



Search

Write

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

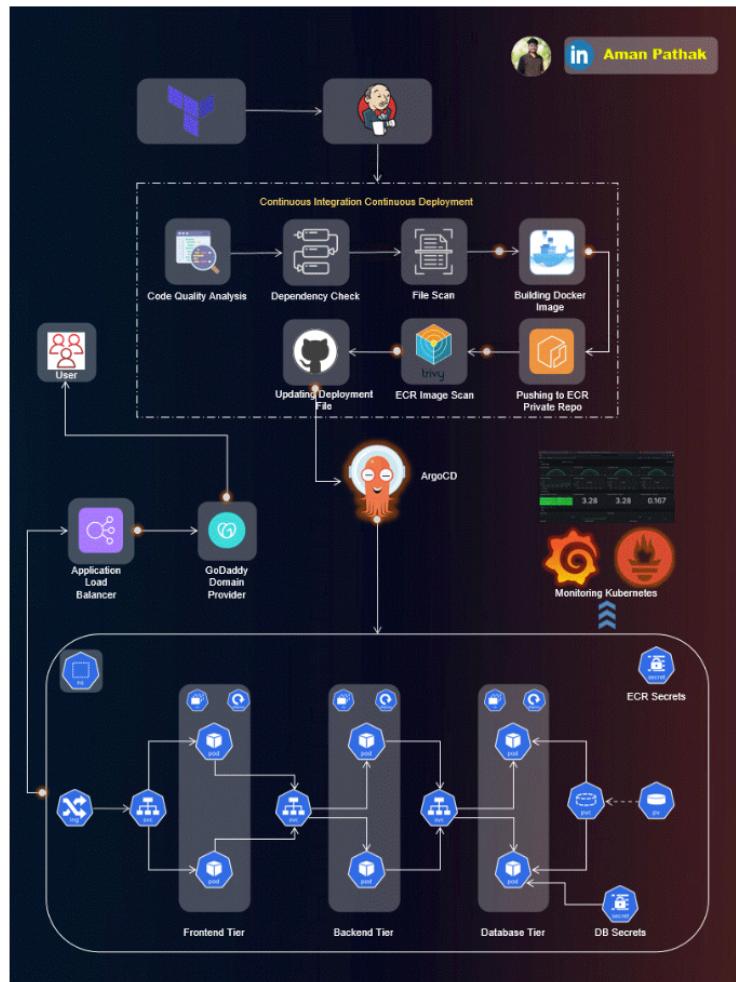
Advanced End-to-End DevSecOps Kubernetes Three-Tier Project using AWS EKS, ArgoCD, Prometheus, Grafana, and Jenkins

Aman Pathak · [Follow](#)

Published in Stackademic · 23 min read · Jan 18

958

10



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 Identity and Access Management (IAM) dashboard. It includes sections for security recommendations, IAM resources (with 1 user, 4 roles, 25 policies, and 11 identities), and user statistics (with 2 users). A sidebar on the left provides navigation links for various IAM management tasks.

Click on Create user

This is the first step of the 'Create user' wizard. It asks for the user name ('DevSecOps-Project') and provides options to provide access to the IAM Management Console or programmatic access via AWS Lambda or Amazon Kinesis. A note about using IAM roles is also present.

Provide the name to your user and click on Next.

This step continues the 'Create user' wizard, showing the user details already entered. It includes fields for user name, access type, and a note about using IAM roles.

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

Click on the Next.

This step shows the 'Set permissions' section where the 'Attach policies directly' checkbox is selected. A search bar is used to find the 'AdministratorAccess' policy, which is highlighted with a red box. Other policies like 'AmazonS3FullAccess' and 'AmazonCloudWatchLogsFullAccess' are also listed.

Click on Create user

This is the final step of the 'Create user' wizard. It reviews the user details (name: DevSecOps-Project, password type: None, password reset: No) and the attached policy (AdministratorAccess). It also shows the 'Add new tag' field and the 'Create user' button.

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

This screenshot shows the user details for 'DevSecOps-Project'. It displays the user's ARN, access type (Stashed), and the last service sign-in date (January 12, 2020, 14:07:03 UTC). The 'Create access key' button is visible at the bottom right.

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

Provide the Description and click on the **Create access key**.

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

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/ $(lsb_release -cs) 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=[REDACTED]
export AWS_SECRET_ACCESS_KEY=[REDACTED]
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=[REDACTED]
export TF_VAR_ENDPOINT=[REDACTED]
export TF_VAR_PROFILE=/home/amanpathak/Downloads/Profile
filelet 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-DevSecOps-Tetris-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).

```
1 terraform {
2   backend "s3" {
3     bucket      = "my-aws-bucket1"
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 > lam-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-Patnak"
8 iam-role       = "Jenkins-iam-role"

```

Initialize the backend by running the below command

```

terraformer init

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS$ 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.
[1] amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS []

```

Run the below command to check the syntax error

```

terraformer validate

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS$ terraform validate
Success! The configuration is valid.

● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS$ terraform fmt
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS []

```

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

```

terraformer plan -var-file=variables.tfvars

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS$ terraformer plan -var-file=variables.tfvars
  )
  + 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_resolution = (known after apply)
  + enable_ip_address_allocation = (known after apply)
  + enable_network_address_usage_metrics = (known after apply)
  + id            = (known after apply)
  + instance_tenancy = (known after apply)
  + ipv6_association_id = (known after apply)
  + ipv6_cidr_block = (known after apply)
  + main_gateway_id = (known after apply)
  + main_route_table_id = (known after apply)
  + owner_id       = (known after apply)
  + tags_all      = [
      + "Name" = "Jenkins-vpc"
    ]
  + tags          = [
      + "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.

● amanpathak@pop-os:~/End-to-End-Kubernetes-Three-Tier-DevSecOps-Project/Jenkins-Server-TFS []

```

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



```
  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_group: Creating...
aws_iam_group: Creation complete after 2s [id=Jenkins-iam-role]
aws_iam_role_attachment: Creation complete after 2s [id=Jenkins-iam-role]
aws_iam_role_policy_attachment: Creation complete after 2s [id=Jenkins-iam-role]
aws_iam_role_policy_attachment: Creation complete after 0s [id=Jenkins-iam-role-policy-attachment]
aws_iam_instance_profile_instance_profile: Creation complete after 1s [id=Jenkins-instance-profile]
aws_internet_gateway: Creating...
aws_internet_gateway: Creation complete after 4s [id=ipmc-0b745ebe82f2f1e3]
aws_subnet: Creating...
aws_route_table: Creating...
aws_route_table: Creation complete after 2s [id=rtr-0d5606e24d724302]
aws_security_group: Creating...
aws_security_group: Creation complete after 2s [id=rsg-09cfe46880004006]
aws_security_group_security_group: Creating complete after 0s [id=rsg-09cfe46880004006]
aws_subnet_public_subnet: Creating... [10s elapsed]
aws_subnet_public_subnet: Creation complete after 10s [id=subnet-057a779a074ca9c8]
aws_route_table_association_rt_association: Creating...
aws_instance: Creating...
aws_instance: Creation complete after 26s [id=idrthassoc-0e0878d86c7990bf9]
aws_instance_ec2: Still creating... [10s elapsed]
aws_instance_ec2: Creation complete after 26s [id=id-0294836e9123c07b]

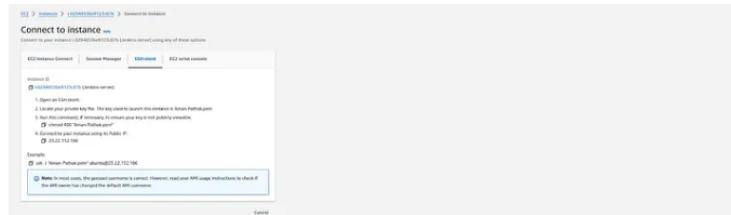
Releasing state lock. This may take a few moments...

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
anamapthagop-es-1>End-to-End-Test-Execution-Three-Tier-development-project/Jenkins-Server-TFS []
```

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

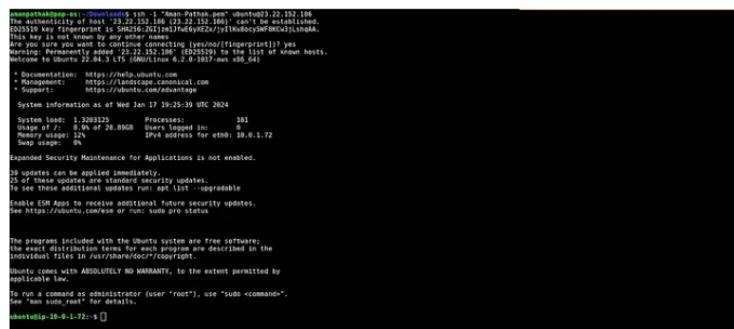


Copy the ssh command and paste it on your local machine



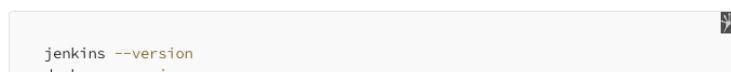
Step 4: Configure the Jenkins

Now, we logged into our Jenkins server.



