

◆ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



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



Aman Pathak · [Follow](#)

Published in Stackademic · 24 min read · Jan 18, 2024

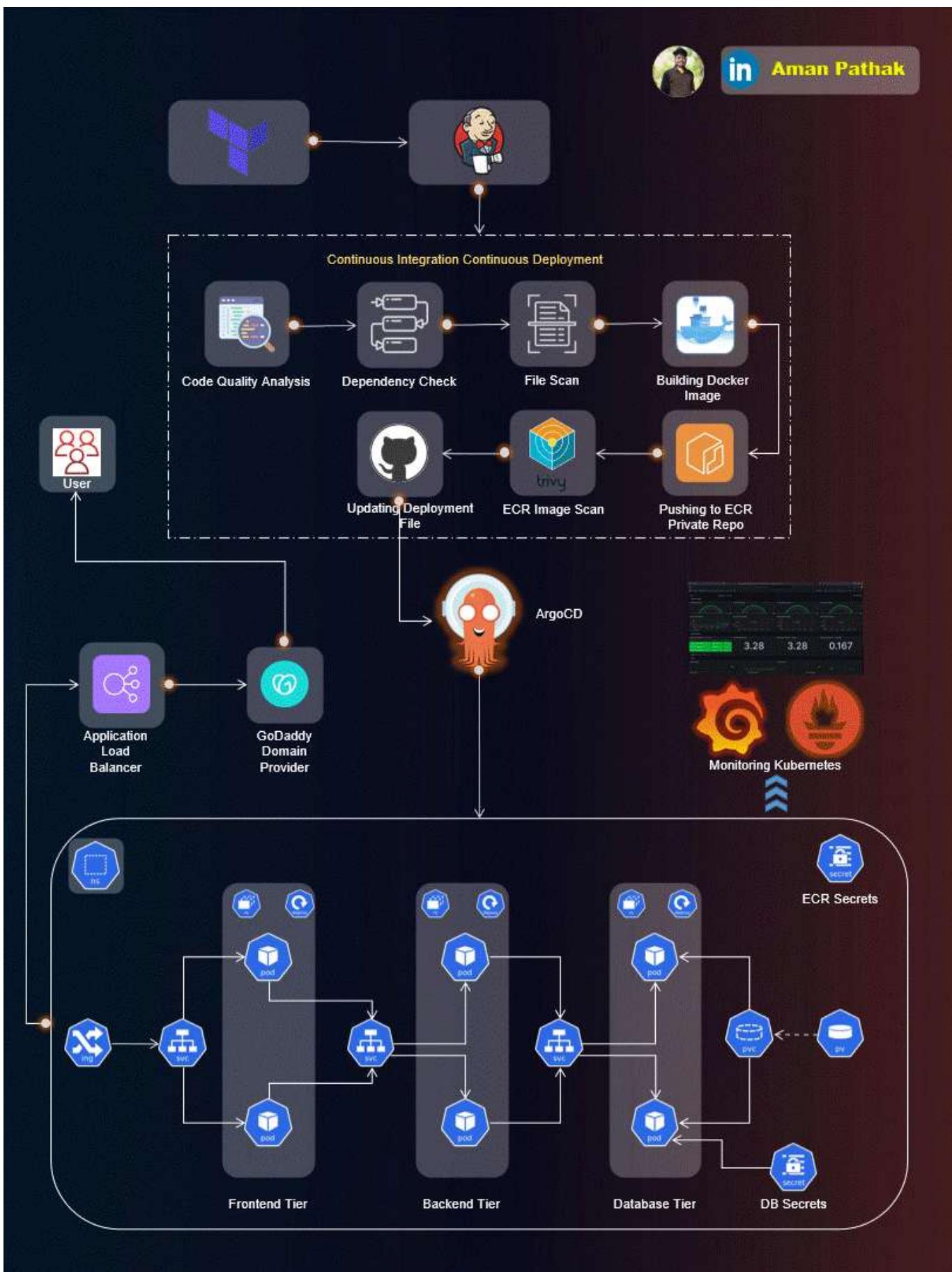
---

👏 2K

🗨 26



...



## Project Introduction:

Welcome to the End-to-End DevSecOps Kubernetes Project guide! In this comprehensive project, we will walk through the process of setting up a robust Three-Tier architecture on AWS using Kubernetes, DevOps best practices, and security measures. This project aims to provide hands-on experience in deploying, securing, and monitoring a scalable application environment.

## Project Overview:

In this project, we will cover the following key aspects:

1. **IAM User Setup:** Create an IAM user on AWS with the necessary permissions to facilitate deployment and management activities.
2. **Infrastructure as Code (IaC):** Use Terraform and AWS CLI to set up the Jenkins server (EC2 instance) on AWS.
3. **Jenkins Server Configuration:** Install and configure essential tools on the Jenkins server, including Jenkins itself, Docker, Sonarqube, Terraform, Kubectl, AWS CLI, and Trivy.
4. **EKS Cluster Deployment:** Utilize eksctl commands to create an Amazon EKS cluster, a managed Kubernetes service on AWS.
5. **Load Balancer Configuration:** Configure AWS Application Load Balancer (ALB) for the EKS cluster.
6. **Amazon ECR Repositories:** Create private repositories for both frontend and backend Docker images on Amazon Elastic Container Registry (ECR).
7. **ArgoCD Installation:** Install and set up ArgoCD for continuous delivery and GitOps.

- 8. Sonarqube Integration:** Integrate Sonarqube for code quality analysis in the DevSecOps pipeline.
- 9. Jenkins Pipelines:** Create Jenkins pipelines for deploying backend and frontend code to the EKS cluster.
- 10. Monitoring Setup:** Implement monitoring for the EKS cluster using Helm, Prometheus, and Grafana.
- 11. ArgoCD Application Deployment:** Use ArgoCD to deploy the Three-Tier application, including database, backend, frontend, and ingress components.
- 12. DNS Configuration:** Configure DNS settings to make the application accessible via custom subdomains.
- 13. Data Persistence:** Implement persistent volume and persistent volume claims for database pods to ensure data persistence.
- 14. Conclusion and Monitoring:** Conclude the project by summarizing key achievements and monitoring the EKS cluster's performance using Grafana.

## **Prerequisites:**

Before starting the project, ensure you have the following prerequisites:

- An AWS account with the necessary permissions to create resources.
- Terraform and AWS CLI installed on your local machine.
- Basic familiarity with Kubernetes, Docker, Jenkins, and DevOps principles.

# Step 1: We need to create an IAM user and generate the AWS Access key

Create a new IAM User on AWS and give it to the AdministratorAccess for testing purposes (not recommended for your Organization's Projects)

Go to the AWS IAM Service and click on Users.

The screenshot shows the AWS IAM Dashboard. On the left, there's a sidebar with navigation links like 'Dashboard', 'Access management', 'Identity providers', and 'Account reports'. The main area has sections for 'Security recommendations', 'IAM resources' (with counts for Open groups, Users, Roles, Policies, and Identity providers), and 'What's new' (listing four recent changes). On the right, there's a panel for the 'AWS Account' (Account ID: 45370222962, Account alias: devsecops) and 'Quick Links' for managing access keys and MFA.

Click on Create user

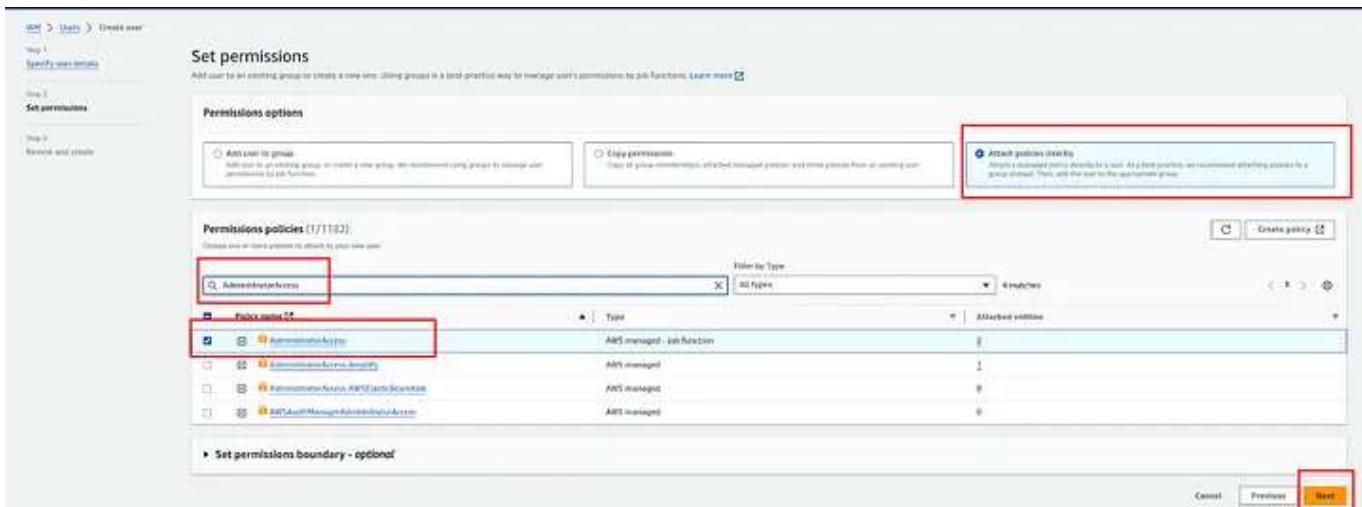
The screenshot shows the 'Users' list page. It displays four users and includes a search bar and navigation controls. A 'Create user' button is located in the top right corner.

Provide the name to your user and click on Next.



Select the **Attach policies directly** option and search for **AdministratorAccess** then select it.

Click on the **Next**.



Click on **Create user**

The screenshot shows the 'Review and create' step in the AWS IAM 'Create New User' wizard. The user is named 'DevSecOps-Project'. Under 'User details', the 'User name' is 'DevSecOps-Project', 'Access key (password) type' is 'None', and 'Project password reset' is set to 'No'. In the 'Permissions summary' section, the user has an 'Administrator permissions' policy attached. Below that, there's a 'Tags - removed' section with a note about tags being removed from the user. At the bottom right, there are 'Create', 'Review', and 'Create user' buttons.

Now, Select your created user then click on **Security credentials** and generate access key by clicking on **Create access key**.

The screenshot shows the 'Security credentials' tab for the 'DevSecOps-Project' user in the AWS IAM console. The 'Access keys' section is highlighted with a red box. It shows a single access key entry for 'devsecops-1' with the status 'Not rotated'. Below this, there are sections for 'Console sign-in' and 'Multi-factor authentication (MFA)'. At the bottom, there is a 'Create access key' button, which is also highlighted with a red box.

Select the **Command Line Interface (CLI)** then select the checkmark for the confirmation and click on **Next**.

**Access key best practices & alternatives** Info

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

**Use case**

- Command Line Interface (CLI)** You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code** You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service** You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service** You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS** You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.
- Other** Your use case is not listed here.

**Alternatives recommended**

- ⚠ Use the AWS CloudShell, a browser-based CLI, to run commands. [Learn more](#) Info
- Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. [Learn more](#) Info

**Confirmation**

I understand the above recommendation and want to proceed to create an access key.

[Cancel](#) [Next](#)

Provide the Description and click on the Create access key.

**Set description tag - optional** Info

The description for this access key will be attached to this user as a tag and shown alongside the access key.

**Description tag value** Describe the purpose of this access key and where it will be used. A good description will help you re-use this access key confidently later.

**EKS-Purpose** Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and \_/-+=@. Info

[Cancel](#) [Previous](#) [Create access key](#)

Here, you will see that you got the credentials and also you can download the CSV file for the future.

**Retrieve access keys** Info

**Access key** If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIAV52KNSRQWMB3YY	ALI59gKdtaByFdZjhOnqOJiaC0HsE5N+LxFk8m <a href="#">Hide</a>

**Access key best practices**

- Never store your access key in plain text, in a code repository, or in code.
- Disable or delete access key when no longer needed.
- Enforce least-privilege permissions.
- Rotate access keys regularly.

For more details about managing access keys, see the [best practices for managing AWS access keys](#).

[Download .CSV file](#) [Done](#)

## Step 2: We will install Terraform & AWS CLI to deploy our Jenkins Server(EC2) on AWS.

Install & Configure Terraform and AWS CLI on your local machine to create Jenkins Server on AWS Cloud

### Terraform Installation Script

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com/ stable main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt install terraform -y
```

### AWSCLI Installation Script

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install unzip -y
unzip awscliv2.zip
sudo ./aws/install
```

Now, Configure both the tools

### Configure Terraform

Edit the file /etc/environment using the below command add the highlighted lines and add your keys in the blur space.

```
sudo vim /etc/environment
```

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
export AWS_ACCESS_KEY_ID=XXXXXXXXXXXXXX
export AWS_SECRET_ACCESS_KEY=XXXXXXXXXXXXXX
export AWS_DEFAULT_REGION=us-east-1
export AWS_CONFIG_FILE="/root/.aws/config"
export TF_VAR_AWS_REGION=us-east-1
export TF_VAR_AWS_ACCOUNT_ID=
export TF_VAR_ENDPOINT=
export TF_VAR PEMFILE=/home/amanpathak/Download
figlet DevOps
```

After doing the changes, restart your machine to reflect the changes of your environment variables.

## Configure AWS CLI

Run the below command, and add your keys

aws configure

```
amanpathak@pop-os:~$ aws configure
AWS Access Key ID [None]: AKIAV52BKN5RIQVMB3YV
AWS Secret Access Key [None]: kLt9vgKldfa4yKd2jh0nq0tJqRC0HuE9N+LKfkBm
Default region name [None]: us-east-1
Default output format [None]: json
```

## **Step 3: Deploy the Jenkins Server(EC2) using Terraform**

Clone the Git repository- <https://github.com/AmanPathak-DevOps/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project>

## Navigate to the Jenkins-Server-TF

Do some modifications to the backend.tf file such as changing the **bucket** name and **dynamodb** table(make sure you have created both manually on AWS Cloud).

```
Jenkins-Server-TF > backend.tf > terraform > required_providers > aws
1 terraform {
2   backend "s3" {
3     bucket      = "my-ews-baket1"
4     region      = "us-east-1"
5     key         = "End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF/terraform.tfstate"
6     dynamodb_table = "Lock-Files"
7     encrypt      = true
8   }
9   required_version = ">=0.13.0"
10  required_providers {
11    aws = {
12      version = ">= 2.7.0"
13      source  = "hashicorp/aws"
14    }
15  }
16 }
```

Now, you have to replace the Pem File name as you have some other name for your Pem file. To provide the Pem file name that is already created on AWS

```
Jenkins-Server-TF > variables.tfvars > iam-role
1 vpc-name      = "Jenkins-vpc"
2 igw-name      = "Jenkins-igw"
3 subnet-name   = "Jenkins-subnet"
4 rt-name       = "Jenkins-route-table"
5 sg-name       = "Jenkins-sg"
6 instance-name = "Jenkins-server"
7 key-name      = "Aman-Pathak"
8 iam-role      = "Jenkins-iam-role"
```

Initialize the backend by running the below command

```
terraform init
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ terraform init
Initializing the backend...
Successfully configured the backend "s3"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v5.31.0...
- Installed hashicorp/aws v5.31.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
○ amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ 
```

Run the below command to check the syntax error

```
terraform validate
```

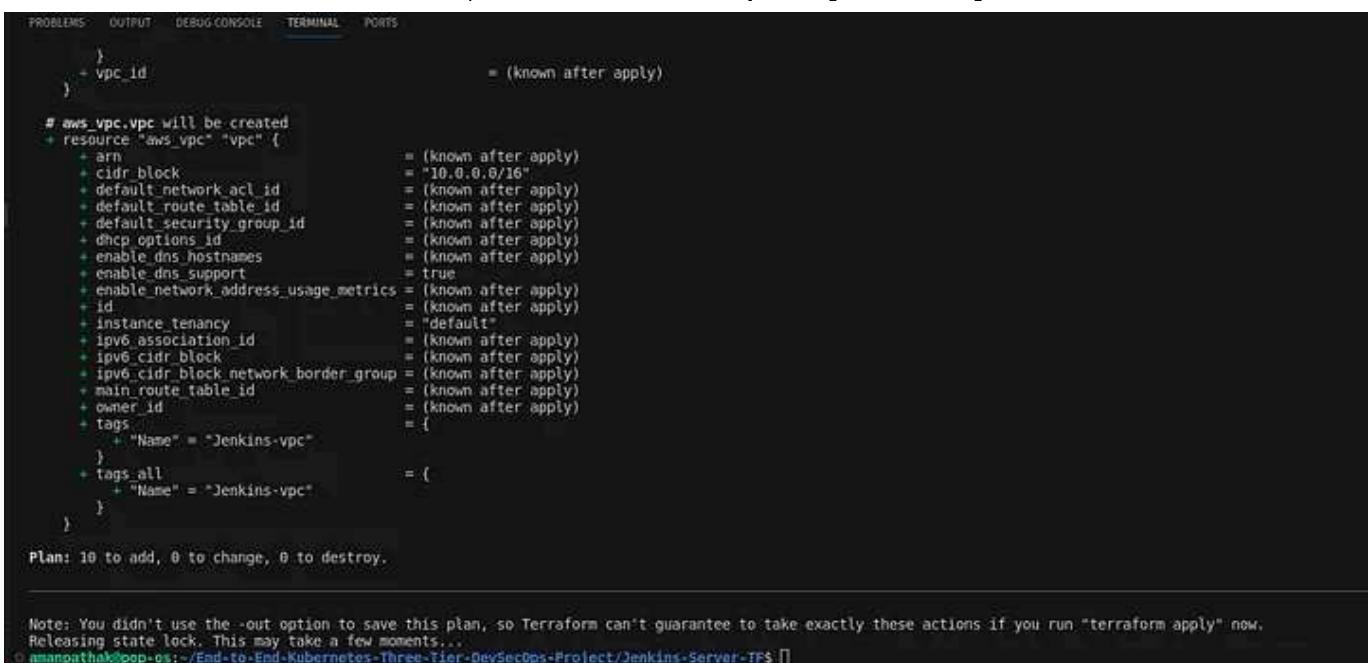
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ terraform validate
Success! The configuration is valid.

○ amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ 
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ terraform fmt
○ amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ 
```

Run the below command to get the blueprint of what kind of AWS services will be created.

```
terraform plan -var-file=variables.tfvars
```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

}
+ vpc_id = (known after apply)

# aws_vpc.vpc will be created
resource "aws_vpc" "vpc" {
  + arn = (known after apply)
  + cidr_block = "10.0.0.0/16"
  + default_network_acl_id = (known after apply)
  + default_route_table_id = (known after apply)
  + default_security_group_id = (known after apply)
  + dhcp_options_id = (known after apply)
  + enable_dns_hostnames = (known after apply)
  + enable_dns_support = true
  + enable_network_address_usage_metrics = (known after apply)
  + id = (known after apply)
  + instance_tenancy = "default"
  + ipv6_association_id = (known after apply)
  + ipv6_cidr_block = (known after apply)
  + ipv6_cidr_block.network_border_group = (known after apply)
  + main_route_table_id = (known after apply)
  + owner_id = (known after apply)
  + tags = {
    + "Name" = "Jenkins-vpc"
  }
  + tags_all = {
    + "Name" = "Jenkins-vpc"
  }
}

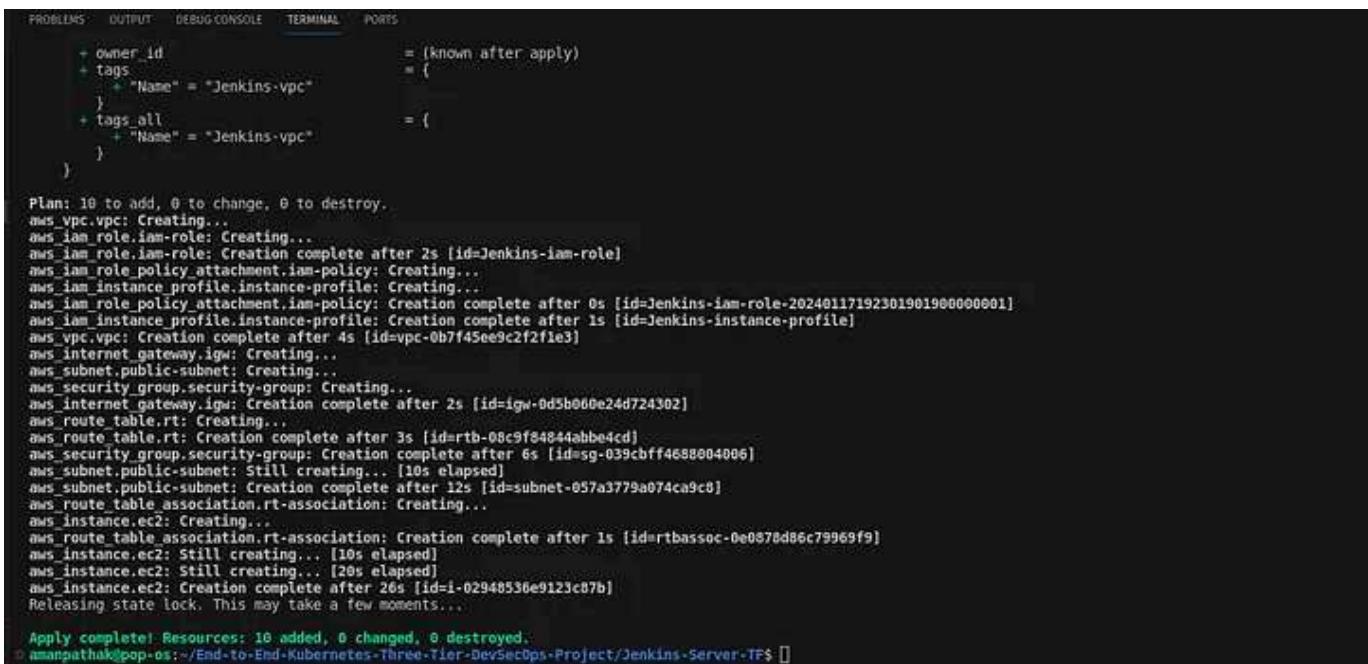
Plan: 10 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
Releasing state lock. This may take a few moments...
amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ []

```

Now, run the below command to create the infrastructure on AWS Cloud which will take 3 to 4 minutes maximum

```
terraform apply -var-file=variables.tfvars --auto-approve
```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

+ owner_id = (known after apply)
+ tags = {
  + "Name" = "Jenkins-vpc"
}
+ tags_all = {
  + "Name" = "Jenkins-vpc"
}

Plan: 10 to add, 0 to change, 0 to destroy.
aws_vpc.vpc: Creating...
aws_iam_role.iam-role: Creating...
aws_iam_role.iam-role: Creation complete after 2s [id=Jenkins-iam-role]
aws_iam_role_policy_attachment.iam-policy: Creating...
aws_iam_instance_profile.instance-profile: Creating...
aws_iam_role_policy_attachment.iam-policy: Creation complete after 0s [id=Jenkins-iam-role-20240117192301901900000001]
aws_iam_instance_profile.instance-profile: Creation complete after 1s [id=Jenkins-instance-profile]
aws_vpc.vpc: Creation complete after 4s [id=vpc-0b7f45ee9c2f2fie3]
aws_internet_gateway.igw: Creating...
aws_subnet.public-subnet: Creating...
aws_security_group.security-group: Creating...
aws_internet_gateway.igw: Creation complete after 2s [id=igw-0d5b060e24d724302]
aws_route_table.rt: Creating...
aws_route_table.rt: Creation complete after 3s [id=rtb-08c9f84844abbe4cd]
aws_security_group.security-group: Creation complete after 6s [id=sg-039cbff4688004006]
aws_subnet.public-subnet: Still creating... [10s elapsed]
aws_subnet.public-subnet: Creation complete after 12s [id=subnet-057a3779a074ca9c0]
aws_route_table_association.rt-association: Creating...
aws_instance.ec2: Creating...
aws_route_table_association.rt-association: Creation complete after 1s [id=rtbassoc-0e0878d86c79969f9]
aws_instance.ec2: Still creating... [10s elapsed]
aws_instance.ec2: Still creating... [20s elapsed]
aws_instance.ec2: Creation complete after 26s [id=i-02948536e9123c87b]
Releasing state lock. This may take a few moments...

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TF$ []

```

Now, connect to your Jenkins-Server by clicking on Connect.

The screenshot shows the AWS CloudWatch Instances console. It displays a table with one row for the instance 'Jenkins-server'. The instance has a Public IP of 23.22.152.186 and is currently running. The table includes columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability zone, Public IPv4 DNS, Public IPv4, Elastic IP, IPv6 IPs, Monitoring, and Security.

Copy the ssh command and paste it on your local machine.

The screenshot shows the 'Connect to instance' dialog box. It lists the instance ID as 'i-02948558e0123c87b (Jenkins-server)'. Below it, there is a sample SSH command: 'ssh -i 'Aman-Patnak.pem'' followed by the instance's Public IP, '23.22.152.186'. There are also instructions for opening an SSH port, locating a private key file, accepting the key fingerprint, and connecting via Public IP.

