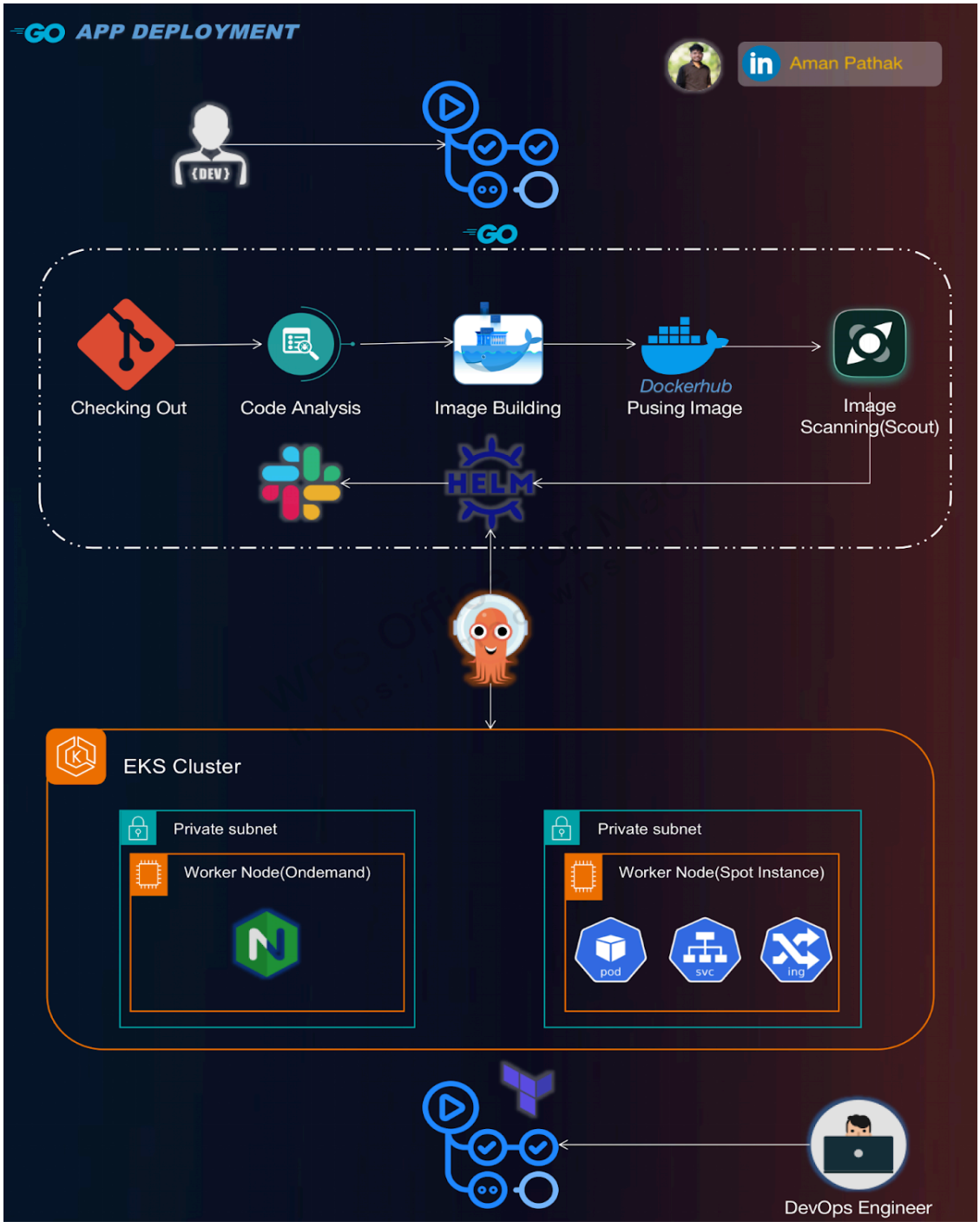
Automate Go App Deployment on Private EKS with **GitHub Actions**, Terraform, Helm, and ArgoCD

- By **Aman Pathak**



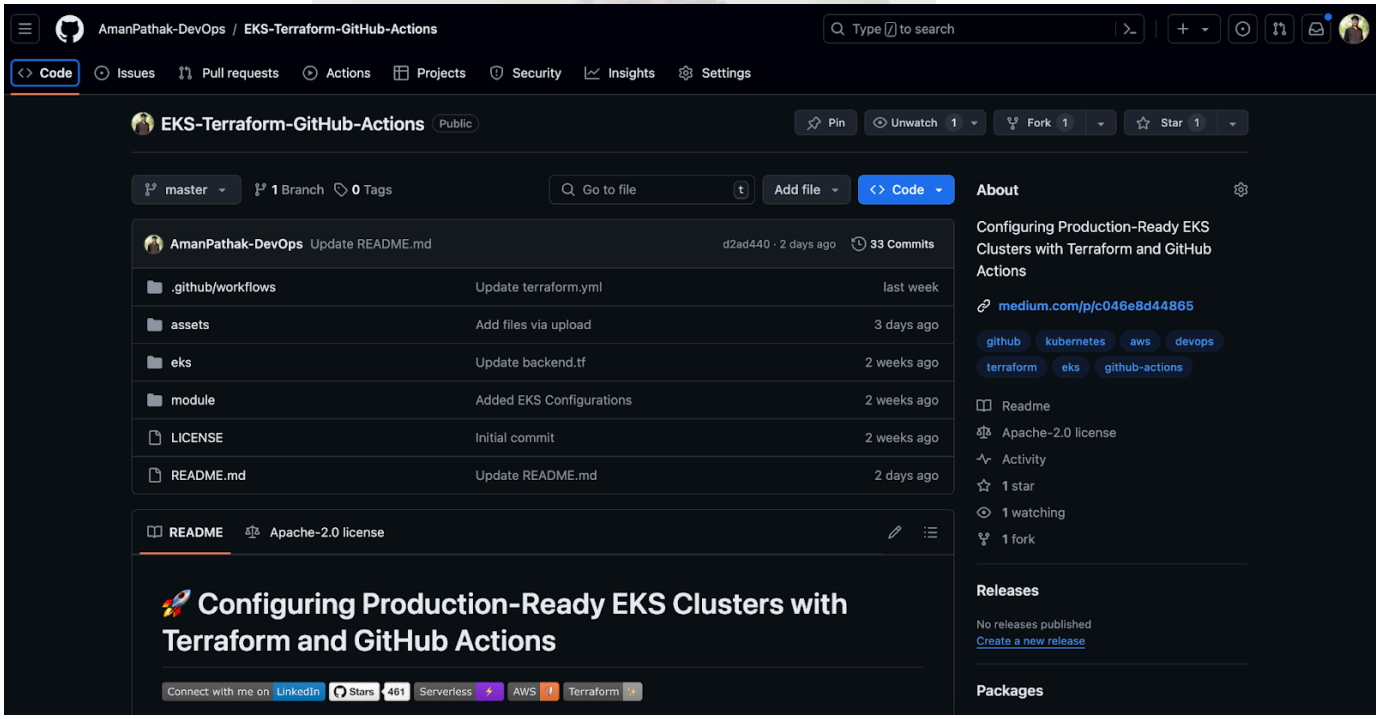
So, I have created three repositories for this project.

- **EKS-Terraform-GitHub-Actions**- Contains terraform code for EKS cluster and other required services along with GitHub actions workflow to deploy the infrastructure through GitHub actions
  https://github.com/AmanPathak-DevOps/EKS-Terraform-GitHub-Actions/tree/master
- **go-portfolio-project**- Contains source code along with GitHub actions workflow including code analysis, docker image creation, image scanning, image tag updation, etc.
  https://github.com/AmanPathak-DevOps/go-portfolio-project
- **go-app-devops-** Contains helm chart where the manifests are present to deploy it on the EKS Cluster.
  https://github.com/AmanPathak-DevOps/go-app-devops

