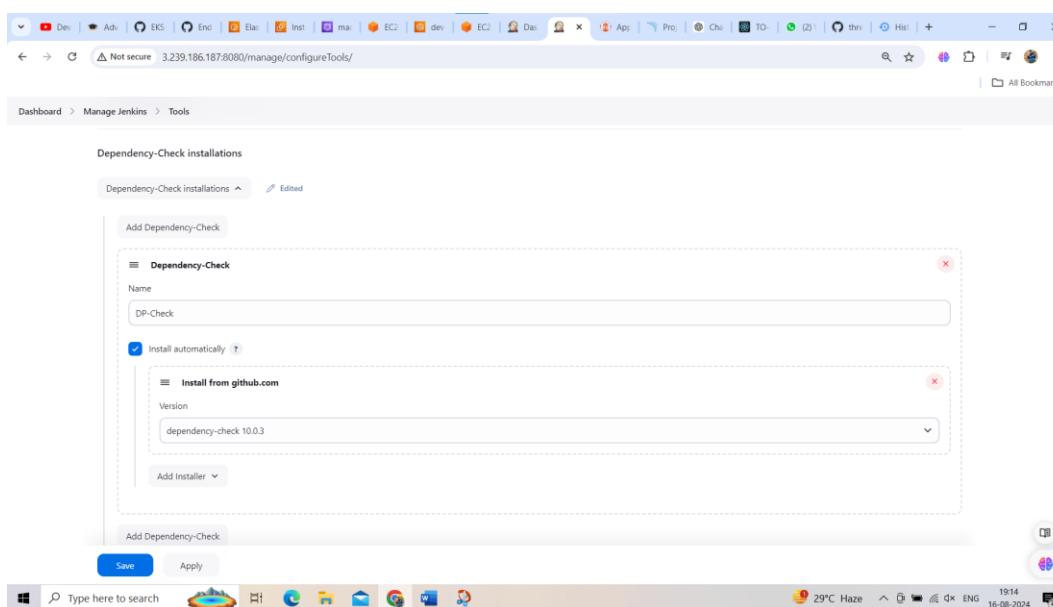
Install from github.com

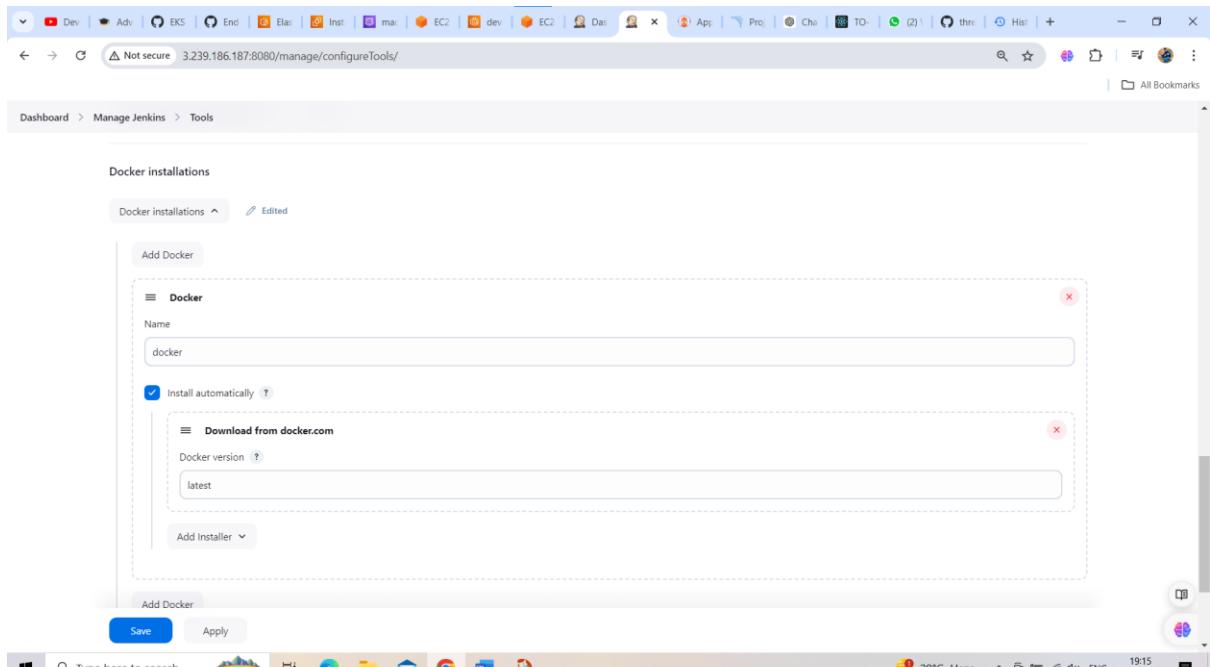
Version: dependency-check 10.0.3

Add Installer

Add Dependency-Check

Save Apply





The screenshot shows the Jenkins Docker installations configuration page. It displays a form for adding a new Docker installation named "docker". The "Install automatically" checkbox is checked, and the "Download from docker.com" section shows "Docker version" set to "latest". Buttons for "Save" and "Apply" are visible at the bottom.