## Step 4: Configure the Jenkins

Now, we logged into our Jenkins server.

```
amxpathak@ip-10-0-1-72:~$ ssh -i "Aman-Patnak.pem" ubuntu@23.22.152.186
The authenticity of host '23.22.152.186 (23.22.152.186)' can't be established.
ED25519 Key fingerprint is SHA256:2G1jzm1JwE6yxEZx/jyITKw8ccy5MF8KwCsjL5hdAA.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '23.22.152.186' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1017-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

 System information as of Wed Jan 17 19:25:39 UTC 2024

System load: 1.3203125   Processes:          181
Usage of /:  8.9% of 28.89GB  Users logged in:    0
Memory usage: 12%
Swap usage:  0%

Expanded Security Maintenance for Applications is not enabled.

39 updates can be applied immediately,
25 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/**/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-1-72:~$ 
```

We have installed some services such as Jenkins, Docker, Sonarqube, Terraform, Kubectl, AWS CLI, and Trivy.

Let's validate whether all our installed or not.

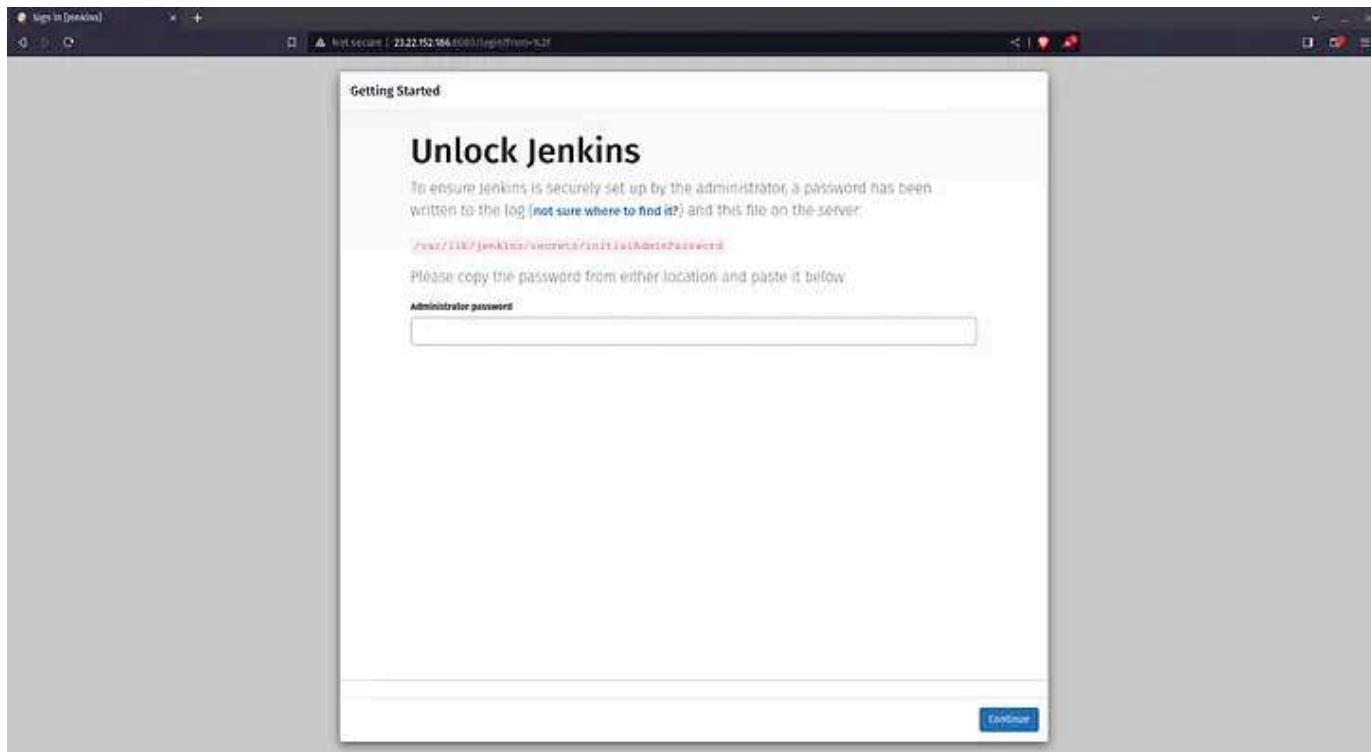
```
jenkins --version
docker --version
docker ps
terraform --version
kubectl version
aws --version
trivy --version
eksctl --version
```

```
ubuntu@ip-10-0-1-72:~$ docker version
Client:
Version: 24.0.5
API version: 1.43
Go version: go1.20.3
Git commit: 24.0.5-0ubuntu1-22.04.1
Built: Mon Aug 21 19:50:14 2023
OS/Arch: linux/amd64
Context: default

Server:
Engine:
Version: 24.0.5
API version: 1.43 (minimum version 1.12)
Go version: go1.20.3
Git commit: 24.0.5-0ubuntu1-22.04.1
Built: Mon Aug 21 19:50:14 2023
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.7.2
GitCommit:
runc:
Version: 1.1.7-0ubuntu1-22.04.1
GitCommit:
docker-init:
Version: 0.19.0
TKEC:
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$ jenkins --version
2.441
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e63ddc502b2f sonarqube:its-community "/opt/sonarqube/dock..." About a minute ago Up About a minute 0.0.0.0:9000->9000/tcp, :::9000->9000/tcp sonar
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$ terraform --version
Terraform v1.6.6
on linux_amd64
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$ kubectl version
Client Version: v1.28.4
Kustomize Version: v5.0.4-0.20230601165947-0ce0bf399ce3
Error from server (Forbidden): <html><head><meta http-equiv="refresh" content="1;url=/login?from=%2Fversion%3Ftimeout%3D32s"/><script id="redirect" data-redirect-url="/login?from=%2Fversion%3Ftimeout%3D32s" src="/static/1f42e217/scripts/redirect.js"></script></head><body style='background-color:white; color:white;'>
Authentication required
<!--
```

```
ubuntu@ip-10-0-1-72:~$ trivy --version
Version: 0.48.3
ubuntu@ip-10-0-1-72:~$ eksctl version
0.167.0
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$ aws --version
aws-cli/2.35.10 Python/3.11.6 Linux/6.2.0-1017-aws exe/x86_64/ubuntu.22 prompt/off
ubuntu@ip-10-0-1-72:~$
```

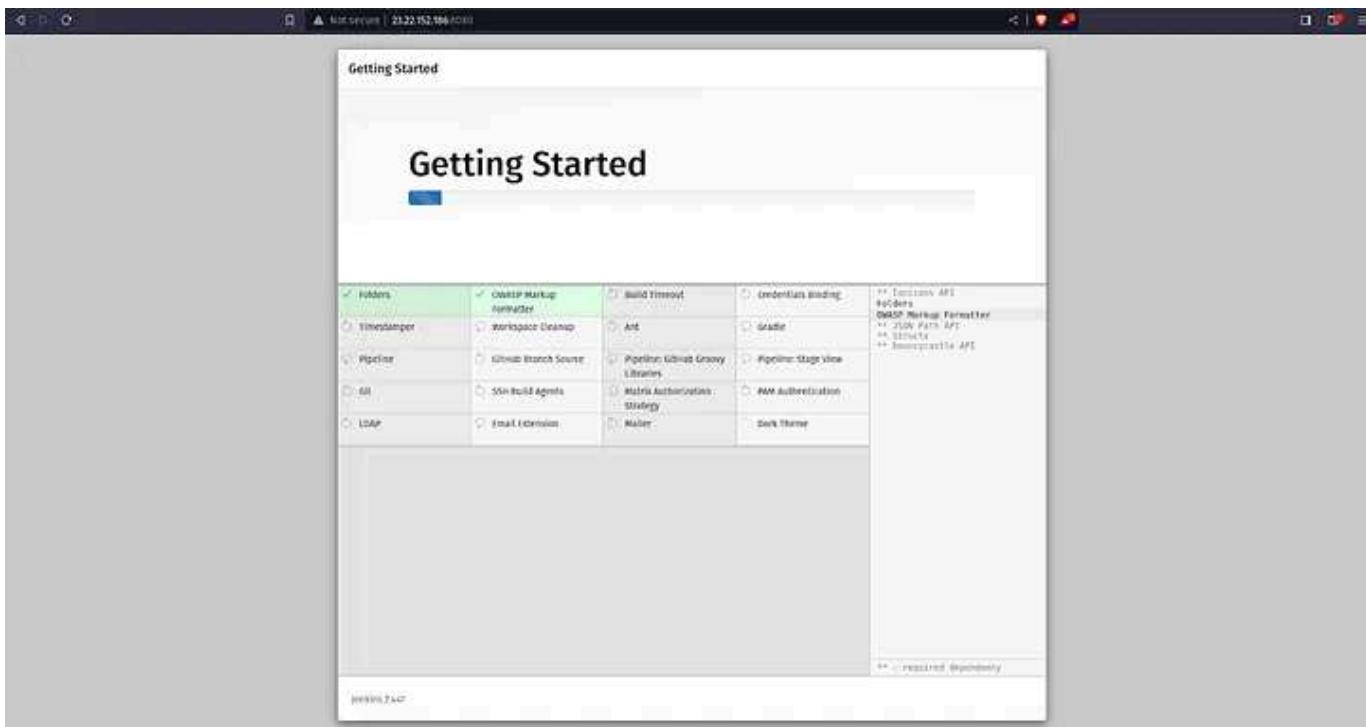
Now, we have to configure Jenkins. So, copy the public IP of your Jenkins Server and paste it on your favorite browser with an 8080 port.



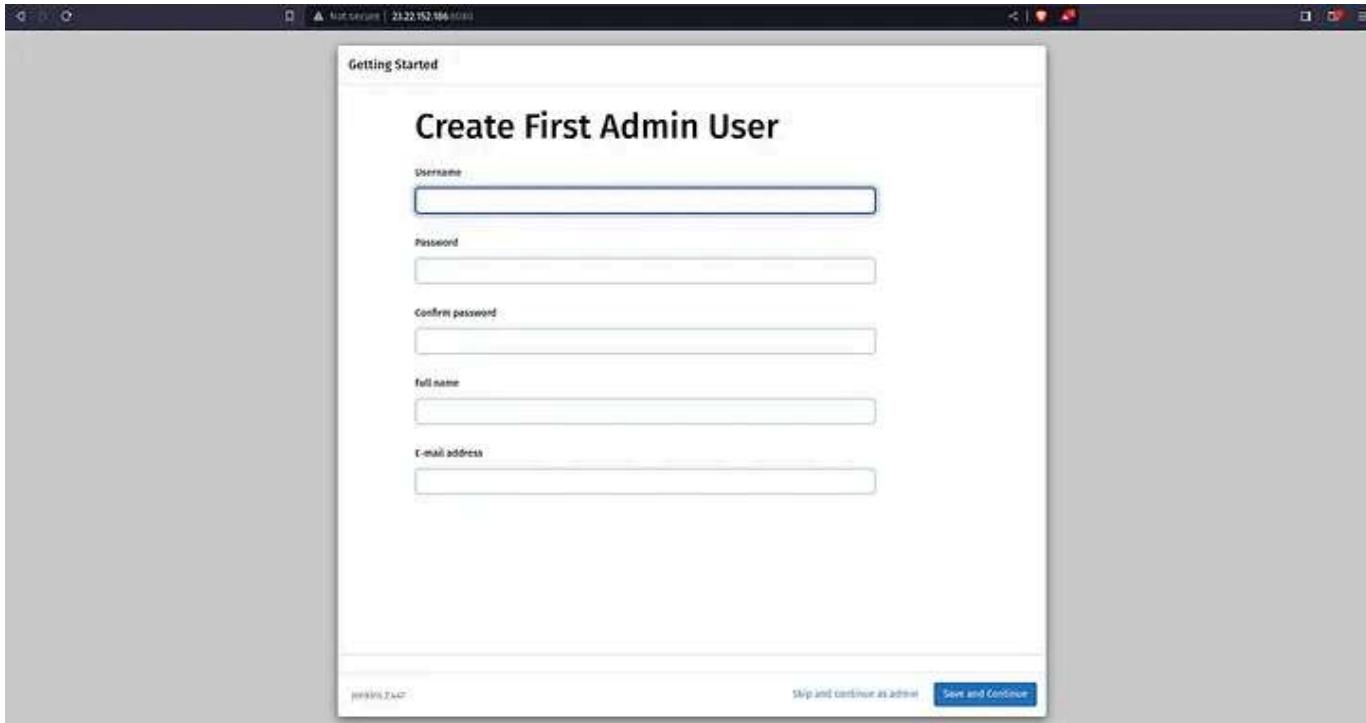
Click on **Install suggested plugins**



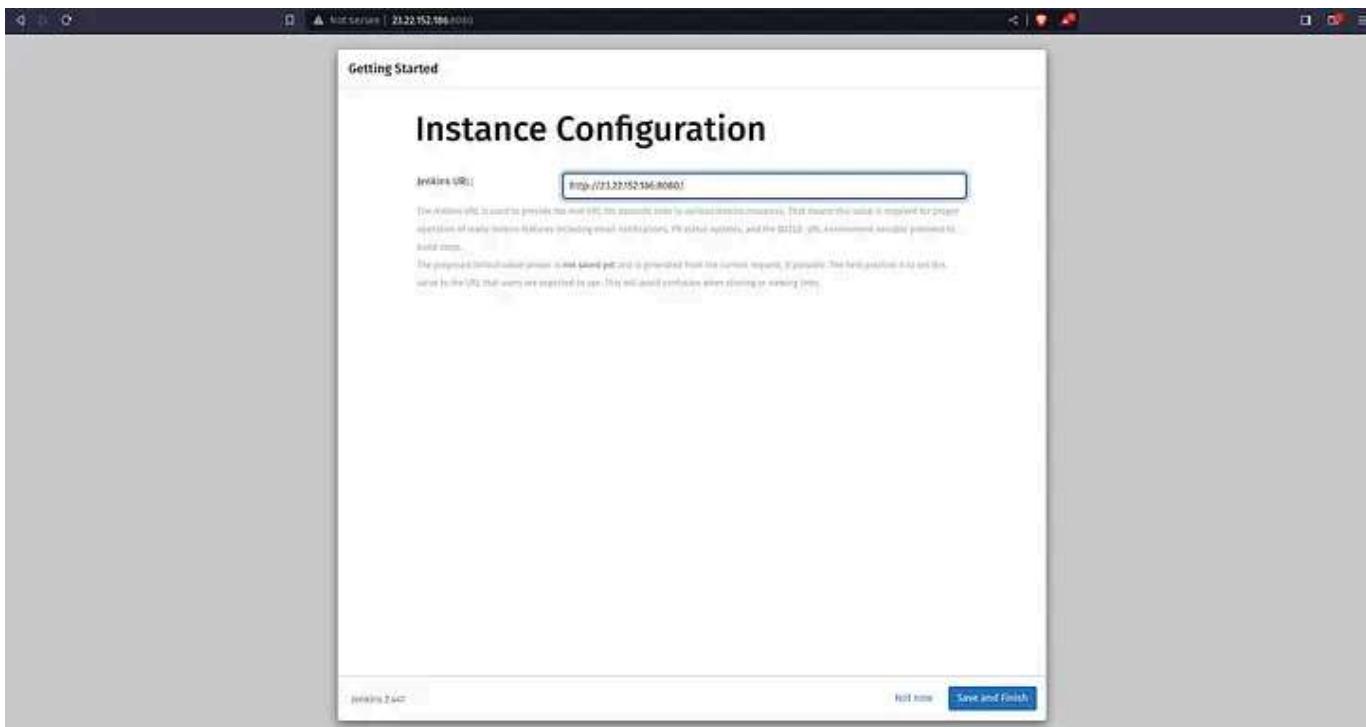
The plugins will be installed



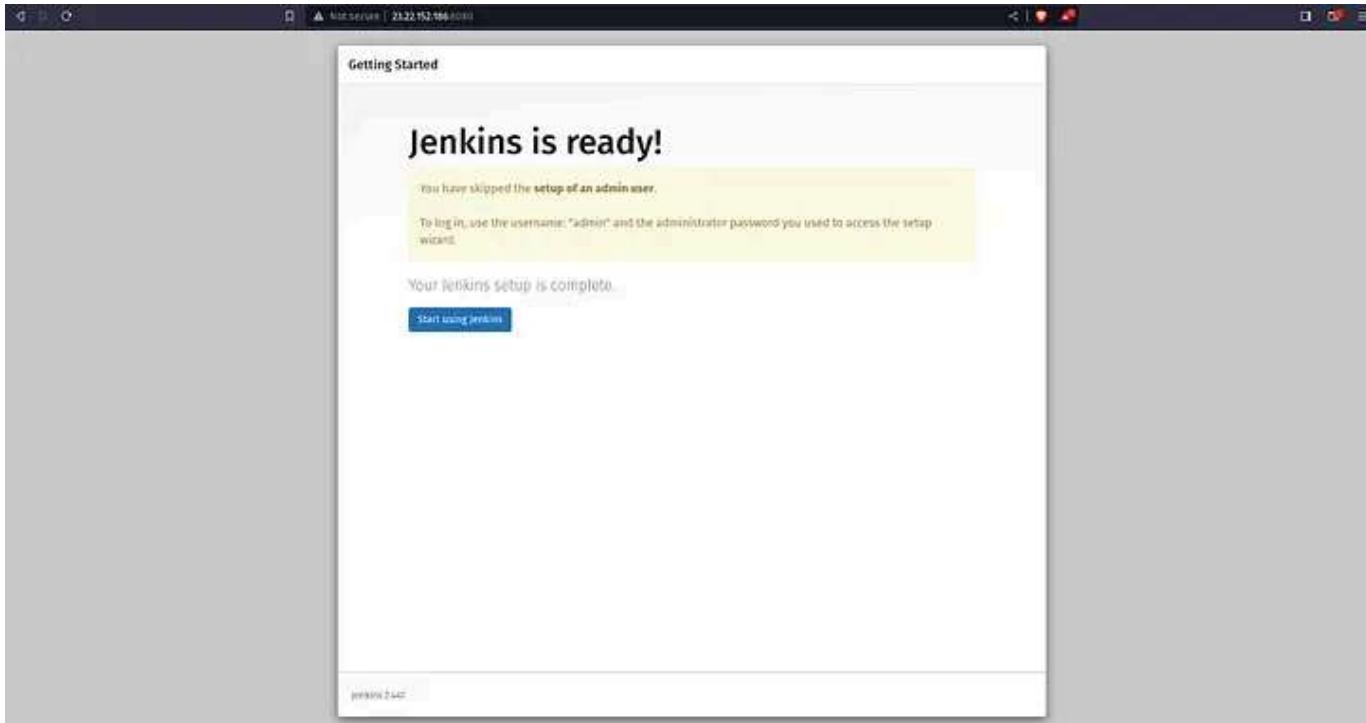
After installing the plugins, continue as admin



Click on **Save and Finish**



Click on Start using Jenkins



The Jenkins Dashboard will look like the below snippet

## Step 5: We will deploy the EKS Cluster using eksctl commands

Now, go back to your Jenkins Server terminal and configure the AWS.

```
ubuntu@ip-10-0-1-72:~$ aws configure
AWS Access Key ID [None]: AKIAV52BKN5RJWCSSA6P
AWS Secret Access Key [None]: l8Hyy+5Jee3aTxmqEFiS6/H6rBLXj0G3mJ89dqdg
Default region name [None]: us-east-1
Default output format [None]: json
ubuntu@ip-10-0-1-72:~$ 
```

Go to Manage Jenkins

Click on Plugins

The screenshot shows the Jenkins 'Manage Jenkins' page under the 'System Configuration' tab. On the left sidebar, 'Manage Jenkins' is selected. The main area displays several configuration sections: 'System' (Configure global settings and paths), 'Tools' (Configure tools, their locations and automatic installers), 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), 'Nodes' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on), 'Clouds' (Add, remove, and configure cloud instances to provision agents on-demand), 'Appearance' (Configure the look and feel of Jenkins), 'Security' (Secure Jenkins; define who is allowed to access the system), 'Credentials' (Configure credentials), 'Credential Providers' (Configure the credential providers and types), and 'Users' (Create / delete / modify users that can log in to this Jenkins). A yellow banner at the top states: "Building on the built-in code can be a security issue. You should set up distributed builds. See the documentation."

Select the Available plugins install the following plugins and click on Install

## AWS Credentials

## Pipeline: AWS Steps

The screenshot shows the Jenkins 'Plugins' page under the 'Available plugins' tab. The search bar contains 'aws step'. Two plugins are listed: 'AWS Credentials' (version 2.18.1) and 'Pipeline: AWS Steps' (version 1.43). Both have the 'Install' button checked.

Plugin	Description	Released
AWS Credentials	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 API Tokens.	6 mo 21 days ago
Pipeline: AWS Steps	This plugin adds Jenkins pipeline steps to interact with the AWS API.	3 yr 1 mo ago

Once, both the plugins are installed, restart your Jenkins service by checking the **Restart Jenkins** option.

Amazon Web Services SDK - IAM  
Amazon Web Services SDK - ECR  
Amazon Web Services SDK - EFS  
Amazon Web Services SDK - SSM  
Amazon Web Services SDK - Lambda  
Amazon Web Services SDK - Logs  
Amazon Web Services SDK - CodeBuild  
Amazon Web Services SDK - Secrets Manager  
Amazon Web Services SDK - AI  
Pipeline AWS Step Functions  
Loading plugin extensions

Install Success Success Success Success Success  
Installing Pending Pending Pending Pending Pending

→ Go back to the top page  
(you can start using the installed plugins right away)  
→  Restart Jenkins when installation is complete and no jobs are running

## Login to your Jenkins Server Again



### Sign in to Jenkins

Username:

Password:

Keep me signed in

**Sign in**

Now, we have to set our AWS credentials on Jenkins

Go to Manage Plugins and click on Credentials

The screenshot shows the Jenkins Manage Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is currently selected), and 'My Views'. Below these are 'Build Queue' and 'Build Executor Status' sections. The main area is titled 'Manage Jenkins' with a sub-section 'System Configuration'. It contains several cards: 'System' (Configure global settings and paths), 'Tools' (Configure tools, their locations and automatic installers), 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), 'Clouds' (Add, remove, and configure cloud instances to provision agents on-demand), 'Appearance' (Configure the look and feel of Jenkins), 'Credential Providers' (Configure the credential providers and types), and 'Users' (Create / delete / modify users that can log in to this Jenkins). A yellow banner at the top right says 'Building on the built-in code can be a security issue. You should set up distributed builds. See the documentation.'

Click on global.

The screenshot shows the Jenkins Credentials page. The URL in the address bar is 'Dashboard > Manage Jenkins > Credentials'. The main title is 'Credentials'. Below it, there's a table header with columns 'T', 'P', 'Store', 'Domain', 'ID', and 'Name'. A sub-section titled 'Stores scoped to Jenkins' is shown, listing a single store named 'System' under the 'Domains' column, with '(global)' selected. At the bottom, there are navigation buttons for 'New', 'S', 'M', and 'L'.

Select AWS Credentials as Kind and add the ID same as shown in the below snippet except for your AWS Access Key & Secret Access key and click on Create.

The screenshot shows the Jenkins 'New credentials' creation interface. The 'Kind' dropdown is set to 'AWS Credentials'. The 'Scope' dropdown is set to 'Global (items, nodes, items, all child items, etc.)'. The 'ID' field contains 'aws-key'. The 'Description' field contains 'aws-key'. The 'Access Key ID' field contains 'AKIAV2KHSRHWCSAJP'. The 'Secret Access Key' field is empty and has a red error message: 'Please specify the Secret Access Key'. The 'IAM Role Support' section is collapsed. An 'Advanced' link is visible at the bottom left. A blue 'Create' button is at the bottom right.

The Credentials will look like the below snippet.

The screenshot shows the Jenkins 'Global credentials (unrestricted)' list page. It displays a single credential entry:

ID	Name	Kind	Description
aws-key	AKIAV2KHSRHWCSAJP (aws-key)	AWS Credentials	aws-key

Now, We need to add GitHub credentials as well because currently, my repository is Private.

This thing, I am performing this because in Industry Projects your repository will be private.

So, add the username and personal access token of your GitHub account.

New credentials

Kind: Username with password

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

Username: AmanPathak-DevOps

Treat username as secret

Password:

ID: GITHUB

Description: GITHUB

Create

Both credentials will look like this.