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

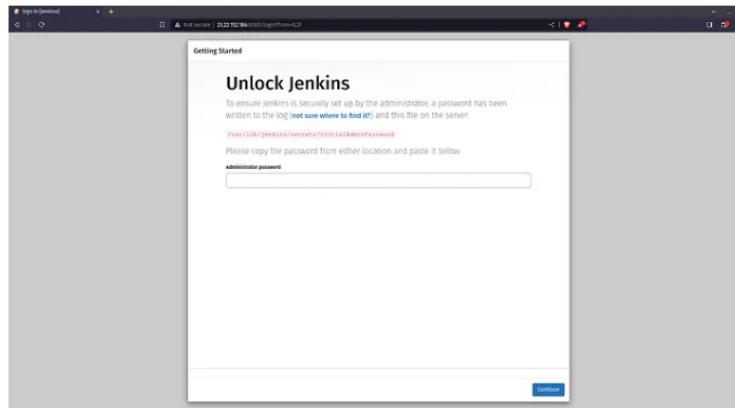
Let's validate whether all our installed or not.



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

```
ubuntu@ip-10-0-1-72:~$ trivy --version
Version: 0.48.3
ubuntu@ip-10-0-1-72:~$ ekctl 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
```

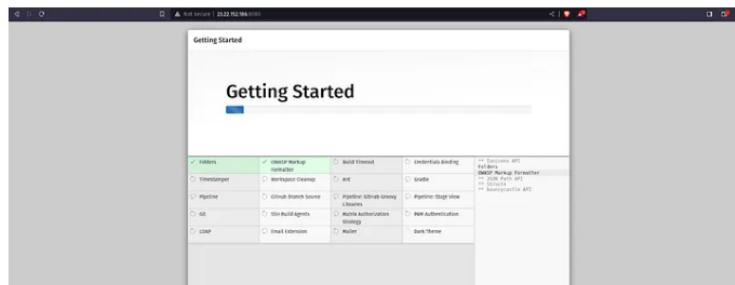
Now, we have to configure Jenkins. So, copy the public IP of your Jenkins Server and paste it on your favorite browser with an 8080 port.

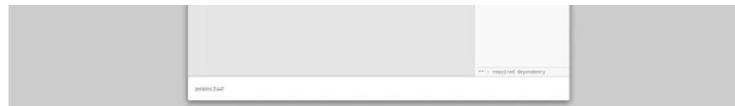


Click on Install suggested plugins

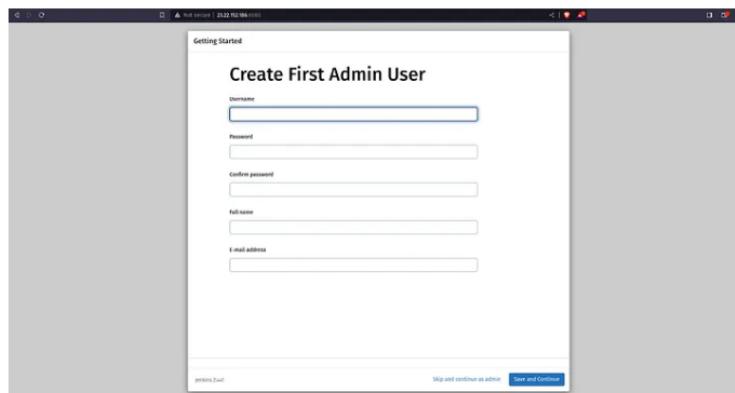


The plugins will be installed

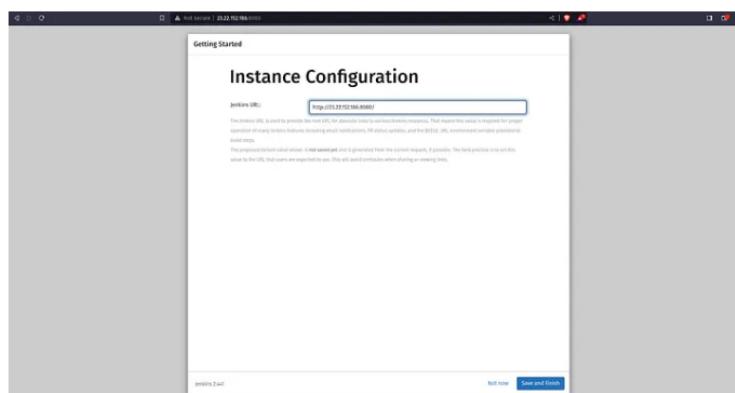




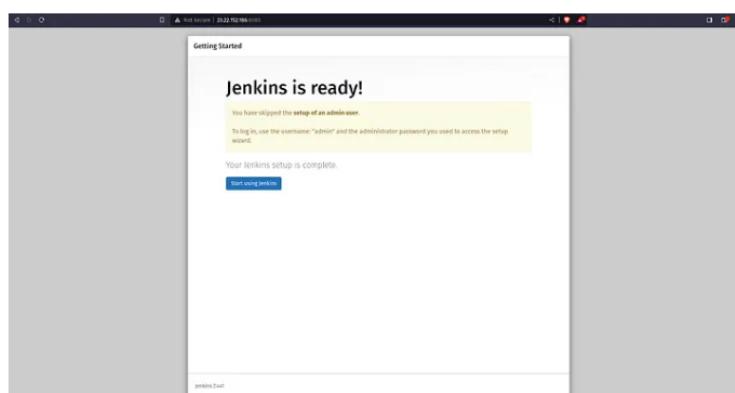
After installing the plugins, continue as admin



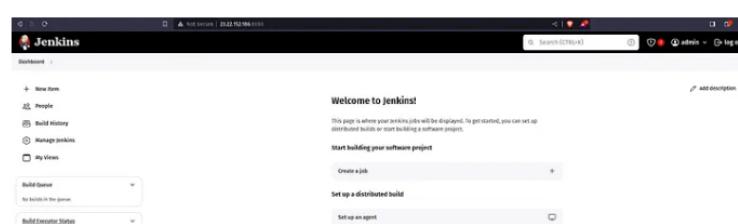
Click on Save and Finish

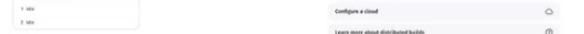


Click on Start using Jenkins



The Jenkins Dashboard will look like the below snippet





jenkins.kubeconfig

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]: AKIAV52BN5RJWCSSA6P
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

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

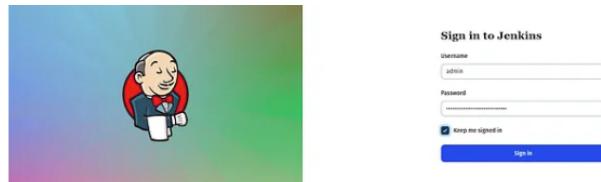
AWS Credentials

Pipeline: AWS Steps

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

Login to your Jenkins Server Again





Now, we have to set our AWS credentials on Jenkins

Go to Manage Plugins and click on Credentials

Click on global.

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 Credentials will look like the below snippet.

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.

Temporary URL: [http://10.0.1.72](#)

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

New credentials

Kind: Username with password

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

Username: amanshukhdev

Password:

Description: GitHub

Create

Both credentials will look like this.

ID	Name	Kind	Description
aws key	AMANSHUKHDEVSSMAP (aws key)	AWS Credentials	aws key
GitHub	AmanshukhDevGitHub	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 t2.medium --nodes-min 2 --nodes-max 2
aws eks update-kubeconfig --region us-east-1 --name Three-Tier-K8s-EKS-Cluster
```

```
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
```

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-K8s-EKS-Cluster
kubectl get nodes
arn:aws:eks:us-east-1:116764280206:cluster/Three-Tier-K8s-EKS-Cluster to /home/ubuntu/.kube/config
NAME           STATUS   ROLES      AGE       VERSION
ip-10-0-1-72.ek2.internal   Ready    <none>   5m3s   v1.28.5-eks-5ebfde
ip-10-0-2-218.ek2.internal   Ready    <none>   5m3s   v1.28.5-eks-5ebfde
ip-10-0-3-244.ek2.internal   Ready    <none>   5m3s   v1.28.5-eks-5ebfde
```

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/v2.1.0/deploy/overlays/k8s-gke.yaml
```

```
ubuntu@ip-10-0-1-72: ~ curl -O https://raw.githubusercontent.com/kubernetesci/sigs/aws-load-balancer-controller/v2.5.4/doc/install/iam_policy.json  
% Total % Received % Xferd Average Speed Time Left Speed  
100 8386 100 8386 0 0 93273 0 ====== 0:01:53  
ubuntu@ip-10-0-1-72: ~  
ubuntu@ip-10-0-1-72: ~ ls  
aws-load-balancer-controller-v2.5.4  
iam_policy.json
```

Create the IAM policy using the below command

```
aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-  
[REDACTED]
```

```
[root@elk-10-0-1-72] ~]# curl -X POST -H "Content-Type: application/json" -d @awsloadbalancercontrollerIAMPolicy.json https://iam.us-east-1.amazonaws.com/api/v2/policies
```