Docker installations

Add Docker

Docker

Name: docker

Install automatically

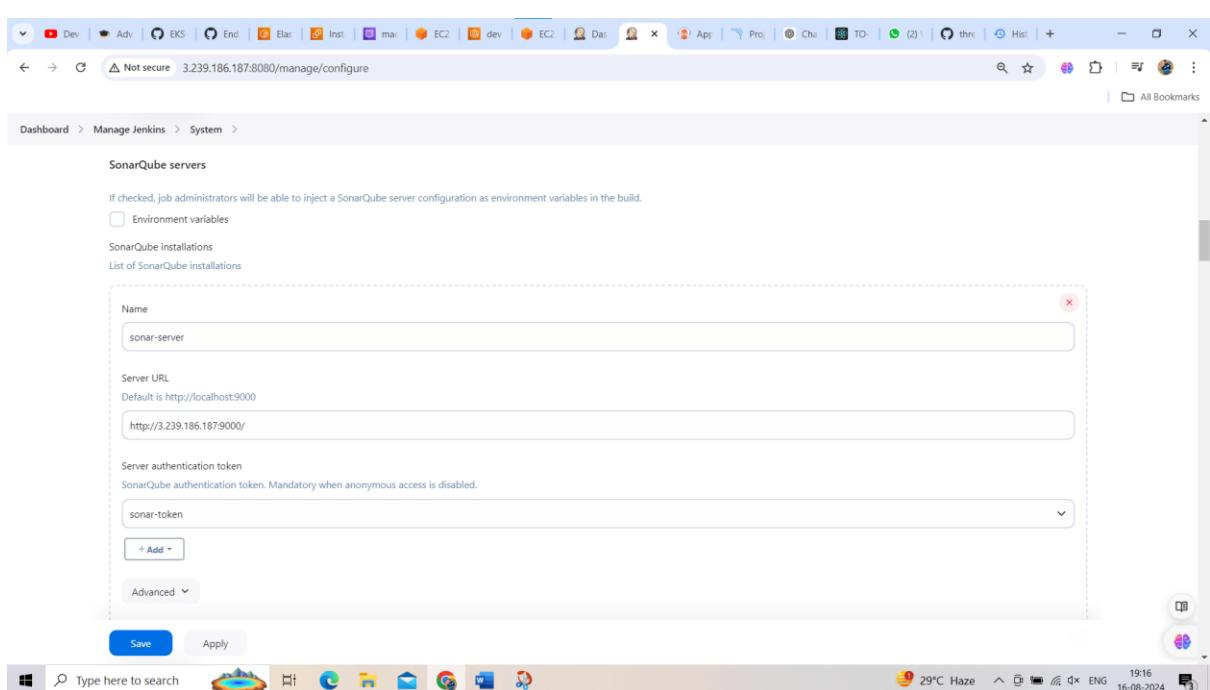
Download from docker.com

Docker version: latest

Add Installer

Add Docker

Save Apply



The screenshot shows the Jenkins SonarQube servers configuration page. It displays a form for adding a new SonarQube server named "sonar-server". The "Server URL" is set to "http://3.239.186.187:9000". The "Server authentication token" field contains "sonar-token". Buttons for "Save" and "Apply" are visible at the bottom.

SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

Environment variables

SonarQube installations

List of SonarQube installations

Name: sonar-server

Server URL
Default is http://localhost:9000
http://3.239.186.187:9000/

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

+ Add Advanced

Save Apply

Step 9: create the Jenkins job for frontend application

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

Stage View

Declarative: Tool Install	Cleaning Workspace	Checkout from Git	SonarQube Analysis	Quality Check	Trivy File Scan	Docker Image Build	ECR Image Pushing	TRIVY Image Scan	Checkout Code	Update Deployment file
109ms	249ms	499ms	11s	306ms	1s	1s	2s	17s	408ms	1s
Aug 16 15:14	No Changes									
109ms	249ms	499ms	11s	306ms (paused for 1s)	1s	1s	2s	17s	408ms	1s

SonarQube Quality Gate

Builds	Passed
server-side processing:	Success

Builds

Today	#1 9:44 AM
	Aug 16 15:14

Step 10 : Create the Jenkins job for Backend Application

Stage View

Declarative: Tool Install	Cleaning Workspace	Checkout from Git	SonarQube Analysis	Quality Check	Trivy File Scan	Docker Image Build	ECR Image Pushing	TRIVY Image Scan	Checkout Code	Update Deployment file
134ms	246ms	525ms	10s	355ms	1s	3s	14s	12s	403ms	1s
Aug 16 15:43	3 commits									
123ms	236ms	556ms	10s	363ms (paused for 1s)	1s	1s	2s	13s	421ms	1s
Aug 16 15:20	No Changes									
145ms	257ms	495ms	10s	347ms (paused for 1s)	1s	4s	25s	11s	386ms	1s

SonarQube Quality Gate

Builds	Passed
server-side processing:	Success

Builds

Today	#4 10:13 AM	#3 9:50 AM	#2 9:49 AM
	Aug 16 15:43		

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

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

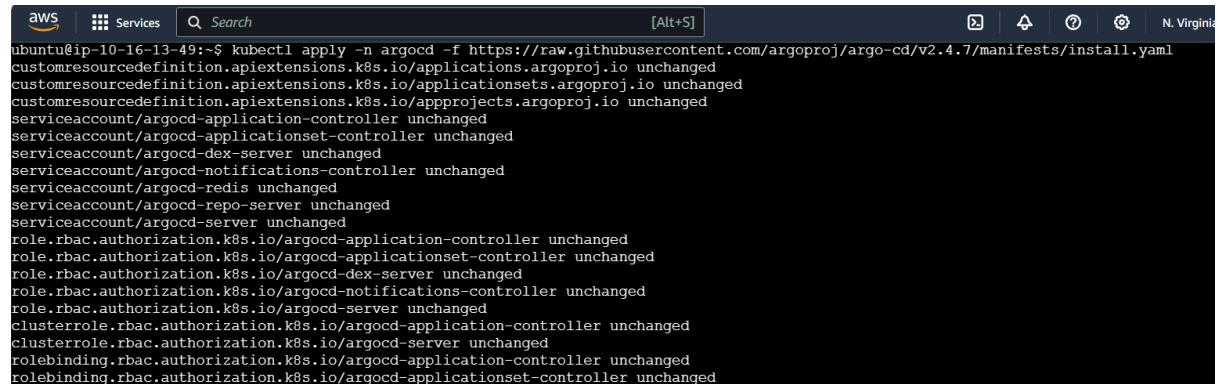
Create an separate namespace for argocd using the command:

```
kubectl create namespace three-tier
```

Now, we will install argoCD:

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

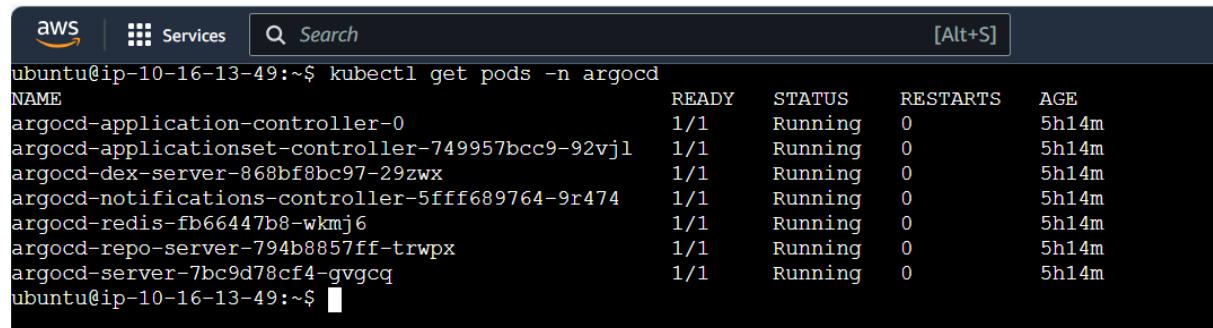
these command will create all the required pods and services of the argocd server in the argocd namespace.



```
aws | Services | Q Search [Alt+S] | N. Virginia
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.apirextensions.k8s.io/applications.argoproj.io unchanged
customresourcedefinition.apirextensions.k8s.io/applicationsets.argoproj.io unchanged
customresourcedefinition.apirextensions.k8s.io/approjects.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
```