Global credentials (unrestricted)

Add Credential

ID	Name	Kind	Description
aws-key	AWS4528C953F1FC55A6P (aws-key)	AWS Credentials	aws-key
GITHUB	AmanPathak-DevOps/***** (GITHUB)	Username with password	GITHUB

Create an eks cluster using the below commands.

```
eksctl create cluster --name Three-Tier-K8s-EKS-Cluster --region us-east-1 --node-type t3.micro
aws eks update-kubeconfig --region us-east-1 --name Three-Tier-K8s-EKS-Cluster
```

```

ubuntu@ip-10-0-1-72: ~ $ eksctl create cluster --name Three-Tier-K8s-EKS-Cluster --region us-east-1 --node-type t2.medium --nodes-min 2 --nodes-max 2
aws eks update-kubeconfig --region us-east-1 --name Three-Tier-K8s-EKS-Cluster
kubectl get nodes

2024-01-17 20:21:01 [i] eksctl version 0.207.0
2024-01-17 20:21:01 [i] using region us-east-1
2024-01-17 20:21:01 [i] setting availability zones to [us-east-1a us-east-1c]
2024-01-17 20:21:01 [i] subnets for us-east-1a - public:102.168.0.0/19 private:102.168.64.0/19
2024-01-17 20:21:01 [i] subnets for us-east-1c - public:102.168.32.0/19 private:102.168.96.0/19
2024-01-17 20:21:01 [i] nodegroup "ng-d24edbd1" will use [AmazonLinux2/1.39]
2024-01-17 20:21:01 [i] using Kubernetes version 1.27
2024-01-17 20:21:01 [i] creating EKS cluster "Three-Tier-K8s-EKS-Cluster" in "us-east-1" region with managed nodes
2024-01-17 20:21:01 [i] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2024-01-17 20:21:01 [i] if you encounter any issues, check CloudFormation console or try `eksctl utils describe-stacks` --region=us-east-1 --cluster=Three-Tier-K8s-EKS-Cluster
2024-01-17 20:21:01 [i] Kubernetes API endpoint access will use default of [publicAccess=true, privateAccess=false] for cluster "Three-Tier-K8s-EKS-Cluster" in "us-east-1"
2024-01-17 20:21:01 [i] CloudWatch loggers will not be enabled for cluster "Three-Tier-K8s-EKS-Cluster" in "us-east-1"
2024-01-17 20:21:01 [i] you can enable with `eksctl utils update-cluster-timestamp --enable-types=[SPECIFY-LOG-TYPES-HERE] (e.g. all) --region=us-east-1 --cluster=Three-Tier-K8s-EKS-Cluster`
2024-01-17 20:21:01 [i]
2 sequential tasks: [ create cluster control plane "Three-Tier-K8s-EKS-Cluster",
  2 sequential sub-tasks: [
    wait for control plane to become ready,
    create managed nodegroup "ng-d24edbd1",
  ]
]

2024-01-17 20:21:01 [i] building cluster stack "eksctl-Three-Tier-K8s-EKS-Cluster-cluster"
2024-01-17 20:21:01 [i] deploying stack "eksctl-Three-Tier-K8s-EKS-Cluster-cluster"
2024-01-17 20:21:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-cluster"
2024-01-17 20:22:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:22:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:23:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:24:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:25:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:26:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:27:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:28:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:29:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:30:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:31:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:32:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:33:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:34:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:35:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:36:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:37:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:38:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:39:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:40:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-controlplane"
2024-01-17 20:41:02 [i] building managed nodegroup stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:42:00 [i] deploying stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:42:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:43:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:44:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:45:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:46:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:47:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:48:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:49:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:50:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:51:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:52:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:53:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:54:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:55:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:56:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:57:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:58:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:59:01 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-nodegroup-ng-d24edbd1"
2024-01-17 20:59:57 [i] waiting for the control plane to become ready
2024-01-17 20:59:58 [x] saved kubeconfig as "/home/ubuntu/.kube/config"
2024-01-17 20:59:58 [i] no tasks
2024-01-17 20:59:58 [i] all EKS cluster resources for "Three-Tier-K8s-EKS-Cluster" have been created
2024-01-17 20:59:58 [i] nodegroup "ng-d24edbd1" has 2 nodes!
```

Once your cluster is created, you can validate whether your nodes are ready or not by the below command

```
kubectl get nodes
```

```
ubuntu@ip-10-0-1-72:~$ aws eks update-kubeconfig --region us-east-1 --name Three-Tier-KBS-EKS-Cluster
Kubectl get nodes
Added new context armazensieks/us-east-1:40762208209621cluster/Three-Tier-KBS-EKS-Cluster to /home/ubuntu/.kube/config
NAME           STATUS   ROLES      AGE     VERSION
ip-10-0-1-130.ec2.internal   Ready    <none>   5m26s   v1.28.5-eks-5e0fddde
ip-10-0-2-218.ec2.internal   Ready    <none>   5m18s   v1.28.5-eks-5e0fddde
ubuntu@ip-10-0-1-72:~$
```

**Step 6: 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/v2.0.0/deploy/managed-nodegroup.yaml
```

```
ubuntu@ip-10-0-1-72:~$ curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam_policy.json
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total   Spent    Left  Speed
100  8386  100  8386    0     0:02:73  0:00:00  0:00:53
ubuntu@ip-10-0-1-72:~$ ls
iam_policy.json
ubuntu@ip-10-0-1-72:~$
```

Create the IAM policy using the below command

```
aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-
```

```
ubuntu@ip-10-0-1-72:~$ aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json
{
  "Policy": {
    "PolicyName": "AWSLoadBalancerControllerIAMPolicy",
    "PolicyId": "APNAV5200NSRNGJ7MPP",
    "Arn": "arn:aws:iam::140762062962:policy/AWSLoadBalancerControllerIAMPolicy",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2024-01-17T19:57:47+00:00",
    "UpdateDate": "2024-01-17T19:57:47+00:00"
  }
}
ubuntu@ip-10-0-1-72:~$
```

Create OIDC Provider

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

```
ubuntu@ip-10-0-1-72:~$ eksctl utils associate-iam-oidc-provider --region=us-east-1 --cluster=Three-Tier-K8s-EKS-Cluster --approve
2024-01-17 10:58:13 [!] Will create IAM OpenID Connect provider for cluster "Three-Tier-K8s-EKS-Cluster" in "us-east-1"
2024-01-17 10:58:15 [+] created IAM OpenID Connect provider for cluster "Three-Tier-K8s-EKS-Cluster" in "us-east-1"
ubuntu@ip-10-0-1-72:~$
```

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

```
eksctl create iamserviceaccount --cluster=Three-Tier-K8s-EKS-Cluster --namespace
```

```
ubuntu@ip-10-0-1-72:~$ eksctl create iamserviceaccount --cluster=Three-Tier-K8s-EKS-Cluster --namespace=kube-system --name=aws-load-balancer-controller --role-name AmazonEKSLoadBalancerControllerRole --attach-policy-arm=arn:aws:iam::40762020962:policy/AWSLoadBalancerControllerIAMPolicy --approve --regionus-east-1
2024-01-17 20:00:01 [i] iamserviceaccount [kube-system/aws-load-balancer-controller] was included (based on the include/exclude rules)
2024-01-17 20:00:01 [i] serviceaccounts that exist in Kubernetes will be excluded. HSM --override-existing-serviceaccounts to override
2024-01-17 20:00:01 [i] 1 task:
  2 sequential sub-tasks:
    - create IAM role for serviceaccount "kube-system/aws-load-balancer-controller",
      create serviceaccount "kube-system/aws-load-balancer-controller",
    - [2024-01-17 20:00:01 [i]] building iamserviceaccount stack "eksctl-Three-Tier-K8s-EKS-Cluster-arn001-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-01-17 20:00:02 [i] deploying stack "eksctl-Three-Tier-K8s-EKS-Cluster-arn001-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-01-17 20:00:02 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-arn001-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-01-17 20:00:32 [i] waiting for CloudFormation stack "eksctl-Three-Tier-K8s-EKS-Cluster-arn001-iamserviceaccount-kube-system-aws-load-balancer-controller"
2024-01-17 20:00:32 [i] created serviceaccount "kube-system/aws-load-balancer-controller"
ubuntu@ip-10-0-1-72:~$
```

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
```

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-10-0-1-72:~$ 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           2m39s
ubuntu@ip-10-0-1-72:~$
```

If the pods are getting Error or CrashLoopBackOff, then use the below command

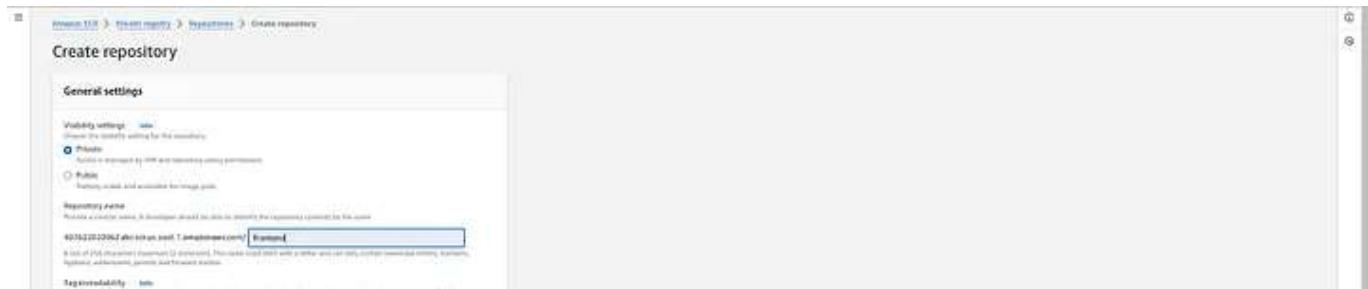
```
helm upgrade -i aws-load-balancer-controller eks/aws-load-balancer-controller \
--set clusterName=<cluster-name> \
--set serviceAccount.create=false \
--set serviceAccount.name=aws-load-balancer-controller \
--set region=us-west-1 --set vpcId=<vpc#> -n kube-system
```

## Step 7: We need to create Amazon ECR Private Repositories for both Tiers (Frontend & Backend)

Click on Create repository



Select the Private option to provide the repository and click on Save.



Do the same for the backend repository and click on Save

Open in app ↗



Now, we have set up our ECR Private Repository and

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

Copy the 1st command for login

Now, run the copied command on your Jenkins Server.

```
ubuntu@ip-10-0-1-72:~$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 407622020962.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-10-0-1-72:~$ 
```

## Step 8: Install & Configure ArgoCD

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

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

```
ubuntu@ip-10-0-1-72:~$ kubectl create namespace three-tier
namespace/three-tier created
ubuntu@ip-10-0-1-72:~$ 
```

As you know, Our two ECR repositories are private. So, when we try to push images to the ECR Repos it will give us the error **Imagepullerror**.

To get rid of this error, we will create a secret for our ECR Repo by the below command and then, we will add this secret to the deployment file.

**Note:** The Secrets are coming from the `.docker/config.json` file which is created while login the ECR in the earlier steps

```
kubectl create secret generic ecr-registry-secret \
--from-file=.dockerconfigjson=${HOME}/.docker/config.json \
--type=kubernetes.io/dockerconfigjson --namespace three-tier
kubectl get secrets -n three-tier
```

```
ubuntu@ip-10-0-1-72: ~$ kubectl create secret generic ecr-registry-secret \
--from-file=dockerconfigjson=$(HOME/.docker/config.json) \
--type=kubernetes.io/dockerconfigjson --namespace three-tier
secret/ecr-registry-secret created
ubuntu@ip-10-0-1-72: ~$ kubectl get secrets -n three-tier
NAME          TYPE           DATA   AGE
ecr-registry-secret  kubernetes.io/dockerconfigjson  1      15s
ubuntu@ip-10-0-1-72: ~$
```

Now, we will install argoCD.

To do that, create a separate namespace for it and apply the argocd configuration for installation.

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2
```

```
ubuntu@ip-10-0-1-72: ~$ kubectl create namespace argocd
namespace/argocd created
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 created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
```

All pods must be running, to validate run the below command

```
kubectl get pods -n argocd
```

```
ubuntu@ip-10-0-1-72: ~$ kubectl get pods -n argocd
NAME                      READY   STATUS    RESTARTS   AGE
argocd-application-controller-0   1/1     Running   0          33s
argocd-applicationset-controller-7659cbf897-thvng   1/1     Running   0          33s
argocd-dex-server-8694489dc5-9mcj   1/1     Running   0          33s
argocd-notification-controller-66f448897d-8fztz   1/1     Running   0          33s
argocd-pilot-7b57c4c298-rz7r7   1/1     Running   0          33s
argocd-secrets-server-6c56d98dc5-6fdpq   1/1     Running   0          33s
argocd-server-769f4f64ab-rx7pc   1/1     Running   0          33s
ubuntu@ip-10-0-1-72: ~$
```

Now, expose the argoCD server as LoadBalancer using the below command

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

```
ubuntu@ip-10-0-1-72: ~ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
ubuntu@ip-10-0-1-72: ~
ubuntu@ip-10-0-1-72: ~
```

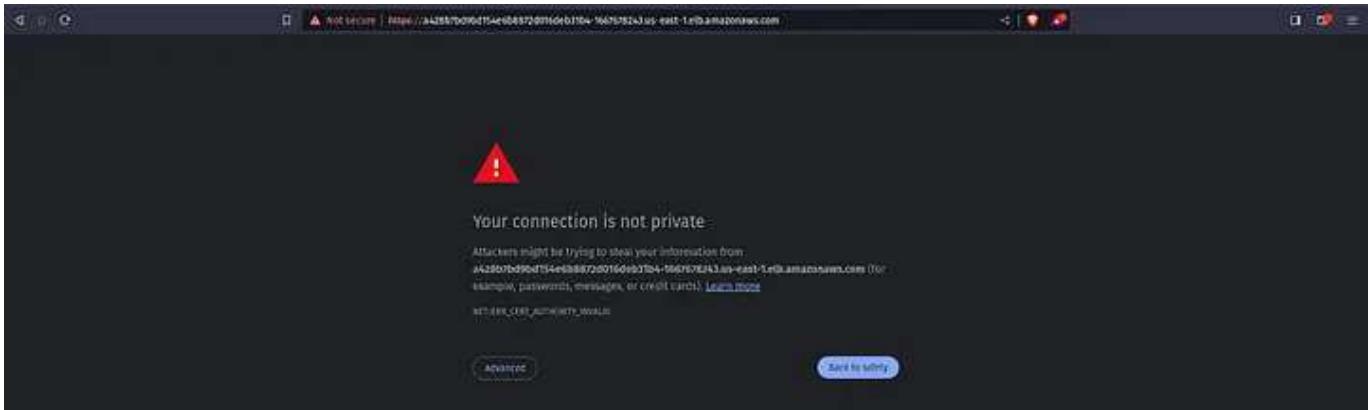
You can validate whether the Load Balancer is created or not by going to the AWS Console



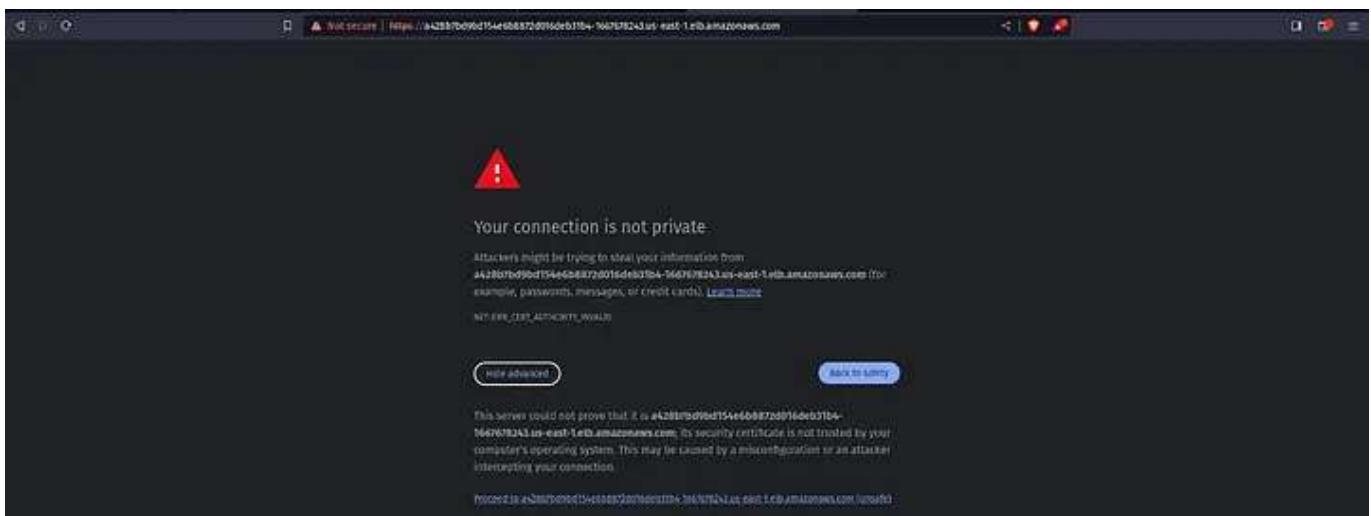
To access the argoCD, copy the LoadBalancer DNS and hit on your favorite browser.

You will get a warning like the below snippet.

Click on Advanced.



Click on the below link which is appearing under Hide advanced



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

To do that, we have a pre-requisite which is jq. Install it by the command below.

```
sudo apt install jq -y
```

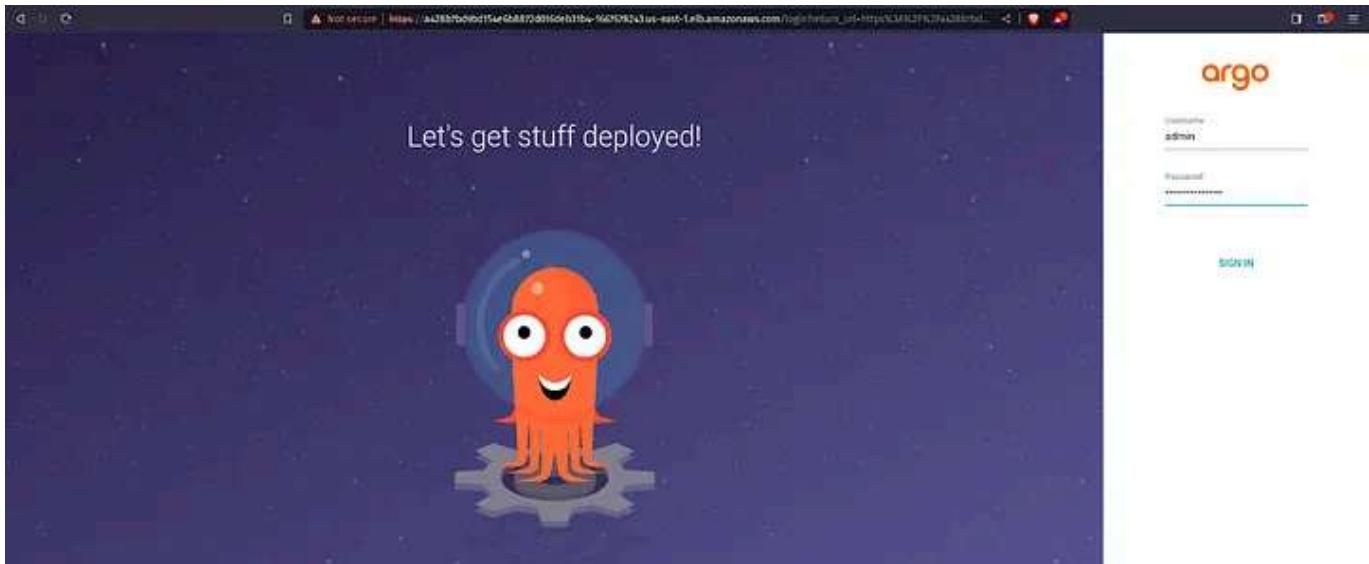
```
ubuntu@ip-10-0-1-72: ~ sudo apt install jq -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libjq1 libonig5
The following NEW packages will be installed:
  jq libjq1 libonig5
0 upgraded, 3 newly installed, 0 to remove and 49 not upgraded.
Need to get 357 kB of archives.
After this operation, 1087 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libonig5 amd64 0.9.7.1-2build1 [172 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libjq1 amd64 1.6-2.1ubuntu3 [133 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 jq amd64 1.6-2.1ubuntu3 [52.5 kB]
Fetched 357 kB in 9s (12.7 kB/s)
Selecting previously unselected package libonig5:amd64.
(Reading database ... 21684 files and directories currently installed.)
Preparing to unpack .../libonig5_0.9.7.1-2build1_amd64.deb ...
Unpacking libonig5:amd64 (0.9.7.1-2build1) ...
```

```
export ARGOCD_SERVER='kubectl get svc argocd-server -n argocd -o json | jq - raw
export ARGO_PWD='kubectl -n argocd get secret argocd-initial-admin-secret -o jso
```

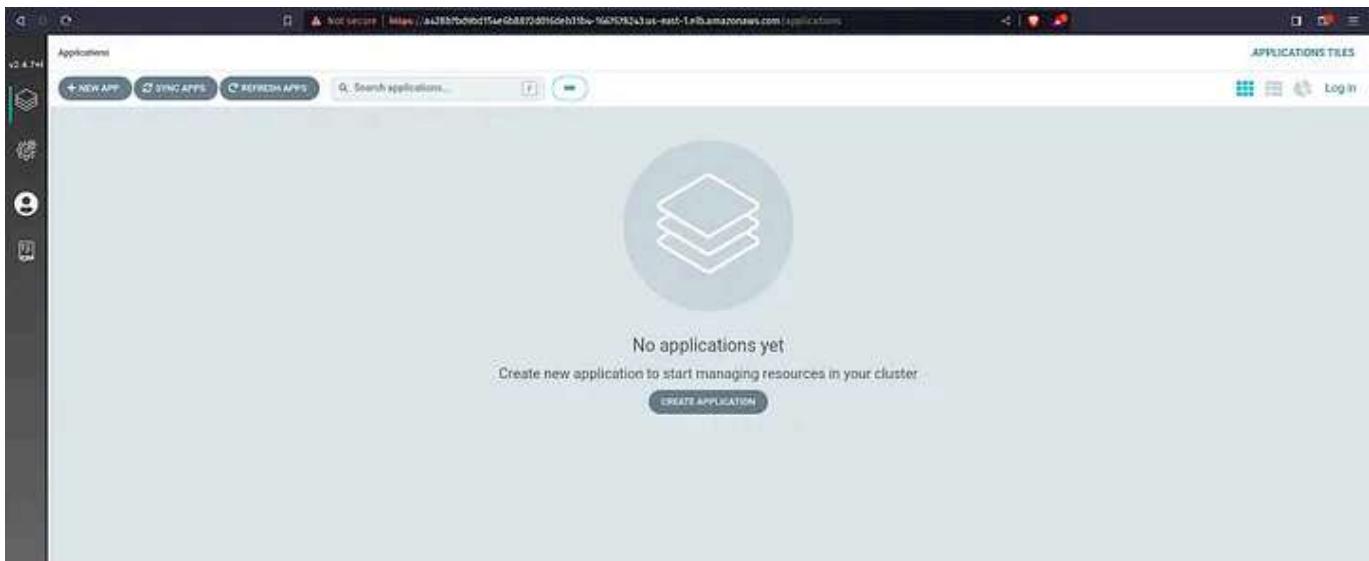
```
echo $ARGO_PWD
```

```
ubuntu@ip-10-0-1-72:~$ export ARGOCD_SERVER=$(kubectl get svc argo-cd-server -n argo-cd -o json | jq --raw-output '.status.loadBalancer.ingress[0].hostname')
ubuntu@ip-10-0-1-72:~$ export ARGO_PWD=$(kubectl -n argo-cd get secret argo-cd-initial-admin-secret -o jsonpath='{.data.password}' | base64 -d)
echo $ARGO_PWD
JN1lnNTg9z4cJ7...
ubuntu@ip-10-0-1-72:~$
```

Enter the username and password in argoCD and click on **SIGN IN**.



Here is our ArgoCD Dashboard.



## Step 9: Now, we have to configure Sonarqube for our DevSecOps Pipeline

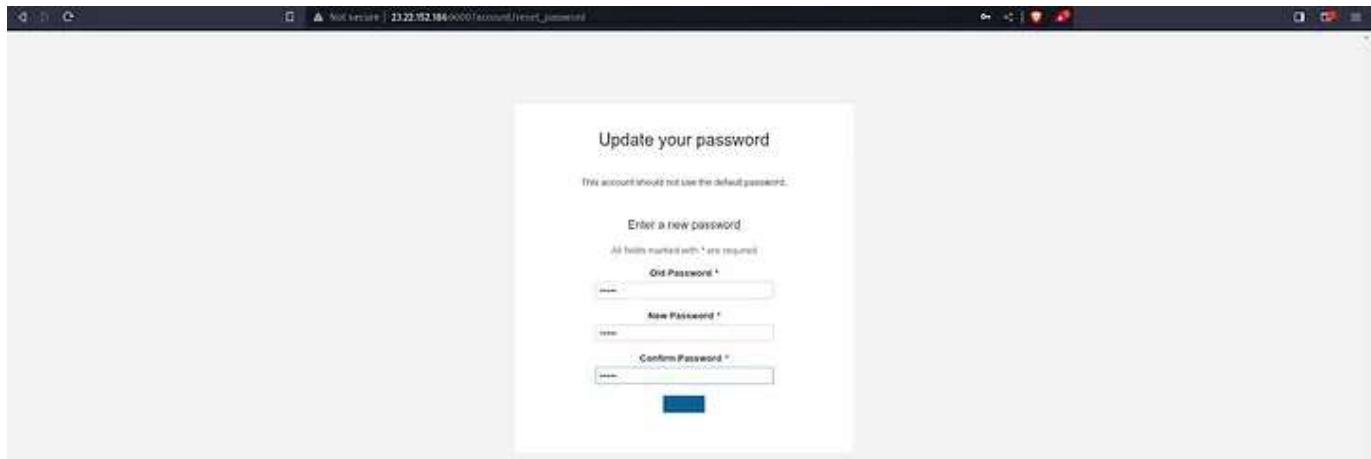
To do that, copy your Jenkins Server public IP and paste it on your favorite browser with a 9000 port

The username and password will be **admin**

Click on Log In.