```
[{"Policy": {"PolicyName": "AWSLoadBalancerControllerIAMPolicy", "Arn": "arn:aws:iam::123456789012:policy/AWSLoadBalancerControllerIAMPolicy", "Description": "A policy for the AWS Load Balancer Controller", "AttachmentCount": 0, "IsAttachable": true, "CreateDate": "2023-01-27T09:57:47+00:00", "UpdateDate": "2023-01-27T09:57:47+00:00"}]
```

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-KBS-EKS-Cluster --approve
2024-05-17 10:58:31 [i] will create IAM Open ID Connect provider for cluster "Three-Tier-KBS-EKS-Cluster" in "us-east-1"
2024-05-17 10:58:31 [i] created IAM Open ID Connect provider for cluster "Three-Tier-KBS-EKS-Cluster" in "us-east-1"
ubuntu@ip-10-0-1-72:~$
```

Create Service Account

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

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

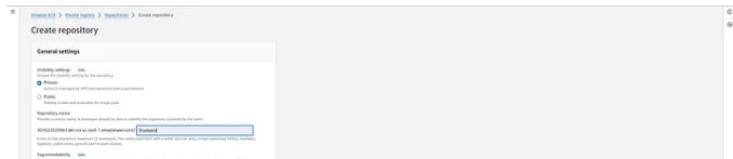
```
kubectl get deployment -n kube-system aws-load-balancer
```

```
root@host-10-0-1-72: ~ kubectl get deployment -n kube-system net-load-balancer-controller
NAME                      READY   AGE
net-load-balancer-controller 3/3    24m
```

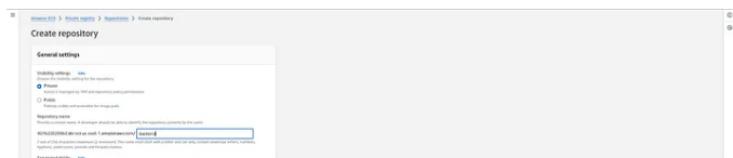
Step 7: We need to create



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



Do the same for the backend repository and click on Save



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.

```
shopt -s -o host刹那
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password $(aws ecr get-login-password --region us-east-1)
Configure a credential helper to remove this warning. See aws ecr get-login-password --no-include-email --region us-east-1 | docker login -u AWS -p \$\(aws ecr get-login-password --region us-east-1\)
Login Succeeded
shopt -s -o host刹那
shopt -s -o host刹那
```

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



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
secret/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
CustomResourceDefinition.argocd.argoproj.io created
CustomResourceDefinition.apirextensions.k8s.io/argocd.argoproj.io created
CustomResourceDefinition.apirextensions.k8s.io/applicationsets.argoproj.io created
ServiceAccount/argocd-application-controller created
ServiceAccount/argocd-dex-controller created
ServiceAccount/argocd-dns-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   1/1     Running   0          33s
argocd-dex-controller          1/1     Running   0          33s
argocd-notifications-controller 1/1     Running   0          33s
argocd-repo-server              1/1     Running   0          33s
argocd-server                  1/1     Running   0          33s
argocd-dns-server               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: ~
```

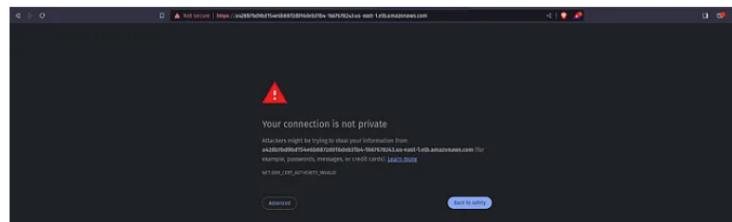
You can validate whether the Load Balancer is created or not by going to the AWS Console



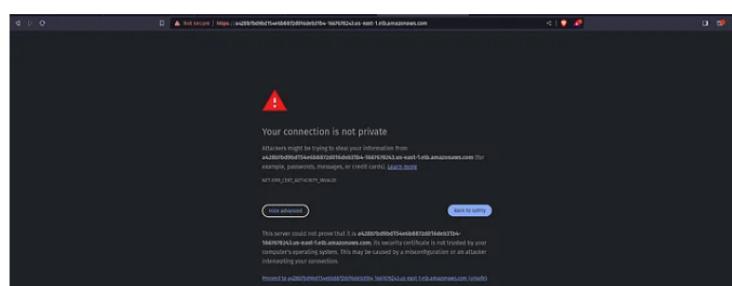
To access the argoCD, copy the LoadBalancer DNS and hit on your favorite browser.

You will get a warning like the below snippet.

Click on Advanced.

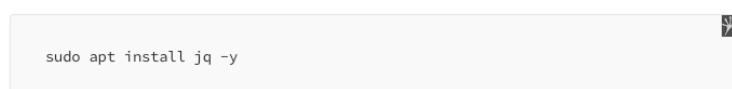


Click on the below link which is appearing under Hide advanced



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.

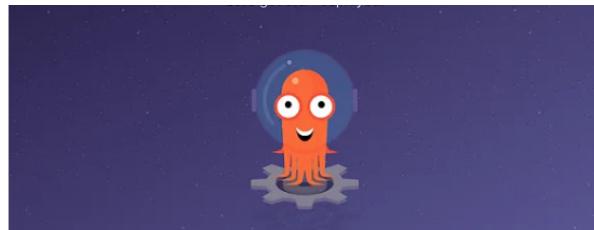