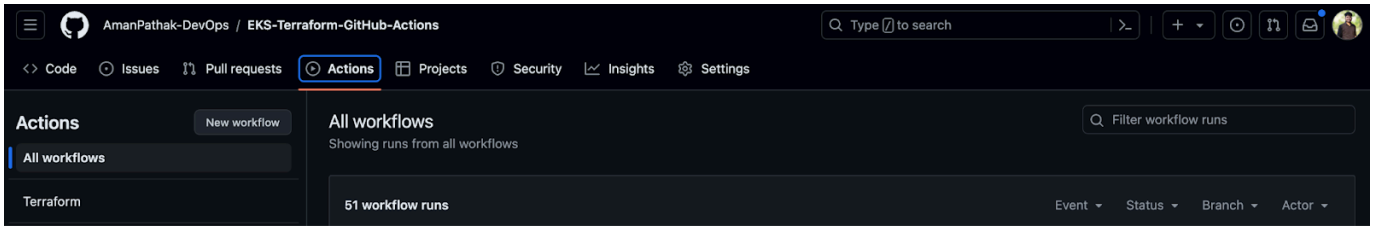
Deploy EKS using GitHub Actions

Repo- https://github.com/AmanPathak-DevOps/EKS-Terraform-GitHub-Actions

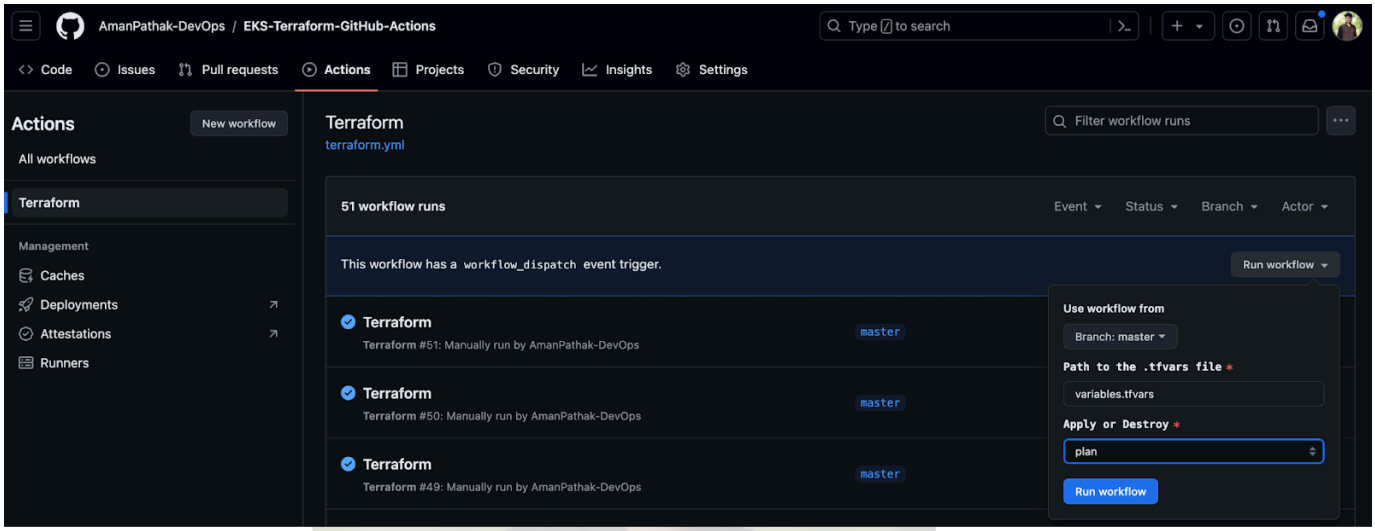
Click on Actions



Click on **Terraform**

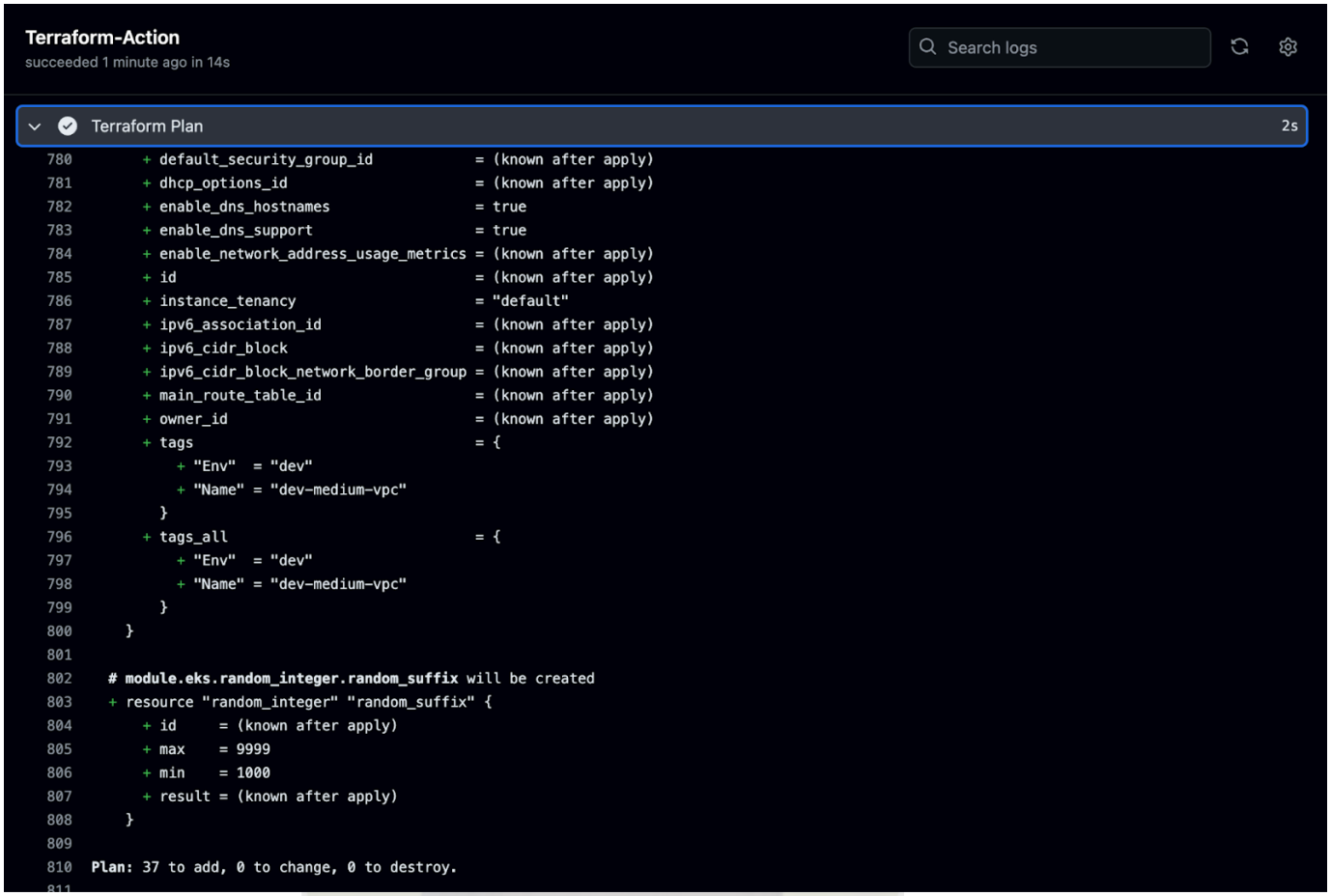


Click on **Run workflow** and run the **plan** to validate what are we going to deploy

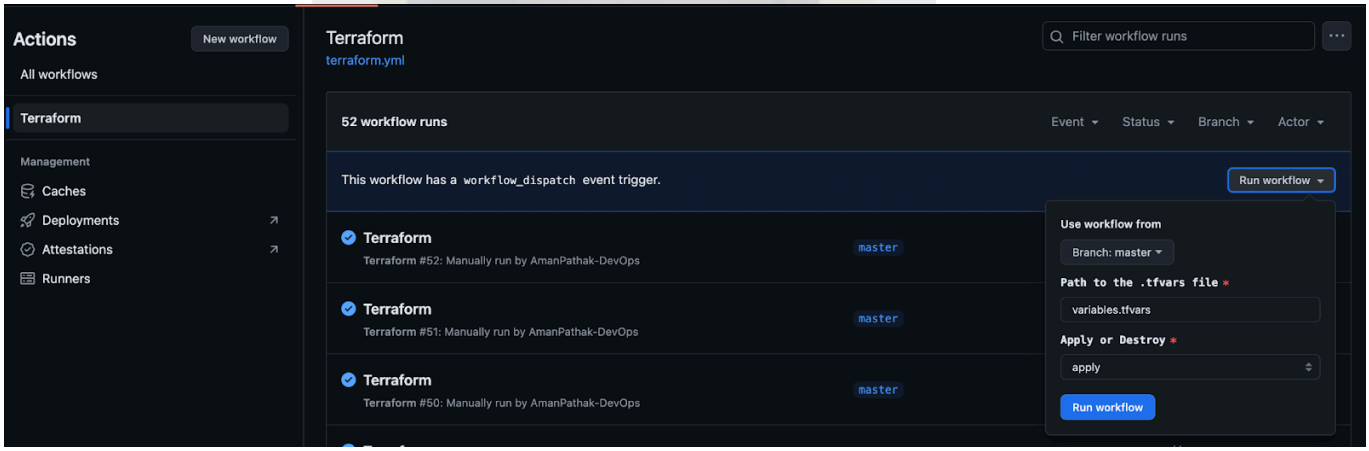


The Plan is successful and you can click on Terraform-Action to view the blueprint of action

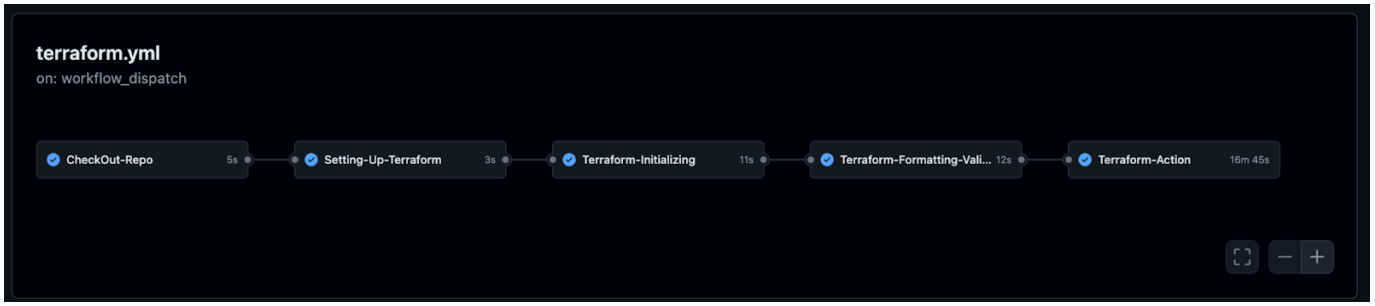
A total 37(repo update 38) AWS resources will be create



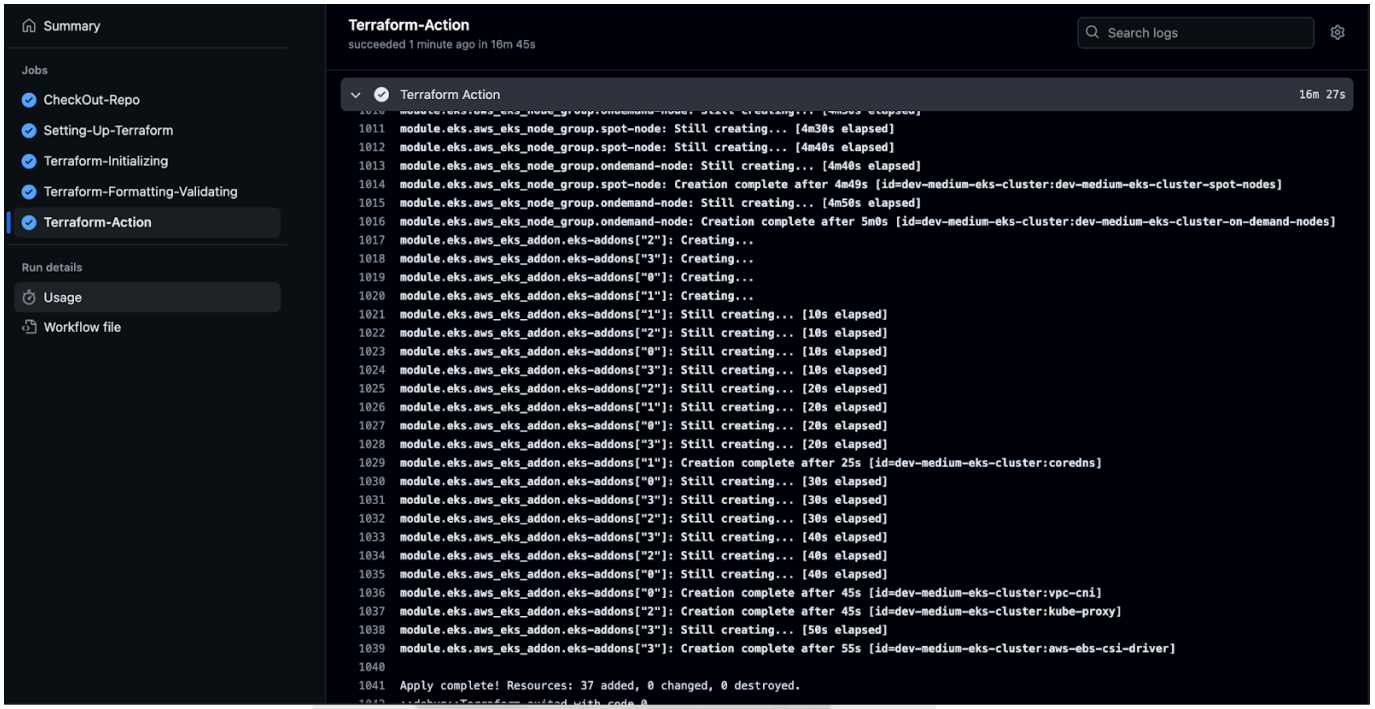
Now, we will run the apply
Go to Terraform workflow and click on **Run workflow**



