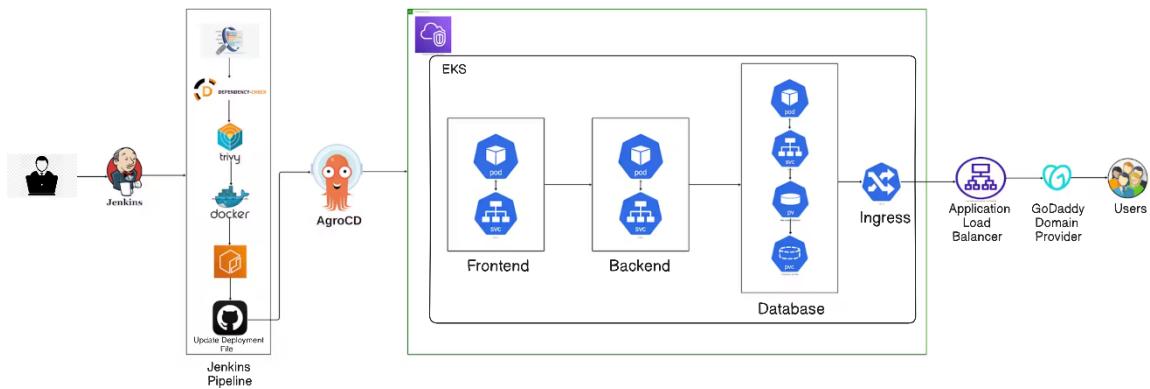


End-to-End DevSecOps Pipeline with Jenkins, ArgoCD, and EKS for Three-Tier Application Deployment

End-to-End DevSecOps Kubernetes Three-Tier Project using AWS EKS



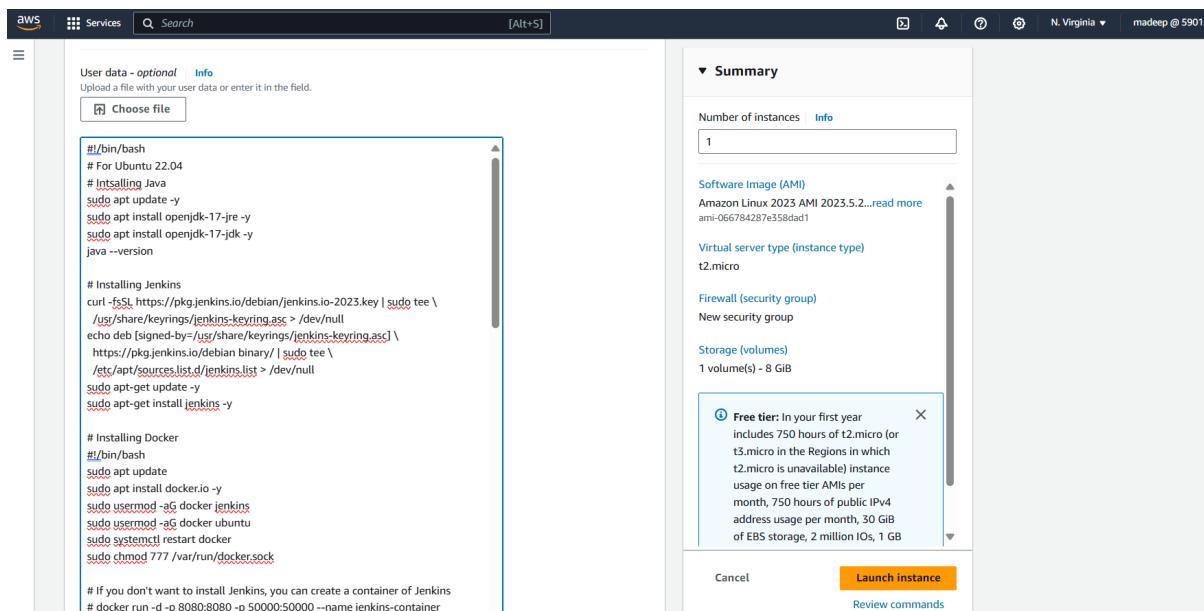
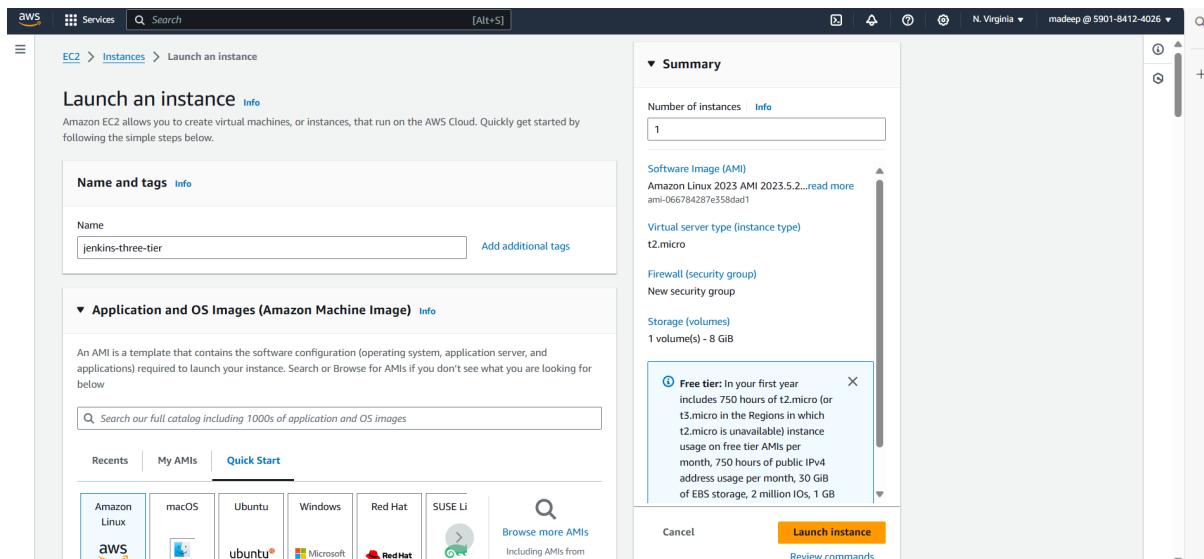
Project Overview:

In this project, we will cover the following steps:

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

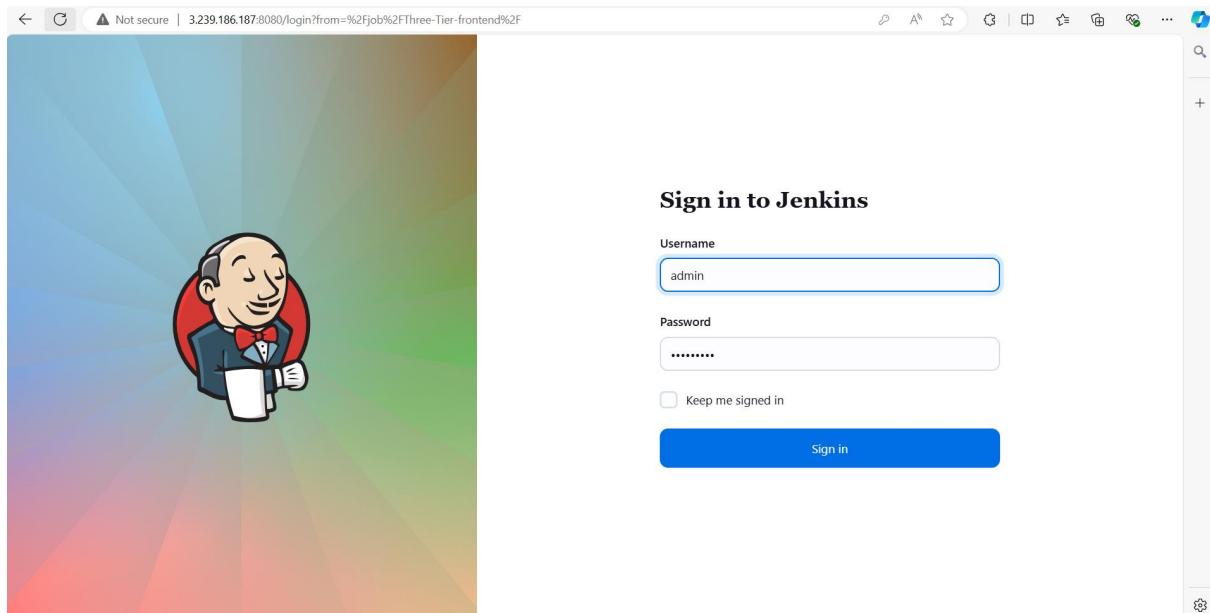
Step-1: Jenkins Server Setup and Install Essential Tools on Jenkins Server

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



Step-2: Configure Jenkins Server

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



- Go to Manage Plugins in Jenkins, select Credentials, choose AWS Credentials as the Kind, add your AWS Access Key & Secret Access Key with a unique ID, and click Create.
- Add GitHub credentials by selecting Username and Personal Access Token, as repositories in industry projects are often private, and save the credentials for use in your pipelines.