```
shantala-10-8-1-72: ~ $ sudo apt install jq -y
Reading package lists... Done
Reading state information... Done
0 upgraded, 1 newly installed, 0 to remove and 40 not upgraded.
Need to get 357 kB of additional disk space.
After this operation, 1087 kB of additional disk space will be used.
Get: 1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libltsql amnd64 1.0-2.1-2ubuntu2 [132 kB]
Get: 2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libltsql amnd64 1.0-2.1-2ubuntu2 [133 kB]
Get: 3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libltsql amnd64 1.0-2.1-2ubuntu2 [32.5 kB]
Fetched 357 kB in 0s (12.7 MB/s)
Reading database ... 16684 files and directories currently installed.
Unpacking libltsql amnd64 ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Processing triggers for ufw ...
shantala-10-8-1-72: ~ $
```

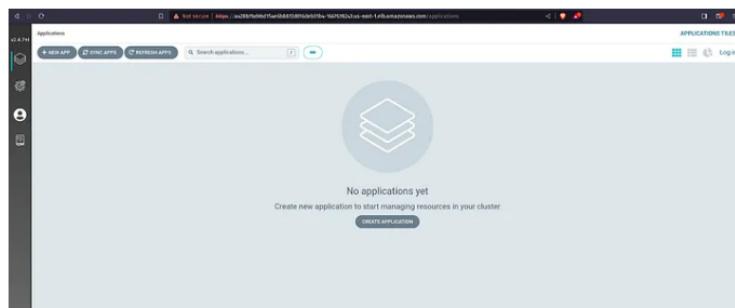
```
shantala-10-8-1-72: ~ $ export ARGOCD_SERVER='`kubectl get svc argocd-server -n argocd -o json | jq - raw`'
export ARGO_PWD='`kubectl -n argocd get secret argocd-initial-admin-secret -o json | base64 -d`'
echo $ARGO_PWD
shantala-10-8-1-72: ~ $
```

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

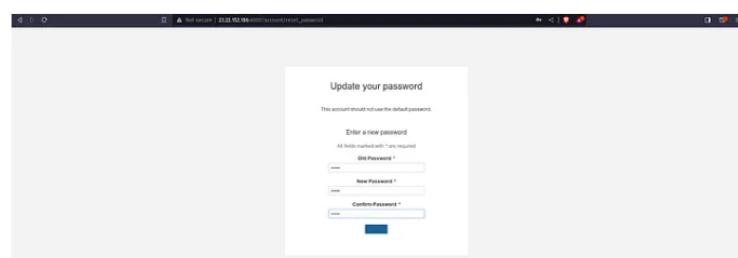
To do that, copy your Jenkins Server public IP and paste it on your favorite browser with a 9000 port

The username and password will be **admin**

Click on Log In.



Update the password



Click on Administration then Security, and select Users

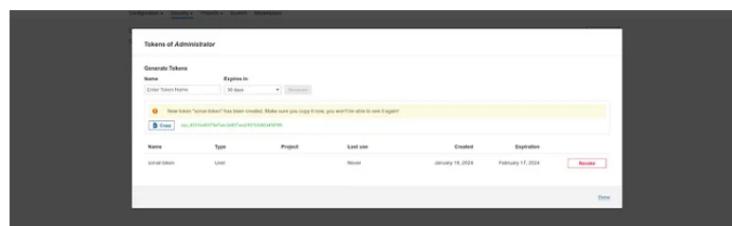
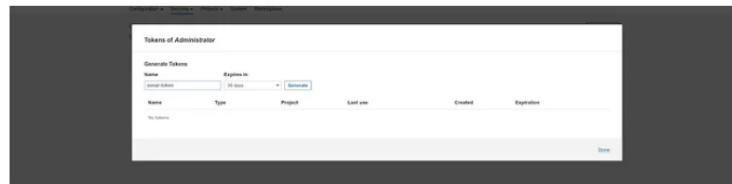


Click on Update tokens



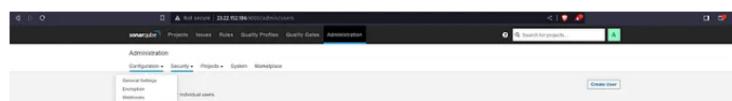


Click on Generate



Now, we have to configure webhooks for quality checks

Click on Administration then, Configuration and select Webhooks

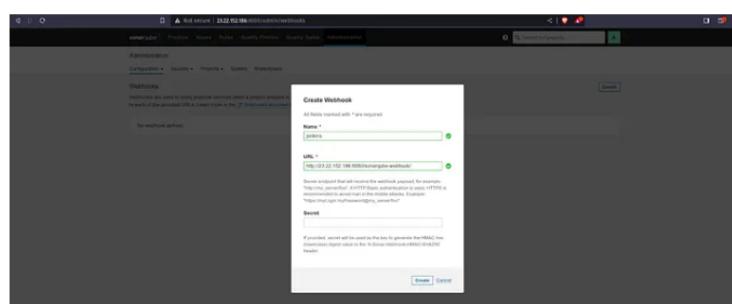


Click on Create



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.

The screenshot shows the SonarQube Administration interface under the 'Webhooks' section. It lists a single webhook entry for 'jenkins' with the URL 'http://127.0.0.1:8080/service/sonar/webhook'. The 'Has secret?' field is set to 'No' and 'Last delivery' is 'Never'. There is a 'Delete' button next to the entry.

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

Click on Manually.

The screenshot shows the 'Create project' page. Under the heading 'How do you want to create your project?', there are several options: 'From Azure DevOps', 'From Bitbucket Server', 'From Bitbucket Cloud', 'From GitHub', and 'From GitLab'. Below these, a link says 'Are you just testing or have an advanced use case? Create a project manually.' A large button labeled 'Manually' is centered below the link.

Provide the display name to your Project and click on Setup

The screenshot shows the 'Create project' page with the 'Project display name' field filled with 'frontend-test'. Other fields like 'Project key' and 'Main branch name' are also visible. At the bottom right, there is a 'Next step' button.

Click on Locally.

The screenshot shows the 'Create project' page with the 'How do you want to analyse your repository?' section. It lists options: 'With Jenkins', 'With GitHub Actions', 'With Bitbucket Pipelines', 'With GitLab CI', 'With Azure Pipelines', and 'Other CI'. Below these, a link says 'Are you just testing or have an advanced use case? Analyse your project locally.' A large button labeled 'Locally' is centered below the link.

Select the Use existing token and click on Continue.

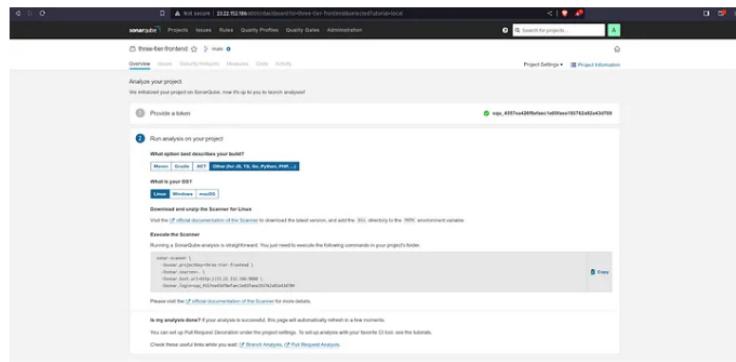
The screenshot shows the 'Analyze your project' page. It asks to 'Provide a token' and has two options: 'Generate a project token' (unchecked) and 'use existing token' (checked). Below this, there is a 'Existing token value' input field containing 'http://127.0.0.1:8080/service/sonar/webhook'. A note states: 'The token is used to identify you during an analysis performance. If it has been compromised, you can revoke it at https://127.0.0.1:8080/settings/tokens/revoke'. At the bottom is a 'Continue' button.

Select Other and Linux as OS.

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

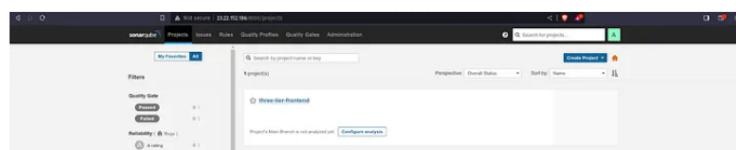
Now, use the command in the Jenkins Frontend Pipeline where Code Quality

Analysis will be performed.



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

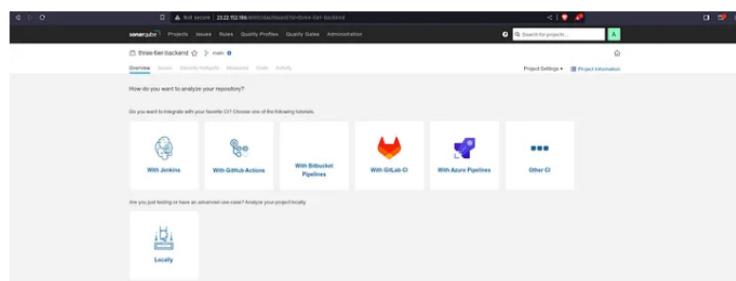
Click on Create Project.



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



Click on Locally.



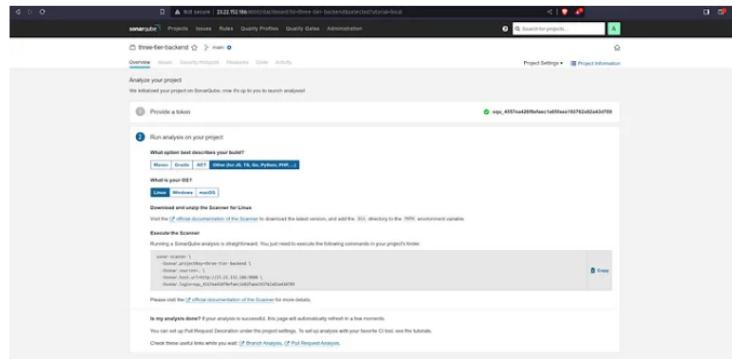
Select the Use existing token and click on Continue.



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 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**



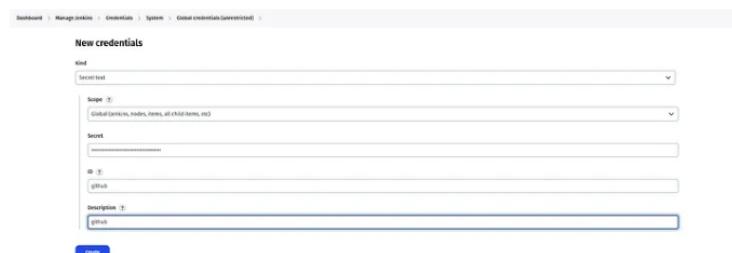
Now, we have to store the GitHub Personal access token to push the deployment file which will be modified in the pipeline itself for the ECR image.

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



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**

The screenshot shows the Jenkins 'New credentials' dialog. 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 the value 'ACCOUNT_ID'. The 'Description' field contains the value 'ACCOUNT_ID'. A blue 'Create' button is at the bottom right.

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' dialog. 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 the value 'REPO_NAME'. The 'Description' field contains the value 'REPO_NAME'. A blue 'Create' button is at the bottom right.

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' dialog. 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 the value 'REPO_NAME'. The 'Description' field contains the value 'REPO_NAME'. A blue 'Create' button is at the bottom right.

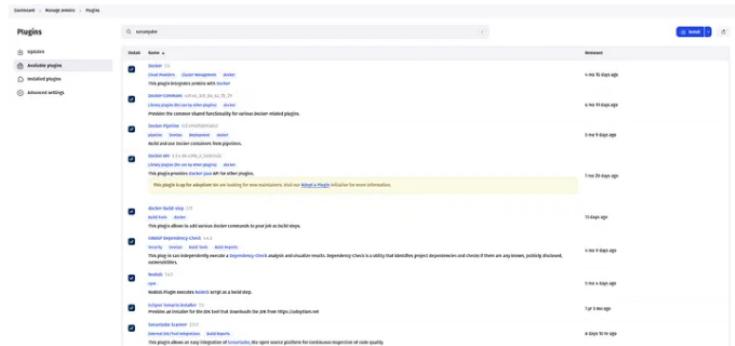
Final Snippet of all Credentials that we needed to implement this project.

The screenshot shows the Jenkins 'Global credentials (unrestricted)' list. It displays three entries: 'ACCOUNT_ID', 'REPO_NAME', and 'REPO_NAME'. A blue 'Add Credential' button is at the top right.

	Name	Type	Value	Action
	AWSKEYSTOREAWSKEY (aws key)	AWS Credentials	aws key	
	GITHUB (GitHub DevOps) (GITHUB)	Username with password	GITHUB	
	secrets-tokens	Secret text	secrets-tokens	
	github	Secret text	github	
	ACCOUNT_ID	Secret text	ACCOUNT_ID	
	ECR_REPO1	Secret text	ECR_REPO1	
	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**



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.



Now, we will configure the sonarqube-scanner

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





Now, we will configure **nodejs**

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



Now, we will configure the OWASP Dependency check

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



Now, we will configure the docker

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



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.

SonarQube servers

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

Environment variables

SonarQube installations
List of SonarQube installations

Name	<input type="text" value="sonar-server"/>	x
Server URL	SonarQube URL (http://localhost:9000)	
<input type="text" value="Http://172.21.12.100:9000"/>		
Server authentication token <small>SonarQube authentication token. Mandatory when anonymous access is disabled.</small>		
<input type="text" value="Sonar token"/>		
Add +		
Advanced ▾		

[Add SonarQube](#)

Pipeline Speed / durability

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

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

[Go to Jenkins Dashboard](#)

Click on New Item

The screenshot shows the Jenkins dashboard with the following elements:

- A top navigation bar with links for "Dashboard", "Manage Jenkins", "Help", and "Logout".
- A sidebar on the left with icons for "New item", "People", "Build History", "Project Relationship", "Check file fingerprint", "Manage Jenkins", and "My Views".
- The main content area features a large "Welcome to Jenkins!" heading, a sub-headline "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.", and two buttons: "Start building your software project" and "Create a job".
- A footer at the bottom with a "Set up a distributed build" link.

Provide the name of your Pipeline and click on OK.

Dashboard All

Enter an item name

There are no build applications
+ Create new

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

 **Matrix project**
Build a matrix project, Jenkins takes advantage of your POM files and drastically reduces the configuration.

 **Pipeline**
Generates long running activities that can span multiple build agents. Suitable for building pipelines (thereby known as workflow) and 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.