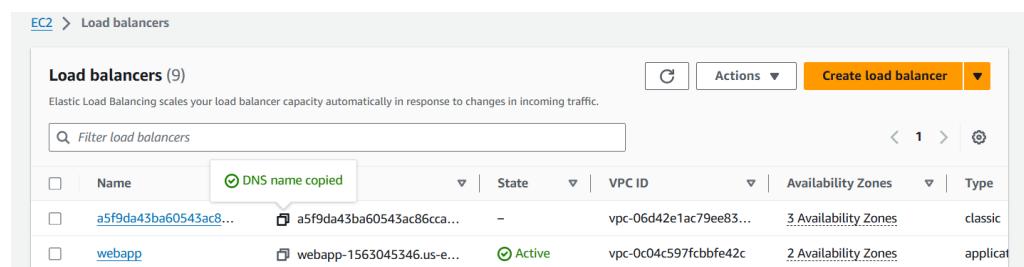


```
aws | Services | Q Search [Alt+S] | N. Virginia
ubuntu@ip-10-16-13-49:~$ kubectl get pods -n argocd
NAME                               READY   STATUS    RESTARTS   AGE
argocd-application-controller-0     1/1    Running   0          5h14m
argocd-applicationset-controller-749957bcc9-92vjl 1/1    Running   0          5h14m
argocd-dex-server-868bf8bc97-29zwx 1/1    Running   0          5h14m
argocd-notifications-controller-5fff689764-9r474 1/1    Running   0          5h14m
argocd-redis-fb66447b8-wkmj6       1/1    Running   0          5h14m
argocd-repo-server-794b8857ff-trwpox 1/1    Running   0          5h14m
argocd-server-7bc9d78cf4-gvgcq    1/1    Running   0          5h14m
ubuntu@ip-10-16-13-49:~$
```

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:



The screenshot shows the AWS CloudWatch Metrics interface with the 'Load balancers' section. It displays a table with 9 entries, each representing a load balancer. The columns include Name, DNS name copied, State, VPC ID, Availability Zones, and Type. One entry for 'a5f9da43ba60543ac8...' has a green status indicator and the message 'DNS name copied'. Another entry for 'webapp' is marked as 'Active'.

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.



Your connection is not private

Attackers might be trying to steal your information from
a5f9da43ba60543ac86ccadd6d0ee81b-1847104018.us-east-1.elb.amazonaws.com
(for example, passwords, messages, or credit cards). [Learn more](#)

NET::ERR_CERT_AUTHORITY_INVALID

[Hide advanced](#)

[Back to safety](#)

This server could not prove that it is a5f9da43ba60543ac86ccadd6d0ee81b-1847104018.us-east-1.elb.amazonaws.com; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to a5f9da43ba60543ac86ccadd6d0ee81b-1847104018.us-east-1.elb.amazonaws.com \(unsafe\)](#)

Let's get stuff deployed!

Username
admin

Password
.....

SIGN IN

argo

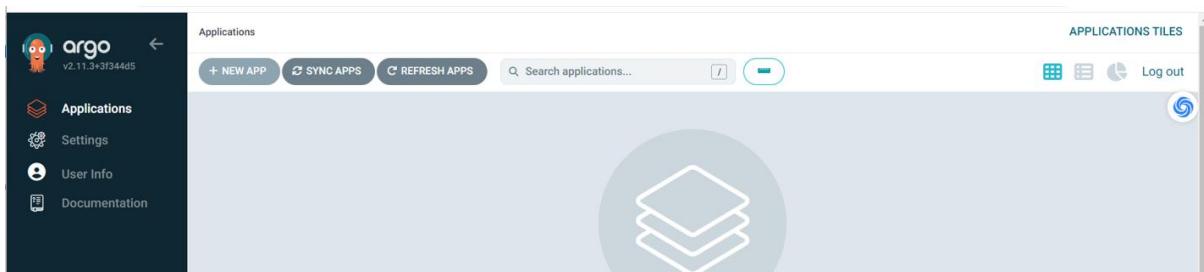
Now, we need to get the password for our argoCD server to perform the deployment.

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

```
aws Services Search [Alt+S] [x] [?] [WFMgTWejLjHugvubuntu@ip-10-16-13-49:~$ kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 -d x-WFMgTWejLjHugvubuntu@ip-10-16-13-49:~$ ]
```

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

Here is our ArgoCD Dashboard.



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

Connect the github repository using HTTPS

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

The screenshot shows the Argo CD Settings interface. On the left is a sidebar with icons for Applications, Settings, User Info, and Documentation. The main area is titled 'Repositories' with the sub-instruction 'Configure connected repositories'. Below it are sections for 'Certificates', 'GnuPG keys', 'Clusters', 'Projects', and 'Accounts', each with a 'Configure' link. In the top right corner, there are 'SETTINGS' and 'Log out' buttons.