Update the password



Click on Administration then Security, and select Users

The screenshot shows the SonarQube administration interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. The main menu has sections for Configuration, Security, Projects, System, and Marketplace. Under the Security section, there are tabs for General, Roles, Global Permissions, and Permissions Templates. A dropdown menu is open over the 'Edit profile' link, showing the same four options.

Click on Update tokens

The screenshot shows the SonarQube administration interface under the Security tab. It lists users with their names and icons. A user named 'Administrator' is shown with a green 'A' icon. There is a 'Create User' button in the top right corner.

Click on Generate

The screenshot shows the 'Tokens of Administrator' page. It features a 'Generate Tokens' form with fields for 'Name' (set to 'sonarqube') and 'Expires in' (set to '30 days'). Below the form is a table titled 'Tokens' with columns for Name, Type, Project, Last use, Created, and Expiration. The table currently shows one entry: 'sonarqube' with a 'Generated' type, created 4 hours ago, and no last use.

Copy the token keep it somewhere safe and click on Done.

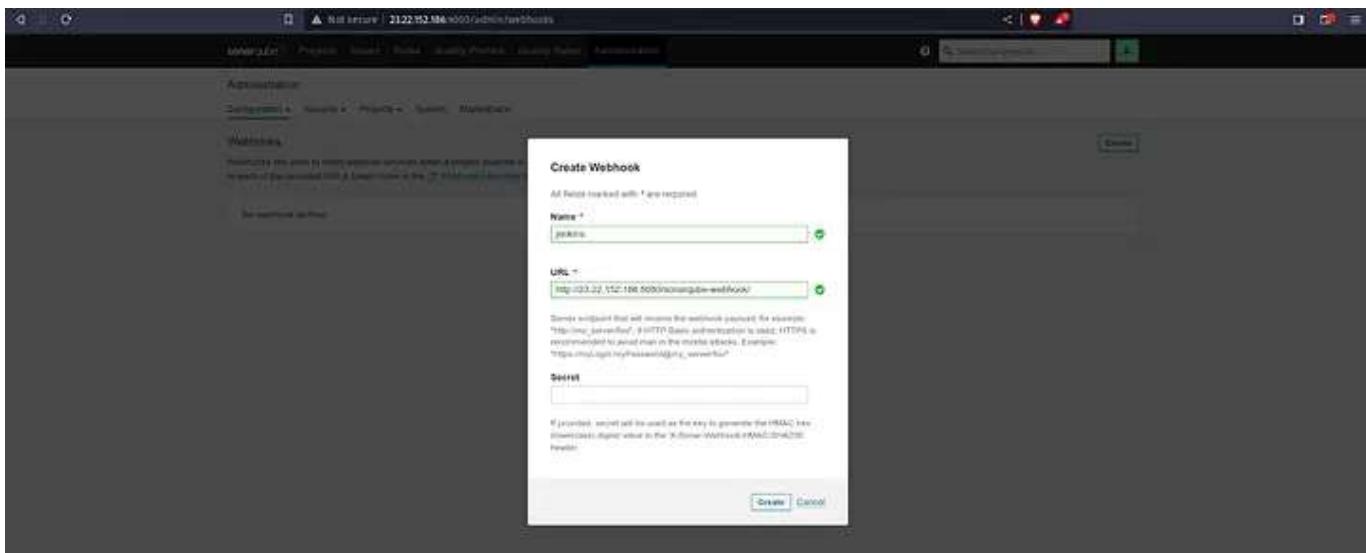
Now, we have to configure webhooks for quality checks.

Click on Administration then, Configuration and select Webhooks

Click on Create

Provide the name of your project and in the URL, provide the Jenkins server public IP with port 8080 add sonarqube-webhook in the suffix, and click on Create.

`http://<jenkins-server-public-ip>:8080/sonarqube-webhook/`



Here, you can see the webhook.

Name	URL	Has secret?	Last delivery	Actions
jenkins	http://23.22.152.186:8080/sonarqube-webhook/	No	Never	

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

Click on Manually.

How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like Repository Import and Pull Request decoration)? Create your project from your favorite DevOps platform. First, you need to set up a DevOps platform configuration.

From Azure DevOps  
Set up global configuration

From Bitbucket Server  
Set up global configuration

From Bitbucket Cloud  
Set up global configuration

From GitHub  
Set up global configuration

From GitLab  
Set up global configuration

Are you just testing or have an advanced use-case? Create a project manually.

< >  
Manually

Provide the display name to your Project and click on Setup

Click on Locally.

Select the Use existing token and click on Continue.

## Select Other and Linux as OS.

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

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

The screenshot shows the SonarQube interface for the 'three-tier-frontend' project. The 'Analyze your project' section is active. Under 'What system best describes your build?', 'Other (for JS, TS, Go, Python, PHP, ...)' is selected. Under 'What is your OS?', 'Linux' is selected. A code snippet for running the scanner is shown:

```
sonar-scanner -Dsonar.projectKey=three-tier-frontend -Dsonar.host.url=http://123.45.67.89:9001 -Dsonar.login=demo_dockerizedFrontendWorkflowAutomation
```

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

Click on Create Project.

The screenshot shows the SonarQube 'Projects' page. The 'three-tier-frontend' project is listed under 'My Favorites'. The 'Create Project' button is located at the top right of the page.

Provide the name of your project name and click on Set up.

All fields marked with \* are required.

**Project display name \***

Use 3-25 characters. Special characters might invalidate this value and prevent creation.

**Project key \***

The project key is a unique identifier for your project. It may contain up to 600 characters. Allowed characters are alphanumeric, '-' (hyphen), '-' (underscore), '-' (percent) and '-' (colon), with at least one non-digit.

**Main branch name \***

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

**Set Up**

Click on Locally.

How do you want to analyze your repository?

Do you want to integrate with your favorite CI? Choose one of the following buttons.

- With Jenkins
- With GitHub Actions
- With Bitbucket Pipelines
- With GitLab CI
- With Azure Pipelines
- Other CI

Are you just testing or have an advanced use-case? Analyze your project locally.

**Locally**

Select the Use existing token and click on Continue.

Analyze your project

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

**1 Provide a token**

Generate a project token

Use existing token

Existing token value:

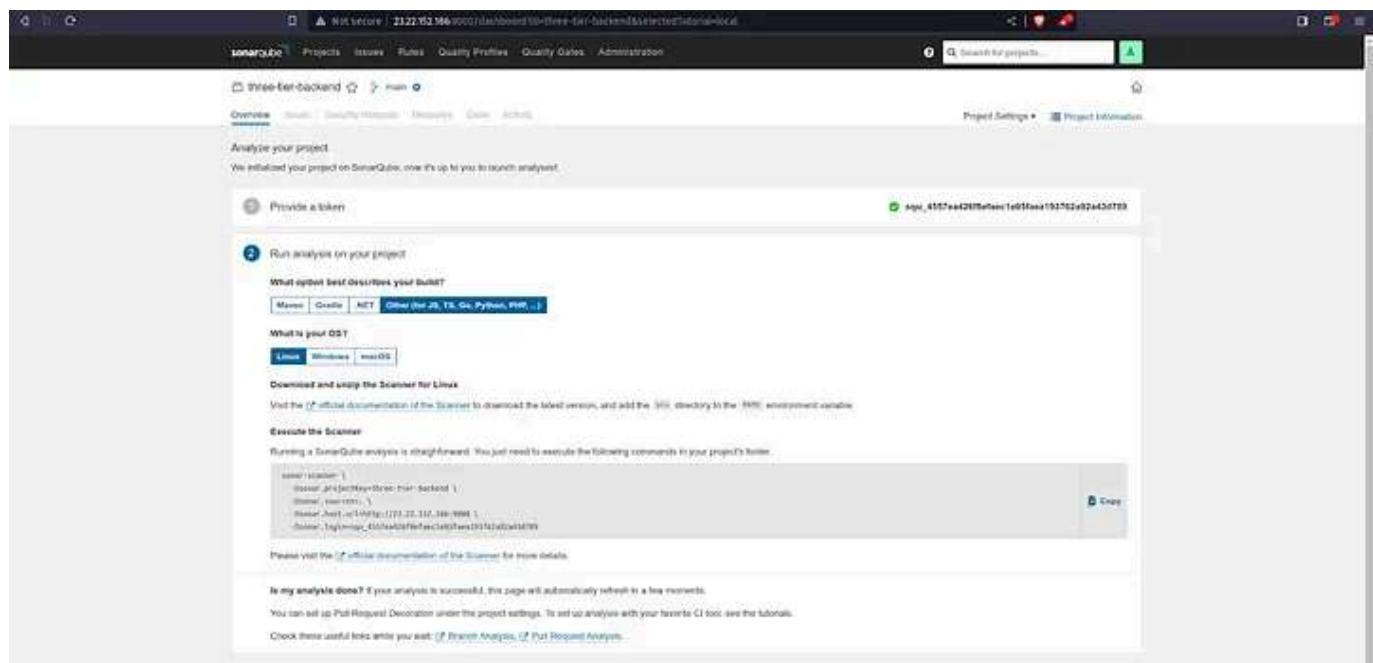
This token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point in time in your profile details.

**Continue**

## Select Other and Linux as OS.

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

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



Now, we have to store the sonar 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**

New credentials

Kind: Secret text

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

Secret:

ID: sonar-token

Description: sonar-token

Create

Now, we have to store the GitHub Personal access token to push the deployment file which will be modified in the pipeline itself for the ECR image.

## Add GitHub credentials

Select the kind as **Secret text** and paste your GitHub Personal access token(not password) in Secret and keep other things as it is.

Click on **Create**

**Note:** If you haven't generated your token then, you have it generated first then paste it into the Jenkins

New credentials

Kind: Secret text

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

Secret:

ID: github

Description: github

**Create**

Now, according to our Pipeline, we need to add an Account ID in the Jenkins credentials because of the ECR repo URI.

Select the kind as **Secret text** paste your AWS Account ID in Secret and keep other things as it is.

Click on **Create**

New credentials

Kind: Secret text

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

Secret:

ID: ACCOUNT\_ID

Description: ACCOUNT\_ID

**Create**

Now, we need to provide our ECR image name for frontend which is **frontend** only.

Select the kind as **Secret text**, paste your frontend repo name in Secret and keep other things as it is.

Click on **Create**

The screenshot shows the Jenkins 'New credentials' page under 'Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Secret text'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'Secret' field contains 'image'. The 'ID' field is 'ECR\_REPO1'. The 'Description' field is 'ECR\_REPO1'. A blue 'Create' button is at the bottom.

Now, we need to provide our ECR image name for the backend which is **backend only**.

Select the kind as **Secret text**, paste your backend repo name in Secret, and keep other things as it is.

Click on **Create**

The screenshot shows the Jenkins 'New credentials' page under 'Manage Jenkins > Credentials > System > Global credentials (unrestricted)'. The 'Kind' dropdown is set to 'Secret text'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc.)'. The 'Secret' field contains 'image'. The 'ID' field is 'ECR\_REPO2'. The 'Description' field is 'ECR\_REPO2'. A blue 'Create' button is at the bottom.

## Final Snippet of all Credentials that we needed to implement this project.

The screenshot shows the Jenkins Global credentials (unrestricted) page. At the top, there is a breadcrumb navigation: Dashboard > Manage Jenkins > Credentials > system > Global credentials (unrestricted). A blue button labeled '+ Add Credential' is located in the top right corner. Below the header, a table lists seven global credentials:

ID	Name	Type	Description
aws-key	AAKU52EKA5KRYWCSM6P (aws-key)	AWS Credentials	aws-key
GITHUB	AminPathak-DevOps/***** (GITHUB)	Username with password	GITHUB
sonar-token	sonar token	Secret text	sonar-token
github	github	Secret text	github
ACCOUNT_ID	ACCOUNT_ID	Secret text	ACCOUNT_ID
ECR_REPO1	ECR_REPO1	Secret text	ECR_REPO1
ECR_REPO2	ECR_REPO2	Secret text	ECR_REPO2

## Step 10: 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

- Docker
- Docker Commons
- Docker Pipeline
- Docker API
- docker-build-step
- Eclipse Temurin installer
- NodeJS
- OWASP Dependency-Check
- SonarQube Scanner

The screenshot shows the Jenkins 'Manage Jenkins' section, specifically the 'Tools' configuration page. On the left, there's a sidebar with options like 'Dashboard', 'Manage Jenkins', and 'Tools'. The main area is titled 'Plugins' and shows a list of installed plugins. The list includes:

- Docker 1.13**: Version 1.13.8, last updated 12/25/20. Description: This plugin integrates Jenkins with Docker.
- Docker Client API 1.24**: Version 1.24.0, last updated 12/25/20. Description: Library plugin that can be used by other plugins - docker.
- Docker Pipeline 1.2.0**: Version 1.2.0, last updated 12/25/20. Description: Provides the common shared functionality for various Docker related plugins.
- Docker API 1.13-04.2019-05.2019**: Version 1.13-04.2019-05.2019, last updated 12/25/20. Description: Library plugin that can be used by other plugins - docker.
- Docker API 1.13-04.2019-05.2019**: Version 1.13-04.2019-05.2019, last updated 12/25/20. Description: This plugin provides Docker Java API for other plugins.
- Docker Build Step 1.10**: Version 1.10.0, last updated 12/25/20. Description: This plugin allows to add various docker commands to your job as build steps.
- Dependency Check 1.4.2**: Version 1.4.2, last updated 12/25/20. Description: This plugin can independently execute a dependency-check analysis and visualize results. Dependency Check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities.
- Holder 1.21**: Version 1.21, last updated 12/25/20. Description: Jenkins plugin executes holder(s) script as a build step.
- AdoptOpenJDK Installer 1.5**: Version 1.5, last updated 12/25/20. Description: Provides an installer for the jdk tool that downloads the jdk from https://adoptopenjdk.net
- SonarQube Scanner 3.10**: Version 3.10, last updated 12/25/20. Description: This plugin allows an easy integration of SonarQube, the open source platform for continuous inspection of code quality.

Now, we have to configure the installed plugins.

Go to Dashboard -> Manage Jenkins -> Tools

We are configuring jdk

Search for **jdk** and provide the configuration like the below snippet.

The screenshot shows the Jenkins 'Manage Jenkins' section, specifically the 'Tools' configuration page, with the 'JDK installations' tab selected. It shows a form to 'Add jdk' with the following fields:

- Name**: jdk
- Install automatically**:
- Install from adoptium.net**:
- Version**: jdk-17.0.5+12
- Add installer**: A dropdown menu.

Now, we will configure the sonarqube-scanner

Search for the sonarqube scanner and provide the configuration like the below snippet.

SonarQube Scanner installations

Add SonarQube Scanner

**SonarQube Scanner**

Name: sonar-scanner

Install automatically

Install from Maven Central

Version: SonarQube Scanner 5.6.1.3006

Add installer

Add SonarQube Scanner

Now, we will configure nodejs

Search for node and provide the configuration like the below snippet.

**NodeJS**

Name: nodejs

Install automatically

Install from nodejs.org

Version: NodeJS 14.0.0

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'

Global npm packages refresh hours

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

72

Add installer

Add NodeJS

Now, we will configure the OWASP Dependency check

Search for **Dependency-Check** and provide the configuration like the below snippet.

**Dependency-Check installations**

Add Dependency-Check

**Dependency-Check**

Name: DP-Check

Install automatically

Install from github.com

Version: dependency check 9.0.9

Add installer

Add Dependency-Check

Now, we will configure the docker

Search for **docker** and provide the configuration like the below snippet.

Add Docker

**Docker**

Name: docker

Install automatically

Download from docker.com

Docker version: latest

Add installer

Add Docker

**Save**   **Apply**

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.

The screenshot shows the Jenkins configuration interface for managing SonarQube servers. The top navigation bar includes 'Dashboard', 'Manage Jenkins', 'System', and 'SonarQube servers'. A note at the top states: 'If checked, Jenkins administrators will be able to inject a SonarQube server configuration as environment variables in the build.' Below this is a checkbox labeled 'Environment variables'. The main section is titled 'SonarQube installations' with a link 'List of SonarQube installations'. It contains fields for 'Name' (set to 'sonar-server'), 'Server URL' (set to 'http://73.23.102.54:9000'), and 'Server authentication token' (set to 'sonar-token'). There is also an 'advanced' dropdown and a 'Save' button at the bottom.

Now, we are ready to create our Jenkins Pipeline to deploy our Backend Code.

Go to Jenkins Dashboard

Click on New Item

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

**Start building your software project**

Create a job     +     Set up a distributed build

Provide the name of your **Pipeline** and click on **OK**.

Enter an Item name

Three-Tier Backend Application (Required field)

**Freestyle project**  
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

This is the Jenkins file to deploy the Backend Code on EKS.

Copy and paste it into the **Jenkins**

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

Click **Apply & Save**.

The screenshot shows the ArgoCD interface for configuring a pipeline. The pipeline is named 'Three-Tier-Backend-Application'. The 'Pipeline Syntax' tab is selected. The pipeline script is written in Groovy and defines several stages: 'Initial', 'Deploy Docker Image', 'Deploy Python API', 'Deploy Java App', 'Deploy Frontend', and 'Deploy Backend'. Each stage includes steps like cloning the repository, building the image, pushing it to ECR, and deploying it to a Kubernetes namespace. A 'try sample Pipeline...' button is visible at the top right.

```

script {
    stage('Initial') {
        step('Deploy Docker Image') {
            action {
                step('Build Docker Image')
                step('Push Docker Image')
                step('Deploy Docker Image')
            }
        }
    }
    stage('Deploy Docker Image') {
        action {
            step('Build Docker Image')
            step('Push Docker Image')
            step('Deploy Docker Image')
        }
    }
    stage('Deploy Python API') {
        action {
            step('Build Python API')
            step('Push Python API')
            step('Deploy Python API')
        }
    }
    stage('Deploy Java App') {
        action {
            step('Build Java App')
            step('Push Java App')
            step('Deploy Java App')
        }
    }
    stage('Deploy Frontend') {
        action {
            step('Build Frontend')
            step('Push Frontend')
            step('Deploy Frontend')
        }
    }
    stage('Deploy Backend') {
        action {
            step('Build Backend')
            step('Push Backend')
            step('Deploy Backend')
        }
    }
}

```

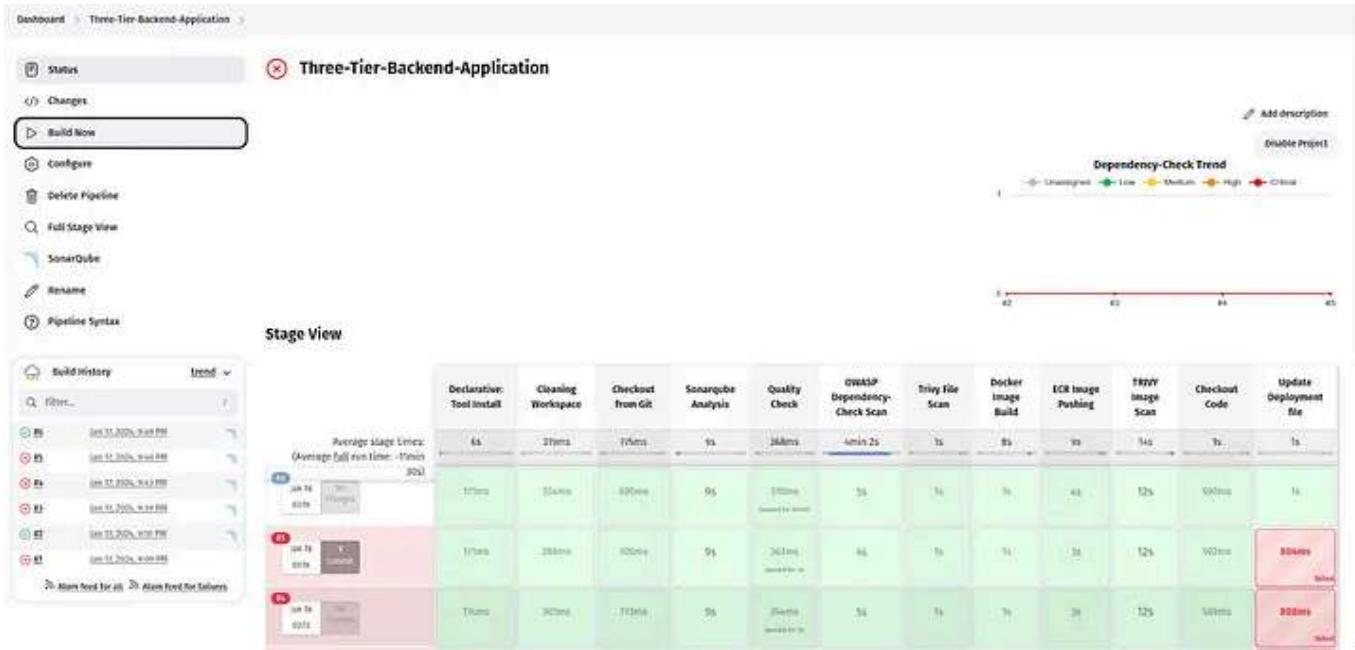
Use Groovy Sandbox

**Save** **Apply**

Now, click on the build.

Our pipeline was successful after a few common mistakes.

**Note:** Do the changes in the Pipeline according to your project.

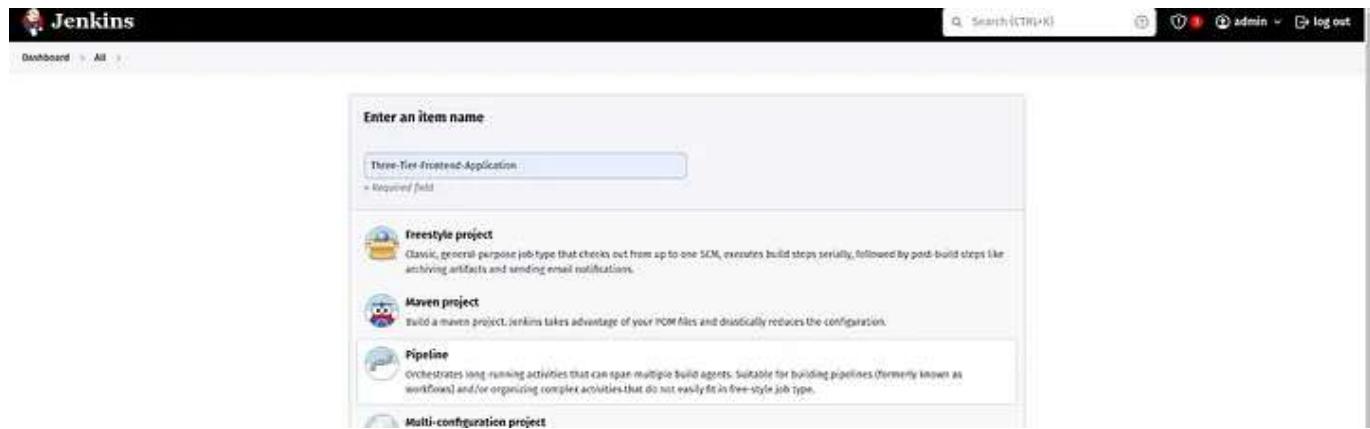


Now, we are ready to create our Jenkins Pipeline to deploy our Frontend Code.

Go to Jenkins Dashboard

Click on New Item

Provide the name of your **Pipeline** and click on **OK**.



This is the Jenkins file to deploy the Frontend Code on EKS.

Copy and paste it into the Jenkins

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

Click Apply & Save.