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	Narasimha76/*****
		System	(global)	ECR_REPO1	ECR_REPO1
		System	(global)	github	github

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

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

VPC Details:

- VPC ID: vpc-06d42e1ac79ee833c
- State: Available
- Tenancy: Default
- Default VPC: No
- IPv4 CIDR: 10.16.0.0/16
- Network Address Usage metrics: Disabled
- DNS hostnames Enabled
- Main route table: rtb-0666a82a7111e6ca8
- IPv6 pool: -
- Route 53 Resolver DNS Firewall rule groups: -
- Owner ID: 590184124026
- DNS resolution Enabled
- Main network ACL: acl-09f15595cf8ac20ff
- IPv6 CIDR (Network border group): -

Resource Map:

- VPC: dev-medium-vpc
- Subnets (6): us-east-1a (dev-medium-subnet-public-1), us-east-1a (dev-medium-subnet-private-1)
- Route tables (3): rtb-0666a82a7111e6ca8, dev-medium-public-route-table, dev-medium-private-route-table
- Network connection: dev-medium-igw, dev-medium-ngw

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

1. Create an ec2 instance for the Jump Server using the below command:

```
aws ec2 run-instances --image-id <image-id> --count 1 --instance-type <type> --key-name <keypair> --security-group-ids <sg-id> --vpc-id <vpc-id> --subnet-id <subnet-id> --region <region>
```
2. Create an AWS EKS cluster using the below command:

```
eksctl create cluster --name <name> --region <region> --node-type <type> --nodes-min <min-num> --nodes-max <max-num>
```
3. Configure the aws credentials in the jump server using aws configure command:

```
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:~$
```

4. After creation of the EKS cluster add the EKS cluster into the Jump server using the below command:

```
aws eks update-kubeconfig --region us-east-1 --name <name of the cluster>
```

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

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

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

1. Download the policy for the LoadBalancer prerequisite:

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

2. Create the IAM policy using the below command:

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

3. Create OIDC Provider:

```
eksctl utils associate-iam-oidc-provider --region=us-east-1 --cluster=Three-Tier-K8s-EKS-Cluster --approve
```

4. Create a Service Account by using below command and replace your account ID:

```
eksctl create iamserviceaccount --cluster=Three-Tier-K8s-EKS-Cluster --namespace=kube-system --name=aws-load-balancer-controller --role-name AmazonEKSLoadBalancerControllerRole --attach-policy-arn=arn:aws:iam::<your_account_id>:policy/AWSLoadBalancerControllerIAMPolicy --approve --region=us-east-1
```

5. Run the below command to deploy the AWS Load Balancer Controller:

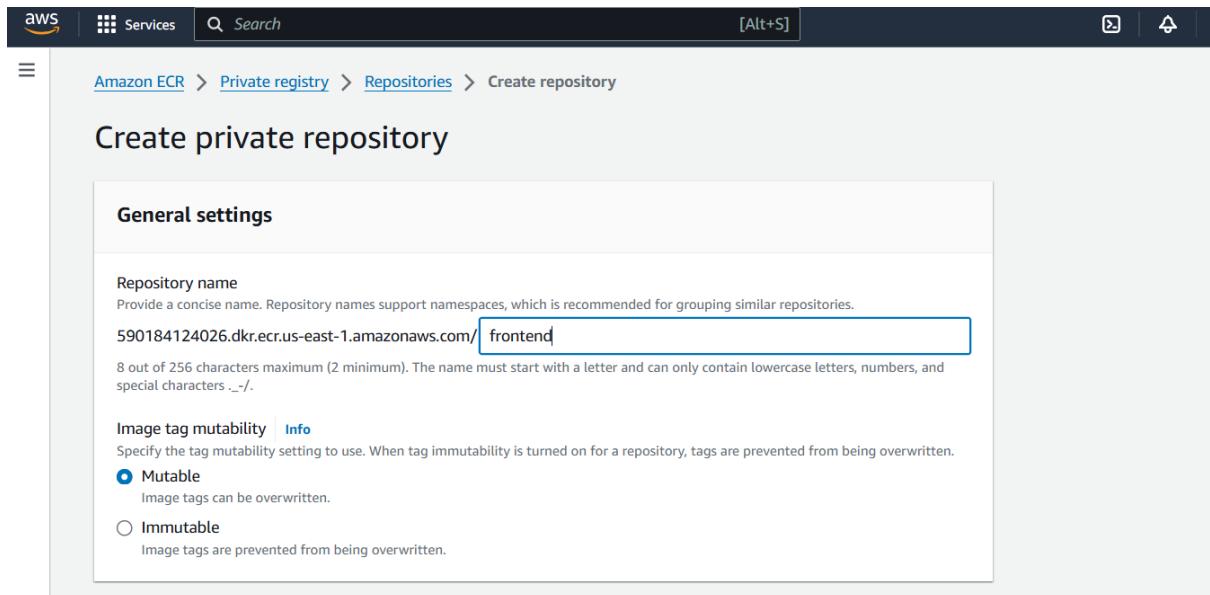
```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller -n kube-system --set clusterName=my-cluster --set serviceAccount.create=false --set serviceAccount.name=aws-load-balancer-controller
```

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

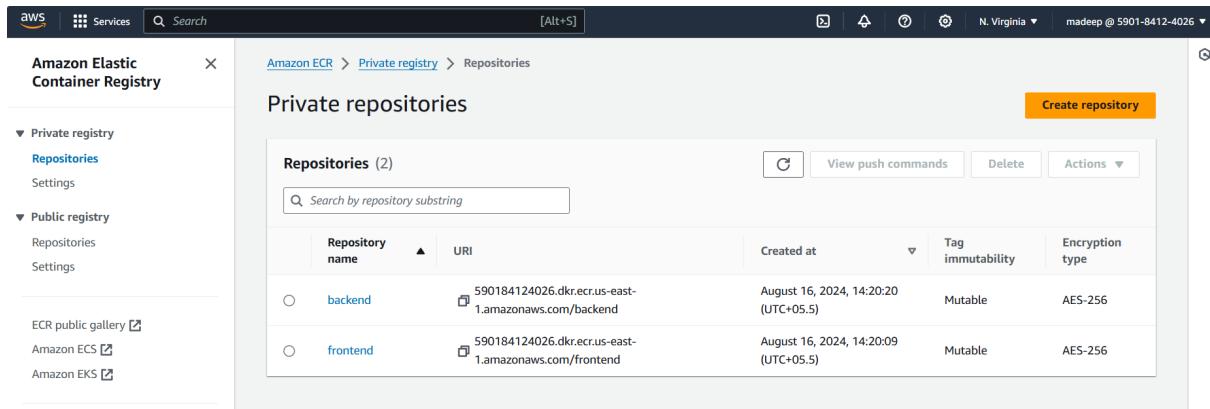
```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

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

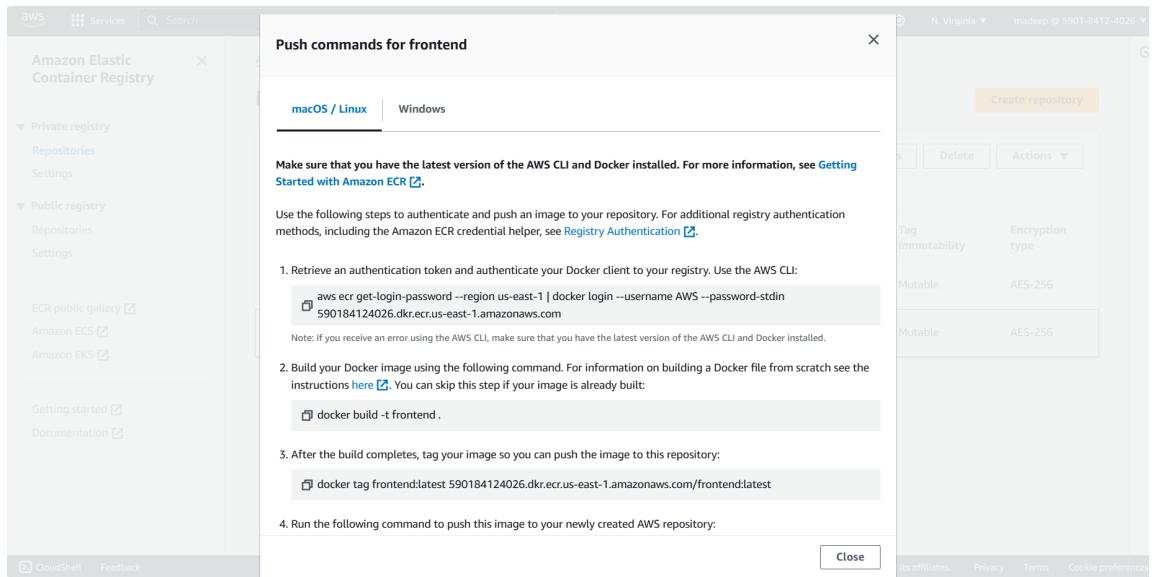
Step 6: We need to create Amazon ECR Private Repositories for both Tiers (Frontend & Backend)



- Similarly create the Backend Repository



- Now, we need to configure ECR locally because we have to upload our images to Amazon ECR.
- Using the push commands we can configure



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

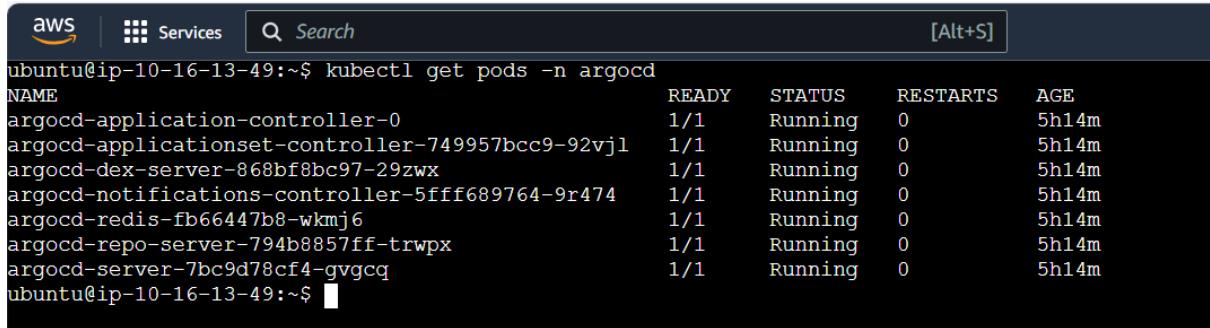
```
ubuntu@ip-10-0-25-107:~$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 590184124026.dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json.
Configure a credential helper to remove this warning.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