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

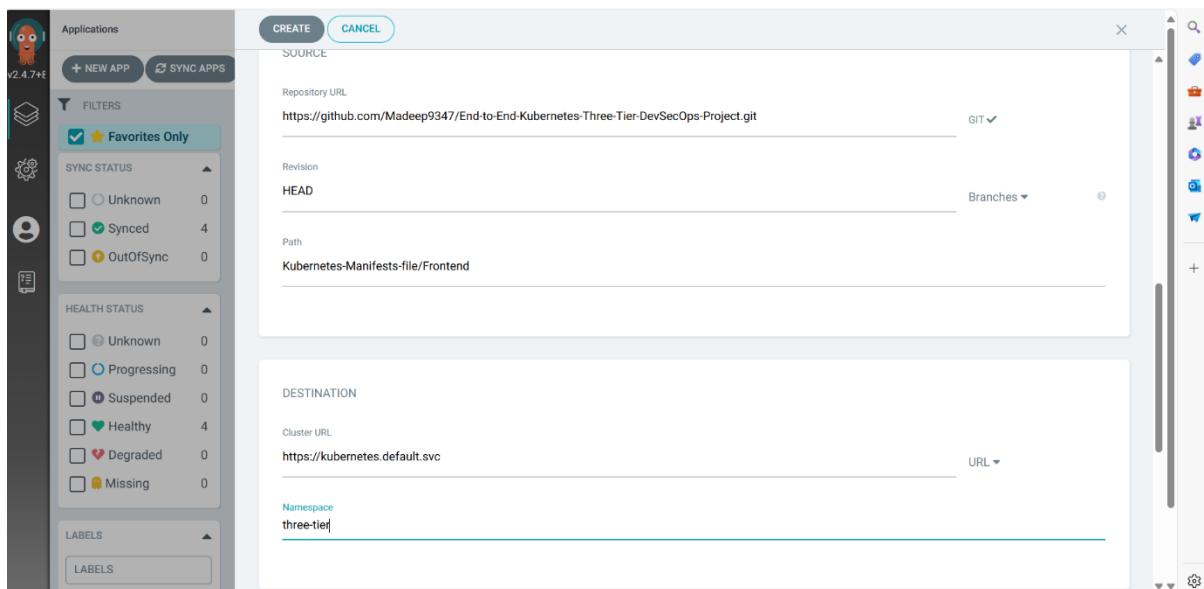
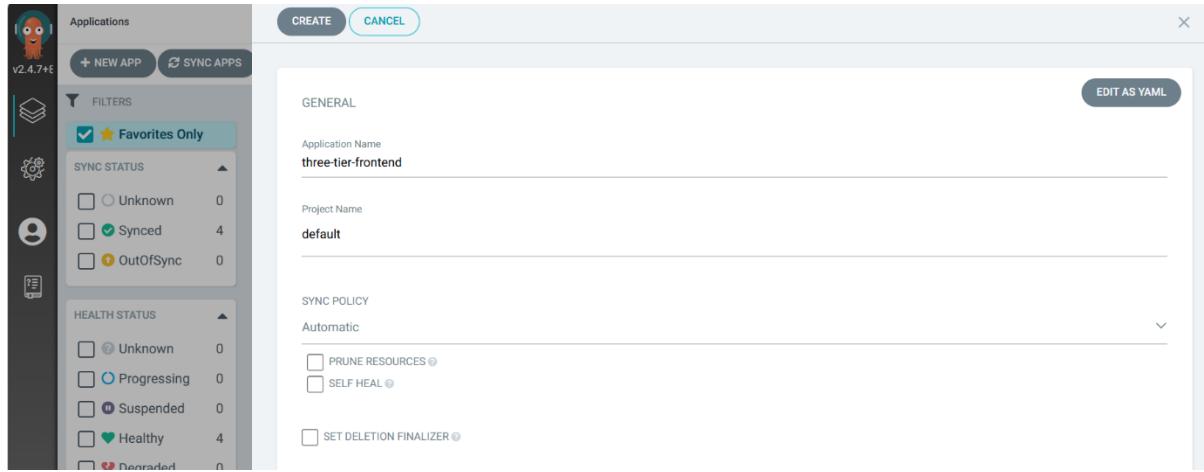
The screenshot shows the 'CONNECT REPO USING HTTPS' dialog box. It has tabs for 'CONNECT', 'SAVE AS CREDENTIALS TEMPLATE', and 'CANCEL'. The 'CONNECT' tab is active. The form fields are: 'Type' set to 'git', 'Project' set to 'default', and 'Repository URL' containing the value 'https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project'. There is also a note above the URL field stating 'CONNECT REPO USING HTTPS'.

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

The screenshot shows the 'REPOSITORIES' list on the Argo CD dashboard. At the top are buttons for '+ CONNECT REPO USING SSH', '+ CONNECT REPO USING HTTPS', '+ CONNECT REPO USING GITHUB APP', and 'REFRESH LIST'. The 'CONNECT REPO USING HTTPS' button is highlighted. The main table has columns: TYPE, NAME, REPOSITORY, and CONNECTION STATUS. One row is visible, showing 'git' as the type, 'https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project' as the repository, and 'Successful' in the connection status column. A 'Log out' button is in the top right.

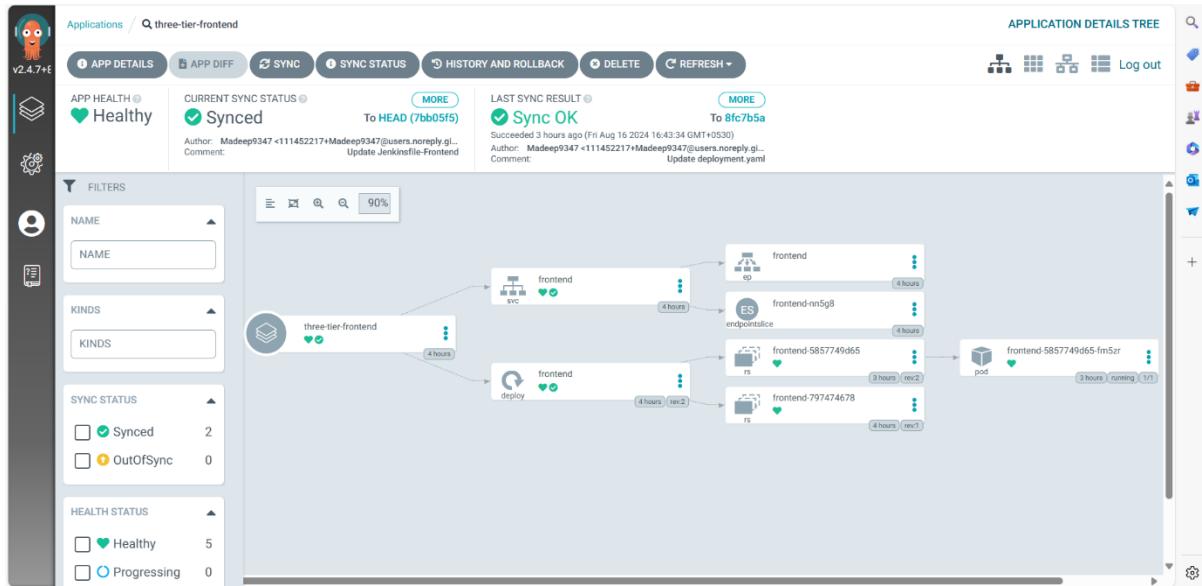
Step 13: Now create the separate apps for Frontend, Backend, Database and the ingress manifest files:

Frontend app creation:

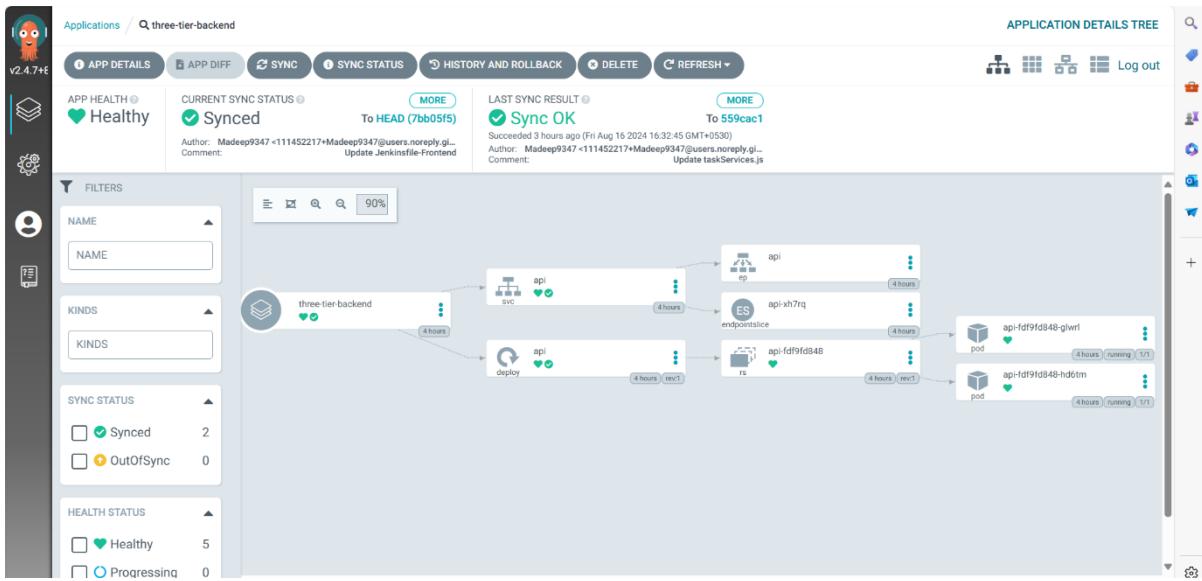


Create these apps same for the Backend, Database and Ingress.