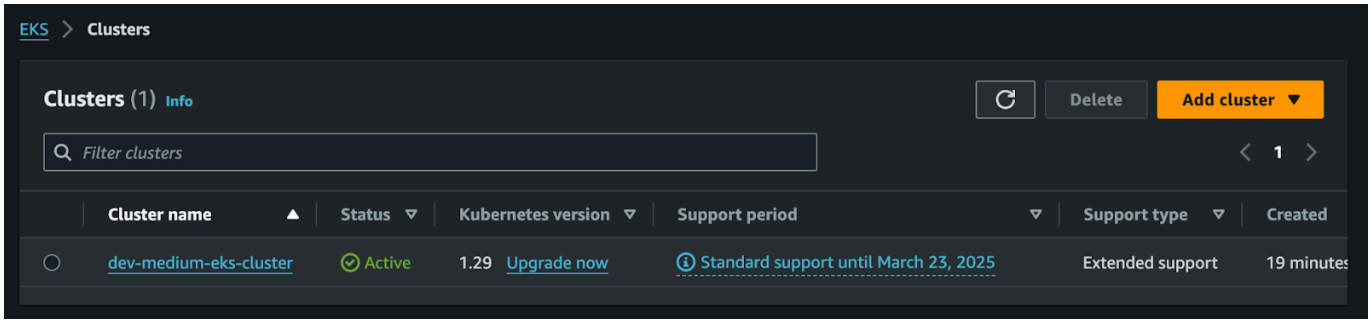
The Apply is successful and you can click on Terraform-Action to view the created resources list



A total of 37(update to 38) AWS resources have been created



You can validate whether the cluster has been created or not by going to the AWS console
Cluster

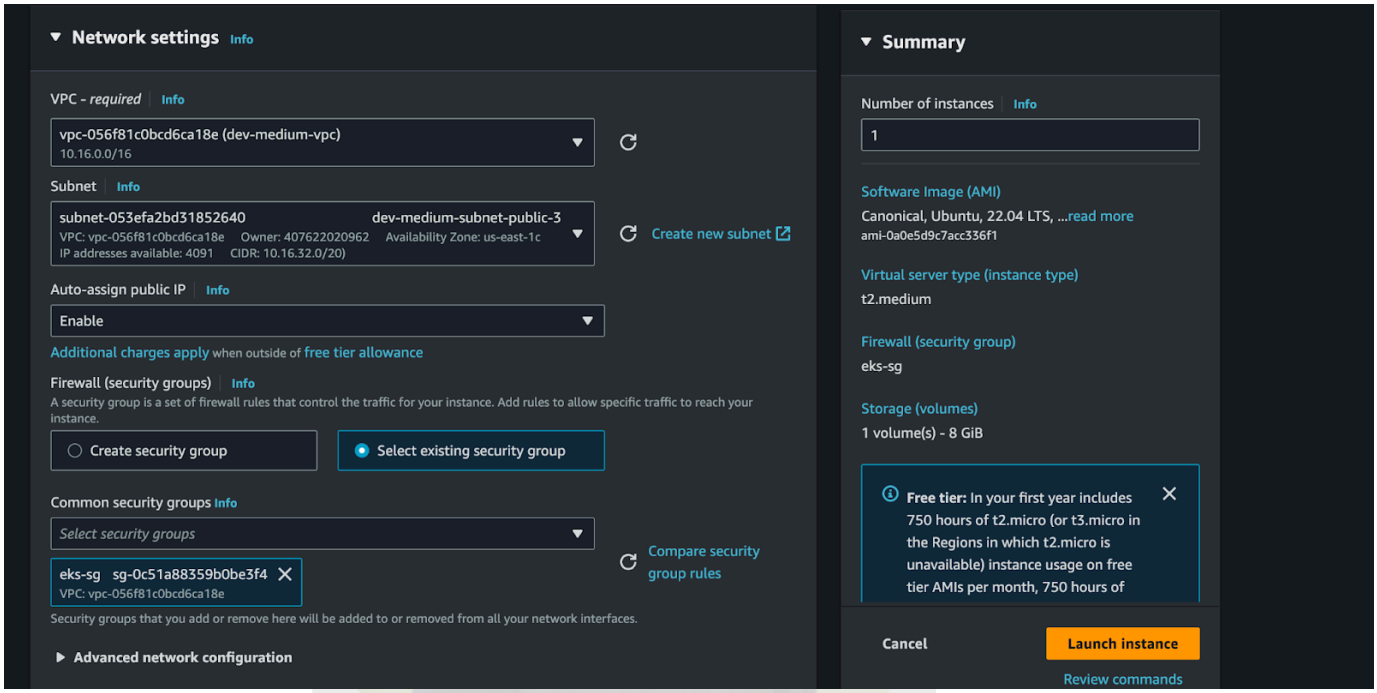


Once you try to connect with your EKS cluster on your local it will add the context.
But while running kubectl commands like get nodes, pods, etc you will get an error
If you look at the snippet, you will observe your local is unable to connect to your server

The reason behind this is that your EKS cluster is Private and your server needs to be in the same VPC as the EKS cluster to access the cluster.

```
aman.pathak@Aman-Pathak ~ % aws eks update-kubeconfig --name dev-medium-eks-cluster --region us-east-1 --profile AmanPathak-DevOps
Added new context arn:aws:eks:us-east-1:407622020962:cluster/dev-medium-eks-cluster to /Users/aman.pathak/.kube/config
aman.pathak@Aman-Pathak ~ % kubectl get nodes

E0801 12:05:51.347798   42034 memcache.go:265] couldn't get current server API group list: Get "https://EC424F3F279368BD6F4627C83C3F1A96.gr7.us-east-1.eks.amazonaws.com/api?timeout=32s": dial tcp 10.16.131.166:443:
i/o timeout
E0801 12:06:21.362274   42034 memcache.go:265] couldn't get current server API group list: Get "https://EC424F3F279368BD6F4627C83C3F1A96.gr7.us-east-1.eks.amazonaws.com/api?timeout=32s": dial tcp 10.16.131.166:443:
```

I have created one more instance and attached the same VPC used in the EKS Cluster.
You can refer to the below snippet



Login to your server and do the following things mentioned below-
Install aws cli, kubectl
Configure aws cli by using the aws configure command and run the kubectl get nodes command

```
ubuntu@ip-10-16-45-215:~$ kubectl get nodes
NAME                                STATUS   ROLES    AGE    VERSION
ip-10-16-166-126.ec2.internal   Ready    <none>   161m   v1.29.3-eks-ae9a62a
ip-10-16-172-139.ec2.internal   Ready    <none>   161m   v1.29.3-eks-ae9a62a
ubuntu@ip-10-16-45-215:~$
```

Install helm which is one of our prerequisites

```
sudo snap install helm --classic
```

```
ubuntu@ip-10-16-45-215:~$ sudo snap install helm --classic
helm 3.15.3 from Snapcrafters installed
ubuntu@ip-10-16-45-215:~$
ubuntu@ip-10-16-45-215:~$ helm version
version.BuildInfo{Version:"v3.15.3", GitCommit:"3bb50bbbdd9c946ba9989fbe4fb4104766302a64", GitTreeState:"clean", GoVersion:"go1.22.5"}
ubuntu@ip-10-16-45-215:~$
```

Install the Nginx ingress controller by using the below command

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.11
.1/deploy/static/provider/aws/deploy.yaml
```

```
ubuntu@ip-10-16-45-215:~$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.11.1/deploy/static/provider/aws/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
ubuntu@ip-10-16-45-215:~$
```

Validate whether your ingress controller pods are running or not. It should be running

```
kubectl get all -n ingress-nginx
```

```
ubuntu@ip-10-16-45-215:~$ kubectl get all -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-ttf4v    0/1     Completed   0          33s
pod/ingress-nginx-admission-patch-b7z74     0/1     Completed   0          33s
pod/ingress-nginx-controller-784997fdc7-dkhxw   1/1     Running     0          33s

NAME                                         TYPE           CLUSTER-IP      EXTERNAL-IP                                                               PORT(S)
                  AGE
service/ingress-nginx-controller             LoadBalancer   172.20.134.63   a39166677c1cc4806a3bb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com   80:30859/
TCP,443:32716/TCP   33s
service/ingress-nginx-controller-admission   ClusterIP      172.20.1.202    <none>                                                                    443/TCP
                  33s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           33s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-784997fdc7   1         1         1       33s

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           5s         33s
job.batch/ingress-nginx-admission-patch    1/1           6s         33s
ubuntu@ip-10-16-45-215:~$
```

Now, we need to install argoCD as per our project-required tool

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