Step-7: Install and Configure ArgoCD

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

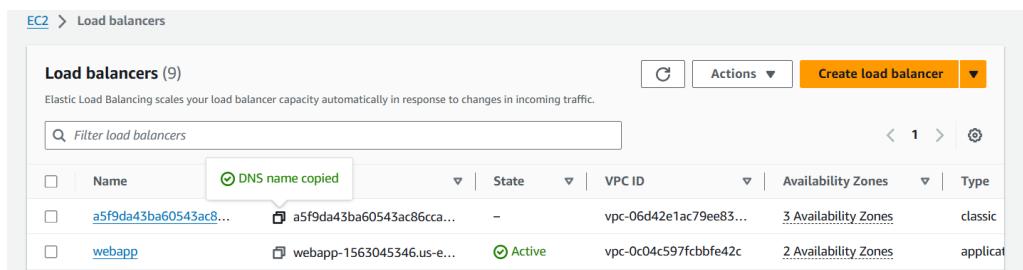
```
ubuntu@ip-10-16-13-49:~$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/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 | Search [Alt+S]
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:



Name	State	VPC ID	Availability Zones	Type
a5f9da43ba60543ac...	-	vpc-06d42e1ac79ee83...	3 Availability Zones	classic
webapp	Active	vpc-0c04c597fcbbfe42c	2 Availability Zones	application

- 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](#)

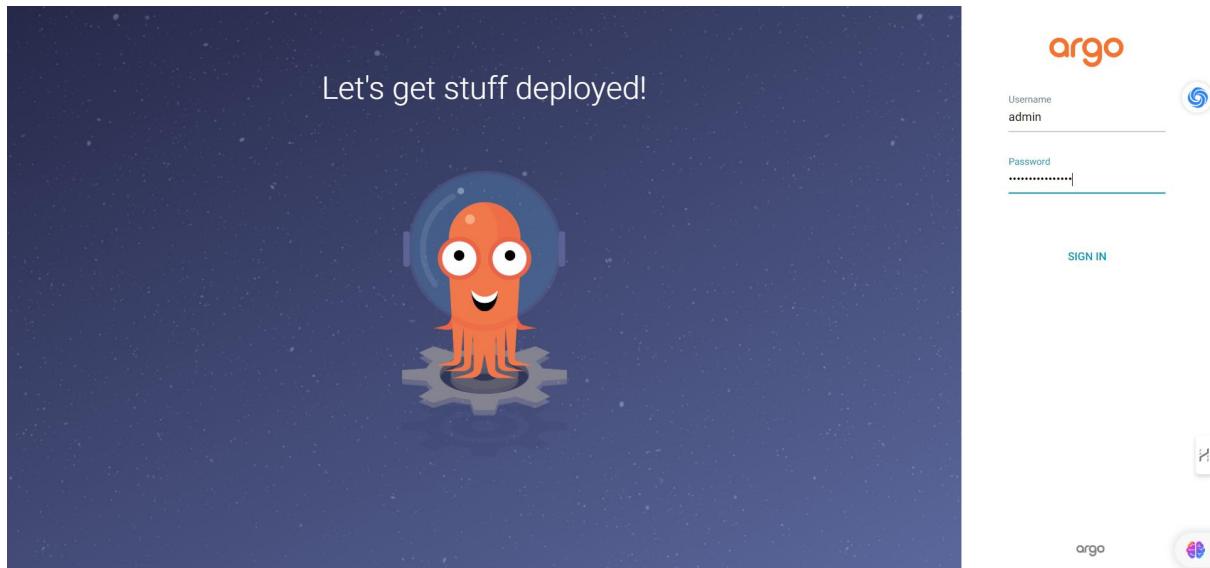
NET::ERR_CERT_AUTHORITY_INVALID

[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 \(unsafe\)](#)



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

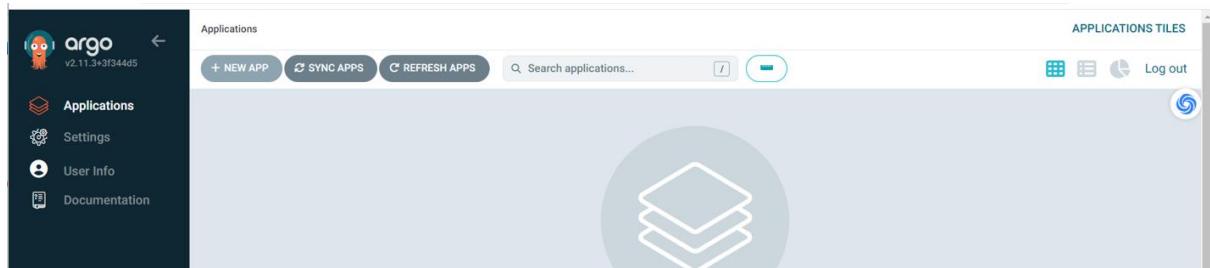
```
kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 -d
```

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

A screenshot of a terminal window. The title bar includes the AWS logo, Services, Search, and keyboard shortcut [Alt+S]. The main area shows a command being run: "kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 -d". The output of the command is partially visible as "x-WfMqTWejLjHugv".

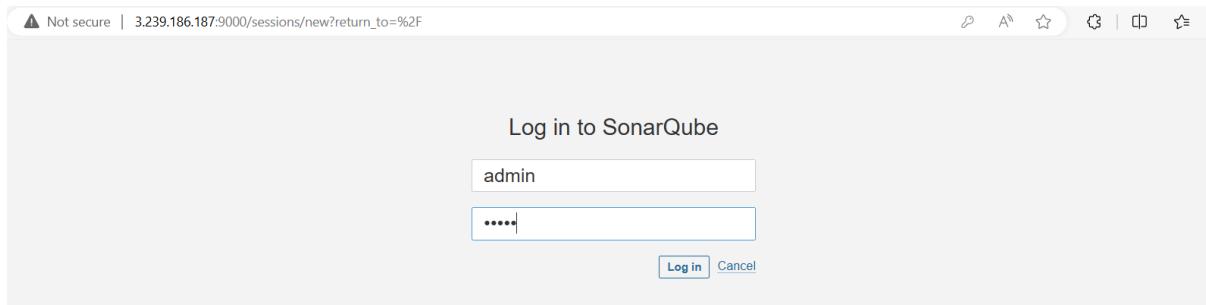
Enter the username (admin) and password in argoCD and click on SIGN IN.

Here is our ArgoCD Dashboard.



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

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



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

Goto: Administration > Security > users

Now, we have to configure webhooks for quality checks.

Click on Administration then, Configuration and select Webhooks

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

Click on Manually

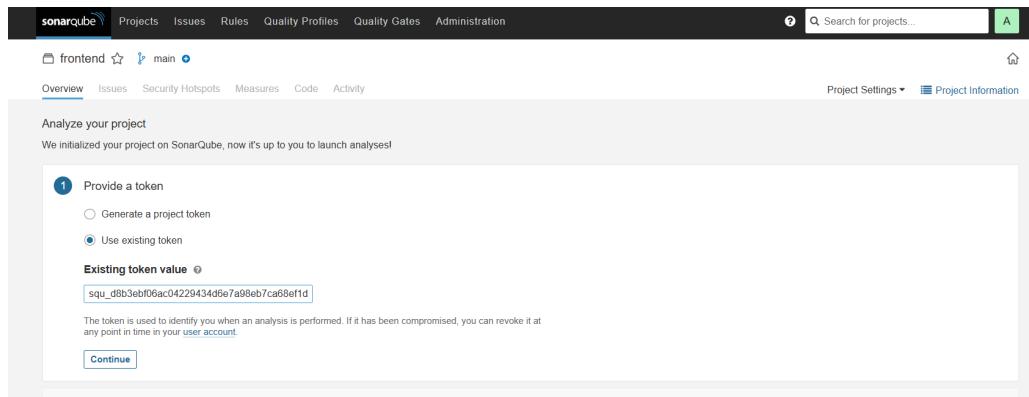
The screenshot shows the SonarQube interface with a dark header bar containing the logo, 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar with placeholder text 'Search for projects...' and a green button with a letter 'A' are also visible. Below the header, a message says 'How do you want to create your project?'. It asks if the user wants to benefit from SonarQube's features like repository import and Pull Request decoration. It then instructs the user to set up a DevOps platform configuration. Five options are shown: 'From Azure DevOps', 'From Bitbucket Server', 'From Bitbucket Cloud', 'From GitHub', and 'From GitLab', each with a small icon and a 'Set up global configuration' link. Below these, a section titled 'Are you just testing or have an advanced use-case? Create a project manually.' shows a button labeled 'Manually' with a double-angle bracket icon above it.

The screenshot shows the 'Create a project' form. At the top, it says 'Create a project' and notes that all fields marked with * are required. The 'Project display name *' field contains 'frontend' with a green checkmark. A note below says 'Up to 255 characters. Some scanners might override the value you provide.' The 'Project key *' field contains 'frontendkey' with a green checkmark. A note below says '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.' The 'Main branch name *' field contains 'main'. A note below says 'The name of your project's default branch' with a 'Learn More' link. At the bottom right of the form is a blue 'Set Up' button.

Click on Locally.

The screenshot shows the SonarQube interface with a dark header bar containing the logo, 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar with placeholder text 'Search for projects...' and a green button with a letter 'A' are also visible. Below the header, a message says 'How do you want to analyze your repository?'. It asks if the user wants to integrate with their favorite CI. It then instructs the user to analyze their project locally. Six options are shown: 'With Jenkins', 'With GitHub Actions', 'With Bitbucket Pipelines', 'With GitLab CI', 'With Azure Pipelines', and 'Other CI', each with a small icon. Below these, a section titled 'Are you just testing or have an advanced use-case? Analyze your project locally.' shows a button labeled 'Locally' with a computer monitor icon above it.

Select the Use existing token and click on Continue.

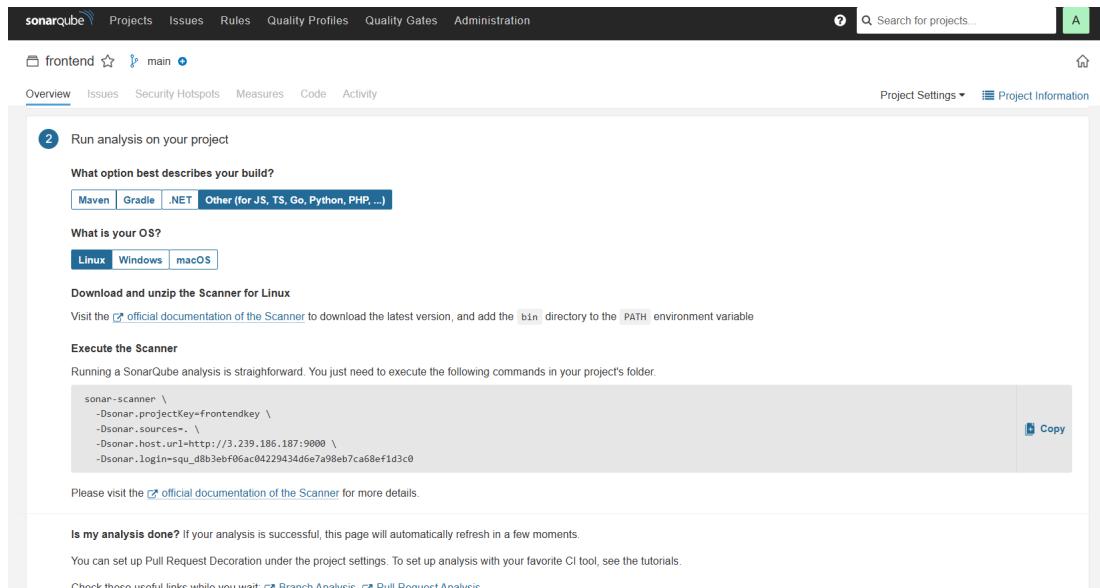


The screenshot shows the SonarQube interface for a project named 'frontend'. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. Below the navigation, there are tabs for Overview, Issues, Security Hotspots, Measures, Code, and Activity. On the right side, there are Project Settings and Project Information buttons. The main content area is titled 'Analyze your project' with the sub-instruction 'We initialized your project on SonarQube, now it's up to you to launch analyses!'. A large callout box labeled '1 Provide a token' contains two options: 'Generate a project token' (radio button) and 'Use existing token' (radio button, which is selected). Below this is a section for 'Existing token value' with a text input field containing the value 'squ_d8b3ebf06ac04229434d6e7a98eb7ca68ef1d'. A note below the input says, '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.' At the bottom of the callout box is a 'Continue' button.

Select Other and Linux as OS.

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

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



The screenshot shows the SonarQube interface for the same 'frontend' project. The navigation bar and tabs are identical to the previous screenshot. The main content area is titled '2 Run analysis on your project'. It asks 'What option best describes your build?' with options for Maven, Gradle, .NET, and 'Other (for JS, TS, Go, Python, PHP, ...)'. The 'Other' option is selected. It also asks 'What is your OS?' with options for Linux, Windows, and macOS. The 'Linux' option is selected. Below this, there's a section titled 'Download and unzip the Scanner for Linux' with instructions to visit the official documentation to download the latest version and add the bin directory to the PATH environment variable. There's also a section titled 'Execute the Scanner' with a command line snippet: 'sonar-scanner \ -Dsonar.projectKey=frontendkey \ -Dsonar.sources= \ -Dsonar.host.url=http://3.239.186.187:9000 \ -Dsonar.login=squ_d8b3ebf06ac04229434d6e7a98eb7ca68ef1d3c0'. A 'Copy' button is next to the command. At the bottom, there are links for 'Is my analysis done?', 'Branch Analysis', and 'Pull Request Analysis'.

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

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Create a project

All fields marked with * are required

Project display name *

 ✓

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

Project key *

 ✓

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 *

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

Set Up

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

backend main

Search for projects... A

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

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=backend1 \
-Dsonar.sources= \
-Dsonar.host.url=http://3.239.186.187:9000 \
-Dsonar.login=squ_08b3ebf06ac04229434d6ea98eb7ca68ef1d3c0
```