The screenshot shows the ArgoCD interface for configuring a pipeline. The left sidebar has tabs for General, Advanced Project Options, and Pipeline, with Pipeline selected. The main area is titled "Definition" and contains a "Pipeline script" input field. The script content is a YAML file defining a pipeline with stages: "Deploy Frontend", "Deploy Middle", and "Deploy Backend". Each stage includes steps like "Deploy", "Wait for Deployment", and "Check Health". A "try sample Pipeline..." button is visible in the top right of the script area. At the bottom are "Save" and "Apply" buttons.

```

apiVersion: argoproj.io/v1alpha1
kind: Pipeline
metadata:
  name: three-tier-front-end-app
spec:
  pipelineSpec:
    stages:
      - name: "Deploy Frontend"
        steps:
          - name: "Deploy Frontend"
            action:
              type: "ArgoCD"
              target:
                app:
                  name: "three-tier-front-end-app"
                  project: "three-tier-front-end-project"
                destination:
                  name: "three-tier-front-end-app"
                  project: "three-tier-front-end-project"
            triggers:
              - type: "Deployment"
                selector:
                  matchLabels:
                    app: "three-tier-front-end-app"
                    tier: "front-end"
            finally:
              - name: "Check Health"
                action:
                  type: "HTTP"
                  url: "http://${{state.outputs[0].url}}/healthz"
                  timeout: "10s"
                  interval: "1s"
                  maxRetries: "3"
            artifacts:
              - name: "Deployed"
                type: "Image"
                path: "three-tier-front-end-app:v1"
      - name: "Deploy Middle"
        steps:
          - name: "Deploy Middle"
            action:
              type: "ArgoCD"
              target:
                app:
                  name: "three-tier-middle-service"
                  project: "three-tier-front-end-project"
                destination:
                  name: "three-tier-middle-service"
                  project: "three-tier-front-end-project"
            triggers:
              - type: "Deployment"
                selector:
                  matchLabels:
                    app: "three-tier-middle-service"
                    tier: "middle"
            finally:
              - name: "Check Health"
                action:
                  type: "HTTP"
                  url: "http://${{state.outputs[0].url}}/healthz"
                  timeout: "10s"
                  interval: "1s"
                  maxRetries: "3"
            artifacts:
              - name: "Deployed"
                type: "Image"
                path: "three-tier-middle-service:v1"
      - name: "Deploy Backend"
        steps:
          - name: "Deploy Backend"
            action:
              type: "ArgoCD"
              target:
                app:
                  name: "three-tier-backend-service"
                  project: "three-tier-front-end-project"
                destination:
                  name: "three-tier-backend-service"
                  project: "three-tier-front-end-project"
            triggers:
              - type: "Deployment"
                selector:
                  matchLabels:
                    app: "three-tier-backend-service"
                    tier: "back-end"
            finally:
              - name: "Check Health"
                action:
                  type: "HTTP"
                  url: "http://${{state.outputs[0].url}}/healthz"
                  timeout: "10s"
                  interval: "1s"
                  maxRetries: "3"
            artifacts:
              - name: "Deployed"
                type: "Image"
                path: "three-tier-backend-service:v1"
    triggers:
      - type: "Manual"
        name: "Deploy"
        annotations:
          - key: "msg"
            value: "Deploy three-tier application"
        actions:
          - type: "ArgoCD"
            target:
              app:
                name: "three-tier-front-end-app"
                project: "three-tier-front-end-project"
              destination:
                name: "three-tier-front-end-app"
                project: "three-tier-front-end-project"
            triggers:
              - type: "Deployment"
                selector:
                  matchLabels:
                    app: "three-tier-front-end-app"
                    tier: "front-end"
            finally:
              - name: "Check Health"
                action:
                  type: "HTTP"
                  url: "http://${{state.outputs[0].url}}/healthz"
                  timeout: "10s"
                  interval: "1s"
                  maxRetries: "3"
            artifacts:
              - name: "Deployed"
                type: "Image"
                path: "three-tier-front-end-app:v1"
    checkpoints:
      - name: "Deployed"
        type: "Image"
        path: "three-tier-front-end-app:v1"
        annotations:
          - key: "msg"
            value: "Deployed three-tier application"
  triggers:
    - type: "Manual"
      name: "Deploy"
      annotations:
        - key: "msg"
          value: "Deploy three-tier application"
      actions:
        - type: "ArgoCD"
          target:
            app:
              name: "three-tier-front-end-app"
              project: "three-tier-front-end-project"
            destination:
              name: "three-tier-front-end-app"
              project: "three-tier-front-end-project"
          triggers:
            - type: "Deployment"
              selector:
                matchLabels:
                  app: "three-tier-front-end-app"
                  tier: "front-end"
            finally:
              - name: "Check Health"
                action:
                  type: "HTTP"
                  url: "http://${{state.outputs[0].url}}/healthz"
                  timeout: "10s"
                  interval: "1s"
                  maxRetries: "3"
            artifacts:
              - name: "Deployed"
                type: "Image"
                path: "three-tier-front-end-app:v1"
    checkpoints:
      - name: "Deployed"
        type: "Image"
        path: "three-tier-front-end-app:v1"
        annotations:
          - key: "msg"
            value: "Deployed three-tier application"
  
```

Now, click on the build.

Our pipeline was successful after a few common mistakes.

Note: Do the changes in the Pipeline according to your project.

The screenshot shows the ArgoCD interface with the "Stage View" tab selected. On the left, there's a sidebar with various project management options like Status, Changes, Build Now, Configure, Delete Pipeline, Full Stage View, SonarQube, Rename, Pipeline Syntax, Build History, and a Filter dropdown. The main area displays a table of build history. The first two rows are in red, indicating failure, while the third row is green, indicating success. The table columns include: Declarative Tool Install, Cleaning Workspace, Checkout from Git, SonarQube Analysis, Quality Check, OWASP Dependency Check Scan, Drift File Scan, Docker Image Build, ECR Image Pushing, TRIVY Image Scan, Checkout Code, and Update Deployment. Below the table, the "SonarQube Quality Gate" section shows a green "Passed" status with a "Success" badge. At the bottom, there's a "Latest Dependency Check" icon and a "Permalinks" section.

	Average stage times: (Average full run time: 3min 10s)	Declarative Tool Install	Cleaning Workspace	Checkout from Git	SonarQube Analysis	Quality Check	OWASP Dependency Check Scan	Drift File Scan	Docker Image Build	ECR Image Pushing	TRIVY Image Scan	Checkout Code	Update Deployment
Jan 16	3min 10s	102ms	210ms	762ms	9%	364ms	8min 28s	2s	4.7%	17s	79s	64ms	1s
Jan 16	3min 10s	102ms	210ms	762ms	9%	370ms	16min 51s	2s	1min 33s	29s	198s	163ms	1s

## Setup 10: 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

Add the prometheus repo by using the below command

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

```
ubuntu@ip-10-0-1-72:~$ helm repo add stable https://charts.helm.sh/stable
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
"stable" has been added to your repositories
"prometheus-community" has been added to your repositories
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "eks" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "stable" chart repository
Update Complete. *Happy Helm-ing!
ubuntu@ip-10-0-1-72:~$
```

Install the Prometheus

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

```
ubuntu@ip-10-0-1-72:~$ helm install stable prometheus-community/kube-prometheus-stack
NAME: stable
LAST DEPLOYED: Wed Jan 17 21:15:47 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace default get pods -l "release=stable"
Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create & configure Alertmanager and Prometheus instances using the Operator.
ubuntu@ip-10-0-1-72:~$ ubuntu@ip-10-0-1-72:~$
```

Now, check the service by the below command

```
kubectl get svc
```

```
ubuntu@ip-10-0-1-72:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
alertmanager-operated   ClusterIP  None         <none>        9093/TCP,9094/TCP,9094/UDP   26s
kubernetes       ClusterIP  10.100.0.1   <none>        443/TCP          49m
prometheus-operated   ClusterIP  None         <none>        9090/TCP          26s
stable-grafana     ClusterIP  10.100.224.15  <none>        80/TCP           30s
stable-kube-prometheus-sta-alertmanager   ClusterIP  10.100.61.97  <none>        9093/TCP,8080/TCP   30s
stable-kube-prometheus-sta-operator       ClusterIP  10.100.44.02  <none>        443/TCP          30s
stable-kube-prometheus-sta-prometheus    ClusterIP  10.100.60.214  <none>        9090/TCP,8080/TCP   30s
stable-kube-state-metrics      ClusterIP  10.100.123.7   <none>        8080/TCP          30s
stable-prometheus-node-exporter      ClusterIP  10.100.133.242  <none>        9180/TCP          30s
ubuntu@ip-10-0-1-72:~$
```

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 stable-kube-prometheus-sta-prometheus
```

Modification in the 48th line from ClusterType to LoadBalancer

```
36  port: 8080
37  protocol: TCP
38  targetPort: 8080
39  - appProtocol: http
40  name: reloader-web
41  port: 8080
42  protocol: TCP
43  targetPort: reloader-web
44  selector:
45  app.kubernetes.io/name: prometheus
46  operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus
47  sessionAffinity: None
48  type: LoadBalanced
49  status:
50  loadBalancer: {}
```

## Edit the stable-grafana service

```
kubectl edit svc stable-grafana
```

```
ubuntu@ip-10-0-1-72:~$ kubectl edit svc stable-grafana
```

Modification in the 39th line from ClusterType to LoadBalancer

```
32      port: 80
33      protocol: TCP
34      targetPort: 3000
35    selector:
36      app.kubernetes.io/instance: stable
37      app.kubernetes.io/name: grafana
38    sessionAffinity: None
39    type: LoadBalanced
40  status:
41    loadBalancer: {}
```

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