```

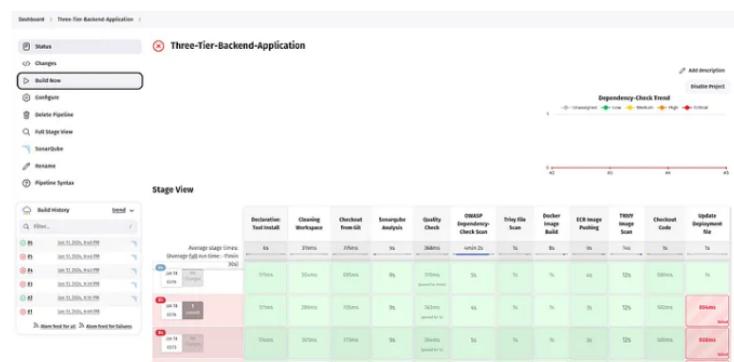
    stage('Build') {
        step('Python')
        step('Java')
        step('Node.js')
    }
    stage('Test') {
        step('Unit Tests')
        step('Integration Tests')
    }
    stage('Deploy') {
        step('Kubernetes Deploy')
    }

```

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.

```

    stage('Build') {
        step('Python')
        step('Java')
        step('Node.js')
    }
    stage('Test') {
        step('Unit Tests')
        step('Integration Tests')
    }
    stage('Deploy') {
        step('Kubernetes Deploy')
    }

```



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.

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 update
stable has been added to your repositories
...
Helm v3.8.0 (crosbymichael), built on Jan 17 2024 at 07:47:47
Using default storage medium: disk
...
Successfully got an update from the "prometheus-community" chart repository
...
Update complete! Happy Helming!
ubuntu@ip-10-0-1-72: ~
```

Install the prometheus

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

```
ubuntu@ip-10-0-1-72: ~ helm install stable prometheus-community/kube-prometheus-stack
NAME: stable
LAST DEPLOYED: Mon Jan 17 20:15:47 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
TESTS: 0
helm: 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: ~
```

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
kubernetes    ClusterIP   10.100.0.1     <none>         443/TCP       39d
kube-dns      ClusterIP   10.100.101.10   <none>         53/TCP       39d
stable-prttn   ClusterIP   10.100.224.15   <none>         80/TCP       39d
stable-prttn   ClusterIP   10.100.41.82   <none>         80/TCP       39d
stable-prttn   ClusterIP   10.100.41.83   <none>         80/TCP       39d
stable-prttn   ClusterIP   10.100.123.7   <none>         8080/TCP     39d
stable-prttn   ClusterIP   10.100.123.242  <none>         8080/TCP     39d
stable-prttn   ClusterIP   10.100.123.242  <none>         80/TCP       39d
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
```

```
[root@k8s-node-1 ~]# kubectl edit svc stable-kube-prometheus-svc-prometheus
```

Modification in the 48th line from ClusterType to LoadBalancer

```
37 port: 8080
38 protocol: TCP
39 target: http://hub-prometheus:9090
40 - appProtocol: http
41   host: hub-prometheus
42 port: 8080
43 protocol: TCP
44 targetPort: reloader-web
45 sessionAffinity: None
46 sessionAffinityConfig:
47   cookieName: hub-prometheus
48 sessionLabel: hub-prometheus
49 status:
50   lastUpdater: ()
```

Edit the stable-grafana service

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

```
ubuntu@ip-10-0-1-32:~$ kubectl edit svc stable-grafana
```

Modification in the 39th line from ClusterType to LoadBalancer

```
32 port: 80
33 protocol: Tcp
34 selector: <nil>
35 selectorType: <nil>
36 selectorName: <nil>
37 app.kubernetes: <nil>/<instance>: stable
38 app.kubernetes: <nil>/<inner>: grafana
39 selectorAnnotations: <nil>
40 type: LoadBalanced
41 <nil>
42 loadBalancer: ()
```

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