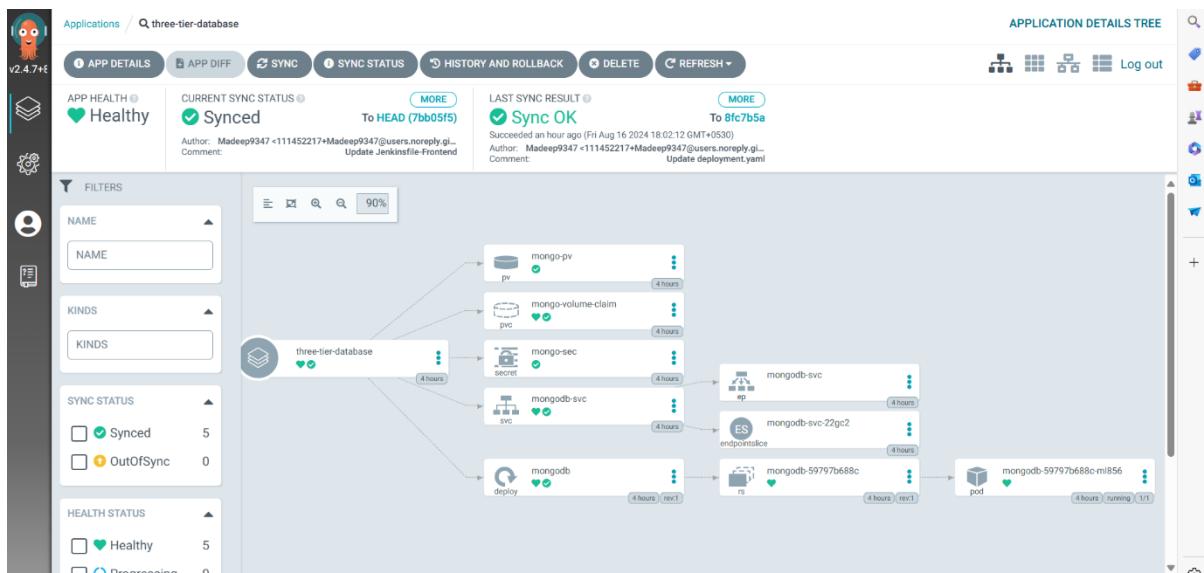
This is the Frontend Application Deployment in ArgoCD:



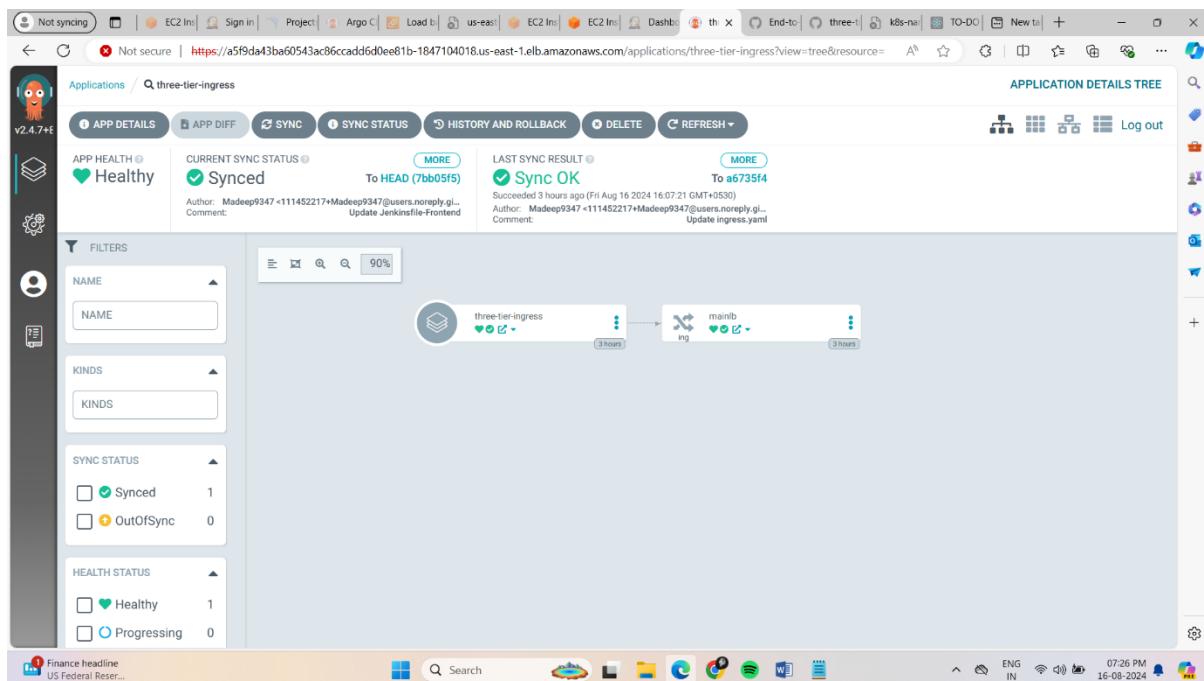
This is the Backend Application Deployment in ArgoCD:



This is the Database Application Deployment in ArgoCD:



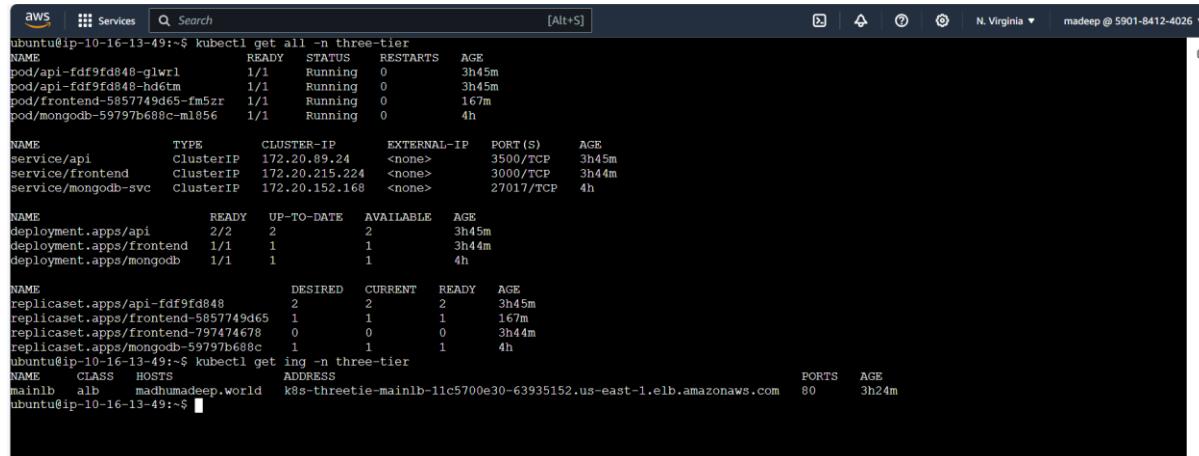
This is the Ingress Application Deployment in ArgoCD:



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

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

Kubectl get all -n three-tier



```
aws Services Search [Alt+S] N. Virginia madeep @ 5901-8412-4026
ubuntu@ip-10-16-13-49:~$ kubectl get all -n three-tier
NAME                                     READY   STATUS    RESTARTS   AGE
pod/api-fdf9fd848-glwrl                1/1    Running   0          3h45m
pod/api-fdf9fd848-hd6tm                1/1    Running   0          3h45m
pod/frontend-5857749d65-fm5zr           1/1    Running   0          167m
pod/mongodb-59797b688c-m1856           1/1    Running   0          4h

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP     PORT(S)      AGE
service/api       ClusterIP   172.20.89.24   <none>        3500/TCP    3h45m
service/frontend ClusterIP   172.20.215.224  <none>        3000/TCP    3h44m
service/mongodb-svc  ClusterIP   172.20.152.168  <none>        27017/TCP   4h

NAME             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api  2/2     2           2           3h45m
deployment.apps/frontend 1/1     1           1           3h44m
deployment.apps/mongodb 1/1     1           1           4h

NAME            DESIRED  CURRENT  READY   AGE
replicaset.apps/api-fdf9fd848  2        2        2        3h45m
replicaset.apps/frontend-5857749d65 1        1        1        167m
replicaset.apps/frontend-797474678 0        0        0        3h44m
replicaset.apps/mongodb-59797b688c 1        1        1        4h
ubuntu@ip-10-16-13-49:~$ kubectl get ing -n three-tier
NAME         CLASS  HOSTS          ADDRESS          PORTS  AGE
mainlb      alb    madhumadeep.world  k8s-threetie-mainlb-11c5700e30-63935152.us-east-1.elb.amazonaws.com  80      3h24m
ubuntu@ip-10-16-13-49:~$
```

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

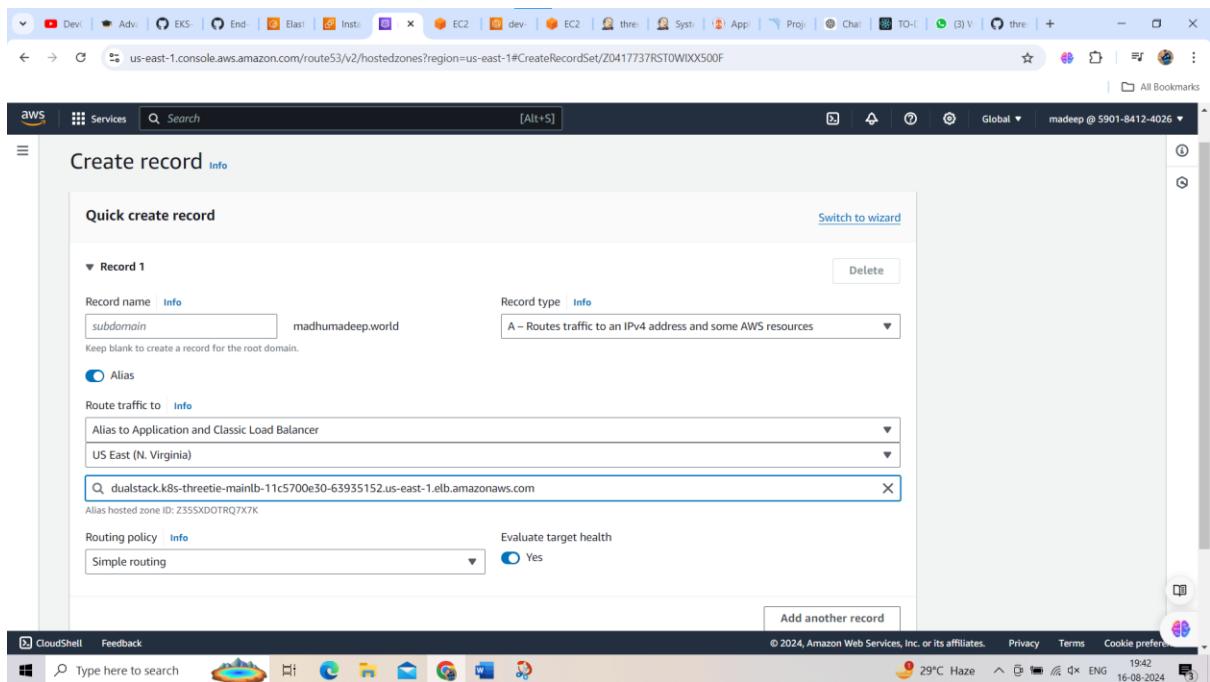
```
spec:
  imagePullSecrets:
  - name: ecr-registry-secret
  containers:
  - name: api
    image: 590184124026.dkr.ecr.us-east-1.amazonaws.com/backend:4
    imagePullPolicy: Always
    env:
      - name: MONGO_CONN_STR
        value: mongodb://mongodb-svc:27017/todo?directConnection=true
      - name: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

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

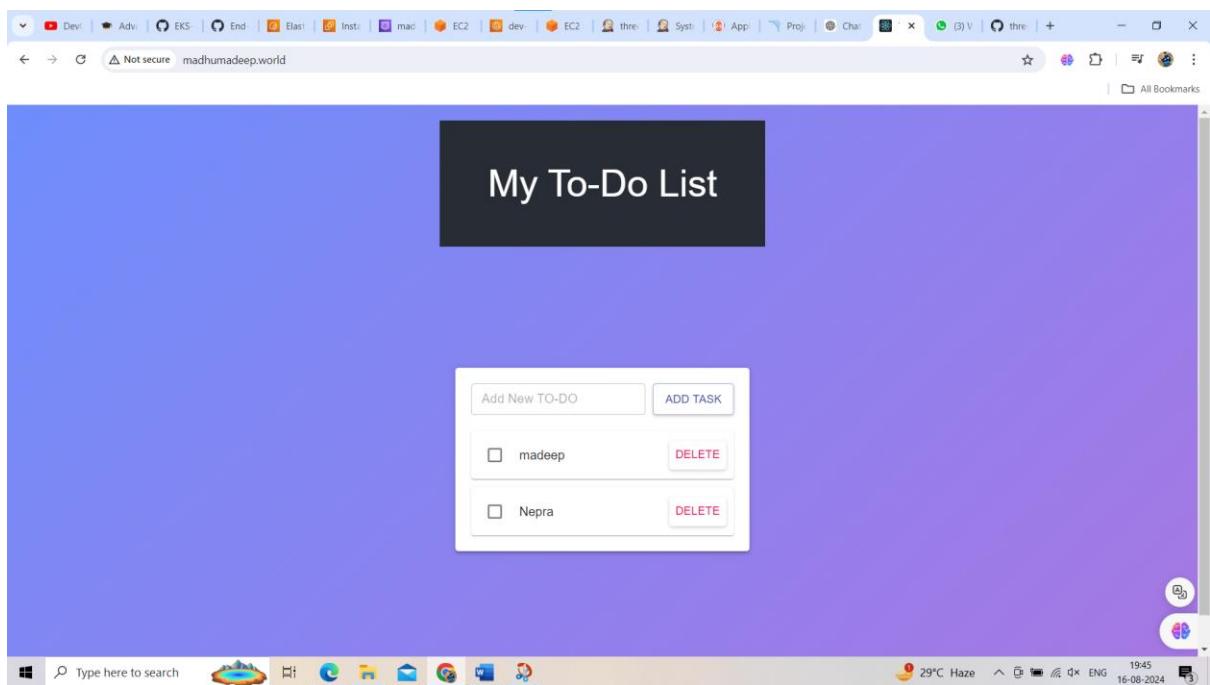
And also edit the domain name in evn

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

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



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



PHASE 3 : Monitoring the EKS cluster

We will achieve the monitoring using Helm

Step1 :Add the prometheus repo by using the below command

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

Step 2: Install the Prometheus

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

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

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

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

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

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

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

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

kubectl edit svc Prometheus-server