[Copy](#)

Please visit the [official documentation of the Scanner](#) for more details.

Is my analysis done? If your analysis is successful, this page will automatically refresh in a few moments.

You can set up Pull Request Decoration under the project settings. To set up analysis with your favorite CI tool, see the tutorials.

Check these useful links while you wait: [Branch Analysis](#), [Pull Request Analysis](#).

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

backend

Project's Main Branch is not analyzed yet. [Configure analysis](#)

frontend

Project's Main Branch is not analyzed yet. [Configure analysis](#)

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

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

Credentials

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

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

Credentials

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

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

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

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

The screenshot shows the Jenkins 'Manage Jenkins > Plugins' interface. On the left, a sidebar lists 'Updates' (6), 'Available plugins', 'Installed plugins' (selected), 'Advanced settings', and 'Download progress'. The main area has a search bar with 'docker' typed in. A table lists four Docker-related plugins:

Name	Enabled
Docker API Plugin 3.3.6-90.ve7c5c7535ddd	<input checked="" type="checkbox"/> (x)
Docker Commons Plugin 443.v921729d5611d	<input checked="" type="checkbox"/> (x)
Docker Pipeline 580.vc0c340686b_54	<input checked="" type="checkbox"/> (x)
Docker plugin 1.6.2	<input checked="" type="checkbox"/> (x)

At the bottom right of the table, a message says: 'This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.'

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

The screenshot shows the Jenkins 'Manage Jenkins > Tools' page. Under 'SonarQube Scanner installations', it shows one entry named 'sonar-scanner' which was 'Edited'. The configuration details are as follows:

- SonarQube Scanner**:
 - Name**: sonar-scanner
 - Install automatically**:
- Install from Maven Central**:
 - Version**: SonarQube Scanner 6.1.0.4477
 - Add Installer**: [Add](#)

At the bottom, there are 'Save' and 'Apply' buttons.

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

Dashboard > Manage Jenkins > Tools
NodeJS installations ^ Edited

Add NodeJS

NodeJS

Name
nodejs

Install automatically ?

Install from nodejs.org

Version
NodeJS 22.6.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail
 Force 32bit architecture

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

Global npm packages refresh hours
Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

Save **Apply**

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

Dependency-Check installations

Dependency-Check installations ^ Edited

Add Dependency-Check

Dependency-Check

Name
DP-Check

Install automatically ?

Install from github.com

Version
dependency-check 10.0.3

Add Installer ▾

Add Dependency-Check

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

Docker installations

Docker installations ^ Edited

Add Docker

Docker

Name: docker

Install automatically ?

Download from docker.com

Docker version ?

latest

Add Installer ▾

Add Docker

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

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

Dashboard > Manage Jenkins > System >

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 ▾

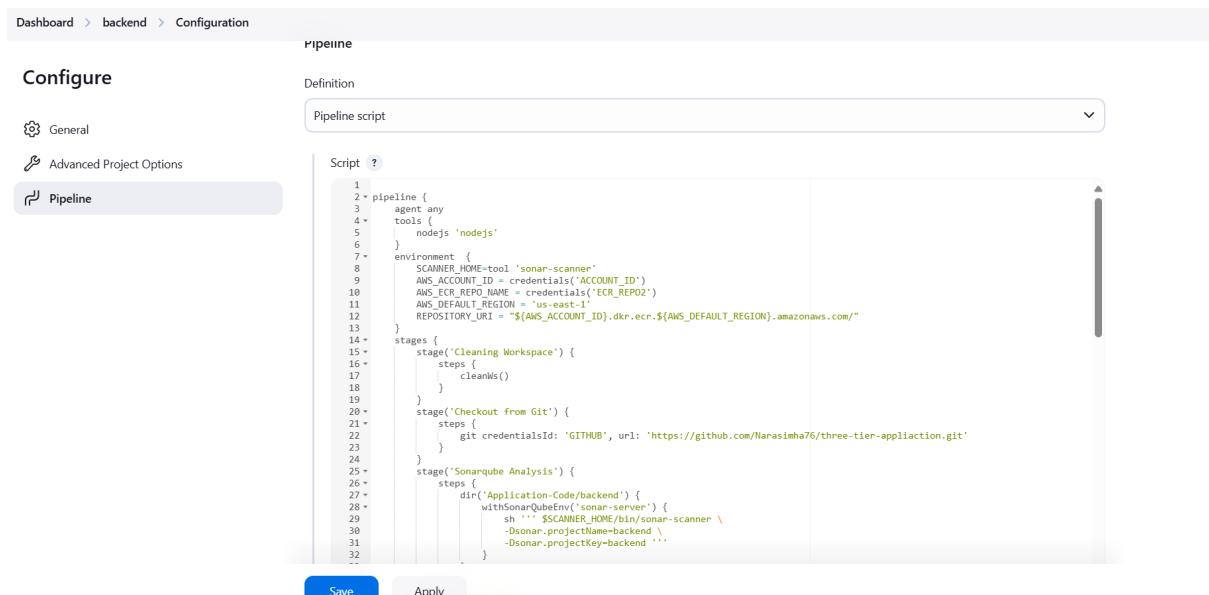
Advanced ▾

Save Apply

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

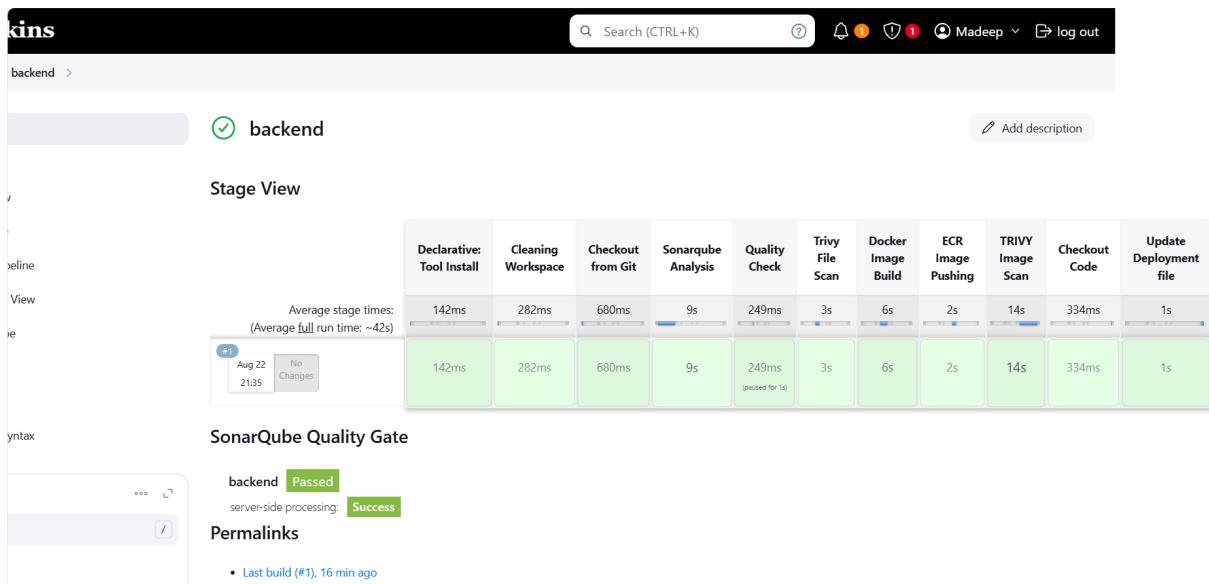
Create an Pipeline for the Backend Application

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



```
1 pipeline {
2   agent any
3   tools {
4     nodejs 'nodejs'
5   }
6   environment {
7     SCANNER_HOME_TOOL 'sonar-scanner'
8     AWS_ACCOUNT_ID = credentials('ACCOUNT_ID')
9     AWS_ECR_REPO_NAME = credentials('ECR_REPO2')
10    AWS_DEFAULT_REGION = 'us-east-1'
11    REPOSITORY_URI = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/"
12  }
13  stages {
14    stage('Cleaning Workspace') {
15      steps {
16        cleanWs()
17      }
18    }
19    stage('Checkout from Git') {
20      steps {
21        git credentialsId: 'GITHUB', url: 'https://github.com/Narasimha76/three-tier-appliaction.git'
22      }
23    }
24    stage('Sonarqube Analysis') {
25      steps {
26        dir("Application-Code/backend") {
27          withSonarQubeEnv('sonar-server') {
28            sh '''$SCANNER_HOME/bin/sonar-scanner \
29              -Dsonar.projectName=backend \
30              -Dsonar.projectKey=backend ...
31          }
32        }
33      }
34    }
35  }
36}
```

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



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

Amazon ECR > Private registry > Repositories > backend

backend

View push commands

Images (3101)

Search artifacts

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
1	Image	August 22, 2024, 21:35:33 (UTC+05.5)	353.94	Copy URI	sha256:fbf44ef3978ffcd...