```
ubuntu@ip-10-16-45-215:~$ kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
namespace/argocd created
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
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-server created
```

Validate whether argocd pods are running or not by using the below command

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

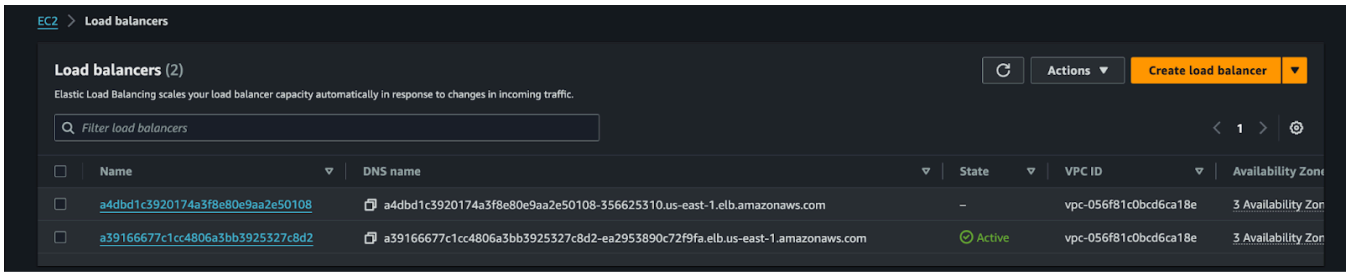
```
ubuntu@ip-10-16-45-215:~$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          60s
argocd-applicationset-controller-749957bcc9-rzxxg   1/1     Running   0          60s
argocd-dex-server-868bf8bc97-bc59f                  1/1     Running   0          60s
argocd-notifications-controller-5fff689764-rffrr    1/1     Running   0          60s
argocd-redis-fb66447b8-xvwqr                        1/1     Running   0          60s
argocd-repo-server-794b8857ff-k87tq                 1/1     Running   0          60s
argocd-server-7bc9d78cf4-tsd7s                      1/1     Running   0          60s
ubuntu@ip-10-16-45-215:~$
```

Now, we need to expose our argocd service to the Loadbalancer type as we need to access it outside of our cluster

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

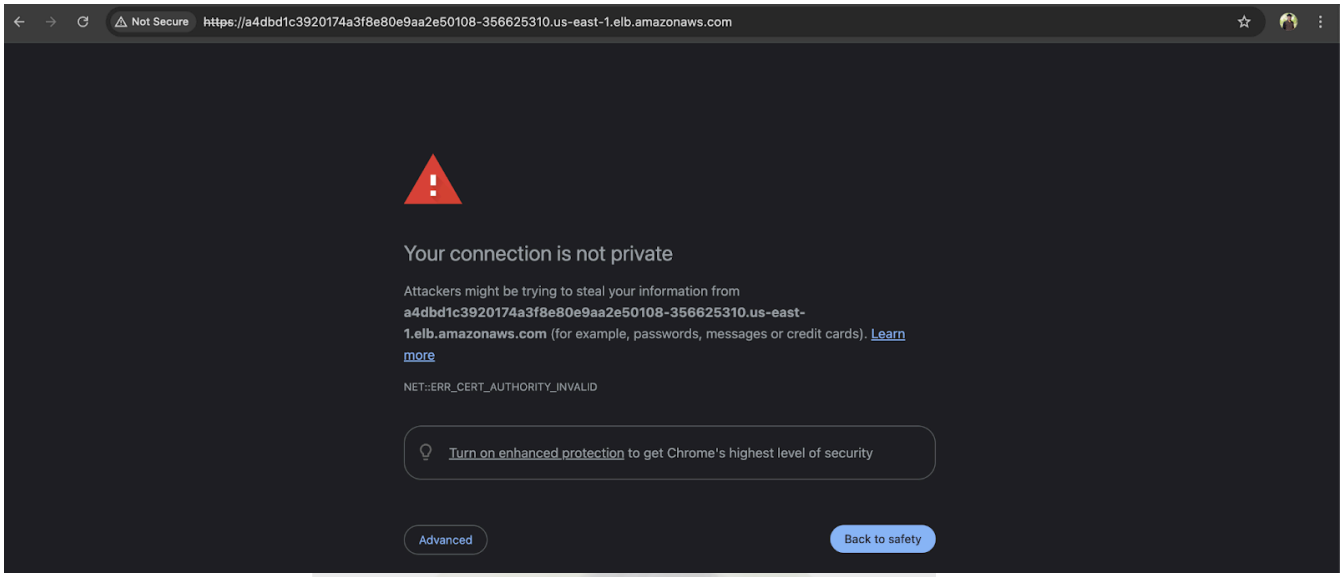
```
ubuntu@ip-10-16-45-215:~$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
service/argocd-server patched
ubuntu@ip-10-16-45-215:~$
```

Validate whether the load balancer has been created or not by going to the AWS console

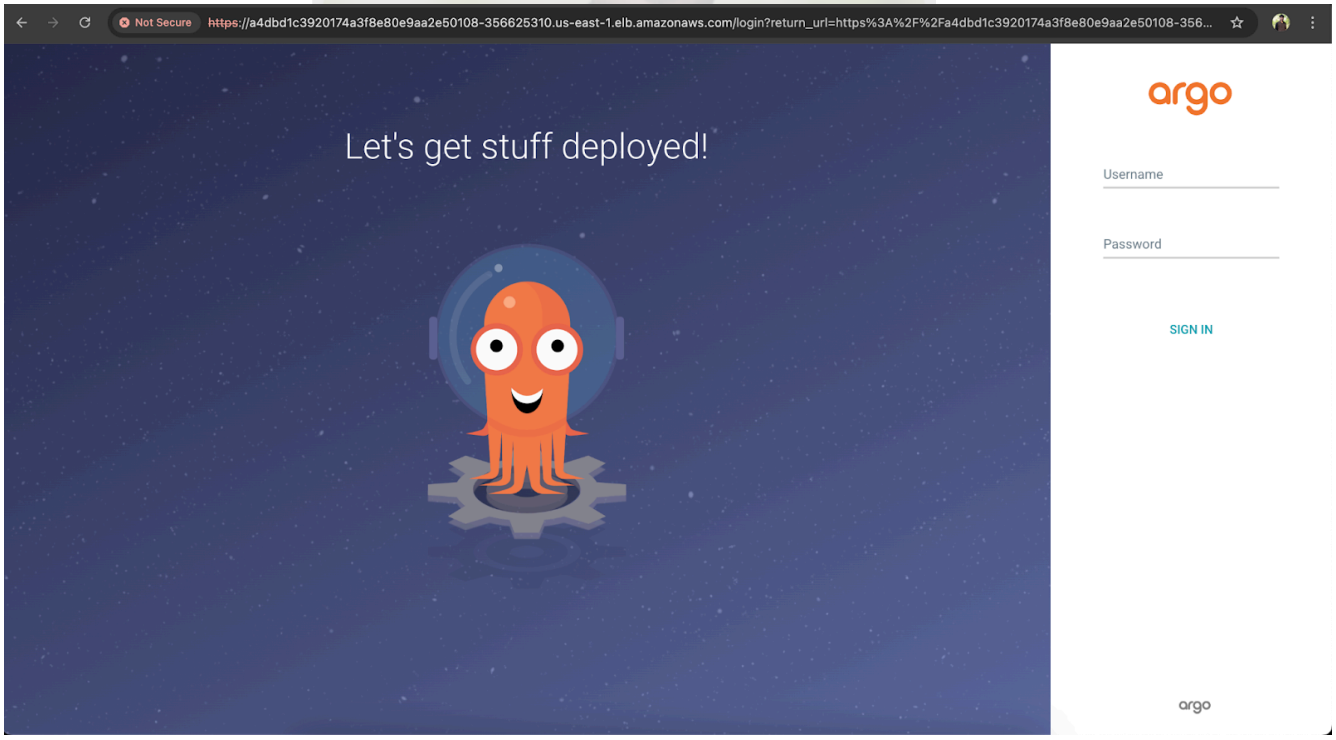


Copy the DNS name of your argocd and hit on your favorite browser

Click on Advanced as SSL is not configured on argoCD



Here is your ArgoCD
Now, the username is admin but we don't know the password.
To get the password, we need to run a few commands in the next steps



Now, we require a password to log in to the argoCD console.
The below command will list the secrets of argocd

```
kubectl get secrets -n argocd
```

```
ubuntu@ip-10-16-45-215:~$ kubectl get secrets -n argocd
NAME                              TYPE     DATA   AGE
argocd-initial-admin-secret       Opaque   1      8m
argocd-notifications-secret       Opaque   0      8m9s
argocd-secret                     Opaque   5      8m9s
ubuntu@ip-10-16-45-215:~$
```

Run the below command and copy the password

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

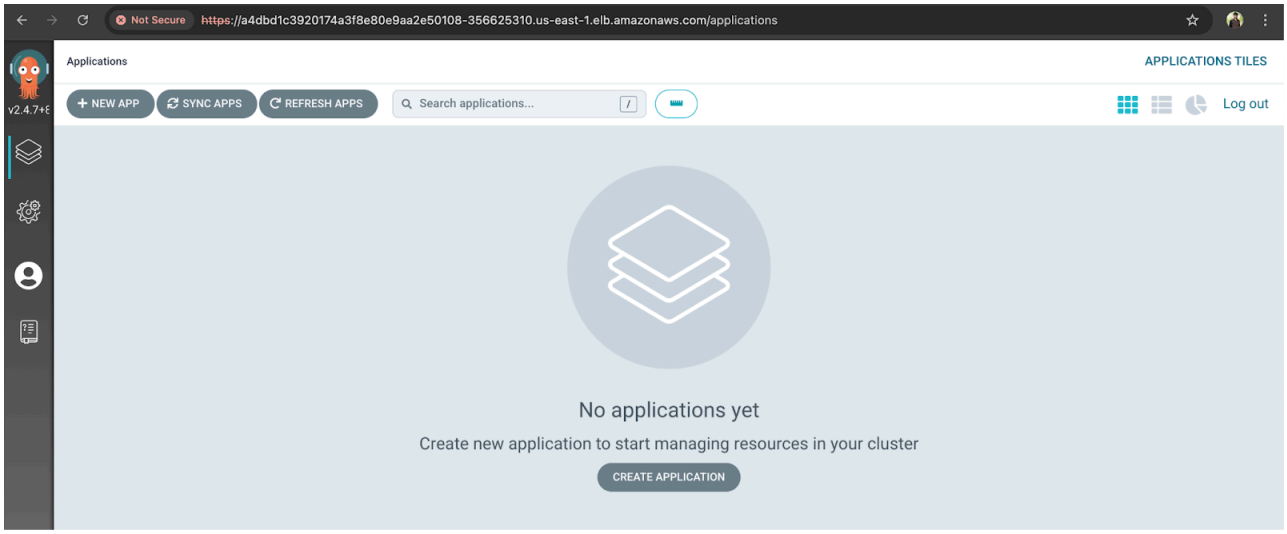
```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  password: VmNMbW9pejYwRmhmajhYQw==
kind: Secret
metadata:
  creationTimestamp: "2024-08-01T14:51:06Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "8011"
  uid: 40d7f05e-f9a5-42e7-91c2-b0963afc9fc9
type: Opaque
~
```

Run the below command to decode the password

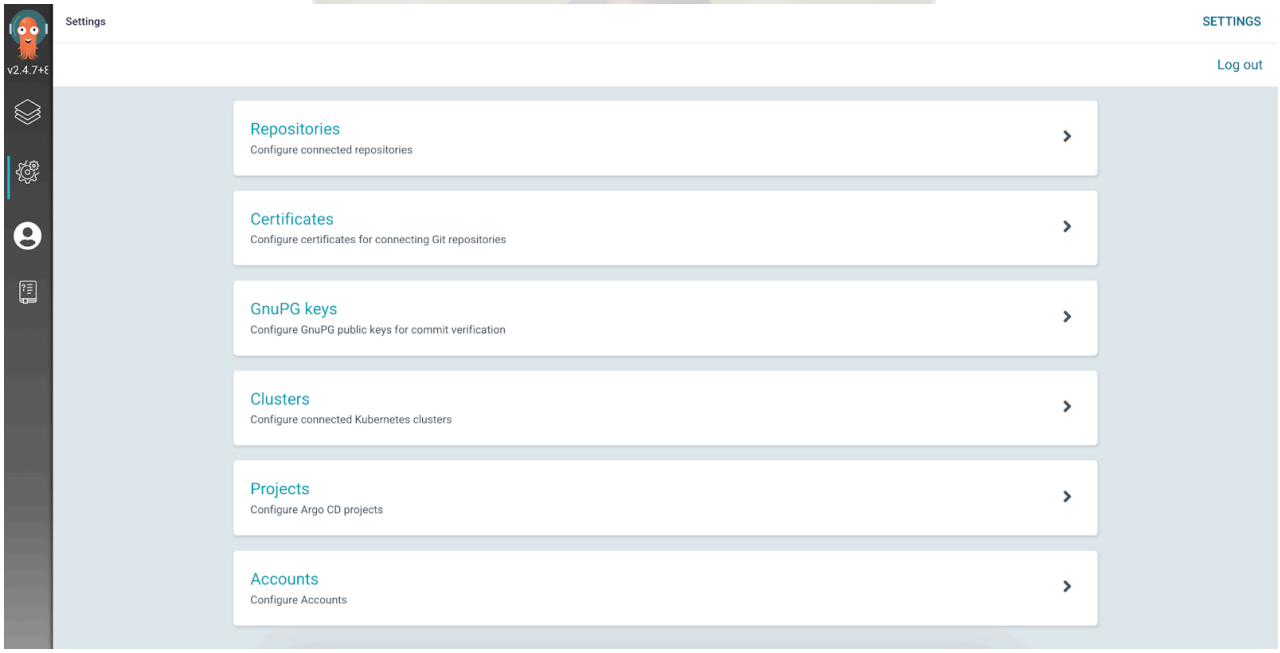
```
echo <your-argocd-password> | base64 --decode
```