```
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
grafana        ClusterIP 172.20.131.173 <none>       80/TCP    14m
kubernetes     ClusterIP 172.20.0.1    <none>       443/TCP   27h
prometheus-alertmanager ClusterIP 172.20.246.8 <none>       9093/TCP  14m
prometheus-alertmanager-headless ClusterIP None        <none>       9093/TCP  14m
prometheus-kube-state-metrics ClusterIP 172.20.24.217 <none>      8080/TCP  14m
prometheus-prometheus-node-exporter ClusterIP 172.20.35.64 <none>      9100/TCP  14m
prometheus-prometheus-pushgateway ClusterIP 172.20.179.123 <none>      9091/TCP  14m
prometheus-server ClusterIP 172.20.130.81 <none>      80/TCP    14m
ubuntu@ip-10-16-13-49:~$ kubectl edit svc prometheus-server
service/prometheus-server edited
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
grafana        ClusterIP 172.20.131.173 <none>       80/TCP    16m
kubernetes     ClusterIP 172.20.0.1    <none>       443/TCP   27h
prometheus-alertmanager ClusterIP 172.20.246.8 <none>       9093/TCP  16m
prometheus-alertmanager-headless ClusterIP None        <none>       9093/TCP  16m
prometheus-kube-state-metrics ClusterIP 172.20.24.217 <none>      8080/TCP  16m
prometheus-prometheus-node-exporter ClusterIP 172.20.35.64 <none>      9100/TCP  16m
prometheus-prometheus-pushgateway ClusterIP 172.20.179.123 <none>      9091/TCP  16m
prometheus-server LoadBalancer 172.20.130.81 a5f57c4555e684ebd8238e531210ele9-193019089.us-east-1.elb.amazonaws.com 80:31696/TCP 16m
ubuntu@ip-10-16-13-49:~$
```

The screenshot shows the Prometheus Targets page. At the top, there's a search bar with the URL 'a5f57c4555e684ebd8238e531210ele9-193019089.us-east-1.elb.amazonaws.com/targets?search=' and a dropdown menu for 'All scrape pools'. Below the search bar are buttons for 'All', 'Unhealthy', and 'Collapse All'. A 'Filter by endpoint or labels' input field is present, along with checkboxes for 'Unknown', 'Unhealthy', and 'Healthy' status filters.

Targets

All scrape pools	All	Unhealthy	Collapse All	Filter by endpoint or labels	Unknown	Unhealthy	Healthy	
kubernetes-apiservers (2/2 up) [show less]								
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error			
https://10.16.135.232/metrics	UP	instance: "10.16.135.232:443" job: "kubernetes-apiservers"	53.167s ago	109.731ms				
https://10.16.168.188/metrics	UP	instance: "10.16.168.188:443" job: "kubernetes-apiservers"	19.491s ago	83.451ms				
kubernetes-nodes (2/2 up) [show less]								
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error			
https://kubernetes.default.svc/api/v1/nodes/ip-10-16-162-211.ec2.internal/proxy/metrics	UP	beta.kubernetes.io.arch="amd64" beta.kubernetes.io.instance.type="t3a.medium" beta.kubernetes.io.os="Linux" eks.amazonaws.com.capacityType="ON_DEMAND" eks.amazonaws.com.nodename="dev-medium-eks-cluster-on-demand-nodes" eks.amazonaws.com.nodegroup.image="ami-08d9654d4ecbb01d5c" failure_domain.beta.kubernetes.io.region="us-east-1" failure_domain.beta.kubernetes.io.zone="us-east-1c"	27.167s ago	121.102ms				

At the bottom of the browser window, there's a taskbar with icons for File, Home, Back, Forward, Stop, Refresh, and Search. The search bar contains the text 'Type here to search'. On the right side of the taskbar, there are system status icons for battery level (33°C Haze), network, and date/time (16:41 17-08-2024).

Step 4: Install Grafana

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

helm repo update

helm install grafana grafana/Grafana

Step 5: Edit the stable-grafana service

Kubectl edit svc grafana

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

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

Step 6: Get the password for grafana from secrets

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

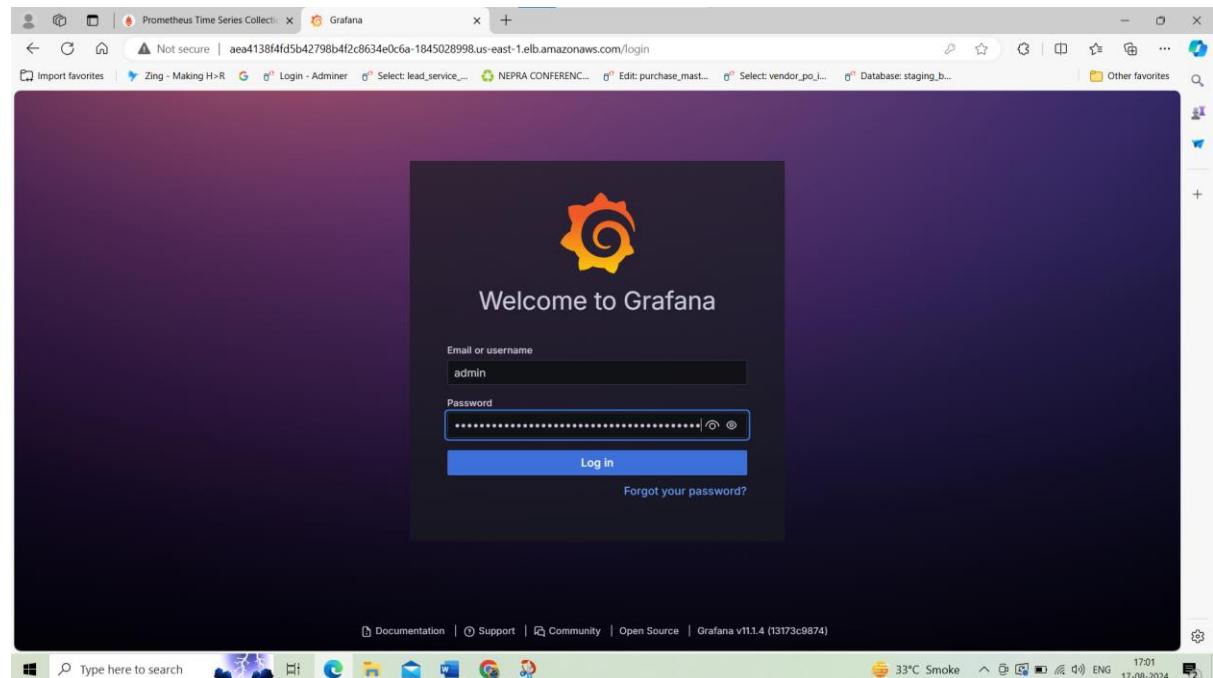
Kubectl edit secret grafana