Now Create another pipeline for the Frontend Application

- This is the Jenkins file to deploy the Frontend Code on EKS.
- Copy and paste it into the Jenkins Pipeline Script

<https://github.com/Narasimha76/three-tier-appliaction/blob/main/Jenkins-Pipeline-Code/Jenkinsfile-Frontend>
- Note: Do the changes in the Pipeline according to your project.

Dashboard > frontend > Configuration

Configure

Pipeline script

Script ?

```

1 pipeline
2 {
3     agent any
4     tools {
5         nodejs 'nodejs'
6     }
7     environment {
8         SCANNER_HOME=tool 'sonar-scanner'
9         AWS_ACCOUNT_ID = credentials('ACCOUNT_ID')
10        AWS_ECR_REPO_NAME = credentials('ECR_REPO1')
11        AWS_DEFAULT_REGION = 'us-east-1'
12        REPOSITORY_URI = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/${AWS_ECR_REPO_NAME}"
13    }
14    stages {
15        stage('Cleaning Workspace') {
16            steps {
17                cleanWs()
18            }
19        }
20        stage('Checkout from Git') {
21            steps {
22                git branch: 'main', credentialsId: 'GITHUB-APP', url: 'https://github.com/Narasimha76/three-tier-appliaction.git'
23            }
24        }
25        stage('Sonarque Analysis') {
26            steps {
27                dir('Application-Code/frontend') {
28                    withSonarQubeEnv('sonar-server') {
29                        sh """${SCANNER_HOME}/bin/sonar-scanner \
30                            -Dsonar.projectName=frontend \
31                            -Dsonar.projectKey=frontend \
32                            """
33                    }
34                }
35            }
36        }
37        stage('Quality Check') {
38            steps {
39            }
40        }
41    }
42 }

```

Save Apply

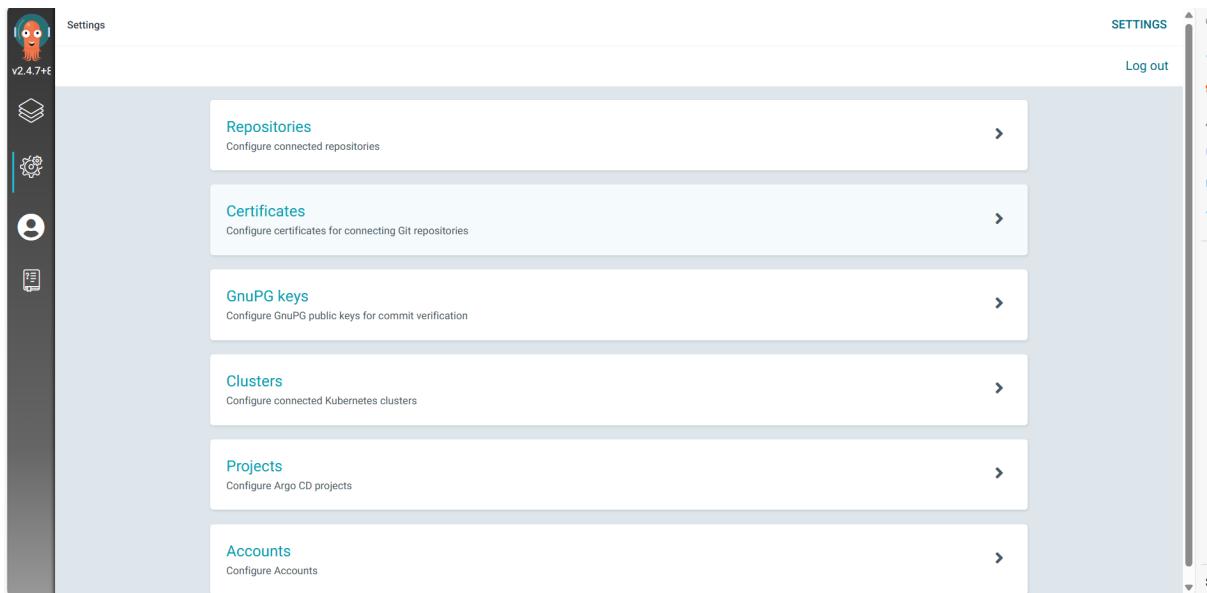
The screenshot shows the Jenkins Pipeline interface for a 'frontend' pipeline. On the left, there's a sidebar with various options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, SonarQube, Stages, Rename, and Pipeline Syntax. The main area has a title 'frontend' with a green checkmark icon. Below it is a 'Stage View' section showing a timeline of stages: Declarative: Tool Install (127ms), Cleaning Workspace (244ms), Checkout from Git (644ms), Sonarqube Analysis (11s), Quality Check (281ms), Docker Image Build (3s), ECR Image Pushing (3s), Checkout Code (389ms), and Update Deployment file (995ms). A tooltip indicates 'Average stage times: (Average full run time: ~22s)'. A summary bar at the bottom shows 'Aug 22 21:55' and 'No Changes'. To the right is a 'SonarQube Quality Gate' section with a green 'Passed' status for 'frontend' and a green 'Success' status for 'server-side processing'. Below this is a 'Builds' section showing one build (#1) from today at 4:25 PM, which is stable. A list of recent builds is also provided.

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

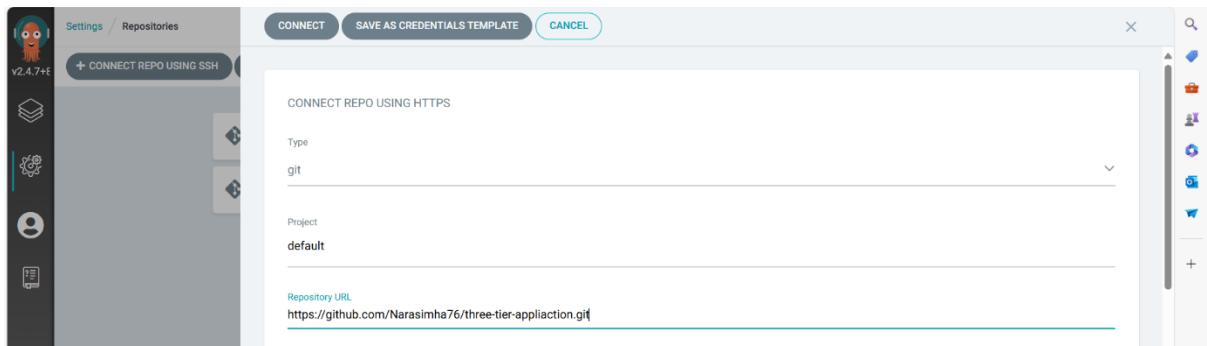
The screenshot shows the Amazon ECR Private registry interface. The path is 'Amazon ECR > Private registry > Repositories > frontend'. The 'frontend' repository page is displayed, with a 'View push commands' button at the top right. Below is a table titled 'Images (3126)' with a search bar. The columns are Image tag, Artifact type, Pushed at, Size (MB), Image URI, and Digest. One row is visible: '1' (Image tag), 'Image' (Artifact type), 'August 22, 2024, 21:55:24 (UTC+05:50)' (Pushed at), '424.83' (Size MB), 'Copy URI' (Image URI), and 'sha256:fbde0bed81b6c8...' (Digest).

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

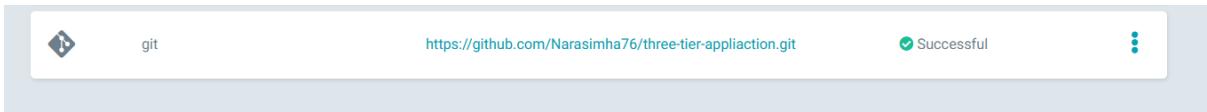
- As our repository is private. So, we need to configure the Private Repository in ArgoCD.
- Click on Settings and select Repositories.
- Add your github repository where the Kubernetes Manifest Files are present.



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



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



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

- Frontend app creation:

GENERAL

Application Name: three-tier-frontend

Project Name: default

Sync Policy: Automatic

SOURCE

Repository URL: <https://github.com/Madeep9347/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project.git>