```
kubectl get svc
```

```
ubuntu@ip-10-0-1-72:~$ kubectl get svc
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP          PORT(S)          AGE
alertmanager-operated   ClusterIP  None         <none>
kubernetes       ClusterIP  10.100.0.1   <none>
prometheus-operated   ClusterIP  None         <none>
stable-grafana     LoadBalancer  10.100.224.15  a7f11e243e5feef6a9f2b2fb8dd4ed7-257659882.us-east-1.elb.amazonaws.com  80:30470/TCP
stable-kube-prometheus-sta-alertmanager   ClusterIP  10.100.61.97  <none>
stable-kube-prometheus-sta-operator        ClusterIP  10.100.41.82  <none>
stable-kube-prometheus-sta-prometheus     LoadBalancer  10.100.88.214  ac73e515d0bef54c26ac1366e02145801-475288234.us-east-1.elb.amazonaws.com  80:30338/TCP,8080:32614/TCP
stable-kube-state-metrics      ClusterIP  10.100.123.7   <none>
stable-prometheus-node-exporter       ClusterIP  10.100.133.242 <none>
                                         PORT(S)          AGE
                                         9093/TCP,9094/TCP,9094/UDP  2m34s
                                         443/TCP          51m
                                         9090/TCP          2m34s
                                         80:30470/TCP          2m38s
                                         9093/TCP,8080/TCP          2m38s
                                         443/TCP          2m38s
                                         8080/TCP          2m38s
                                         9100/TCP          2m38s
```

You can also validate from your console.

The screenshot shows the AWS CloudFormation Load Balancers page. It displays three load balancers with the following details:

Name	ARN	State	VPC ID	Availability Zones	Type	Date created
alT1x243e5fca046a0f29c2f10d0246d7	arn:aws:cloudformation:us-east-1:123456789012:resource/LoadBalancers/alT1x243e5fca046a0f29c2f10d0246d7	Active	vpc-084c211abcf7e638	2 Availability Zones	classic	January 18, 2024, 02:47 (UTC-05:00)
alT1x243e5fca046a0f29c2f10d0246d7	arn:aws:cloudformation:us-east-1:123456789012:resource/LoadBalancers/alT1x243e5fca046a0f29c2f10d0246d7	Active	vpc-084c211abcf7e638	2 Availability Zones	classic	January 18, 2024, 02:47 (UTC-05:00)
alT1x243e5fca046a0f29c2f10d0246d7	arn:aws:cloudformation:us-east-1:123456789012:resource/LoadBalancers/alT1x243e5fca046a0f29c2f10d0246d7	Active	vpc-084c211abcf7e638	2 Availability Zones	classic	January 18, 2024, 02:47 (UTC-05:00)

Now, access your Prometheus Dashboard

Paste the <Prometheus-LB-DNS>:9090 in your favorite browser and you will see like this

The screenshot shows the Prometheus dashboard. The top navigation bar includes links for Metrics, Alerts, Graph, Status, and Help. Below the navigation, there are several configuration checkboxes: Use local time (unchecked), Enable query history (unchecked), Enable autocomplete (checked), Enable highlighting (checked), and Enable inter (checked). The main interface has two tabs: Table and Graph. The Graph tab is selected, showing a single panel 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.

Click on Status and select Target.

You will see a lot of Targets

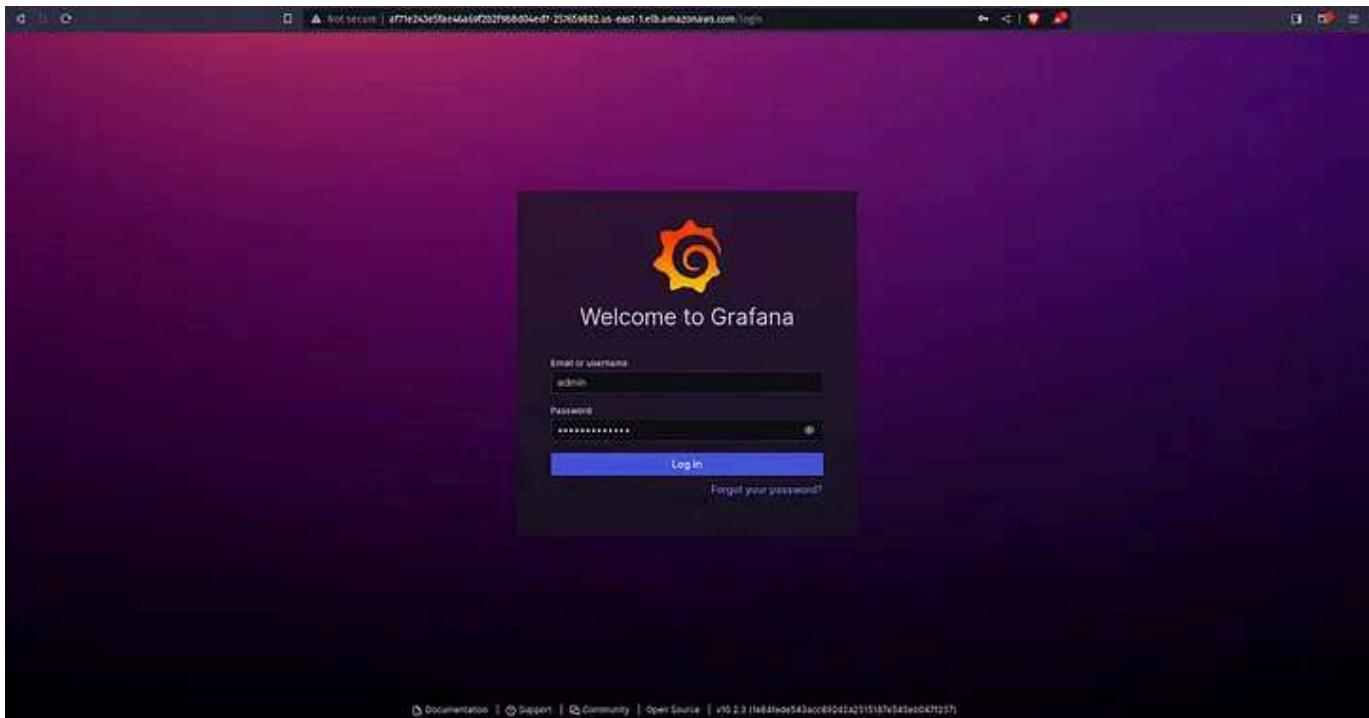
The screenshot shows the Prometheus Targets page with four service monitors listed:

- serviceMonitor/default/stable-kube-prometheus-etc-alertmanager/0**: 1/1 up. Endpoint: https://12.168.21.102:4003/metrics. Labels include: alertmanager="stable-kube-prometheus-etc-alertmanager", instance="https://12.168.21.102:4003/metrics". Last Scrape: 26.606s ago. Scrape Duration: 8.030ms.
- serviceMonitor/default/stable-kube-prometheus-etc-prometheus/1**: 1/1 up. Endpoint: https://12.168.21.102:4000/metrics. Labels include: prometheus="stable-kube-prometheus-etc-prometheus", instance="https://12.168.21.102:4000/metrics". Last Scrape: 27.71s ago. Scrape Duration: 2.476ms.
- serviceMonitor/default/stable-kube-prometheus-etc-apiserver/0**: 2/2 up. Endpoint: https://12.168.21.102:4001/metrics. Labels include: kube-apiserver="stable-kube-prometheus-etc-apiserver", instance="https://12.168.21.102:4001/metrics". Last Scrape: 15.168s ago. Scrape Duration: 156.678ms.
- serviceMonitor/default/stable-kube-prometheus-etc-coredns/0**: 2/2 up. Endpoint: https://12.168.21.102:4004/metrics. Labels include: coredns="stable-kube-prometheus-etc-coredns", instance="https://12.168.21.102:4004/metrics". Last Scrape: 9.49 ago. Scrape Duration: 208.325ms.

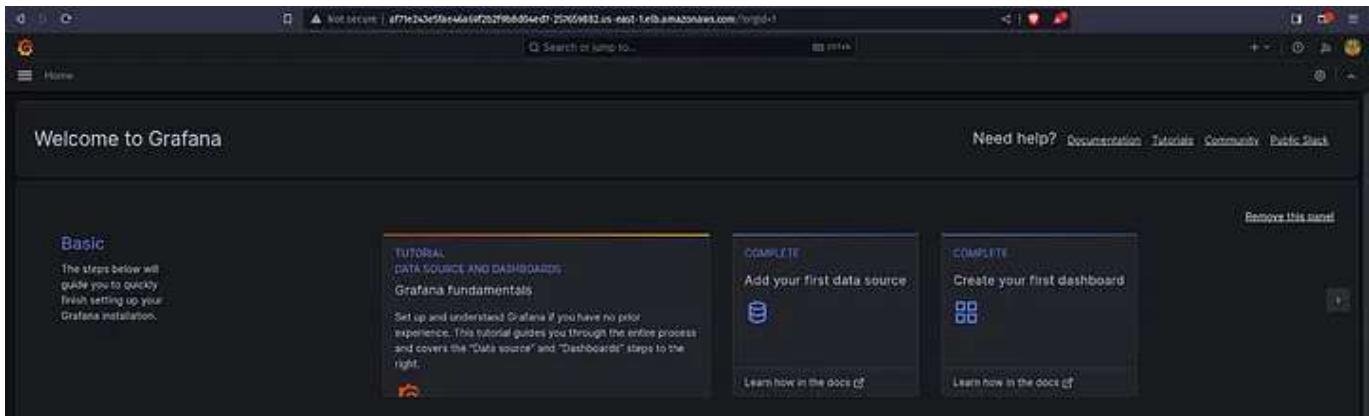
Now, access your **Grafana Dashboard**

Copy the ALB DNS of Grafana and paste it into your favorite browser.

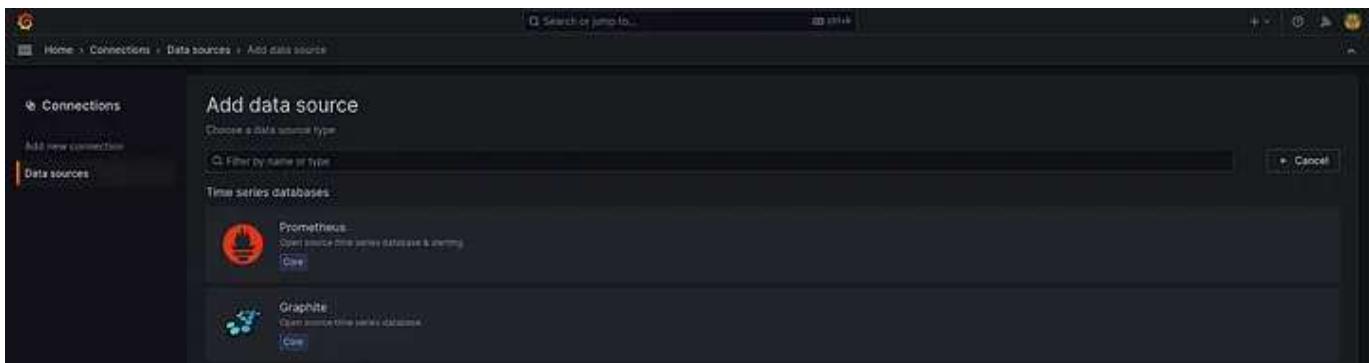
The username will be **admin** and the password will be **prom-operator** for your Grafana LogIn.



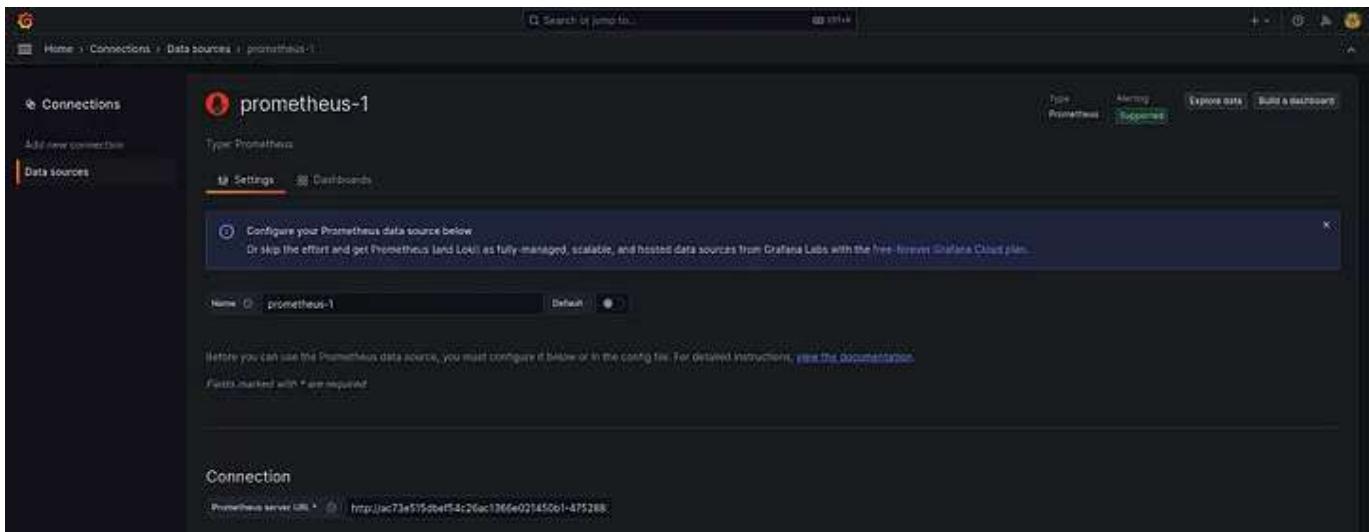
Now, click on Data Source



Select Prometheus



In the Connection, paste your <Prometheus-LB-DNS>:9090.



If the URL is correct, then you will see a green notification/

Click on Save & test.

The screenshot shows the Grafana interface for managing data sources. The left sidebar has a 'Data sources' tab selected. The main area is titled 'Query editor' and contains several configuration sections:

- Connections**: Default editor is set to 'Builder'. There is an option to 'Add new connection'.
- Data sources**: A single data source named 'prometheus-1' is listed. It is configured with:
  - Prometheus type: 'Choose'
  - Cache level: 'Low'
  - Incremental querying (beta): Enabled
  - Disable recording rules (beta): Enabled
- Performance**: Includes 'Custom query parameters' (set to 'Example: max\_source\_resolution=1m&format') and 'HTTP method' (set to 'POST').
- Exemplars**: A '+ Add' button is present.

A green success message at the bottom states: 'Successfully queried the Prometheus API. Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.' Buttons for 'Delete' and 'Save & test' are at the bottom.

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

Click on Dashboard.

The screenshot shows the Grafana navigation bar. The 'Dashboard' tab is highlighted in blue, indicating it is the active section. Other tabs include 'Home', 'Starred', and 'Logs'.

Once you click on Dashboard. You will see a lot of Kubernetes components monitoring.