```
apiVersion: v1
data:
  admin-password: MERPWXdGUUhNRmVXdXdkY1A3TUM4dkZjR3FZNm82V1N1cFBmY21qZA==
  admin-user: YWRtaW4=
```

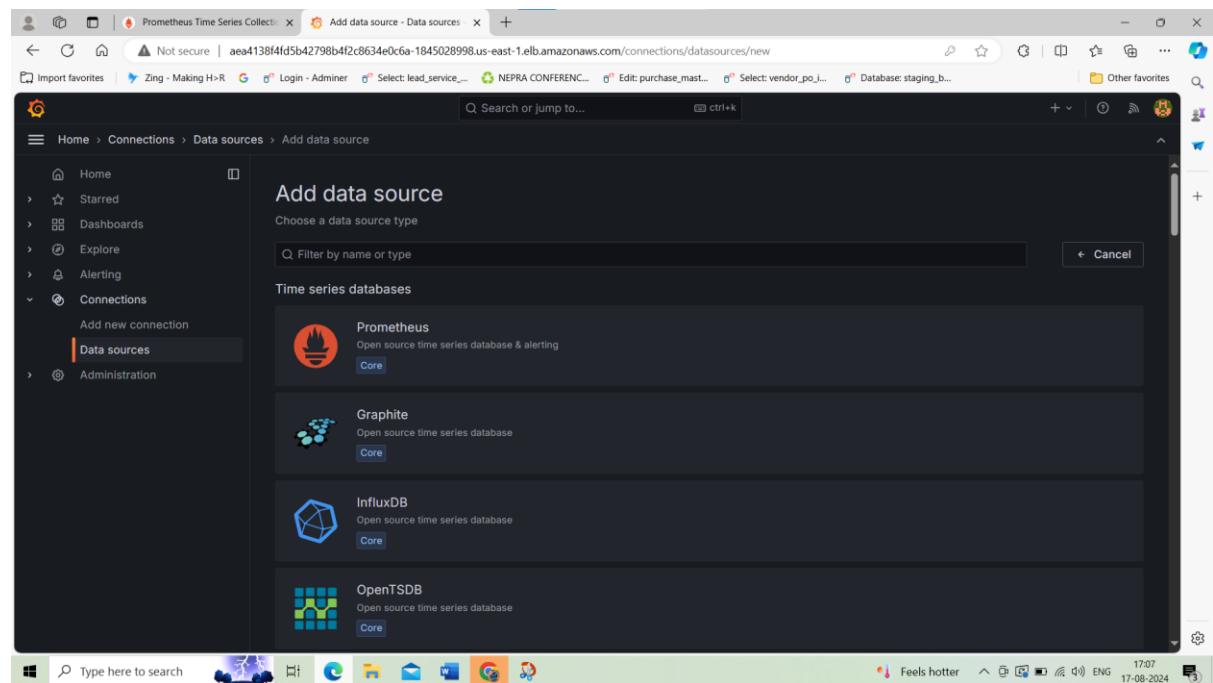
Copy the password and decode with base64

```
ubuntu@ip-10-16-13-49:~$ echo MERPWXdGUUhNRmVXdXdkY1A3TUM4dkZjR3FZNm82V1N1cFBmY21qZA== | base64 --decode  
0DOYWFQHMFeWuwdbP7MC8vFcGqY6o6WSupPfcmjduuntu@ip-10-16-13-49:~$
```

Step7: Now access the grafana using loadbalancer



Step 8: select the data source as prometheus



Enter your loadbalancer domain name and test

Connection

Prometheus server URL * ⓘ 531210e1e9-193019089.us-east-1.elb.amazonaws.com/

✓ Successfully queried the Prometheus API.

Next, you can start to visualize data by [building a dashboard](#), or by querying data in the [Explore view](#).

[Delete](#)

[Save & test](#)

Step 9: Create the Dashboard

Import the dashboard

The screenshot shows the Grafana interface with the title bar "Prometheus Time Series Collector" and "New dashboard - Dashboards". The address bar shows the URL "aea41384fd5b42798b4f2c0634e0c6a-1845028998.us-east-1.elb.amazonaws.com/dashboard/new?orgId=1". The left sidebar has a tree structure with "Home", "Starred", "Dashboards" (which is selected), "Explore", "Alerting", "Connections" (with "Add new connection" and "Data sources" options), and "Administration". The main content area has a heading "Start your new dashboard by adding a visualization" and a sub-instruction "Select a data source and then query and visualize your data with charts, stats and tables or create lists, markdowns and other widgets.". Below this is a blue button "+ Add visualization". At the bottom of the main area are two sections: "Import panel" (with a sub-instruction "Add visualizations that are shared with other dashboards." and a button "+ Add library panel") and "Import a dashboard" (with a sub-instruction "Import dashboards from files or grafana.com." and a button "Import dashboard"). The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

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.