```
ubuntu@ip-10-16-45-215:~$ kubectl edit secret argocd-initial-admin-secret -n argocd
Edit cancelled, no changes made.
ubuntu@ip-10-16-45-215:~$
ubuntu@ip-10-16-45-215:~$ echo VmNMbW9pejYwRmhmajhYQw== | base64 --decode
VcLmoiz60Fhfj8XCubuntu@ip-10-16-45-215:~$
```

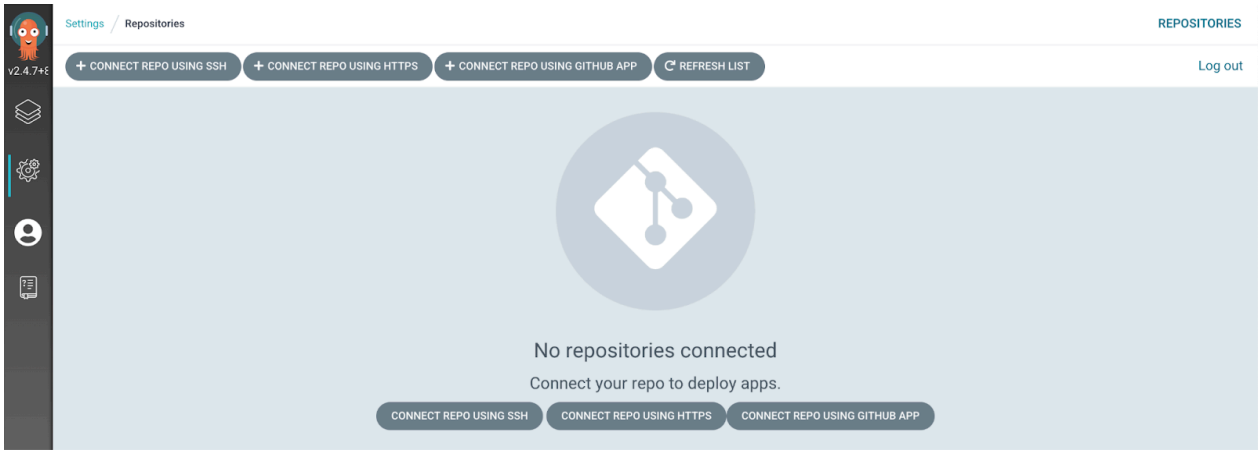
Now, log in to your argoCD console



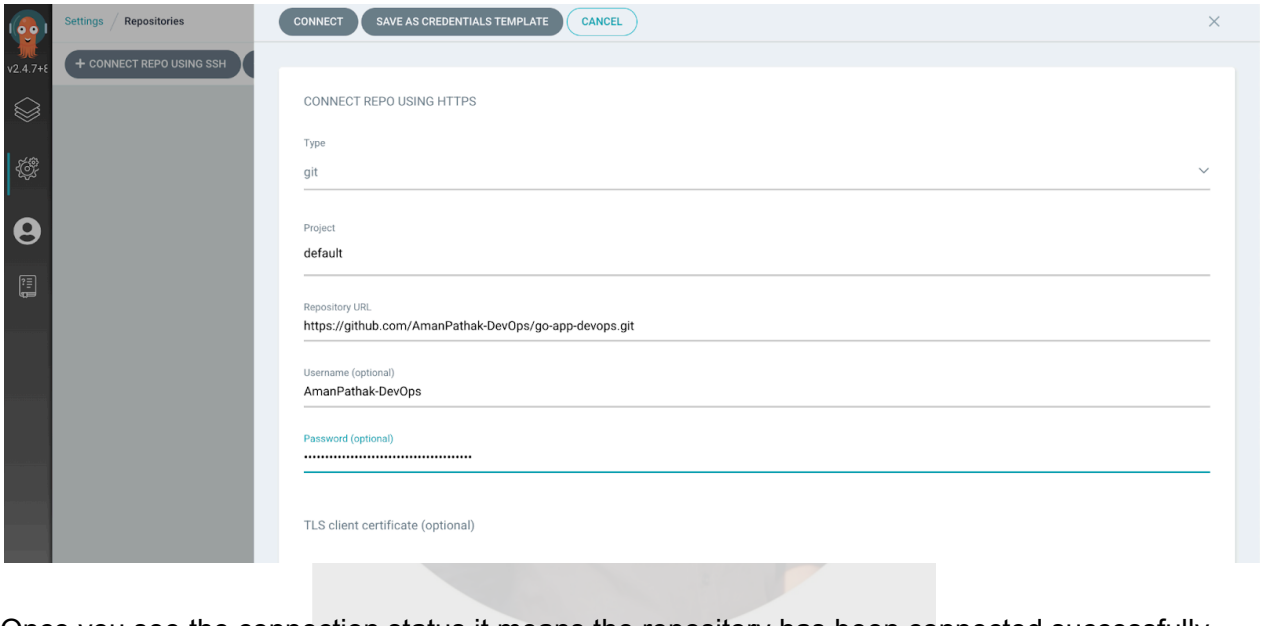
We have set everything to deploy our application
But there is one more thing that is connected to the repository in argoCD
Why are we doing that?
So, our helm chart or manifest file is always stored in a private repository. To clone that repo, we need to set up our repository in argoCD.
Click on the settings icon on the left and then, click on Repositories



For now, we are going to connect the repo using HTTPS which means we need a Personal Access token(PAT)
So, kindly generate the Personal Access token(PAT) of your GitHub account

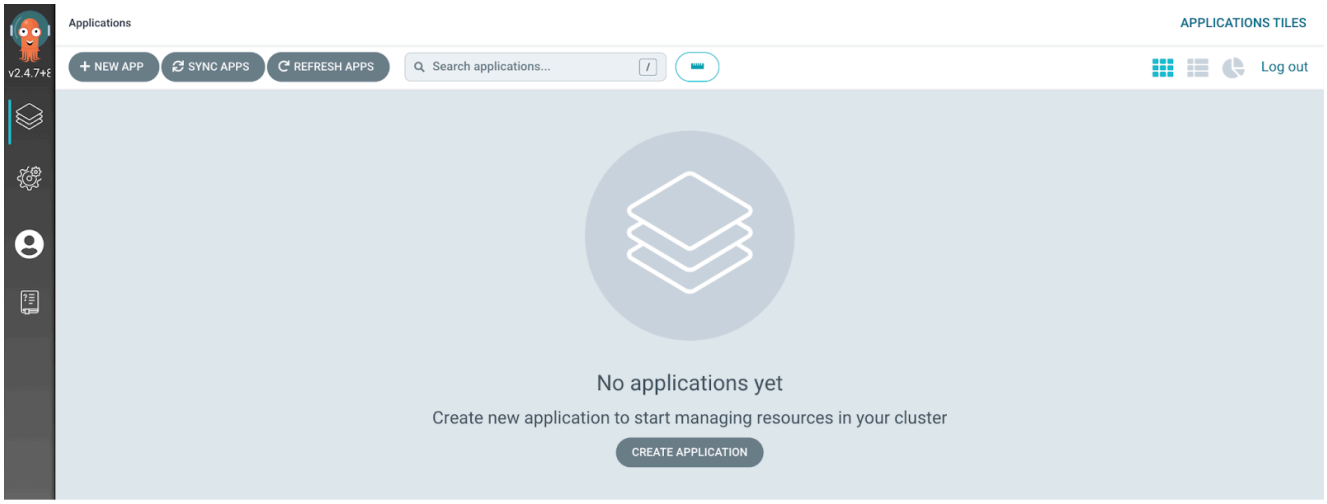