```
kubectl get svc
```

```

ubuntu@ip-10-0-1-71: ~ $ kubectl get svc
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         2d24h
elasticsearch  ClusterIP  10.106.8.1   <none>        9200/TCP,9092/TCP,9094/TCP,9094/UDP  2d24h
kube-dns       ClusterIP  10.96.1.1    <none>        53/TCP          2d24h
stable-grafana ClusterIP  10.106.224.15  <none>        3000/TCP       2d24h
stable-prometheus  stable-elasticsearch  ClusterIP  10.106.81.1   <none>        9090/TCP       2d24h
stable-prometheus  stable-grafana      ClusterIP  10.106.81.1   <none>        3000/TCP       2d24h
stable-prometheus  stable-prometheus  ClusterIP  10.106.88.216  <none>        9090/TCP       2d24h
stable-prometheus  stable-prometheus  ClusterIP  10.106.88.216  <none>        9090/TCP       2d24h
stable-prometheus  stable-prometheus  ClusterIP  10.106.133.242  <none>        9090/TCP       2d24h
ubuntu@ip-10-0-1-71: ~ $
```

You can also validate from your console.



Now, access your Prometheus Dashboard

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



Click on Status and select Target.

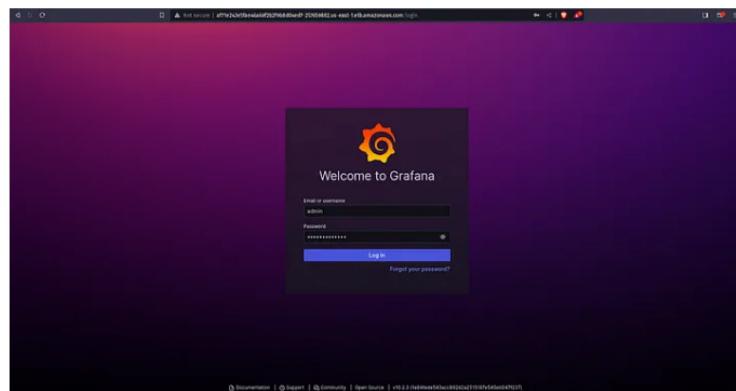
You will see a lot of Targets

A screenshot of the Prometheus Targets page. It shows three sections of targets. The first section has one target: 'serviceMonitor/default/stable-kube-prometheus-etc-externals(0/1 up)'. The second section has two targets: 'serviceMonitor/default/stable-kube-prometheus-etc-externals(1/1 up)' and 'serviceMonitor/default/stable-kube-prometheus-etc-apiservers(0/2 up)'. The third section has two targets: 'serviceMonitor/default/stable-kube-prometheus-etc-apiservers(1/2 up)' and 'serviceMonitor/default/stable-kube-prometheus-etc-controlPlane(2/2 up)'. Each target row includes columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error.

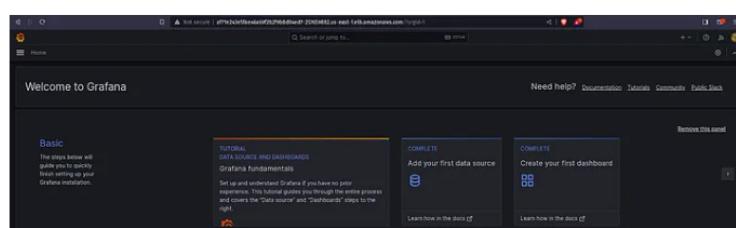
Now, access your Grafana Dashboard

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

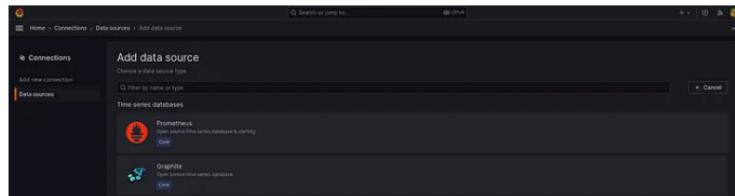
The username will be **admin** and the password will be **prom-operator** for your Grafana LogIn.



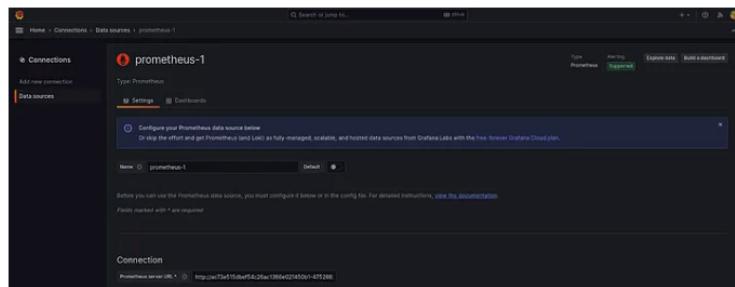
Now, click on Data Source



Select Prometheus

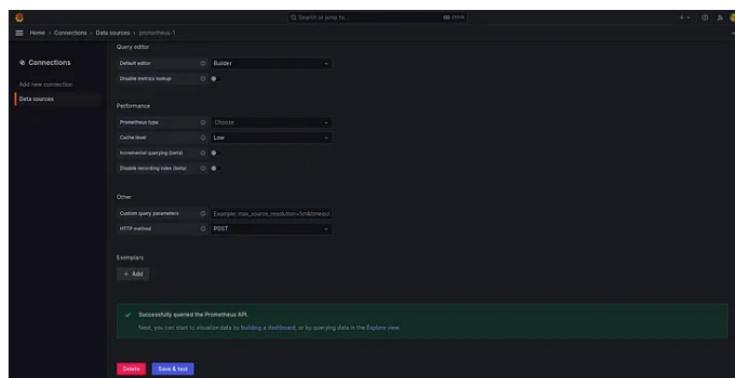


In the Connection, paste your <Prometheus-LB-DNS>:9090.



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

Click on Save & test.

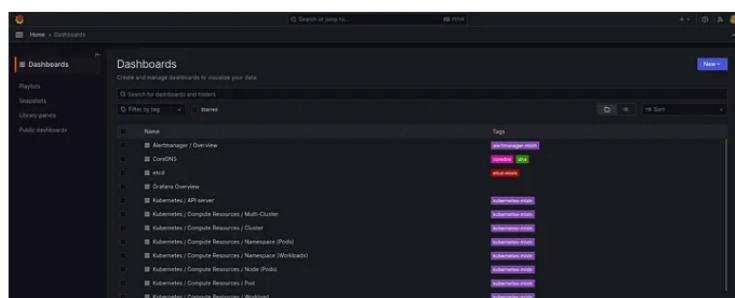


Now, we will create a dashboard to visualize our Kubernetes Cluster Logs.

Click on Dashboard.



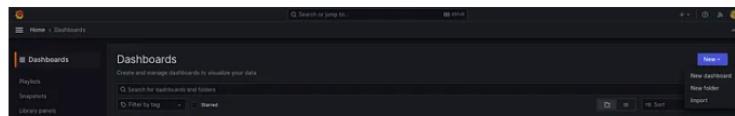
Once you click on Dashboard. You will see a lot of Kubernetes components monitoring.





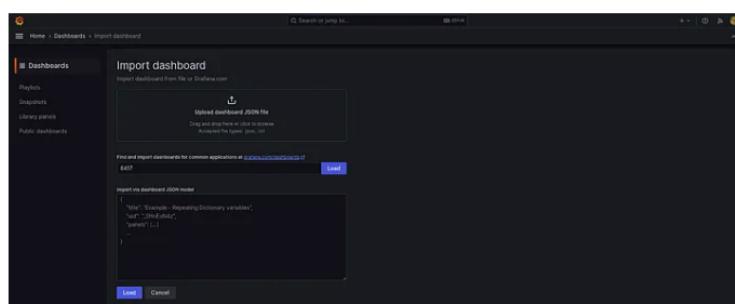
Let's try to import a type of Kubernetes Dashboard.

Click on New and select Import

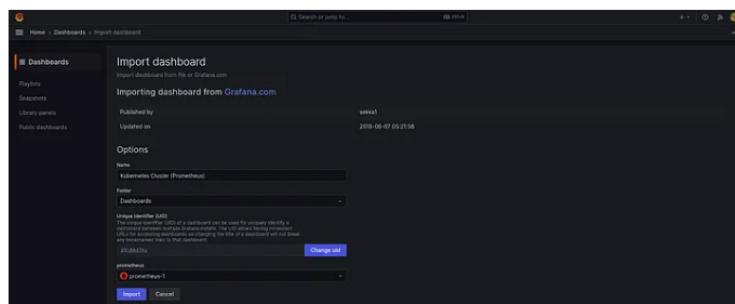


Provide 6417 ID and click on Load

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



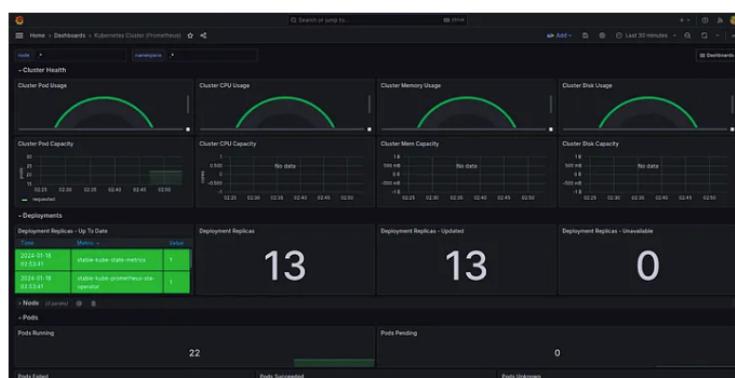
Select the data source that you have created earlier and click on Import.



Here, you go.

You can view your Kubernetes Cluster Data.