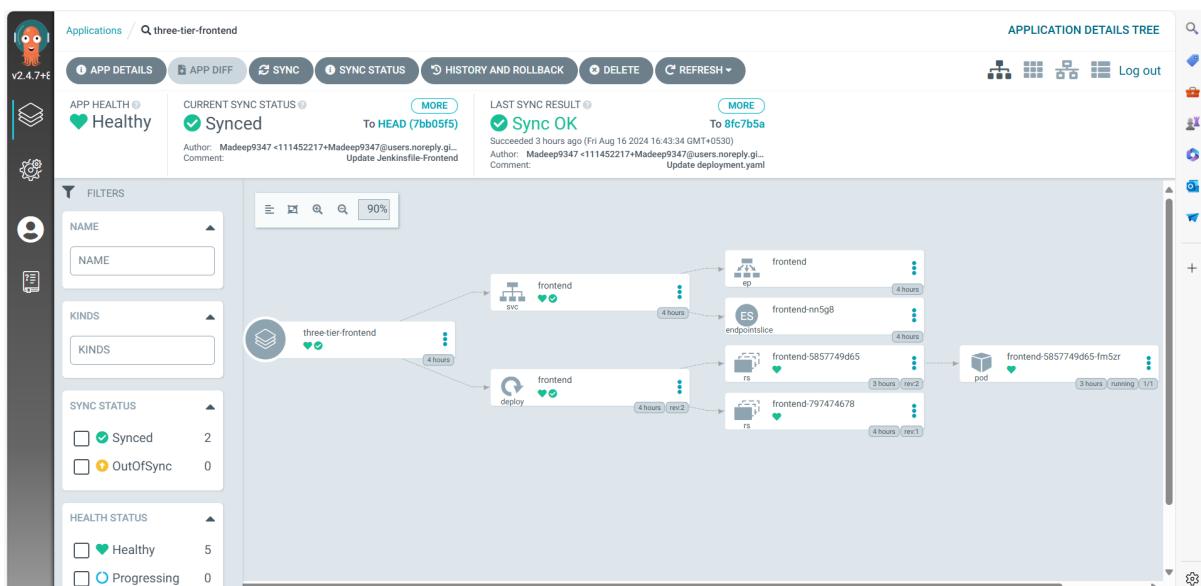
Revision: HEAD

DESTINATION

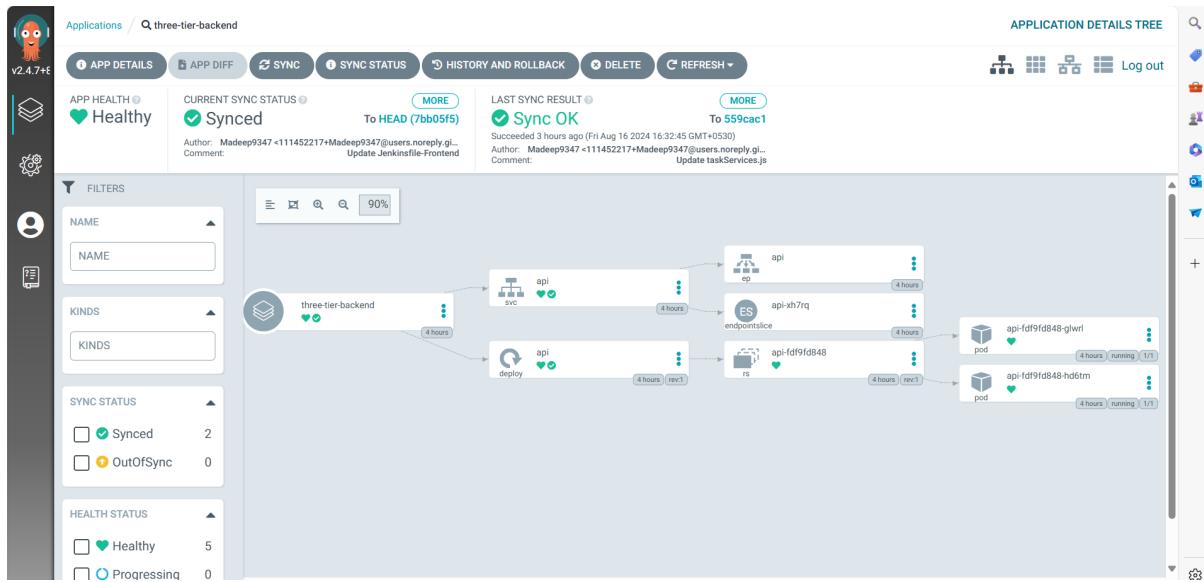
Cluster URL: <https://kubernetes.default.svc>

Namespace: three-tier

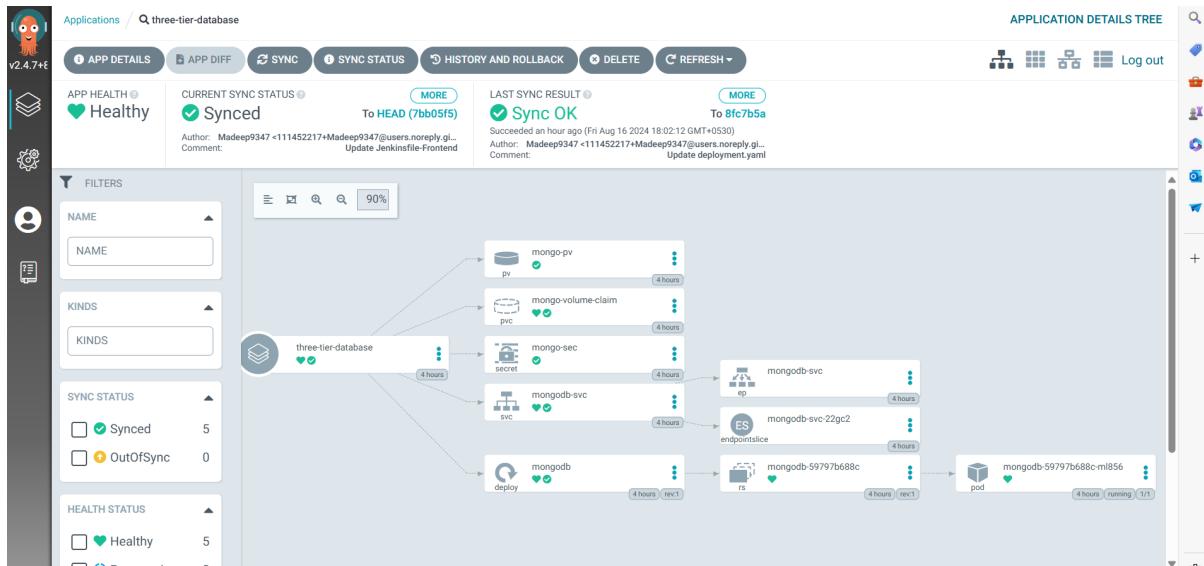
- This is the Frontend Application Deployment in ArgoCD:



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



- This is the Database Application Deployment in ArgoCD:



- This is the Ingress Application Deployment in ArgoCD:

The screenshot shows the ArgoCD interface for the 'three-tier-ingress' application. At the top, there are tabs for APP DETAILS, APP DIFF, SYNC, SYNC STATUS, HISTORY AND ROLLBACK, DELETE, and REFRESH. The SYNC STATUS section shows 'Synced' with a green checkmark and 'To HEAD (7bb05f5)'. The LAST SYNC RESULT section shows 'Sync OK' with a green checkmark and 'To a6735f4'. Below these are filter panels for NAME, KINDS, SYNC STATUS (Synced, OutOfSync), and HEALTH STATUS (Healthy, Progressing). A dependency graph shows 'three-tier-ingress' connected to 'mainlb' with a green arrow, both having a 'Sync OK' status and a 3-hour duration.

- If you observe, we have configured the Persistent Volume & Persistent Volume Claim. So, if the pods get deleted then, the data won't be lost. The Data will be stored on the host machine.
- To ensure all the pods and the service are running are not check in the eks cluster using command:
Kubectl get all -n three-tier

```
ubuntu@ip-10-16-13-49:~$ kubectl get all -n three-tier
NAME           READY   STATUS    RESTARTS   AGE
pod/api-fdf9fd848-q1wrl   1/1     Running   0          3h45m
pod/api-fdf9fd848-hdstm   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-threetier-mainlb-1lc5700e30-63935152.us-east-1.elb.amazonaws.com  80      3h24m
ubuntu@ip-10-16-13-49:~$
```

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

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

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

2. Edit Backend deployment file

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

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

We will achieve the monitoring using Helm

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

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

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

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

```
aws | Services | Search
app.kubernetes.io/part-of: prometheus
app.kubernetes.io/version: v2.54.0
helm.sh/chart: prometheus-25.26.0
name: prometheus-server
namespace: default
resourceVersion: "2291522"
uid: 5f57c455-5e68-4ebd-8238-e531210e1e97
spec:
  clusterIP: 172.20.130.81
  clusterIPs:
  - 172.20.130.81
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 9090
  selector:
    app.kubernetes.io/component: server
    app.kubernetes.io/instance: prometheus
    app.kubernetes.io/name: prometheus
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

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

```
aws | Services | Search
app.kubernetes.io/managed-by: Helm
app.kubernetes.io/name: grafana
app.kubernetes.io/version: 11.1.4
helm.sh/chart: grafana-8.4.5
name: grafana
namespace: default
resourceVersion: "2296798"
uid: ea4138f4-fd5b-4279-8b4f-2c8634e0c6a8
spec:
  clusterIP: 172.20.131.173
  clusterIPs:
  - 172.20.131.173
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: service
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: grafana
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

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

```
ubuntu@ip-10-16-13-49:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP
grafana        LoadBalancer 172.20.131.173  aea4138f4fd5b42798b4f2c8634e0c6a-1462404586.us-east-1.elb.amazonaws.com  80:30119/TCP
kubernetes     ClusterIP  172.20.0.1    <none>
prometheus-alertmanager ClusterIP 172.20.246.8  <none>
prometheus-alertmanager-headless ClusterIP None    <none>
prometheus-kube-state-metrics ClusterIP 172.20.24.217 <none>
prometheus-prometheus-node-exporter ClusterIP 172.20.35.64  <none>
prometheus-prometheus-pushgateway ClusterIP 172.20.179.123 <none>
prometheus-server LoadBalancer 172.20.130.81  a5f57c4555e684ebd8238e531210ele9-2124946716.us-east-1.elb.amazonaws.com  80:32333/TCP

```

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

The screenshot shows the Prometheus web interface. At the top, there are several input fields: 'Use local time' (unchecked), 'Enable query history' (unchecked), 'Enable autocomplete' (checked), 'Enable highlighting' (checked), and 'Enable linter' (checked). Below these is a search bar labeled 'Expression (press Shift+Enter for newlines)' with a magnifying glass icon. To the right of the search bar are buttons for 'Table' (selected), 'Graph', 'Evaluation time' (with arrows for navigation), and 'Execute'. Further down, there's a large empty panel area with the message 'No data queried yet'. At the bottom left is a blue 'Add Panel' button, and at the bottom right is a 'Remove Panel' link.