Provide the information accordingly

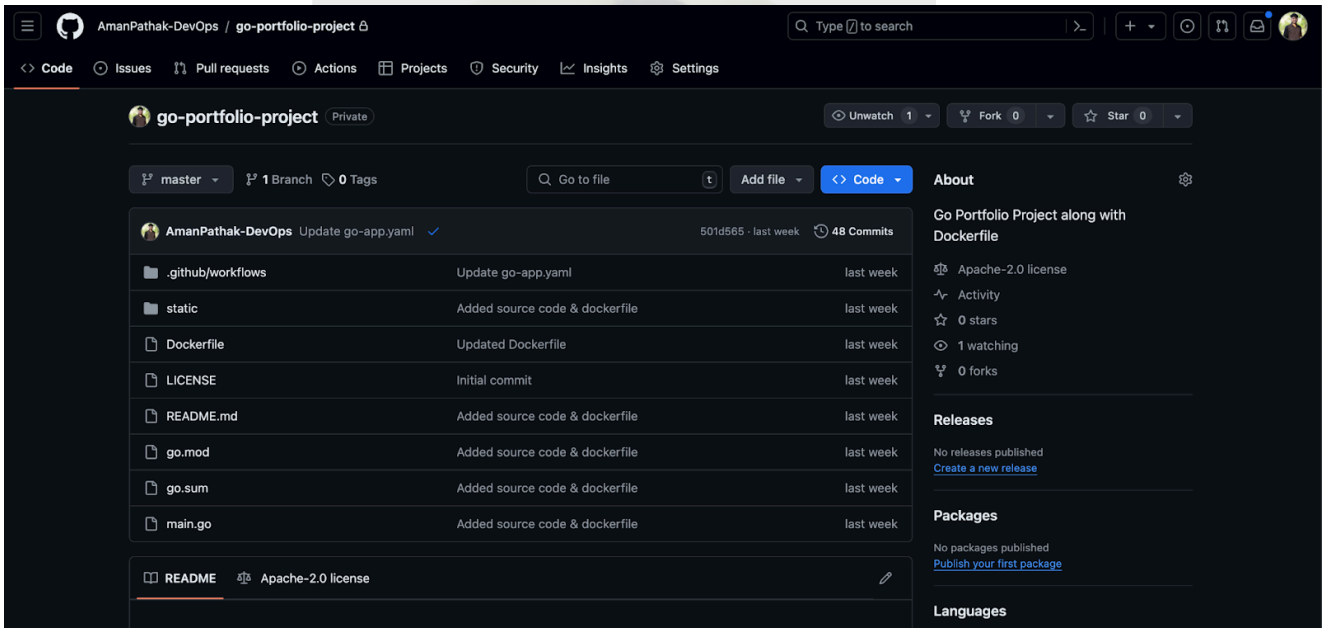


Once you see the connection status it means the repository has been connected successfully

Create an application on argocd



Let's create a workflow for our source code

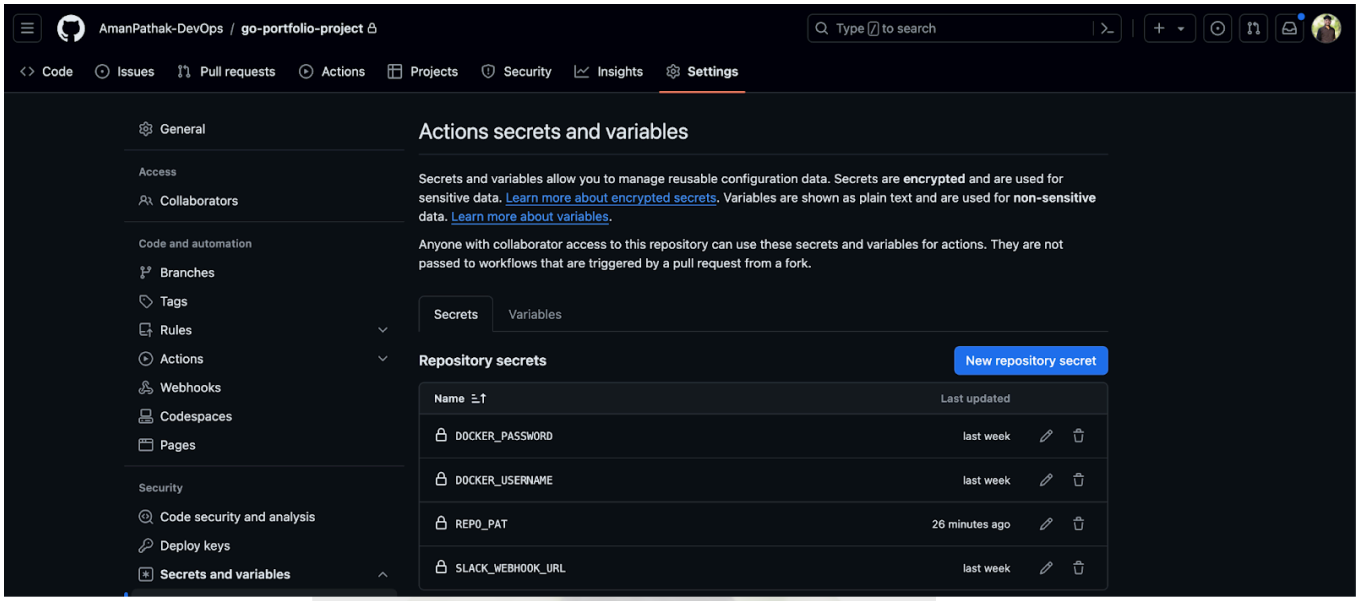


Navigate to Repo's **Settings** -> **Security**(on the left side) -> **Secrets and variables** and click on **Actions**

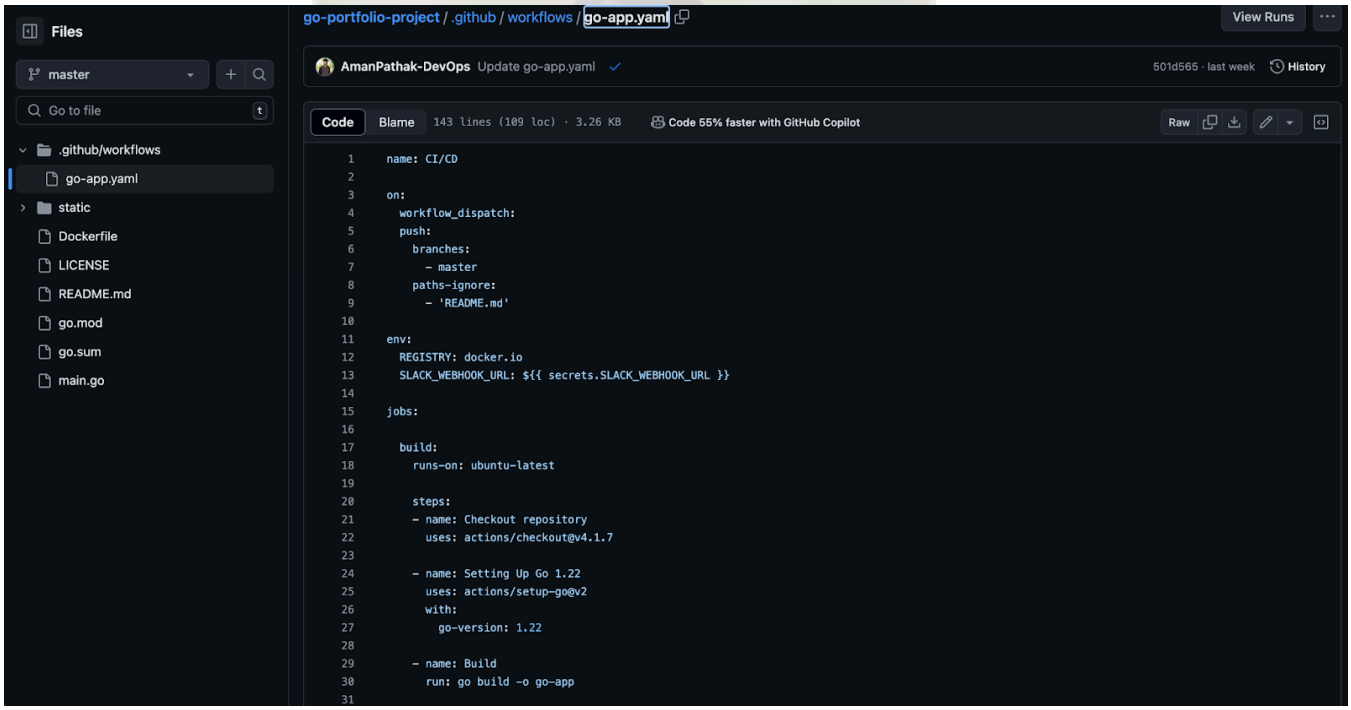
Create required secrets in which docker PAT, docker username, GitHub PAT, and slack webhook URL

Note: I am configuring Slack notification, if you want to do it the same then, refer to this video https://www.youtube.com/watch?v=f8Tr3unIdNw and add a Slack webhook URL as well. But if you don't want to integrate Slack then, you don't need to add the secret. Also, skip the last job in

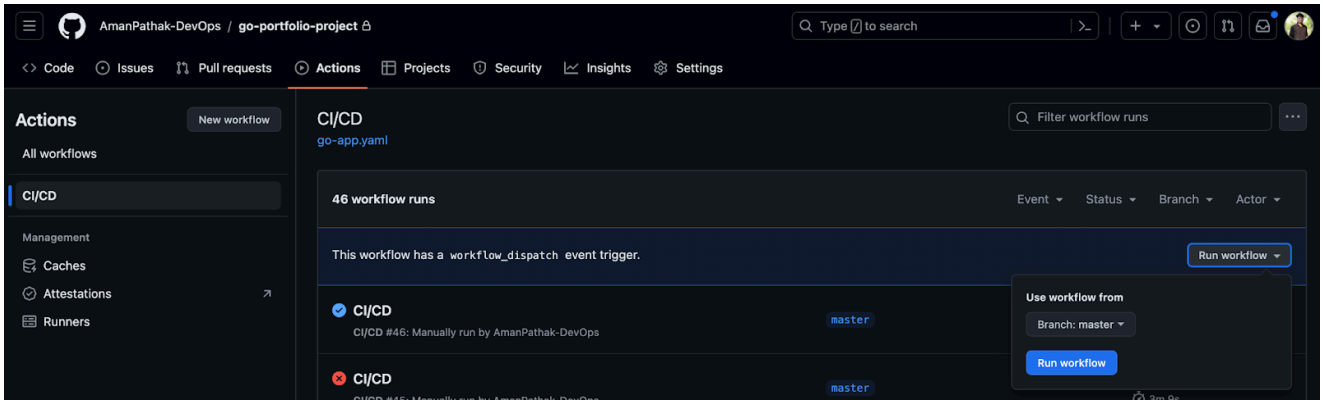
the workflow.



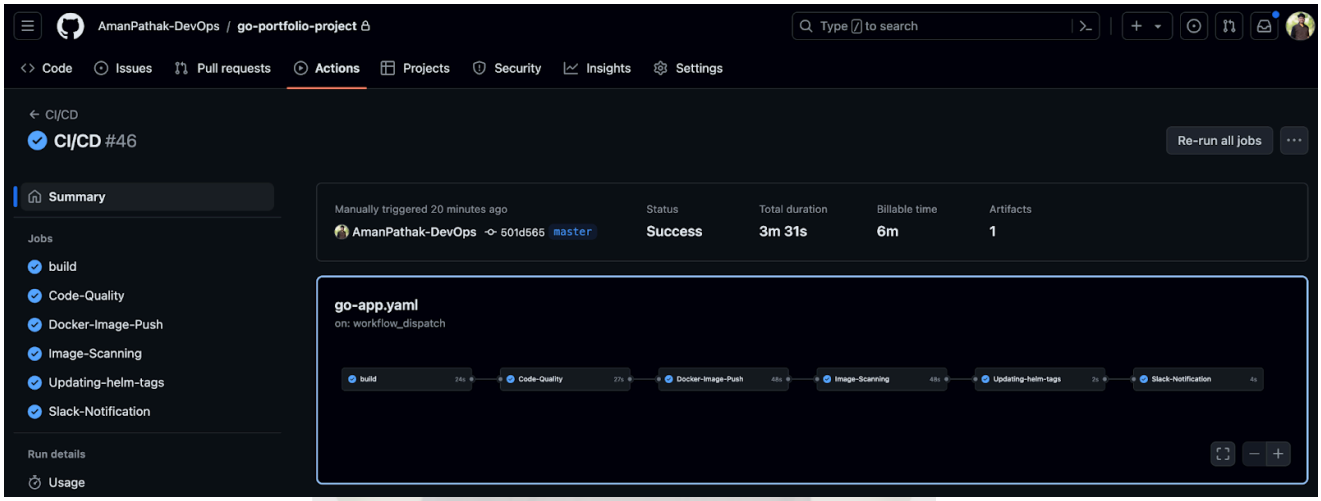