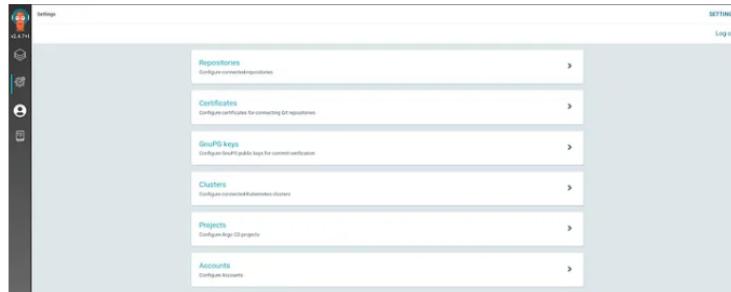
Feel free to explore the other details of the Kubernetes Cluster.



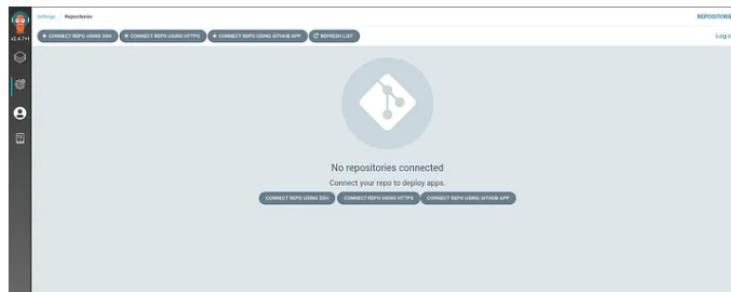
Step 11: We will deploy our Three-Tier Application using ArgoCD.

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

Click on **Settings** and select **Repositories**

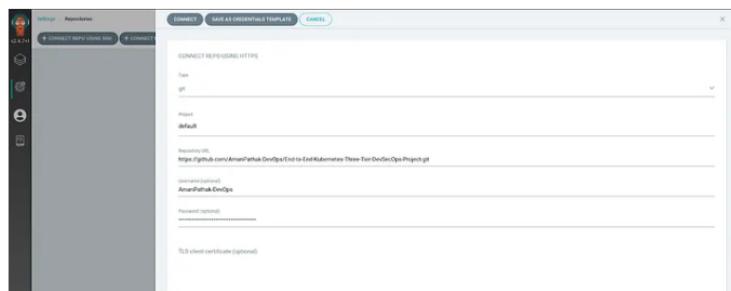


Click on CONNECT REPO USING HTTPS



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

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



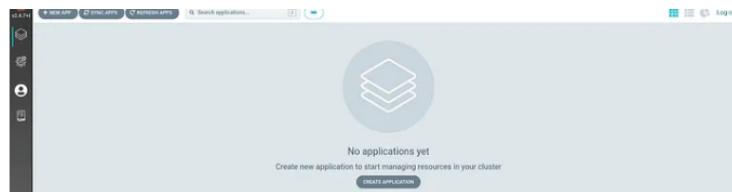
If your Connection Status is Successful it means repository connected successfully.



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

Click on CREATE APPLICATION.





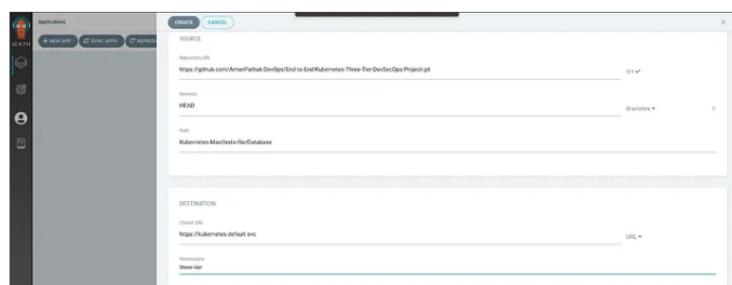
Provide the details as it is provided in the below snippet and scroll down.



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.

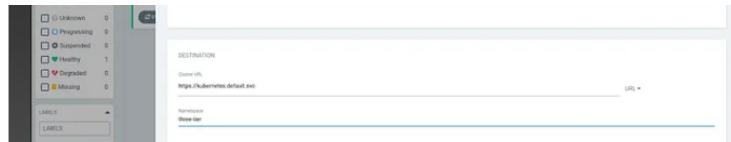


Select the same repository that you configured in the earlier step.

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.

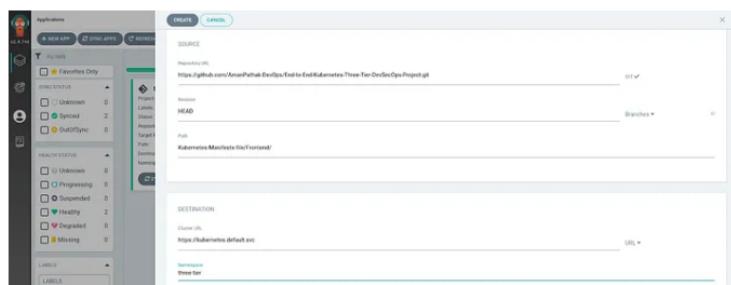
Provide the details as it is provided in the below snippet and scroll down.



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

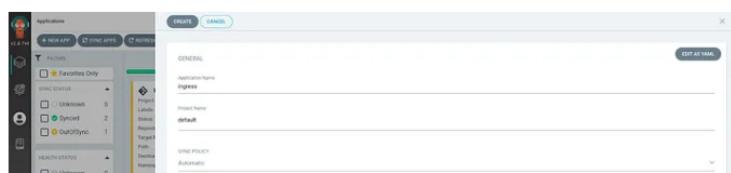
In the Path, provide the location where your Manifest files are presented and provide other things as shown in the below screenshot.

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.

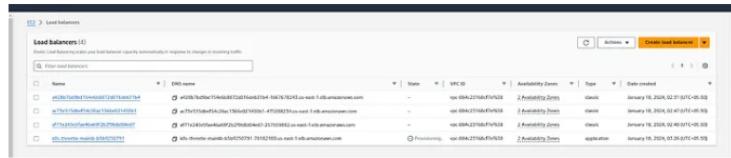
Click on CREATE.





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

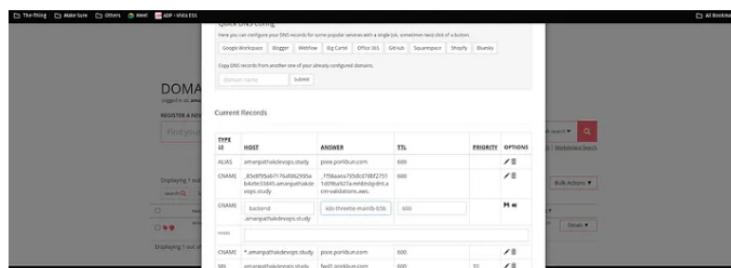
You can check out the load balancer named with k8s-three.



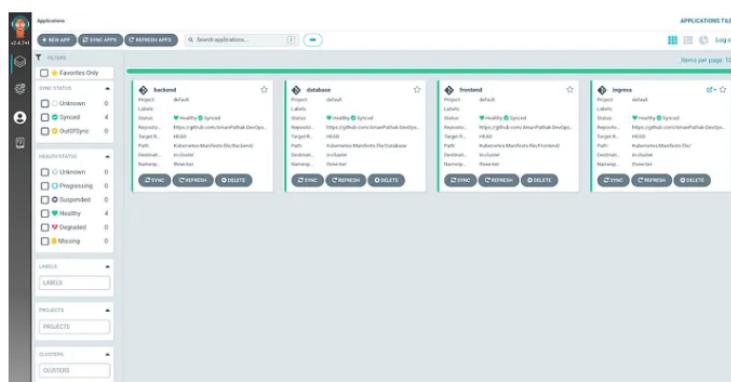
Now, Copy the ALB-DNS and go to your Domain Provider in my case porkbun is the domain provider.

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

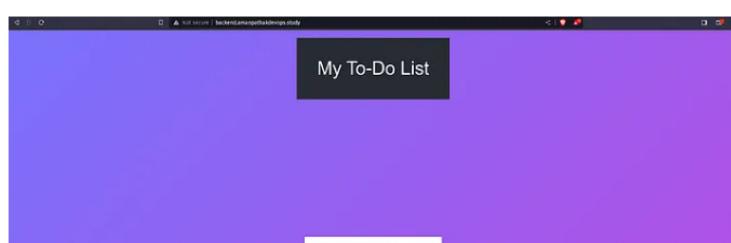
Note: I have created a subdomain backend.amanpathakdevops.study



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

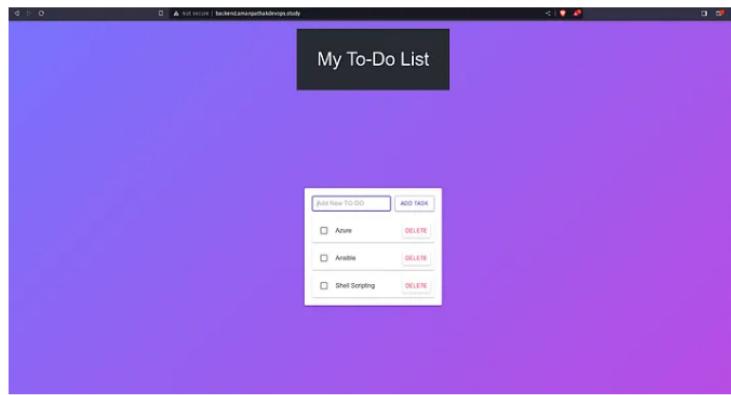


Now, hit your subdomain after 2 to 3 minutes in your browser to see the magic.

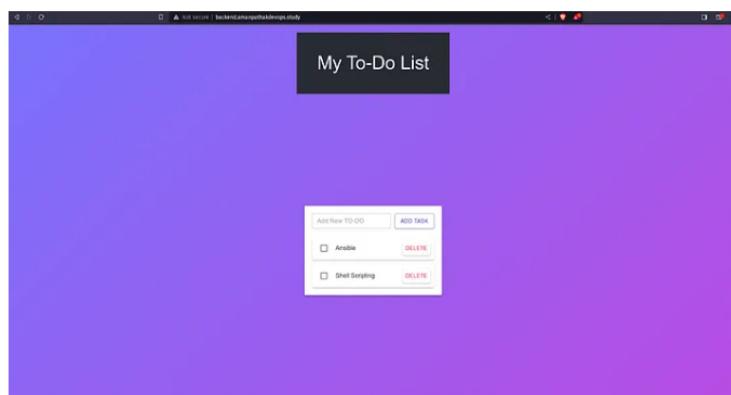




You can play with the application by adding the records.



You can play with the application by deleting the records.



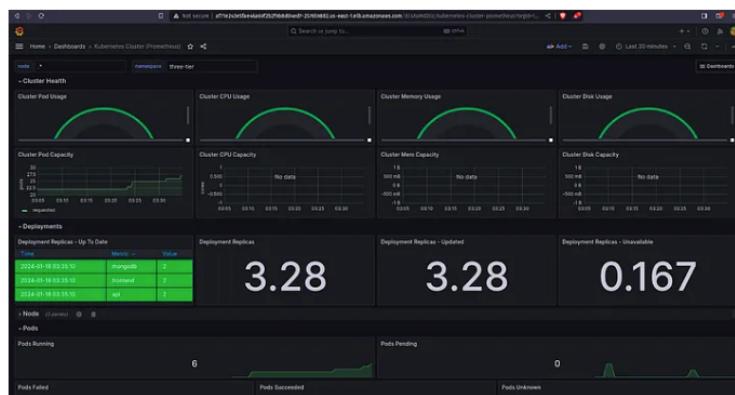
Now, you can see your Grafana Dashboard to view the EKS data such as pods, namespace, deployments, etc.



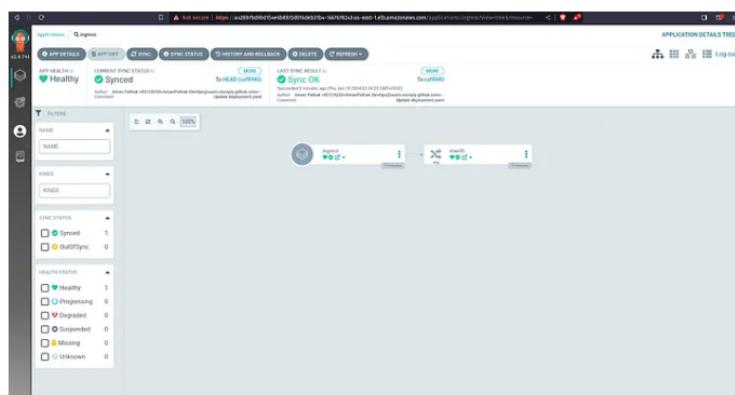
If you want to monitor the three-tier namespace.

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

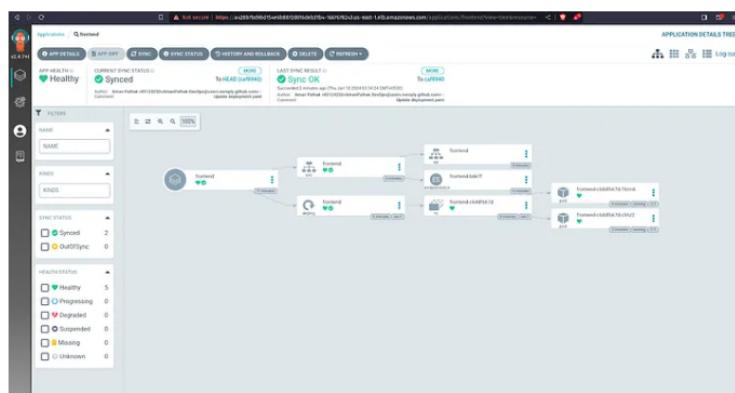
You will see the deployments that are done by ArgoCD



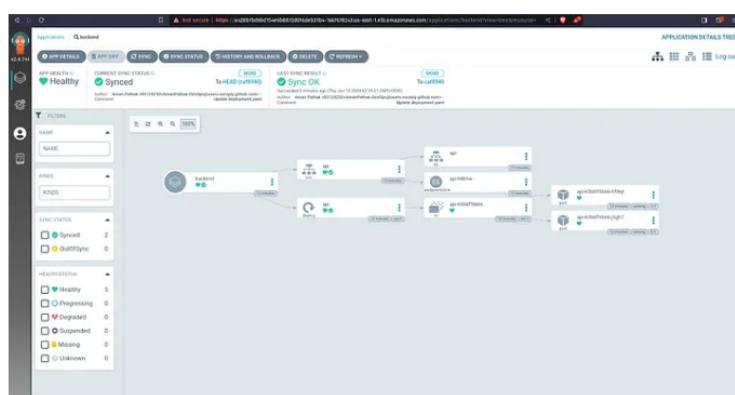
This is the Ingress Application Deployment in ArgoCD



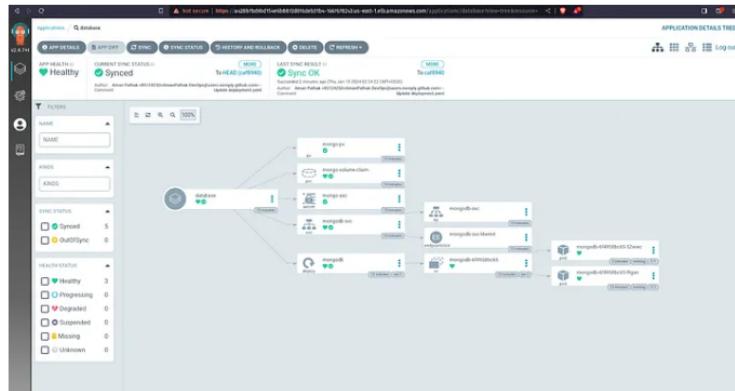
This is the Frontend Application Deployment in ArgoCD



This is the Backend Application Deployment in ArgoCD

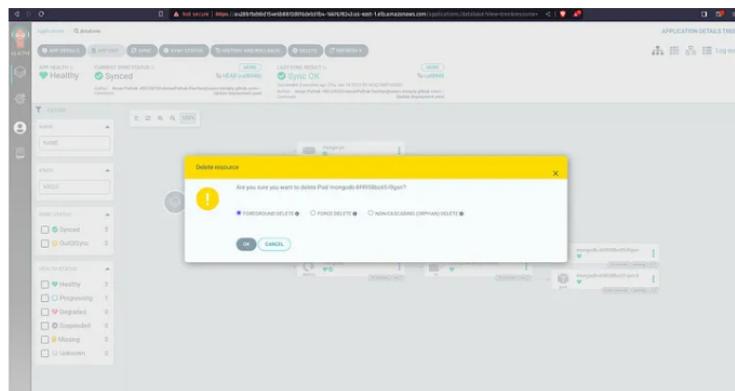