The screenshot shows the 'Targets' section of the Prometheus dashboard. At the top, there are buttons for 'All scrape pools' (selected), 'All', 'Unhealthy', and 'Collapse All'. To the right is a search bar labeled 'Filter by endpoint or labels' with a magnifying glass icon. Below these are two tables:

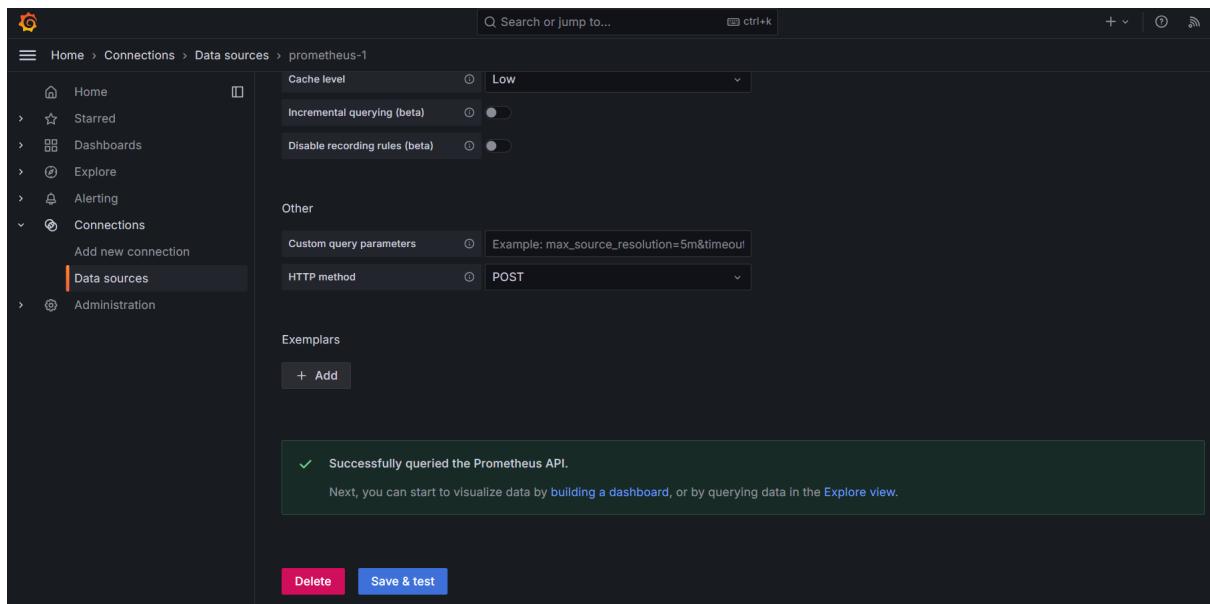
- kubernetes-apiservers (2/2 up)**: Shows two healthy targets. The first is 'https://10.16.135.232/metrics' with labels 'instance="10.16.135.232:443"', 'job="kubernetes-apiservers"', and 'scrape_duration="28.123s ago"'. The second is 'https://10.16.168.188/metrics' with similar labels.
- kubernetes-nodes (2/2 up)**: Shows one healthy target. The endpoint is 'https://kubernetes.default.svc/api/v1/nodes/ip-10-16-162-211.ec2.internal/proxy/metrics'. The labels include '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_nodegroup="dev-medium-eks-cluster-on-demand-nodes"', 'eks.amazonaws.com_nodegroup_image="ami-08d9654decbb01d5c"', 'failure_domain_beta_kubernetes_io_region="us-east-1"', 'failure_domain_beta_kubernetes_io_zone="us-east-1c"', 'instance="ip-10-16-162-211.ec2.internal"', and 'job="kubernetes-nodes"'.

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

The screenshot shows the Grafana Home page. On the left, there's a sidebar with links like Home, Starred, Dashboards, Explore, Alerting, Connections (with options for Add new connection and Data sources), and Administration. The main area has a title "Welcome to Grafana" and a "Basic" section with a "TUTORIAL" card about "DATA SOURCE AND DASHBOARDS" and "Grafana fundamentals". Below this, there are sections for "Dashboards" (listing Starred dashboards, Recently viewed dashboards, and K8s Resource Monitoring) and "Latest from the blog" (an article titled "Exemptions in Adaptive Metrics").

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

The screenshot shows the "Connections > Data sources" page for a Prometheus data source named "prometheus-1". The sidebar on the left shows the same navigation as the previous screenshot. The main panel shows the "Settings" tab for the "prometheus-1" data source. It includes fields for "Name" (set to "prometheus-1"), "Type" (set to "Prometheus"), and "Alerting" status ("Supported"). Below this, there's a "Connection" section with a "Prometheus server URL" input field containing the value "http://a5f57c4555e684ebd8238e531210e1e9-1930190". There's also an "Authentication" section at the bottom.



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

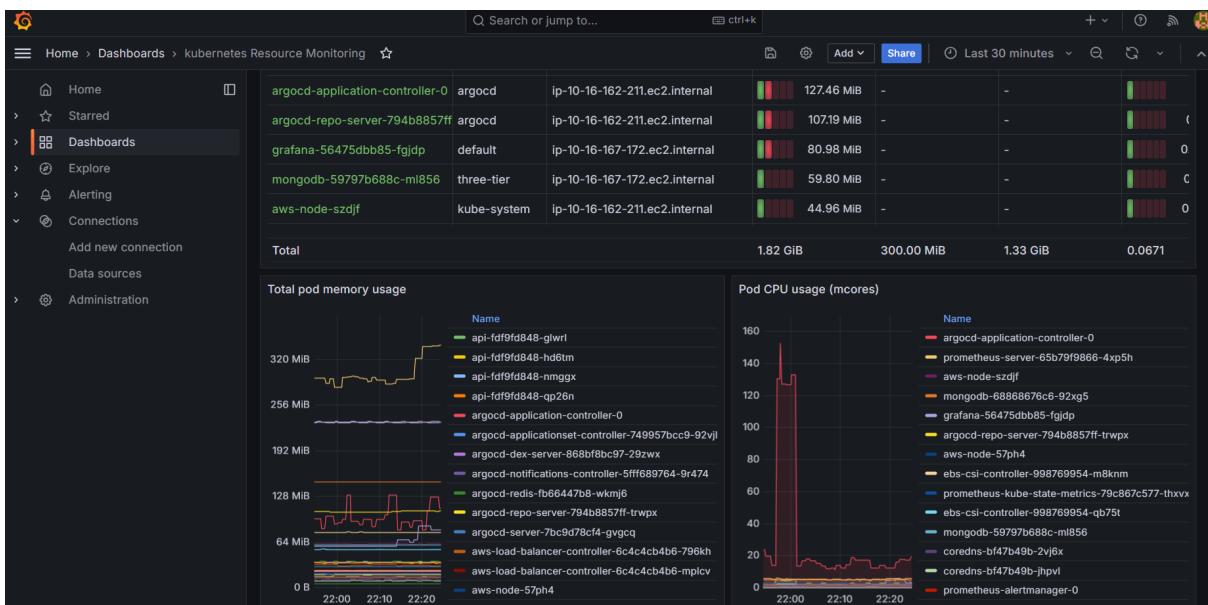
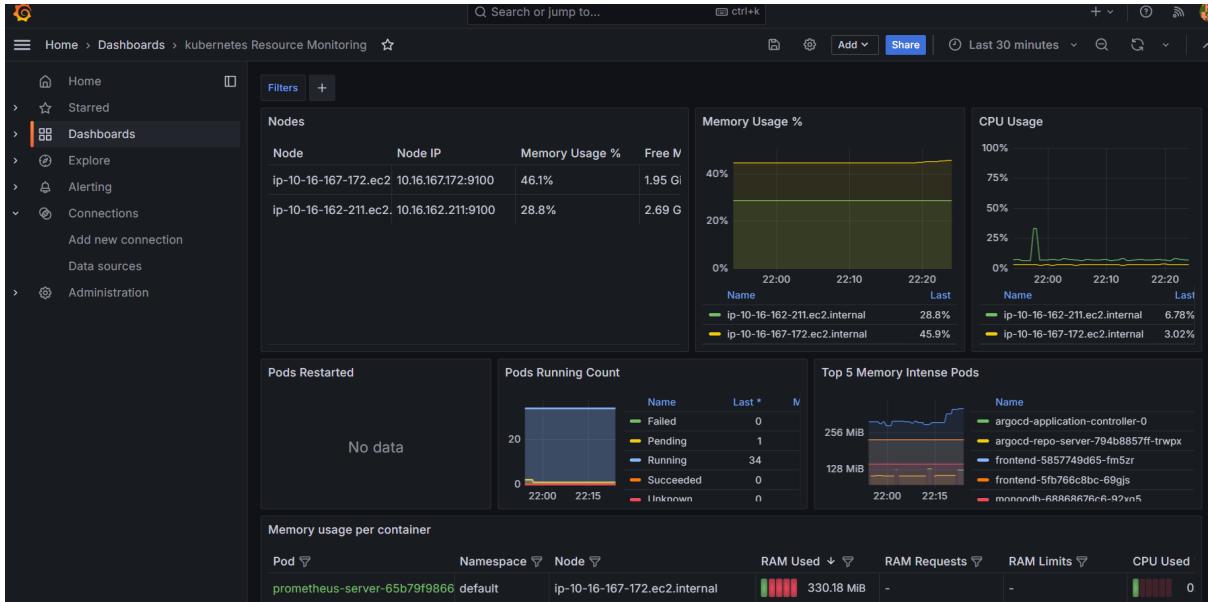
Now, we will create a dashboard to visualize our Kubernetes Cluster Logs

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

The screenshot shows the 'Import dashboard' page in Grafana. On the left, there's a sidebar with navigation links: Home, Starred, Dashboards (which is selected and highlighted in orange), Explore, Alerting, Connections (with 'Add new connection' and 'Data sources' sub-links), and Administration. The main area has a title 'Import dashboard' and a subtitle 'Import dashboard from file or Grafana.com'. It features a large dashed box for uploading a JSON file, with instructions: 'Upload dashboard JSON file', 'Drag and drop here or click to browse', and 'Accepted file types: .json, .txt'. Below this is a search bar with the placeholder 'Find and import dashboards for common applications at grafana.com/dashboards' and a 'Load' button. Underneath is a code editor-like area titled 'Import via dashboard JSON model' containing a snippet of JSON code. At the bottom are 'Load' and 'Cancel' buttons.

This screenshot shows the 'Import dashboard' page after selecting a dashboard from Grafana.com. The sidebar and top navigation are identical to the first screenshot. The main area now displays the title 'Importing dashboard from Grafana.com' and information about the source: 'Published by xroker' and 'Updated on 2022-11-08 22:28:19'. A section titled 'Options' allows setting the 'Name' (set to 'kubernetes Resource Monitoring') and 'Folder' (set to 'Dashboards'). Below these are fields for 'Unique identifier (UID)' (containing 'kubernetes-resource-monitoring') and 'Prometheus' (containing 'prometheus-1'). At the bottom are 'Import' and 'Cancel' buttons.

- You can view your Kubernetes Cluster Data.