The screenshot shows the Grafana interface for managing dashboards. On the left, there's a sidebar with links for Dashboards, Playlists, Snapshots, Library panels, and Public dashboards. The main area is titled 'Dashboards' and contains a search bar and a filter section. A large list of dashboards is displayed, each with a preview icon, name, and tags. The tags are color-coded: purple for 'alertmanager', green for 'dns', red for 'etcd', and blue for 'grafana'. Some dashboards have multiple tags. The list includes entries like 'Alertmanager / Overview', 'CoreDNS', 'etcd', 'Grafana Overview', 'Kubernetes / API server', and various Kubernetes Compute Resources and Networking dashboards.

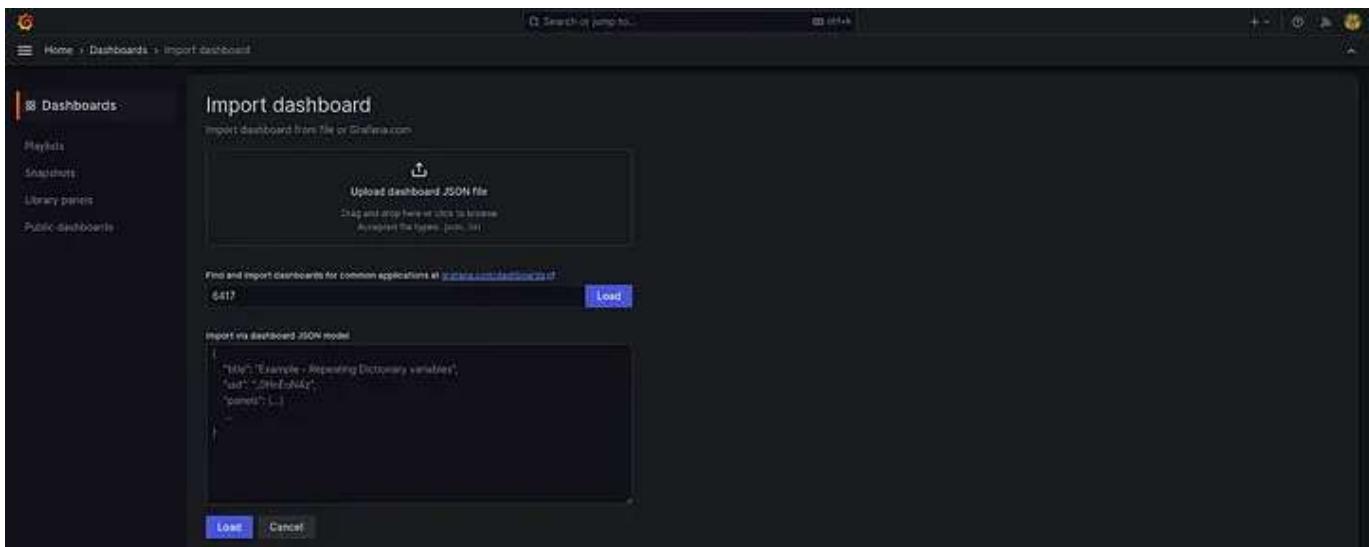
Let's try to import a type of Kubernetes Dashboard.

Click on New and select Import

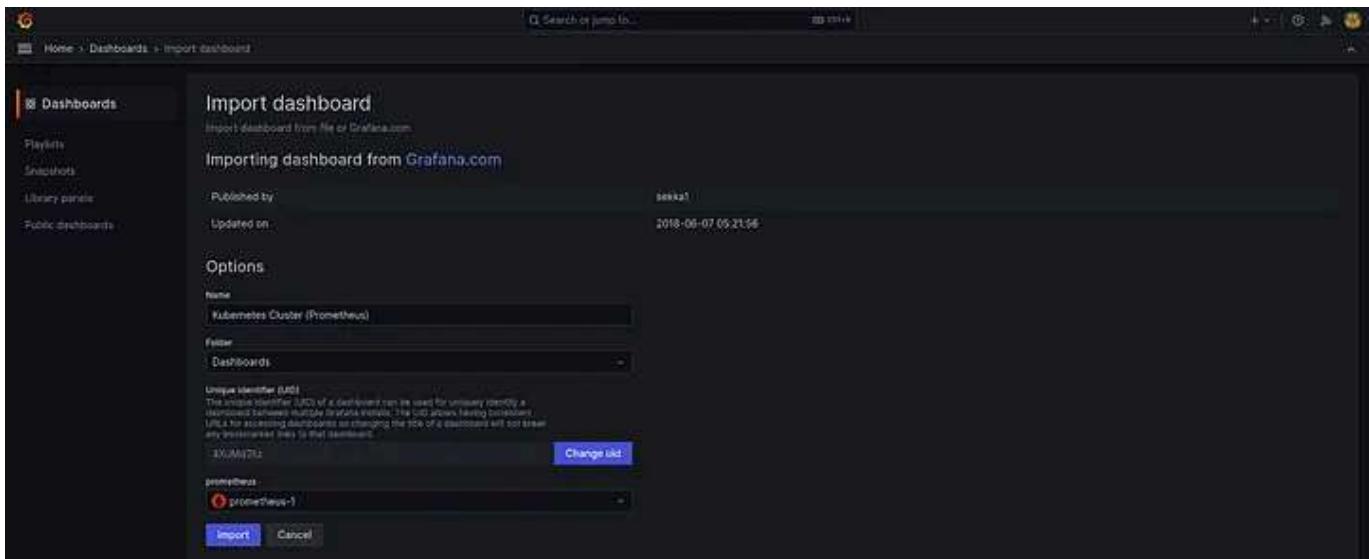
This screenshot is similar to the previous one but focuses on the top right corner of the interface. It shows the 'New' button followed by a dropdown menu with options: 'New dashboard', 'New folder', and 'Import'. The 'Import' option is highlighted with a blue border, indicating it is the selected action.

Provide 6417 ID and click on Load

**Note:** 6417 is a unique ID from Grafana which is used to Monitor and visualize Kubernetes Data



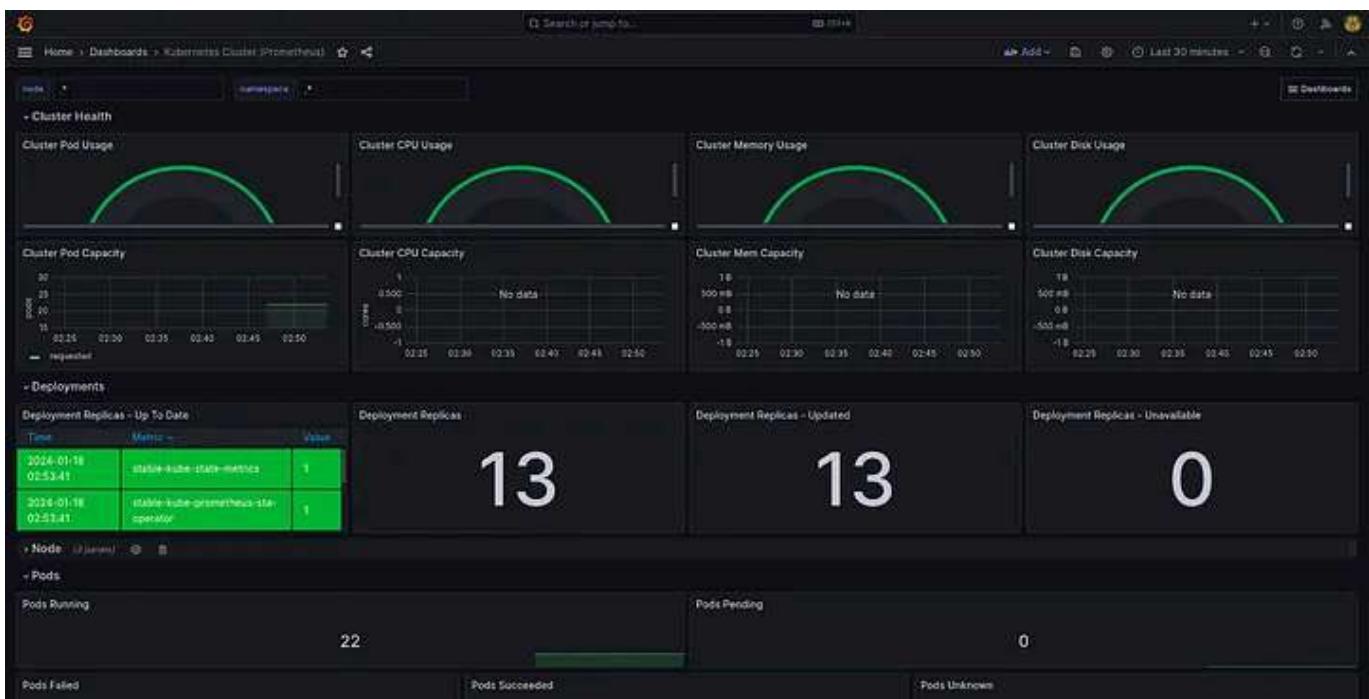
Select the data source that you have created earlier and click on **Import**.



Here, you go.

You can view your Kubernetes Cluster Data.

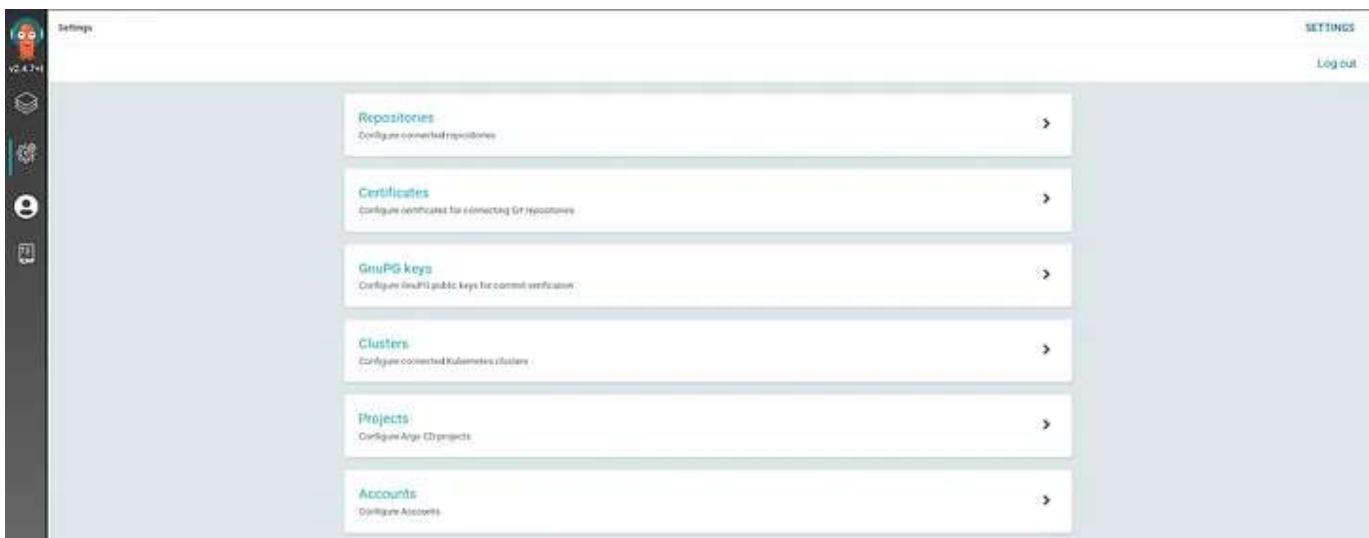
Feel free to explore the other details of the Kubernetes Cluster.



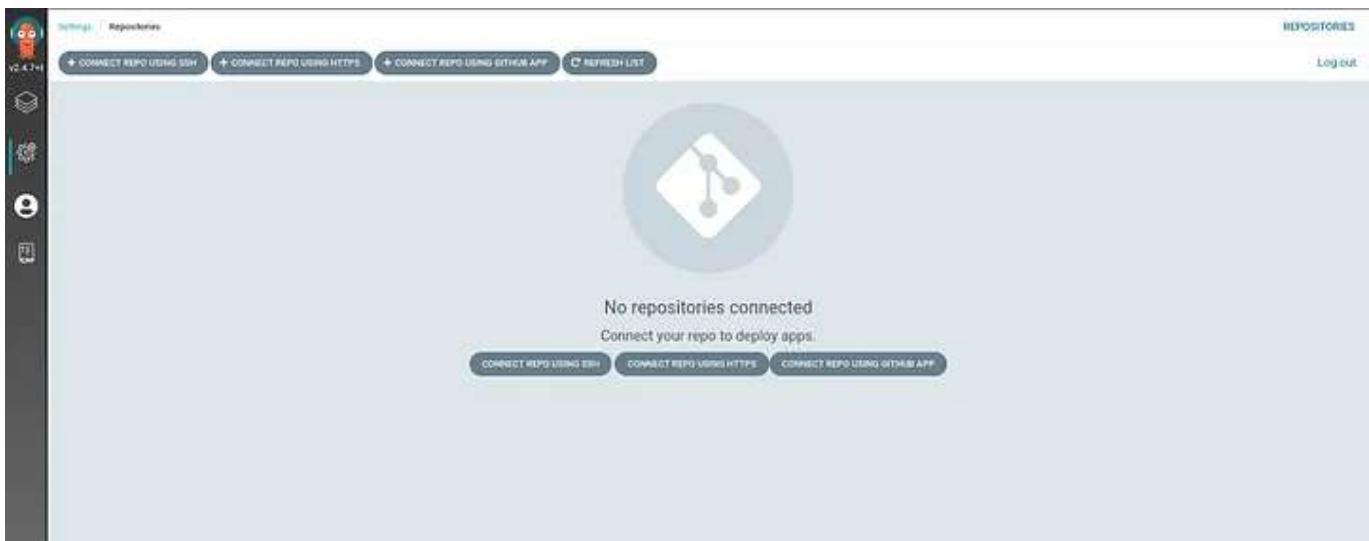
## Step 11: We will deploy our Three-Tier Application using ArgoCD.

As our repository is private. So, we need to configure the Private Repository in ArgoCD.

Click on Settings and select Repositories

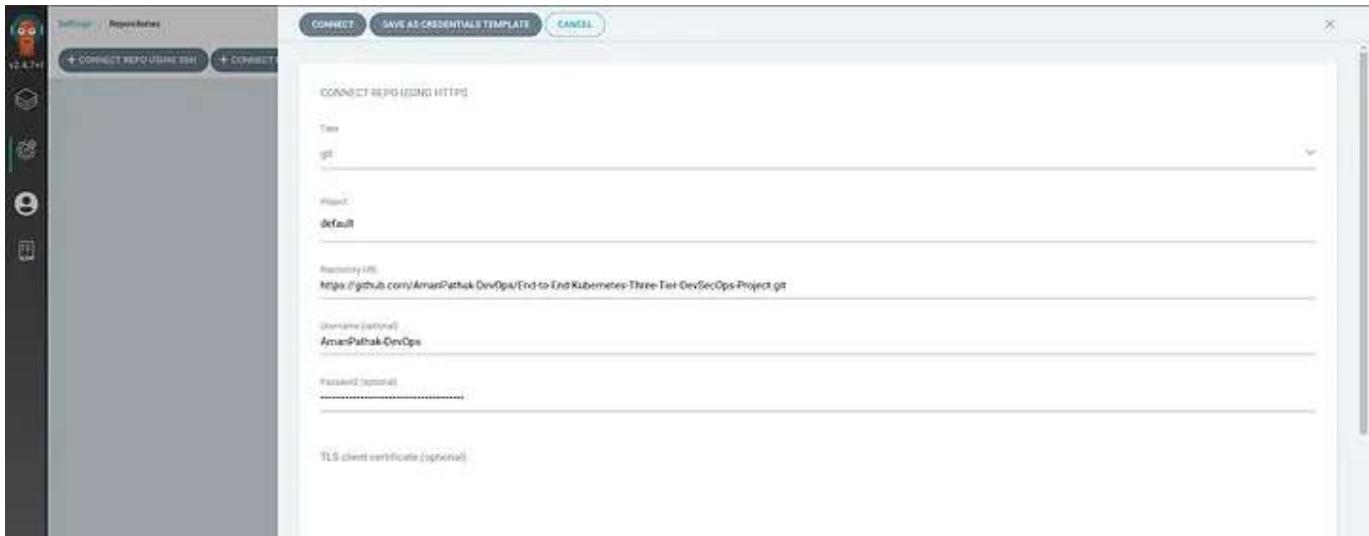


Click on CONNECT REPO USING HTTPS



Now, provide the repository name where your Manifests files are present.

Provide the username and GitHub Personal Access token and click on CONNECT.

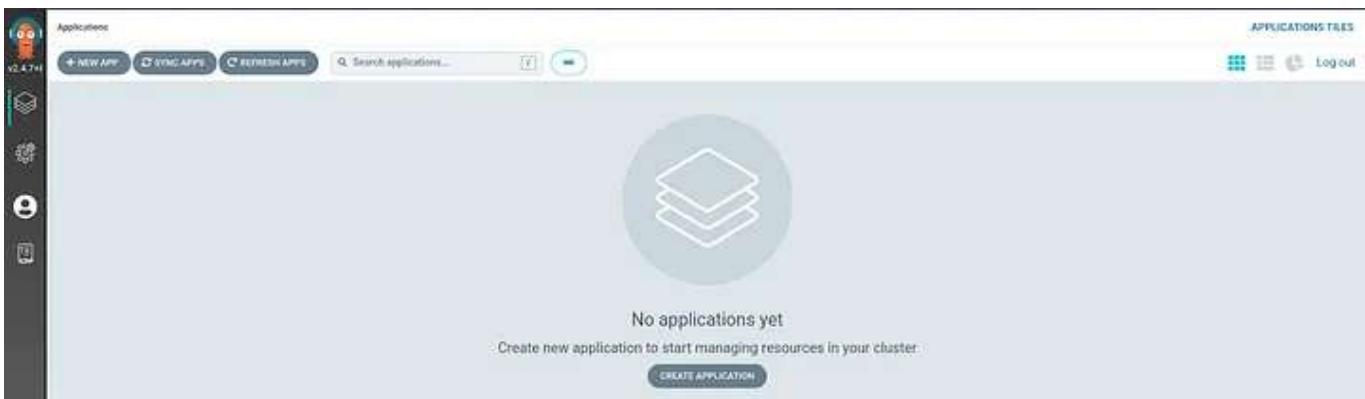


If your Connection Status is Successful it means repository connected successfully.



Now, we will create our first application which will be a database.

Click on **CREATE APPLICATION**.



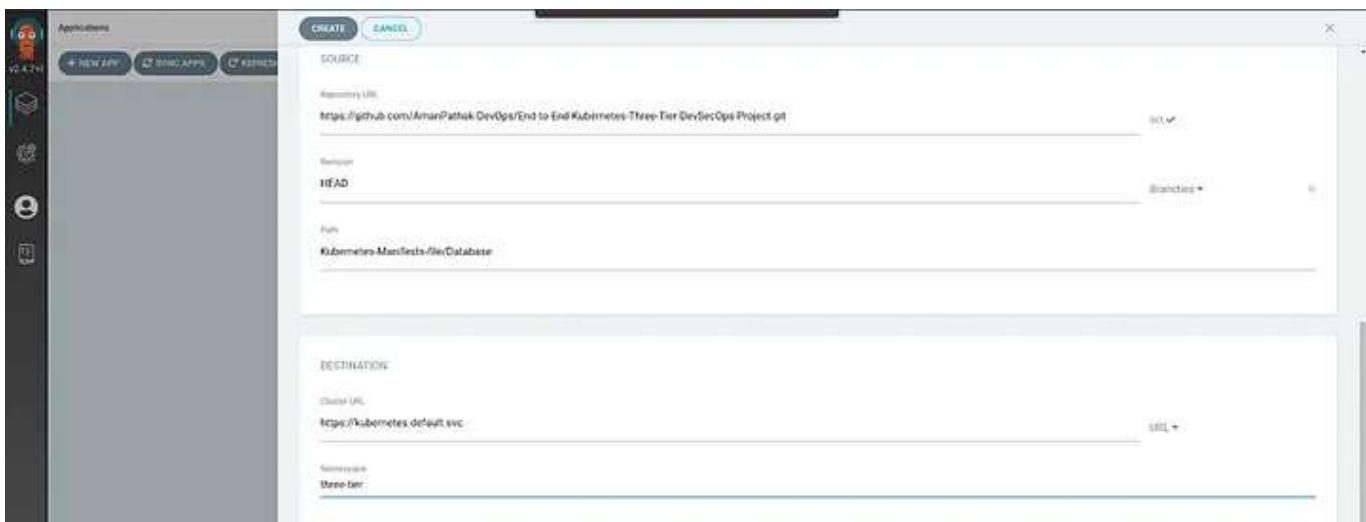
Provide the details as it is provided in the below snippet and scroll down.



Select the same repository that you configured in the earlier step.

In the **Path**, provide the location where your Manifest files are presented and provide other things as shown in the below screenshot.

Click on **CREATE**.



While your database Application is starting to deploy, We will create an application for the backend.

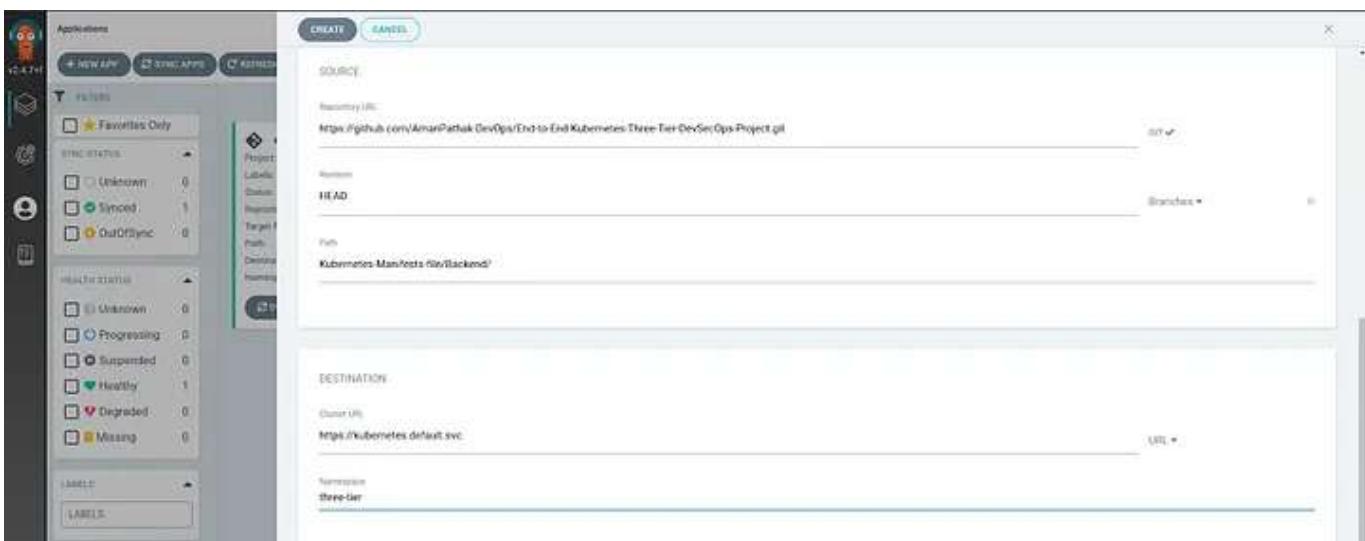
Provide the details as it is provided in the below snippet and scroll down.



Select the same repository that you configured in the earlier step.

In the **Path**, provide the location where your Manifest files are presented and provide other things as shown in the below screenshot.

Click on **CREATE**.



While your backend Application is starting to deploy, We will create an application for the frontend.

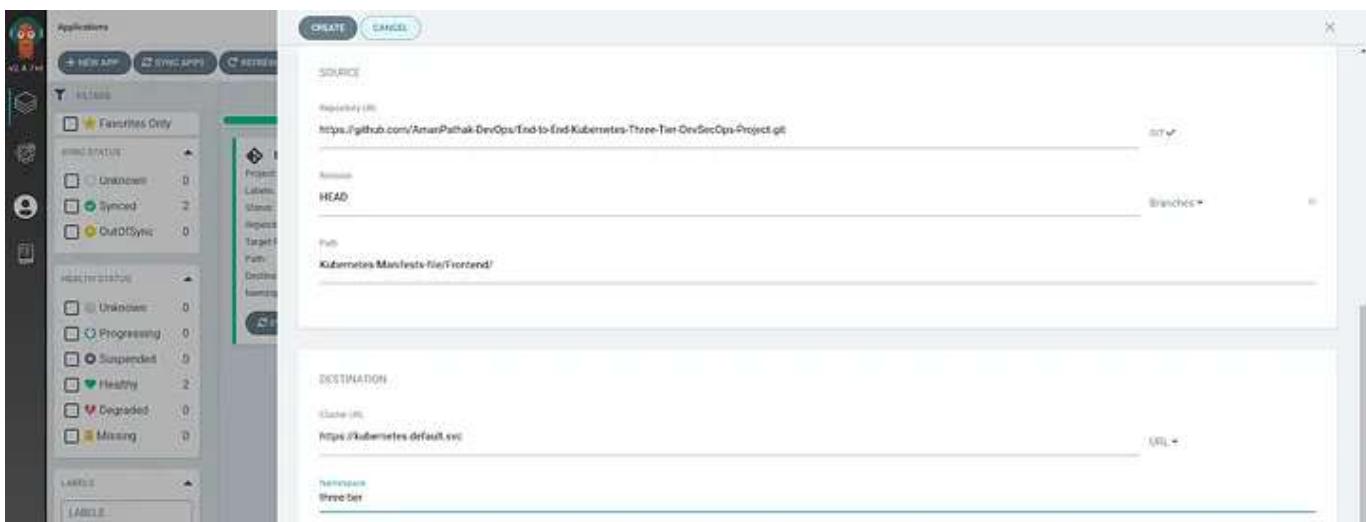
Provide the details as it is provided in the below snippet and scroll down.



Select the same repository that you configured in the earlier step.

In the **Path**, provide the location where your Manifest files are presented and provide other things as shown in the below screenshot.

Click on **CREATE**.



While your frontend Application is starting to deploy, We will create an application for the ingress.

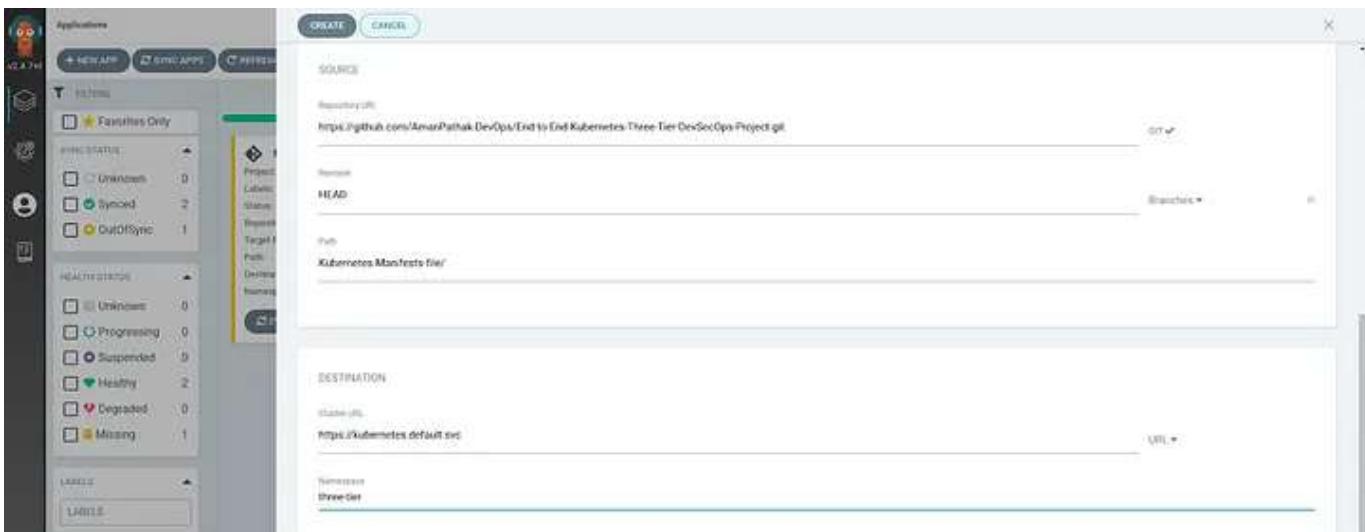
Provide the details as it is provided in the below snippet and scroll down.



Select the same repository that you configured in the earlier step.

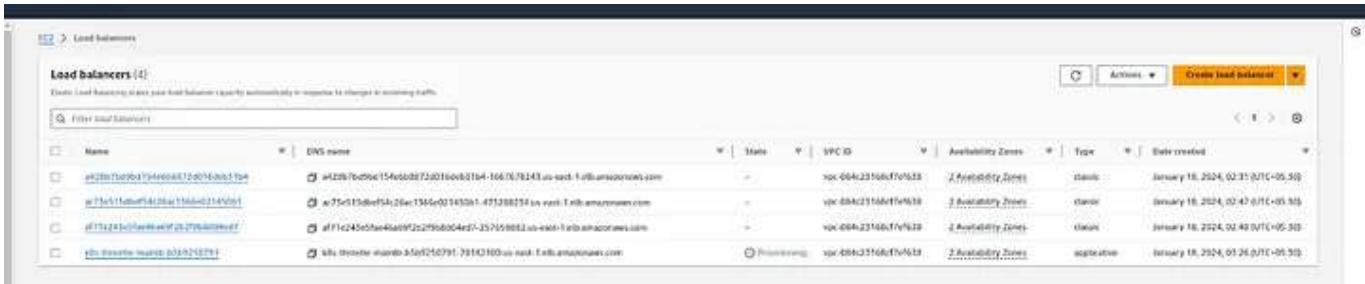
In the **Path**, provide the location where your Manifest files are presented and provide other things as shown in the below screenshot.

Click on **CREATE**.



Once your Ingress application is deployed. It will create an **Application Load Balancer**

You can check out the load balancer named with k8s-three.



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

Go to DNS and add a CNAME type with hostname backend then add your ALB in the Answer and click on Save

**Note:** I have created a subdomain backend.amanpathakdevops.study

The screenshot shows the Cloudflare DNS interface with the domain 'amangpathakdevops.study' selected. It displays a table of current records:

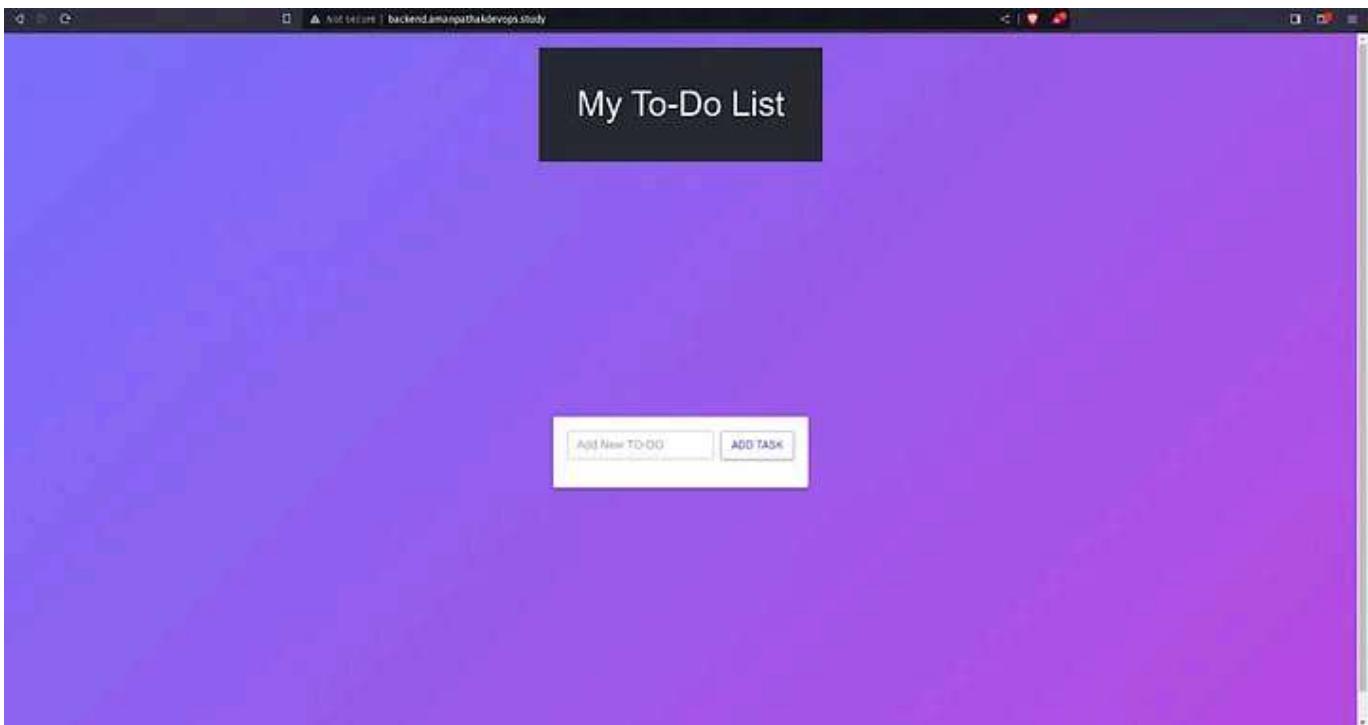
Type	Host	Answer	TTL	Priority	Options
ALIAS	alias/pathakdevops.study	prox.porkbun.com	600		
CNAME	85e8795ab71764f902993a ba49e33d45.amangpathakde vops.study	192.168.0.250.dns.27384223 1.16.96.132.2.mn148gdpdr4 .m-validation.aws	600		
CNAME	backend	http://threelb-mainlb-500 .amangpathakdevops.study	600		
MX	*.amangpathakdevops.study	prox.porkbun.com	600		
MX	amangpathakdevops.study	fw01.porkbun.com	600	10	

You can see all 4 application deployments in the below snippet.

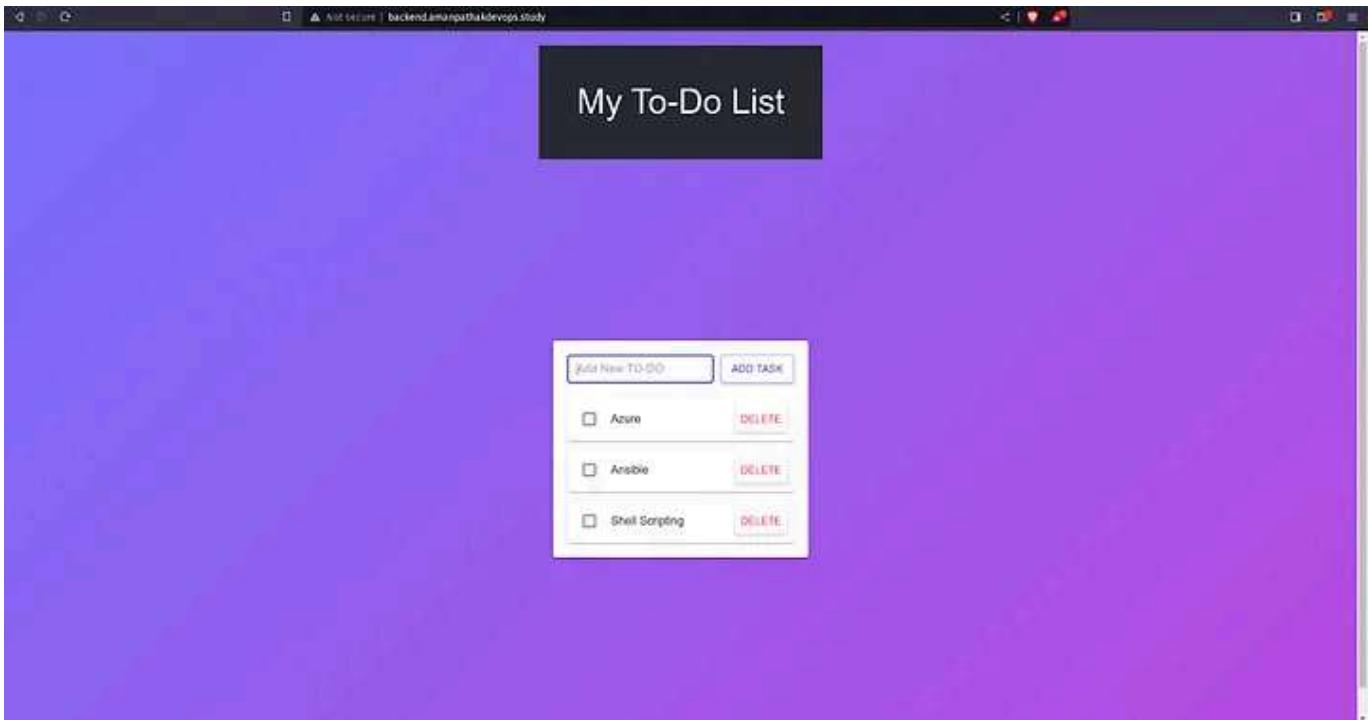
The screenshot shows the ArgoCD UI displaying four application deployments:

- backend**: Project: default, Status: Healthy Synced, Repository: https://github.com/amangpathak/DevOps..., Target Ref: HEAD, Path: Kubernetes-Manifests/Backend/
- database**: Project: default, Status: Healthy Synced, Repository: https://github.com/amangpathak/DevOps..., Target Ref: HEAD, Path: Kubernetes-Manifests/Database/
- frontend**: Project: default, Status: Healthy Synced, Repository: https://github.com/amangpathak/DevOps..., Target Ref: HEAD, Path: Kubernetes-Manifests/Frontend/
- ingress**: Project: default, Status: Healthy Synced, Repository: https://github.com/amangpathak/DevOps..., Target Ref: HEAD, Path: Kubernetes-Manifests/Ingress/

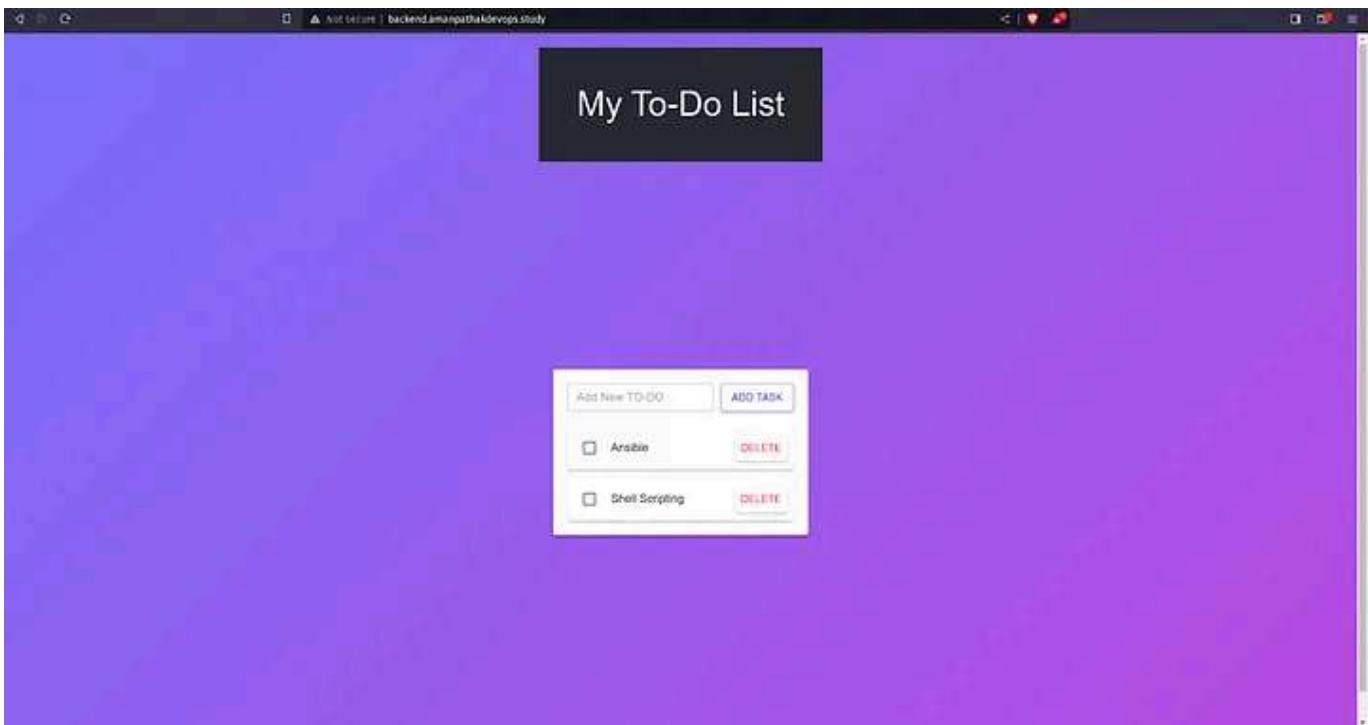
Now, hit your subdomain after 2 to 3 minutes in your browser to see the magic.



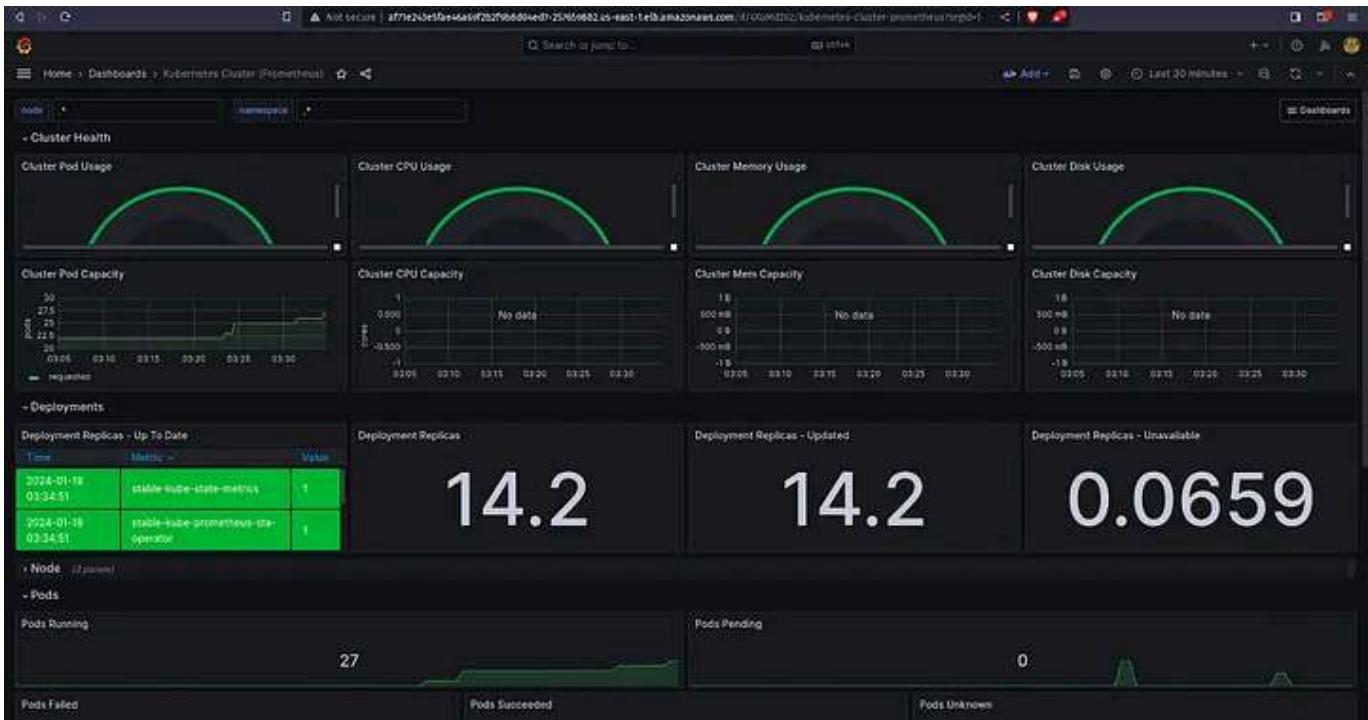
You can play with the application by adding the records.



You can play with the application by deleting the records.



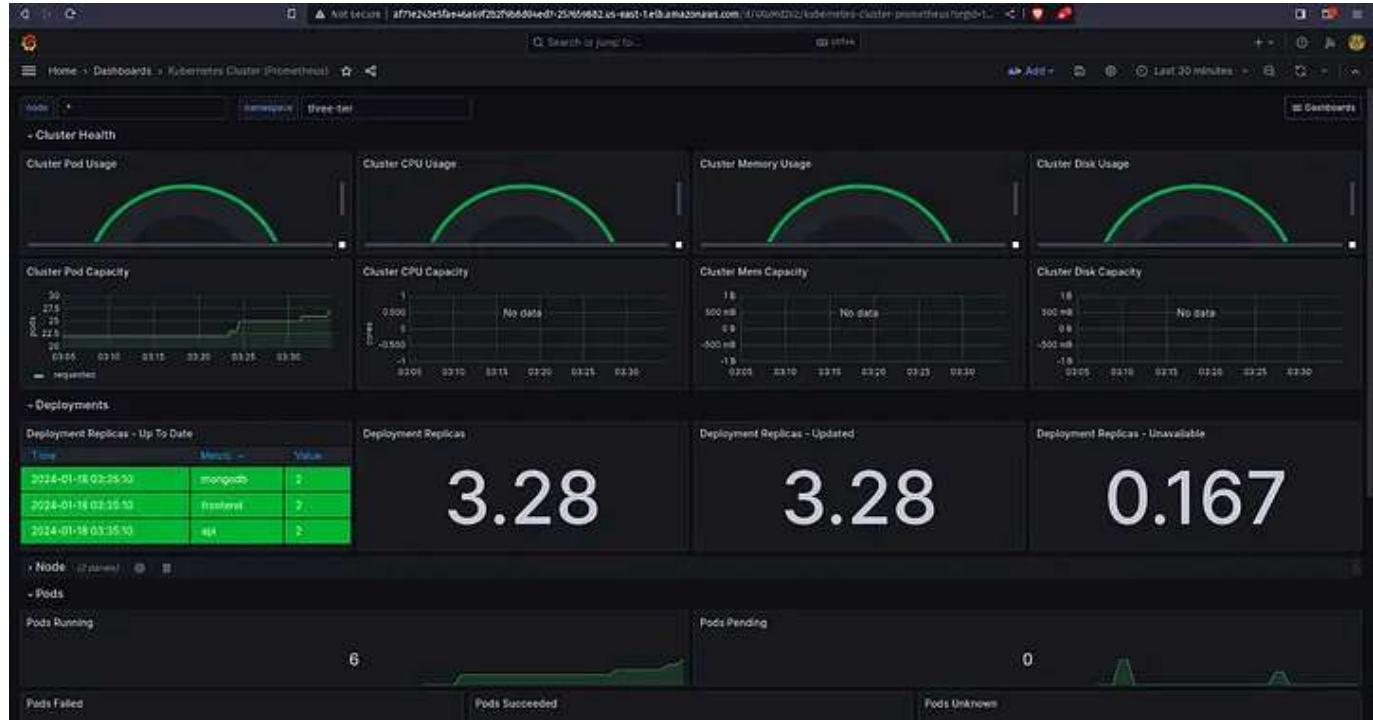