This is the Database Application Deployment in ArgoCD

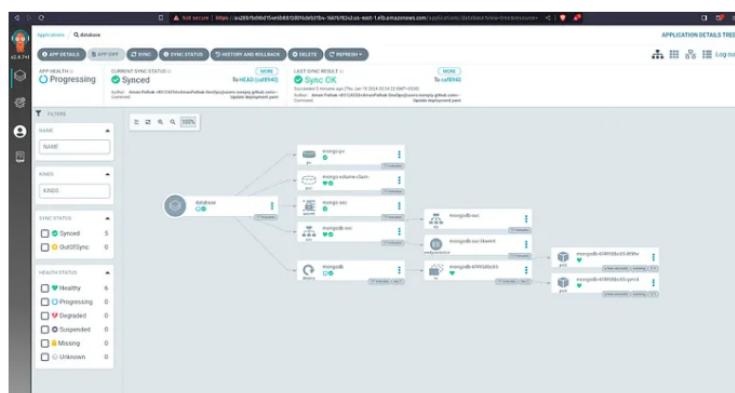


If you observe, we have configured the Persistent Volume & Persistent Volume Claim. So, if the pods get deleted then, the data won't be lost. The Data will be stored on the host machine.

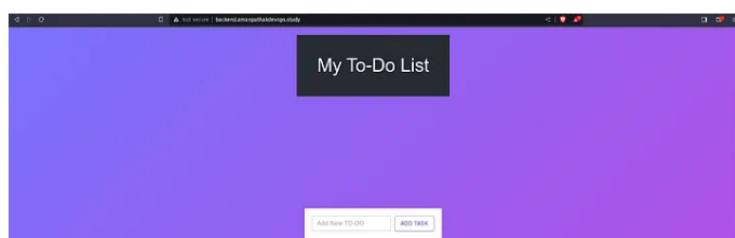
To validate it, delete both Database pods.

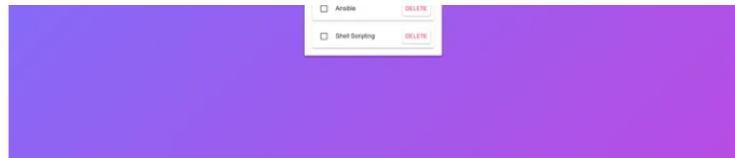


Now, the new pods will be started.



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.

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

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

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

958 10

Bookmark Share More



Written by Aman Pathak

3.6K Followers · Writer for Stackademic

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

Follow

Message

More from Aman Pathak and Stackademic



Aman Pathak in DevOps.dev

End-to-End DevSecOps Kubernetes Project

End-to-End DevSecOps Kubernetes Project

25 min read · Dec 30, 2023

427 6

...



Tomas Svojanovsky in Stackademic

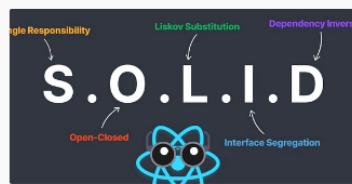
I Failed as a Lead Developer. What I've Learned?

Navigating the Transition: Lessons Learned on the Journey from Senior Developer to Le...

4 min read · Dec 25, 2023

1.6K 25

...



ismail harmanda in Stackademic

Frontend Masters: Solid Principles in React / React Native

11 min read · Jan 13

792 9

...



Aman Pathak in Stackademic

DevSecOps Mastery: A Step-by-Step Guide to Deploying Tetris on...

End-to-End Kubernetes Project using Kubernetes, AWS EKS, Terraform, Jenkins,...

16 min read · Jan 12

217

...

See all from Aman Pathak

See all from Stackademic

Recommended from Medium



Aman Pathak in Stackademic

DevSecOps Mastery: A Step-by-Step Guide to Deploying Tetris on...

End-to-End Kubernetes Project using Kubernetes, AWS EKS, Terraform, Jenkins,...

16 min read · Jan 12

217 6

...

Seifeddine Rajhi

Most common mistakes to avoid when using Kubernetes: Anti...

A simple guide to avoid these Pitfalls 🚧

10 min read · 4 days ago

330 6

...

Lists

General Coding Knowledge

20 stories · 839 saves

Productivity

237 stories · 296 saves

Natural Language Processing

1128 stories · 598 saves

 Jacob Bennett in Level Up Coding

The 5 paid subscriptions I actually use in 2024 as a software engineer

Tools I use that are cheaper than Netflix

 · 5 min read · Jan 4

 5K  62

  ...

 Guillermo Quiros in ITNEXT

K8Studio Kubernetes IDE

It's been an exhilarating journey since we first embarked on the K8studio project four year...

4 min read · Jan 16

 673  9

  ...

 Henry Ha (Here & Now)

Breakdown of the Digital Banking System Architecture

Mapping the Money Maze: Exploring the Layers of Digital Banking Architecture; #5 in...

 · 7 min read · Jan 7

 333 

  ...

 Talha Şahin

High-Level System Architecture of Booking.com

Hello everyone! In this article, we will take an in-depth look at the possible high-level...

8 min read · Jan 10

 1.7K  13

  ...

[See more recommendations](#)