Step-14: Create the DNS record for the ALB

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

```

aws | Services | Search [Alt+S]
ubuntu@ip-10-16-13-49:~$ kubectl get all -n three-tier
NAME           READY   STATUS    RESTARTS   AGE
pod/api-fdf9fd848-glwr1   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-ml856 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
ubuntu@ip-10-16-13-49:~$ kubectl get ing -n three-tier
NAME      CLASS      HOSTS      ADDRESS      PORTS      AGE
mainlb    alb        madhumadeep.world  k8s-threetier-mainlb-11c5700e30-63935152.us-east-1.elb.amazonaws.com  80      3h24m
ubuntu@ip-10-16-13-49:~$ 

```

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

Route 53 > Hosted zones > madhumadeep.world

madhumadeep.world Info

Hosted zone details Edit hosted zone

Records (3) Info

Automatic mode is the current search behavior optimized for best filter results. [To change modes go to settings.](#)

	Record ...	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...)	Health ..
<input type="checkbox"/>	madhuma...	NS	Simple	-	No	ns-931.awsdns-52.net. ns-1184.awsdns-20.org. ns-400.awsdns-50.com. ns-1962.awsdns-53.co.uk.	172800	-
<input type="checkbox"/>	madhuma...	SOA	Simple	-	No	ns-931.awsdns-52.net. awsd...	900	-
<input type="checkbox"/>	docker.ma...	A	Simple	-	No	15.206.70.80	300	-

- Choose Simple Routing

Route 53 > Hosted zones > madhumadeep.world > Create record

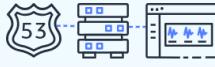
Step 1
Choose routing policy

Step 2
Configure records

Choose routing policy Info

The routing policy determines how Amazon Route 53 responds to queries.

Routing policy Switch to quick create

<input checked="" type="radio"/> Simple routing Use if you want all of your clients to receive the same response(s).	<input type="radio"/> Weighted Use when you have multiple resources that do the same job, and you want to specify the proportion of traffic that goes to each resource. For example: two or more EC2 instances.
	
<input type="radio"/> Geolocation Use when you want to route traffic based on the location of your users.	<input type="radio"/> Latency Use when you have resources in multiple AWS Regions and you want to route traffic to the Region that provides the best latency.
	
<input type="radio"/> Failover	<input type="radio"/> Multivalue answer

Define simple record

Record name Info
To route traffic to a subdomain, enter the subdomain name. For example, to route traffic to blog.example.com, enter blog. If you leave this field blank, the default record name is the name of the domain.

.madhumadeep.world

Keep blank to create a record for the root domain.

Record type Info
The DNS type of the record determines the format of the value that Route 53 returns in response to DNS queries.

Choose when routing traffic to AWS resources for EC2, API Gateway, Amazon VPC, CloudFront, Elastic Beanstalk, ELB, or S3. For example: 192.0.2.44.

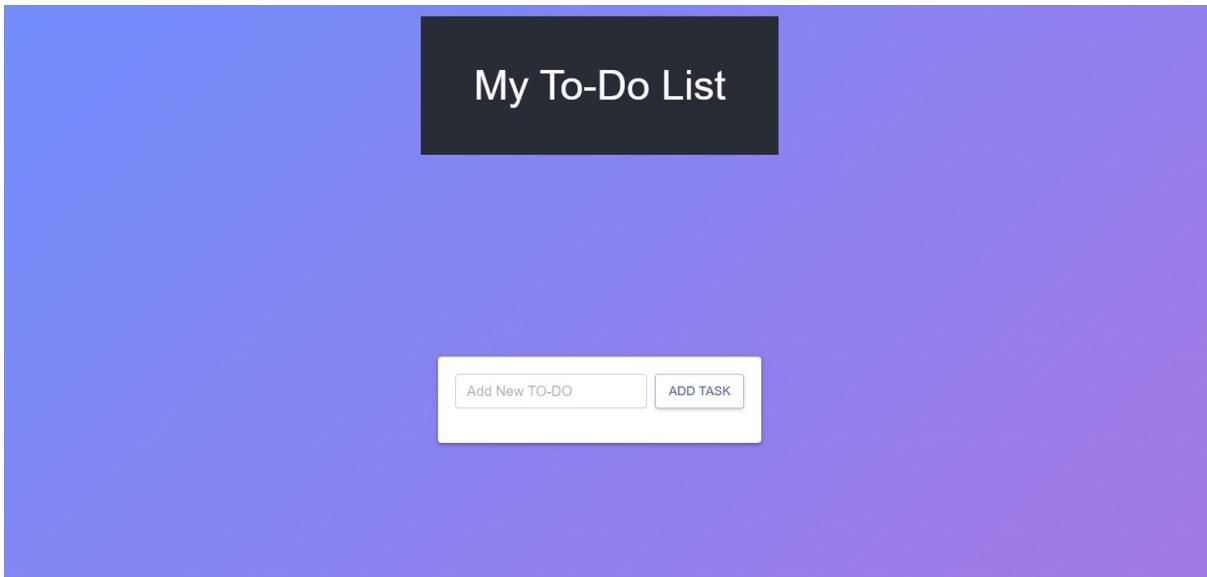
Value/Route traffic to Info
The option that you choose determines how Route 53 responds to DNS queries. For most options, you specify where you want to route internet traffic.

Evaluate target health
Select Yes if you want Route 53 to use this record to respond to DNS queries only if the specified AWS resource is healthy.

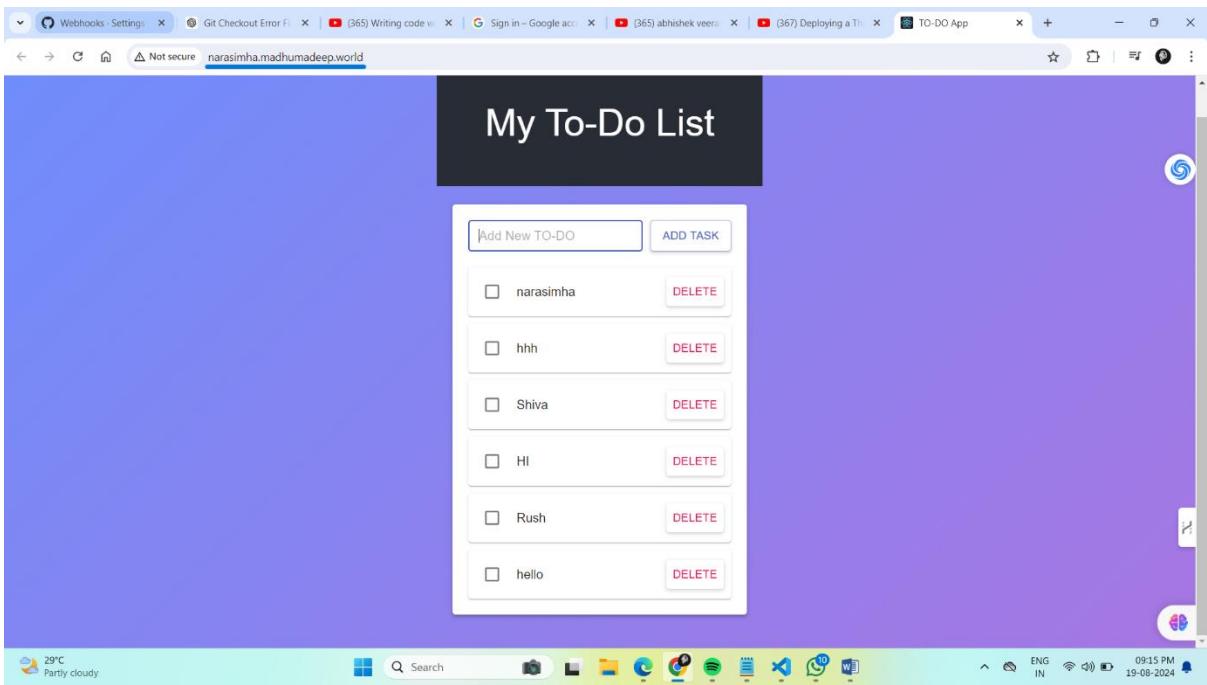
Yes

- Click on Define a simple record.

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



- Now test the application by adding some data



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

- Using the command "exec -it <mongodbd-container-name> -n <namespace> -- bin/sh

```

aws Services Search [Alt+S] N. Virginia madeep @ 5901-8412-40
ubuntu@ip-10-16-13-49:~$ kubectl exec -it mongodb-68868676c6-92xq5 -n narasimha --bin/sh
error: unknown flag: --bin/sh
See 'kubectl exec --help' for usage.
ubuntu@ip-10-16-13-49:~$ kubectl exec -it mongodb-68868676c6-92xq5 -n narasimha -- bin/sh
# mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("c001lc46-c527-4955-a80b-ab241b548104") }
MongoDB server version: 4.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2024-08-16T12:35:36.756+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2024-08-16T12:35:36.756+00:00: You are running this process as the root user, which is not recommended
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

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

```

aws Services Search [Alt+S] N. Virginia madeep @ 590
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2024-08-16T12:35:36.756+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
  2024-08-16T12:35:36.756+00:00: You are running this process as the root user, which is not recommended
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
todo 0.000GB
> use todo
switched to db todo
> db.tasks.find()
{
  "_id": ObjectId("66bf496d5a6e63e068e79ffc"),
  "completed": false,
  "task": "narasimha",
  "__v": 0
}
{
  "_id": ObjectId("66bf496f3b62fce42963909c"),
  "completed": false,
  "task": "hhh",
  "__v": 0
}
{
  "_id": ObjectId("66bf4d8c5a6e634a62e79fff"),
  "completed": false,
  "task": "Shiva",
  "__v": 0
}
{
  "_id": ObjectId("66bf50375a6e635ea7e7a003"),
  "completed": false,
  "task": "HI",
  "__v": 0
}
{
  "_id": ObjectId("66c0ab215a6e63d183e7a005"),
  "completed": false,
  "task": "Rush",
  "__v": 0
}
{
  "_id": ObjectId("66c366f95a6e636233e7a008"),
  "completed": false,
  "task": "hello",
  "__v": 0
}
```

```

> show dbs
admin 0.000GB
config 0.000GB
local 0.000GB
todo 0.000GB
> use todo
switched to db todo
> db.tasks.find()
{
  "_id": ObjectId("66bf496d5a6e63e068e79ffc"),
  "completed": false,
  "task": "narasimha",
  "__v": 0
}
{
  "_id": ObjectId("66bf496f3b62fce42963909c"),
  "completed": false,
  "task": "hhh",
  "__v": 0
}
{
  "_id": ObjectId("66bf4d8c5a6e634a62e79fff"),
  "completed": false,
  "task": "Shiva",
  "__v": 0
}
{
  "_id": ObjectId("66bf50375a6e635ea7e7a003"),
  "completed": false,
  "task": "HI",
  "__v": 0
}
{
  "_id": ObjectId("66c0ab215a6e63d183e7a005"),
  "completed": false,
  "task": "Rush",
  "__v": 0
}
{
  "_id": ObjectId("66c366f95a6e636233e7a008"),
  "completed": false,
  "task": "hello",
  "__v": 0
}
```

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

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