This is our workflow code and you can check it out by clicking on the repo link https://github.com/AmanPathak-DevOps/go-portfolio-project/blob/master/.github/workflows/go-app.yaml



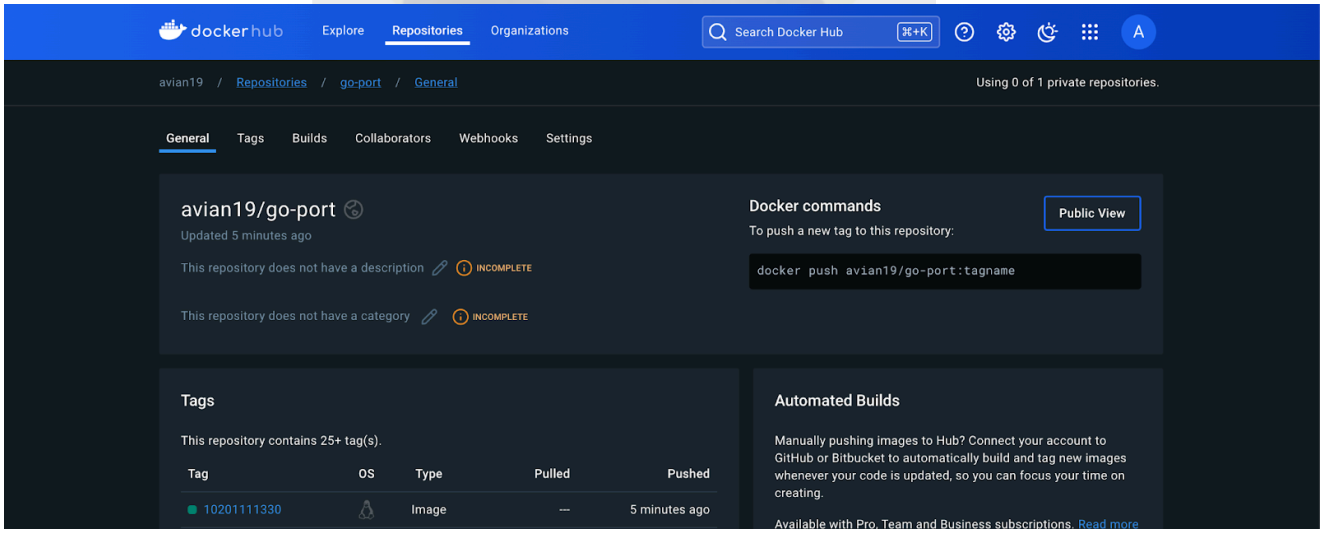
Now, we are set to run our workflow/pipeline



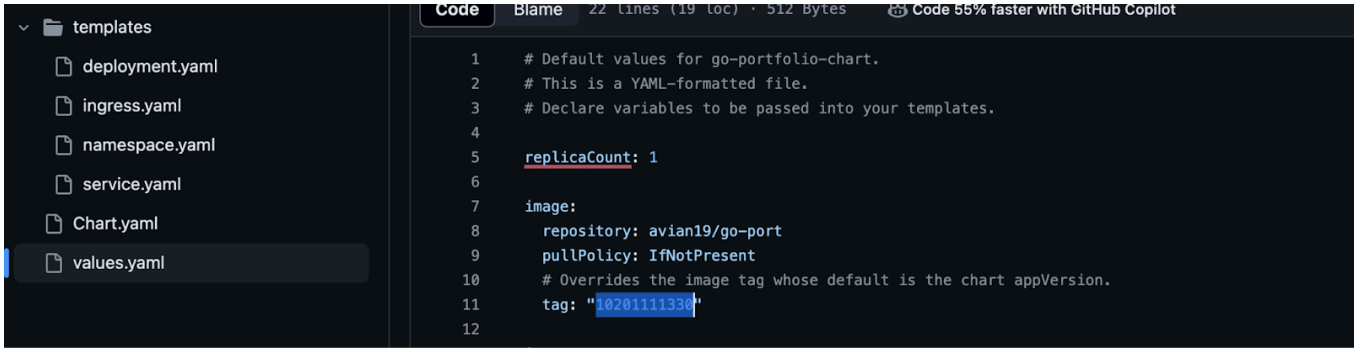
Workflow got successful



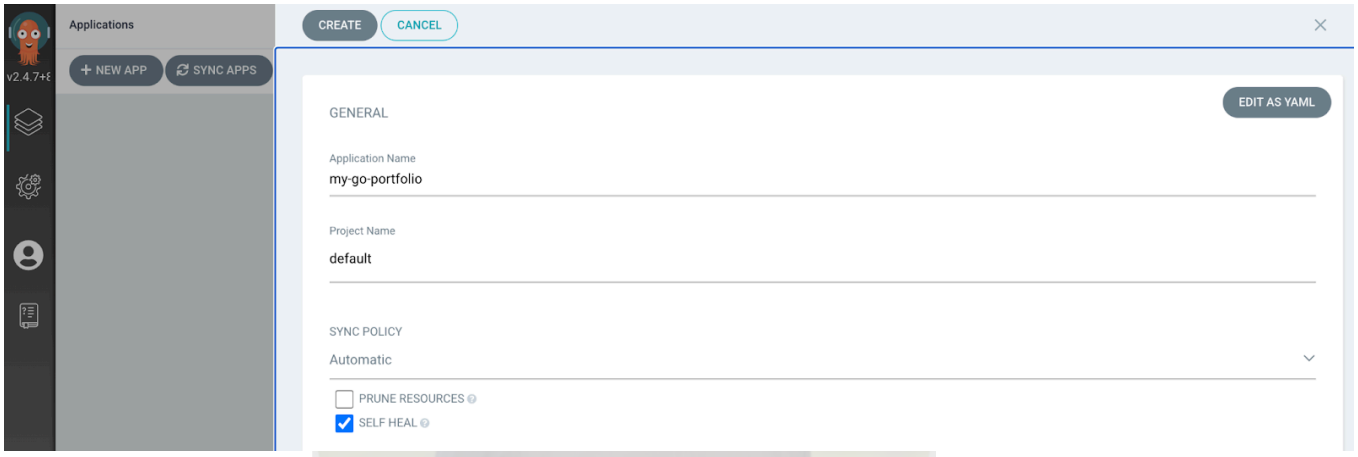
The image has been pushed

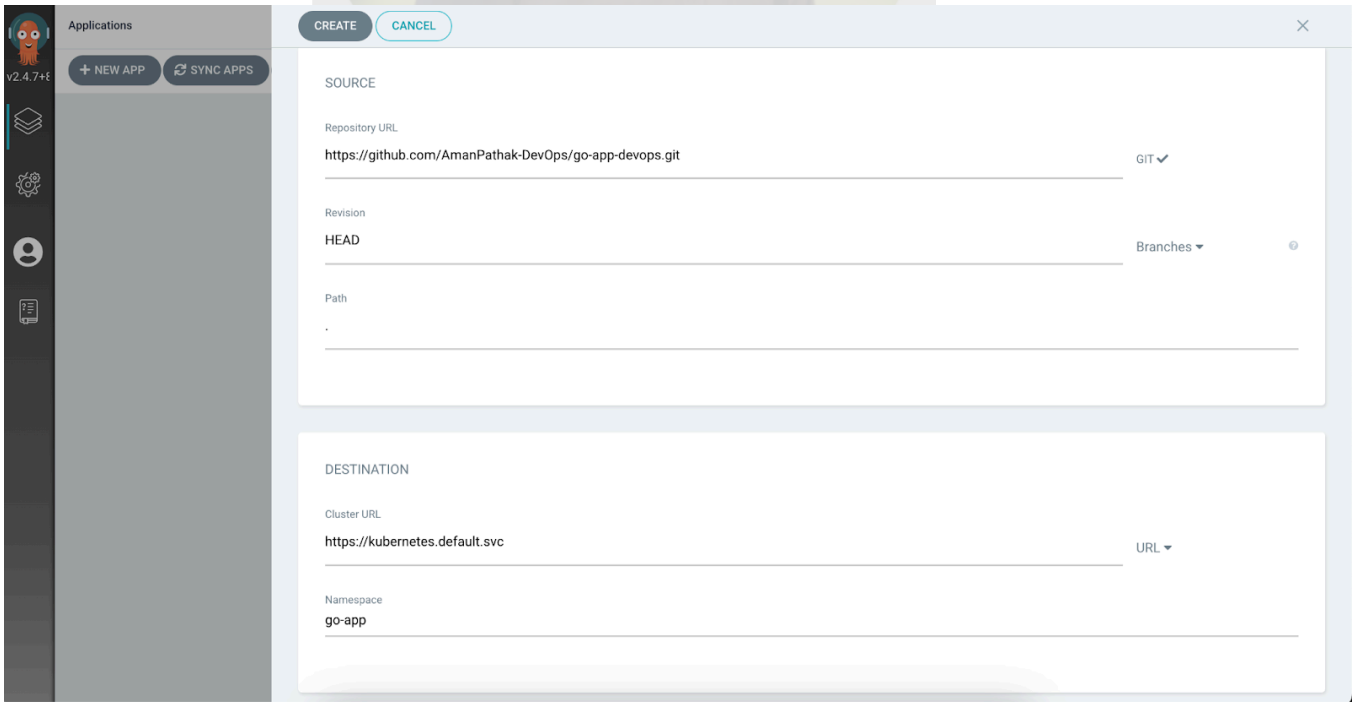


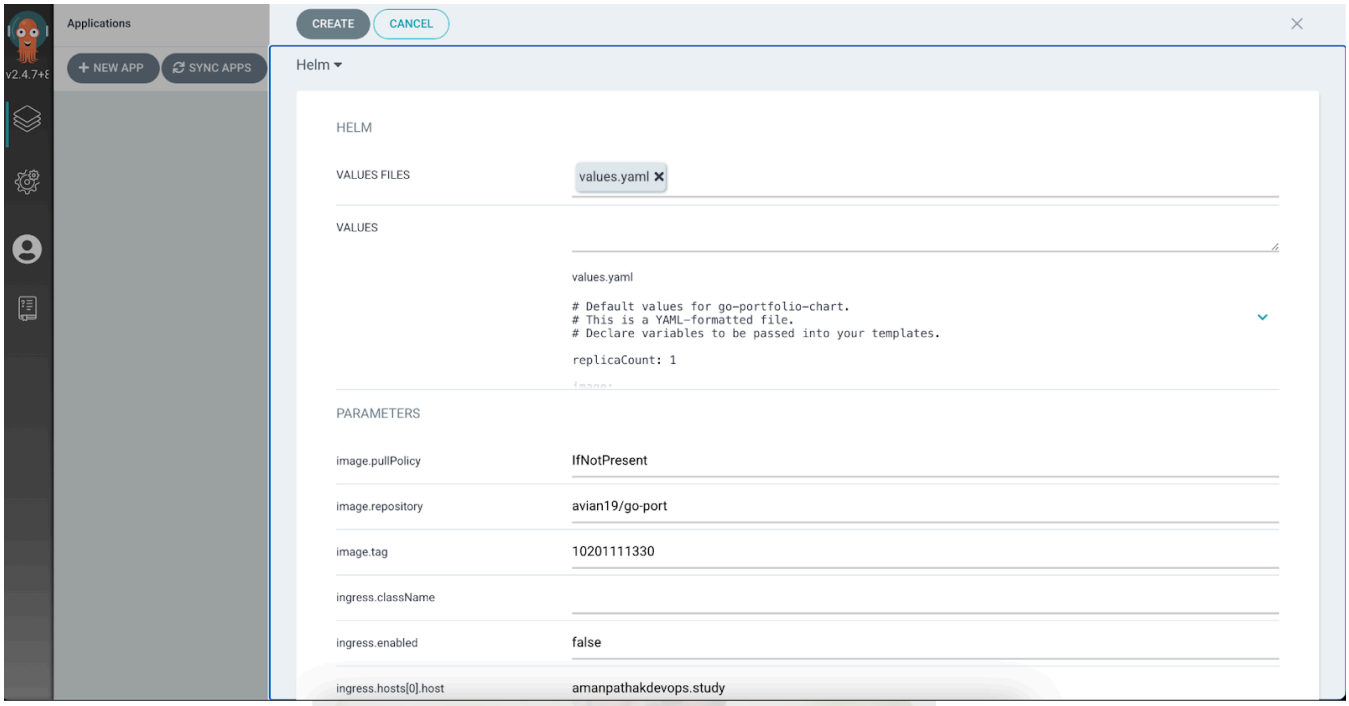
The image tag has been updated in values.yaml file



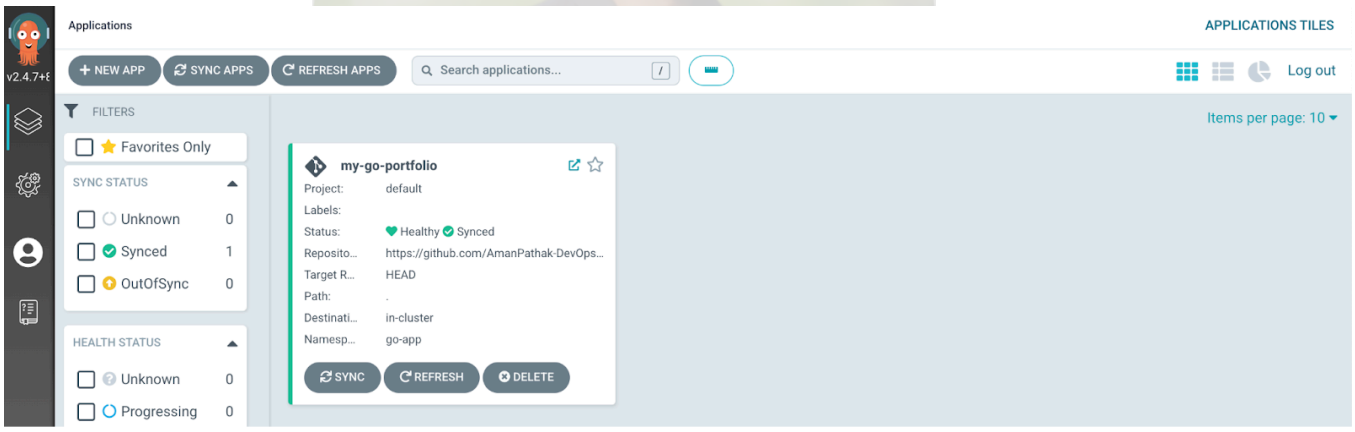
Now, we will create an application on argocd to deploy our application on Kubernetes.
Provide the application name, repo URL, etc as shown in the below snippet



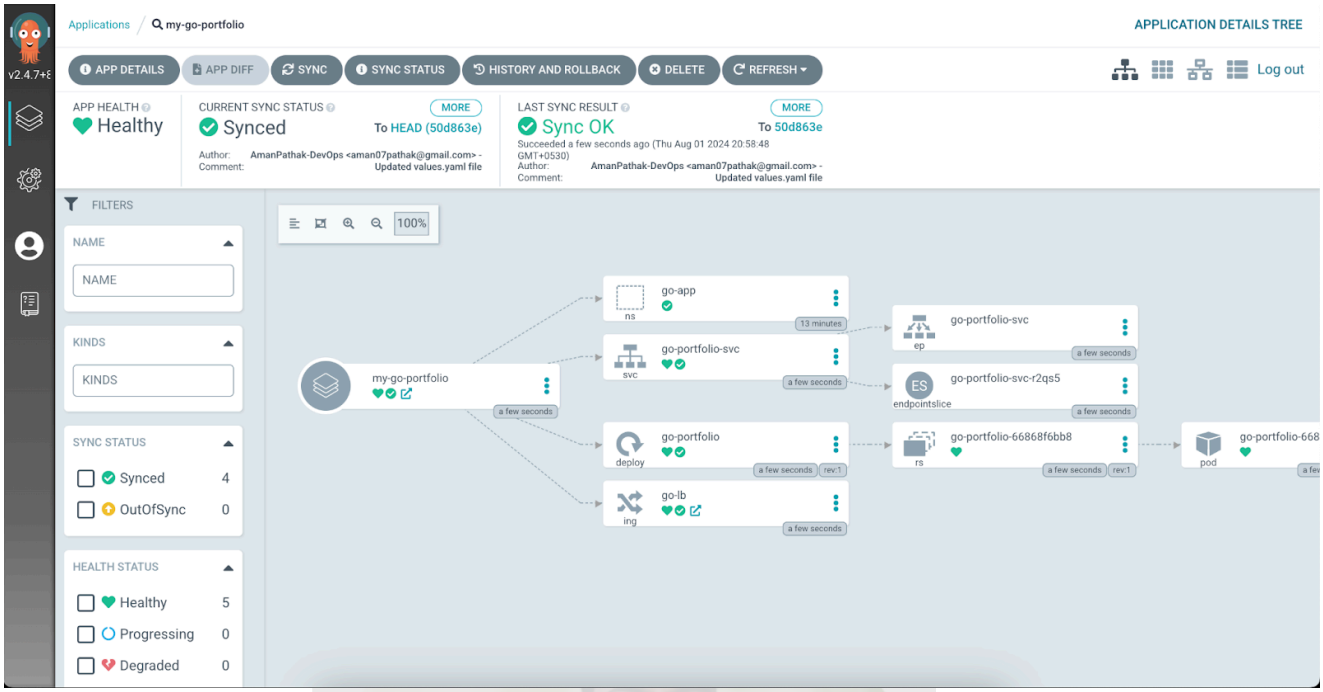




As soon as you create the application, your application will be deployed.



You can click on the application to view all the resources that have been created by argoCD



You can also validate from your jump server by running the below command.