Now, you can see your Grafana Dashboard to view the EKS data such as pods, namespace, deployments, etc.



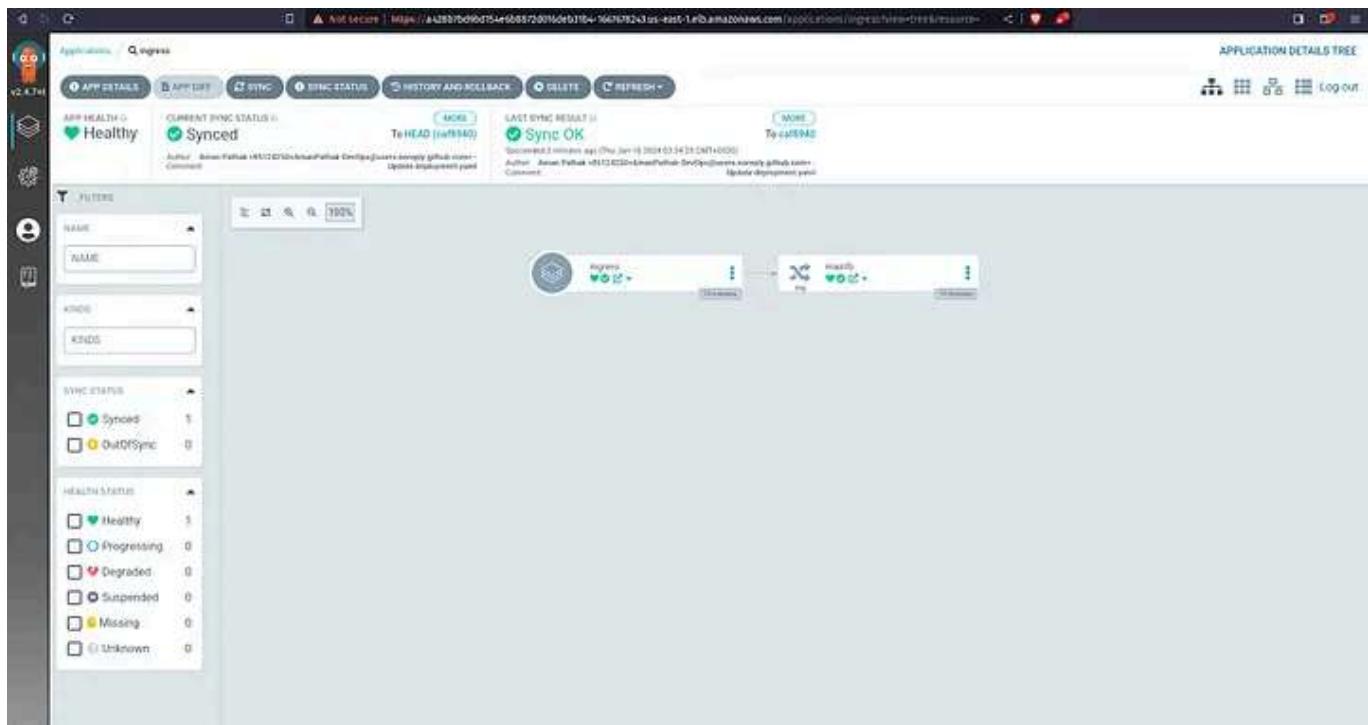
If you want to monitor the three-tier namespace.

In the namespace, replace three-tier with another namespace.

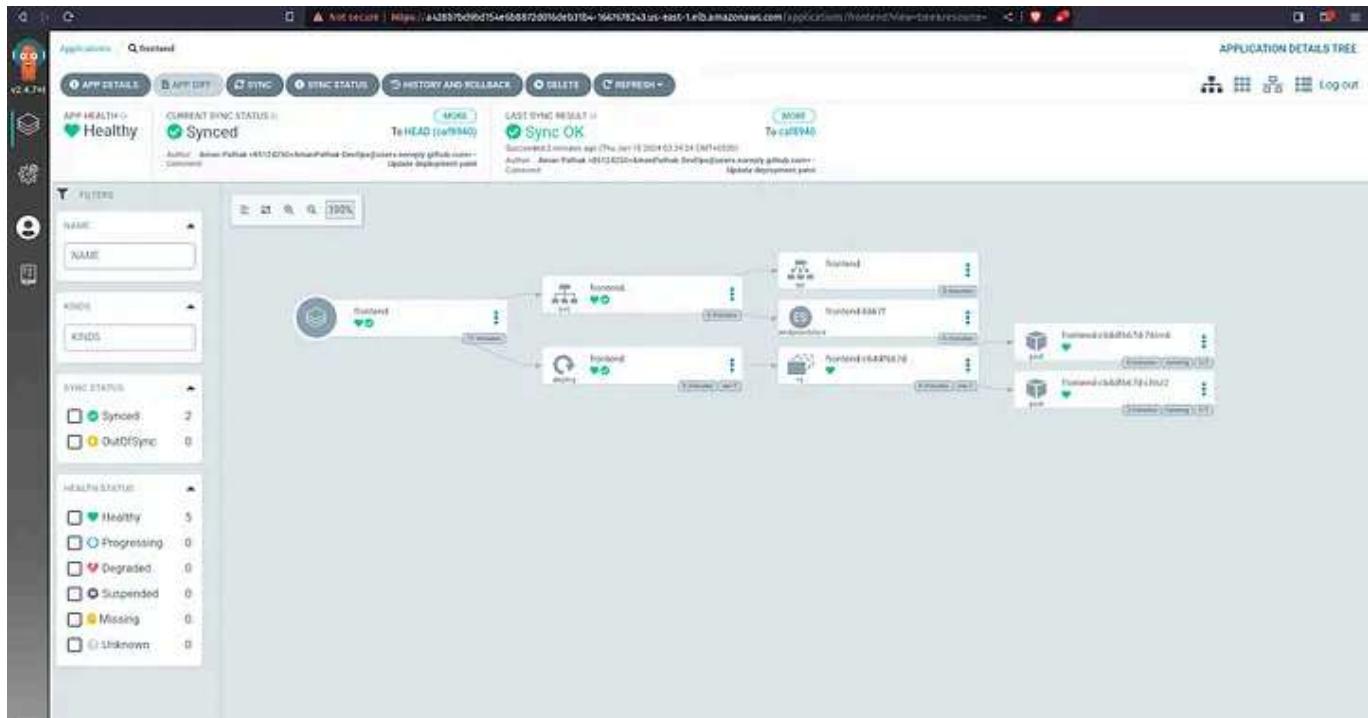
You will see the deployments that are done by ArgoCD



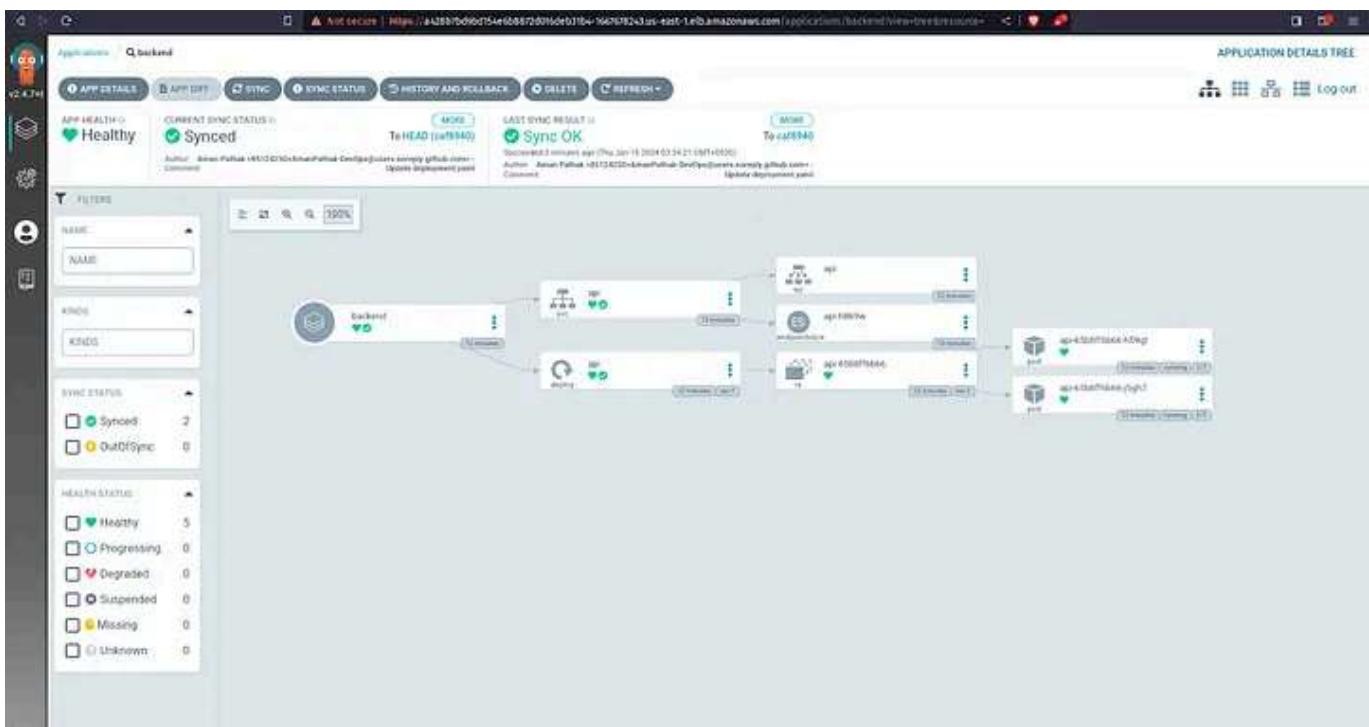
This is the Ingress Application Deployment in ArgoCD



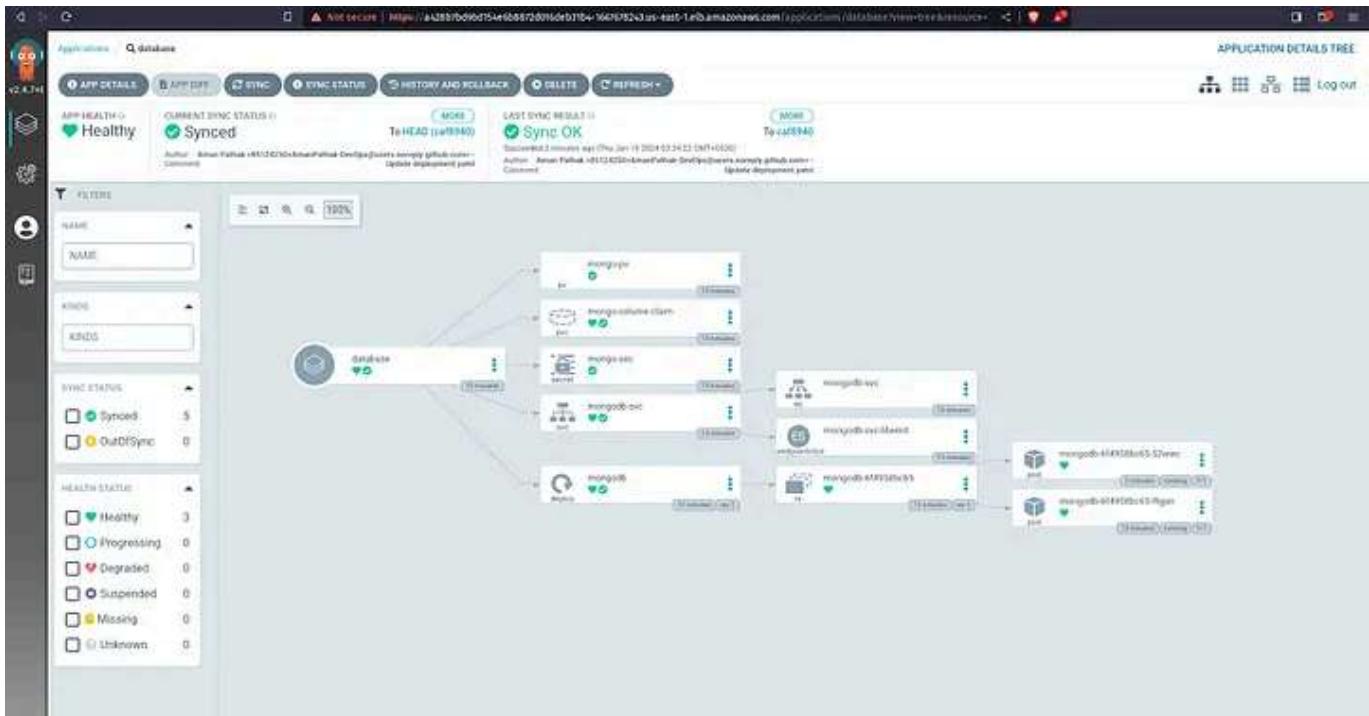
This is the Frontend Application Deployment in ArgoCD



This is the Backend Application Deployment in ArgoCD

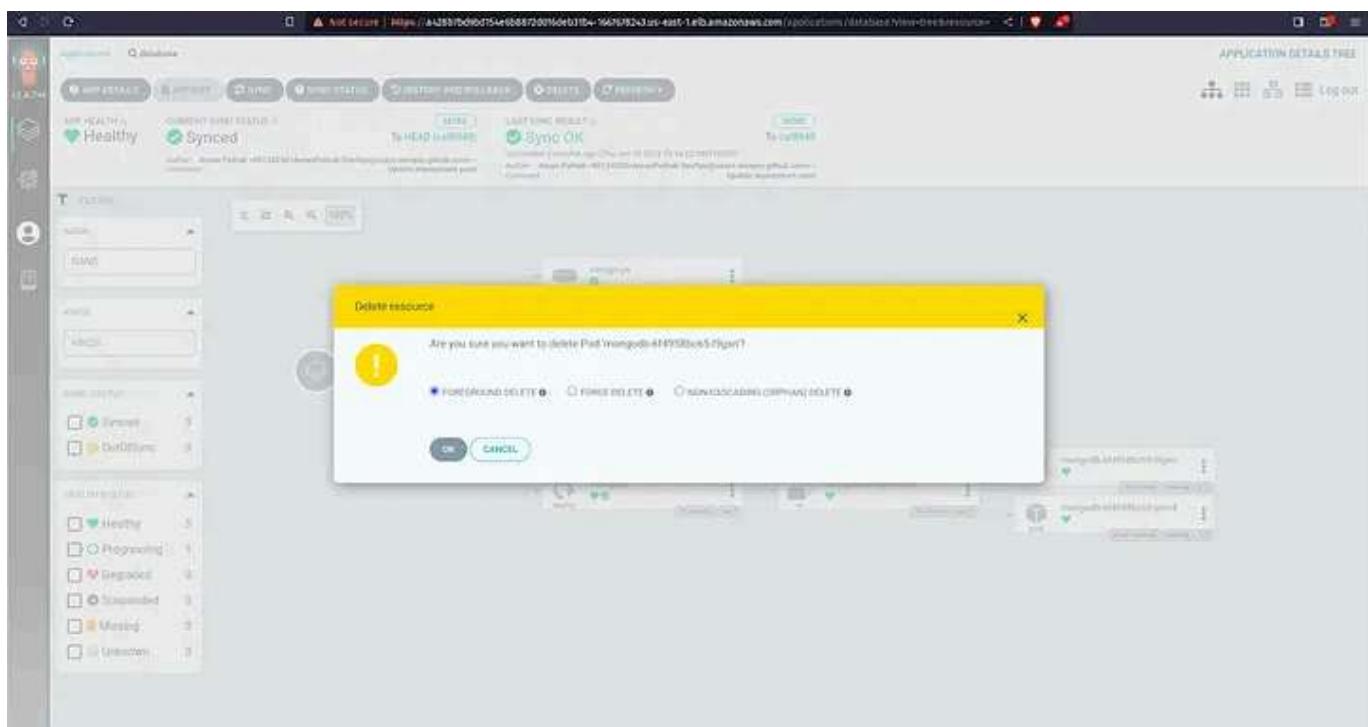


This is the Database Application Deployment in ArgoCD

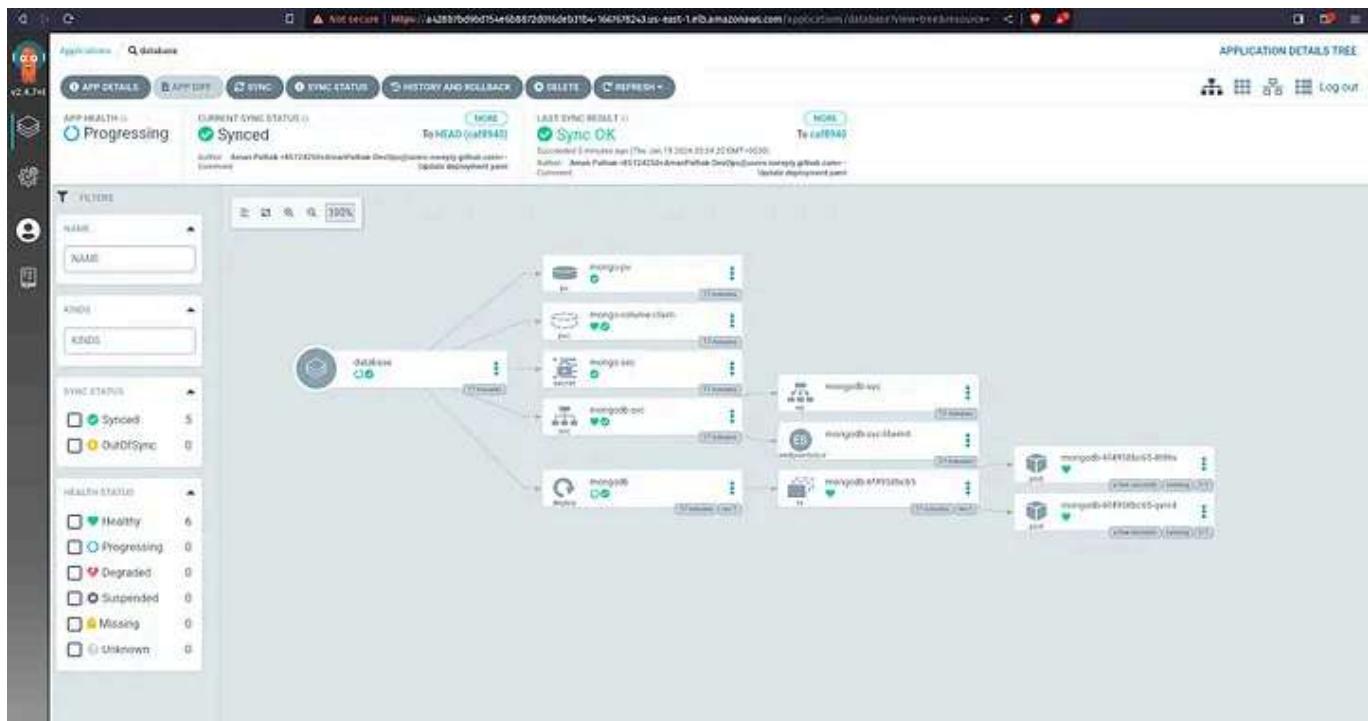


If you observe, we have configured the Persistent Volume & Persistent Volume Claim. So, if the pods get deleted then, the data won't be lost. The Data will be stored on the host machine.

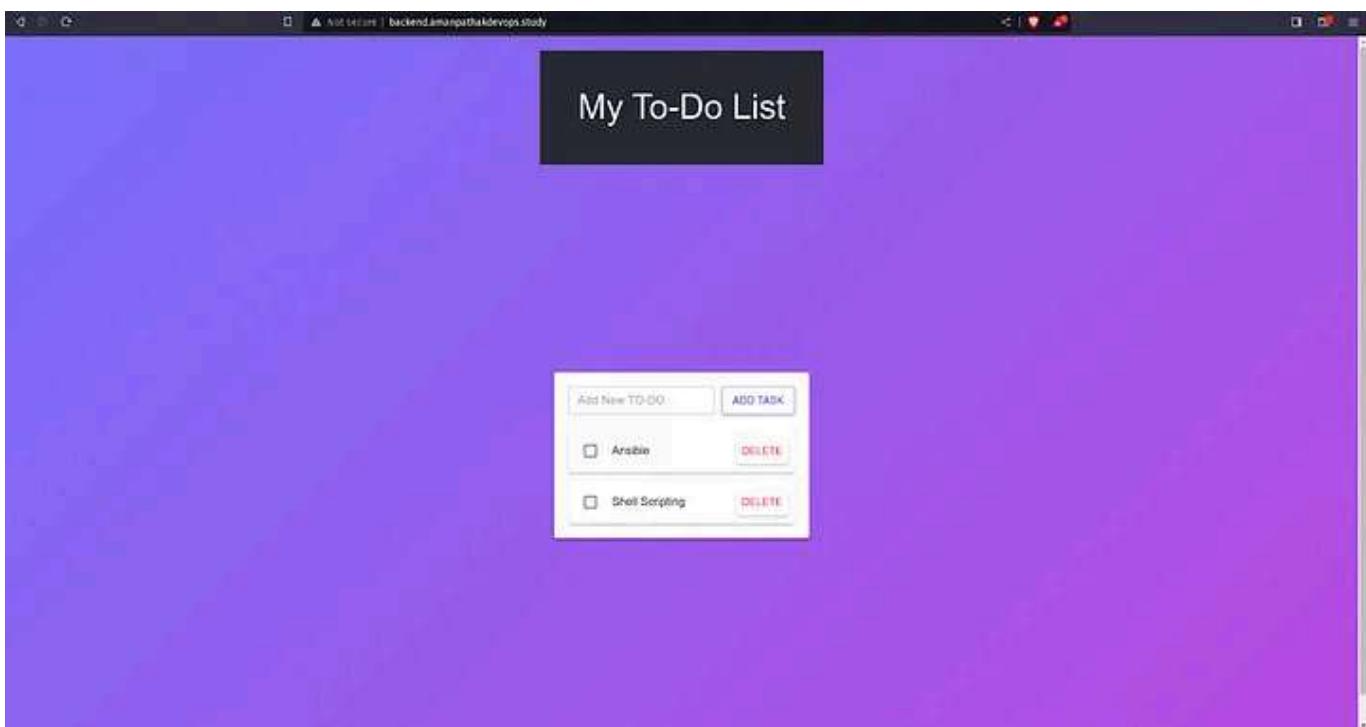
To validate it, delete both Database pods.



Now, the new pods will be started.



And Your Application won't lose a single piece of data.



## Conclusion:

In this comprehensive DevSecOps Kubernetes project, we successfully:

- Established IAM user and Terraform for AWS setup.
- Deployed Jenkins on AWS, configured tools, and integrated it with Sonarqube.
- Set up an EKS cluster, configured a Load Balancer, and established private ECR repositories.
- Implemented monitoring with Helm, Prometheus, and Grafana.
- Installed and configured ArgoCD for GitOps practices.

- Created Jenkins pipelines for CI/CD, deploying a Three-Tier application.
- Ensured data persistence with persistent volumes and claims.

## Support my work-

<https://www.buymeacoffee.com/aman.pathak>

Stay connected on LinkedIn: [LinkedIn Profile](#)

Stay up-to-date with GitHub: [GitHub Profile](#)

Want to discuss trending technologies in DevOps & Cloud

Join the Discord Server- <https://discord.gg/jdzF8kTtw2>

Feel free to reach out to me, if you have any other queries.

Happy Learning!

## Stackademic

Thank you for reading until the end. Before you go:

- Please consider clapping and following the writer! 
- Follow us [X](#) | [LinkedIn](#) | [YouTube](#) | [Discord](#)
- Visit our other platforms: [In Plain English](#) | [CoFeed](#) | [Venture](#)

Kubernetes

DevOps

Devsecops

Jenkins

AWS



## Published in Stackademic

[Follow](#)

30K Followers · Last published 3 days ago

Stackademic is a learning hub for programmers, devs, coders, and engineers. Our goal is to democratize free coding education for the world.



## Written by Aman Pathak

[Follow](#)

9.6K Followers · 5 Following

DevOps & Cloud Engineer | AWS Community Builder | AWS Certified | CKA Certified | Azure | Terraform | Docker | Ansible | CI/CD Jenkins | Oracle Certified

## Responses (26)



What are your thoughts?

[Respond](#)

Tommy Kaufmann

Jan 22, 2024

...

This is great work - hats off to you! I would strongly recommend, if you haven't already on your YouTube page, taking this entire post and turning into an instructional video. If you display this as a guided-walkthrough on a video, while lecturing how & why you did certain things, I think it would be superb.



15

1 reply

[Reply](#)



Avant Aditya

Jan 23, 2024

...

This looks good for beginners, but in industry standards, even Jenkins server creation, configuration should be done by terraform.



62 1 reply [Reply](#)



Truong Hoang Phu Loc

Jan 23, 2024

...

Hi Aman Pathak

I really appreciate your post. It's very informative and helpful.

Apart from that, here's the critical point I see on your post. In Configure Terraform section, you blurred your secret. However, in AWS CLI section, it seems that you... [more](#)



5 1 reply [Reply](#)

[See all responses](#)

## More from Aman Pathak and Stackademic



In Stackademic by Aman Pathak

**Tips and Tricks to Clear the CKA Exam**

In Stackademic by Dylan Cooper

**Zig's New Masterpiece: HashiCorp Co-founder's Latest Creation Tak...**

Hi guys, I hope you are doing well. Recently, I cleared my CKA exam with 75% marks, whic...

Dec 20, 2024 • 60 👏 5



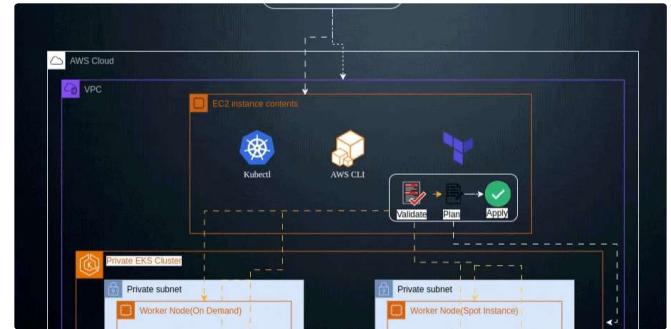
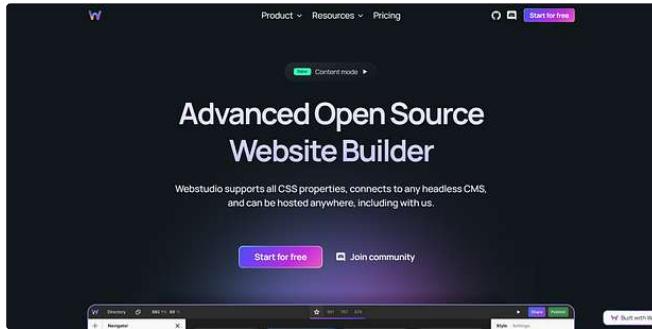
...

Recently, a terminal emulator written in Zig reached its official 1.0 release, making waves...

Jan 13 • 867 👏 12



...



In Stackademic by Let's Code Future

## 15 Open-Source Alternatives to Popular SaaS Tools & Apps 🚀✨

In today's fast-paced digital world, relying on SaaS tools like Webflow, Firebase, or...

Jan 16 • 897 👏 8



...

In Towards Dev by Aman Pathak

## Configure ArgoCD, Prometheus, Grafana & AWS Load Balancer...

Introduction

Sep 3, 2024 • 243 👏 1



...

See all from Aman Pathak

See all from Stackademic

## Recommended from Medium



In devsecops-community by Karthick Dkk

## 5 DevSecOps Project Ideas for Improve Your Skills

Here's a step-by-step breakdown for each of the 5 DevSecOps projects for beginners.



Oct 21, 2024

4



...



In Level Up Coding by Akhilesh Mishra

## Run Multi-Container Applications With Docker Compose

Docker composes to run a web application with Flask and Postgres containers.



Sep 10, 2024

190

1



...

## Lists



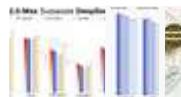
### General Coding Knowledge

20 stories · 1889 saves



### Productivity

244 stories · 672 saves

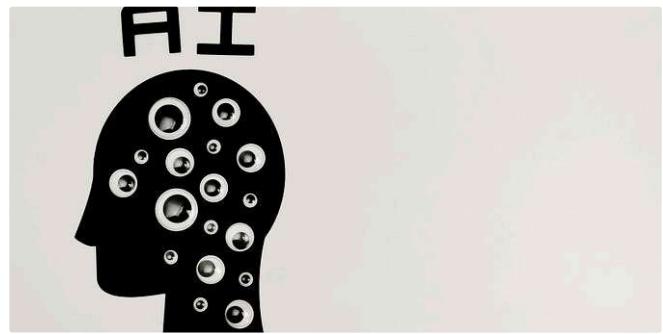


### Natural Language Processing

1908 stories · 1563 saves



In AWS in Plain En... by Mélony Qin (aka cloud...



Zudonu Osomudeya

## What I Wish I Knew When I Got Started with Kubernetes

Get acquainted with the key things you need to know to get started with Kubernetes,...



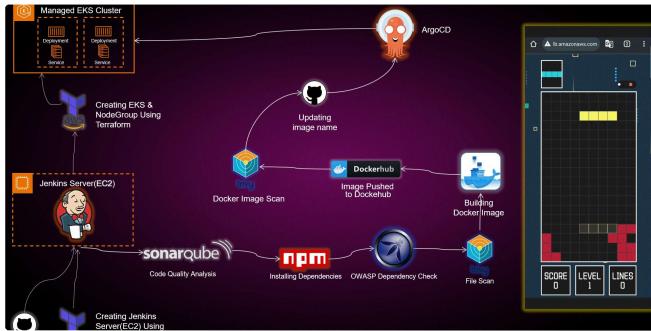
Mar 12, 2024



335



1



Ikeri Ebenezer

## End-to-End DevSecOps Kubernetes Project using AWS EK...

Project Introduction

Aug 15, 2024



149



3



See more recommendations

## Every DevOps Engineer Must Know How to Use AI in 2025

Why Every DevOps Engineer Must Know How to Use AI in 2025



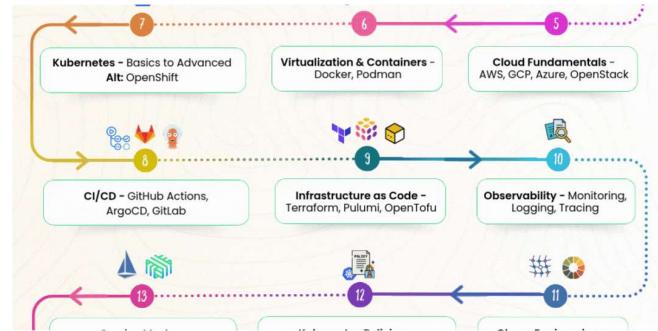
Dec 30, 2024



24



1



Rohit Ghumare

## DevOps Roadmap 2025

In today's rapidly evolving tech landscape, DevOps has become more than just a...



Jan 15



244



6