```
kubectl get all -n go-app
```

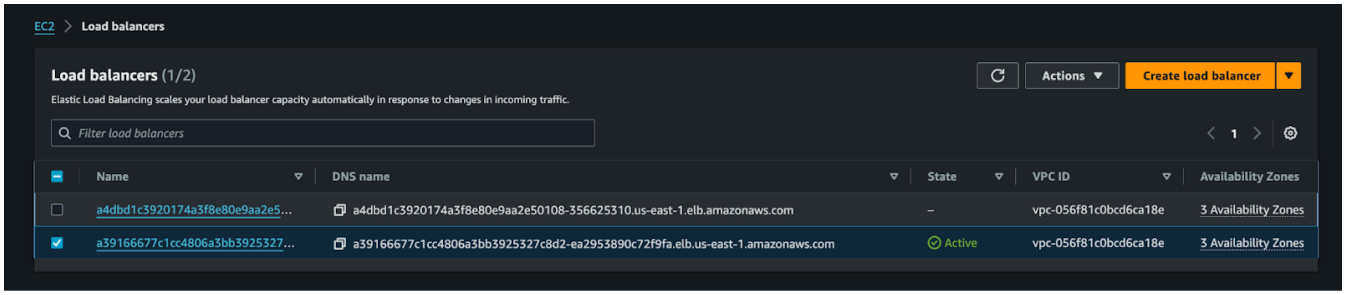
```
ubuntu@ip-10-16-45-215:~$ kubectl get all -n go-app
NAME                                 READY   STATUS    RESTARTS   AGE
pod/go-portfolio-66868f6bb8-zsw9w    1/1     Running   0          115s

NAME                         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/go-portfolio-svc     ClusterIP   172.20.88.100    <none>        80/TCP    115s

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/go-portfolio     1/1     1            1           115s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/go-portfolio-66868f6bb8     1         1         1       115s
ubuntu@ip-10-16-45-215:~$
```

Check the ingress address by using the below command

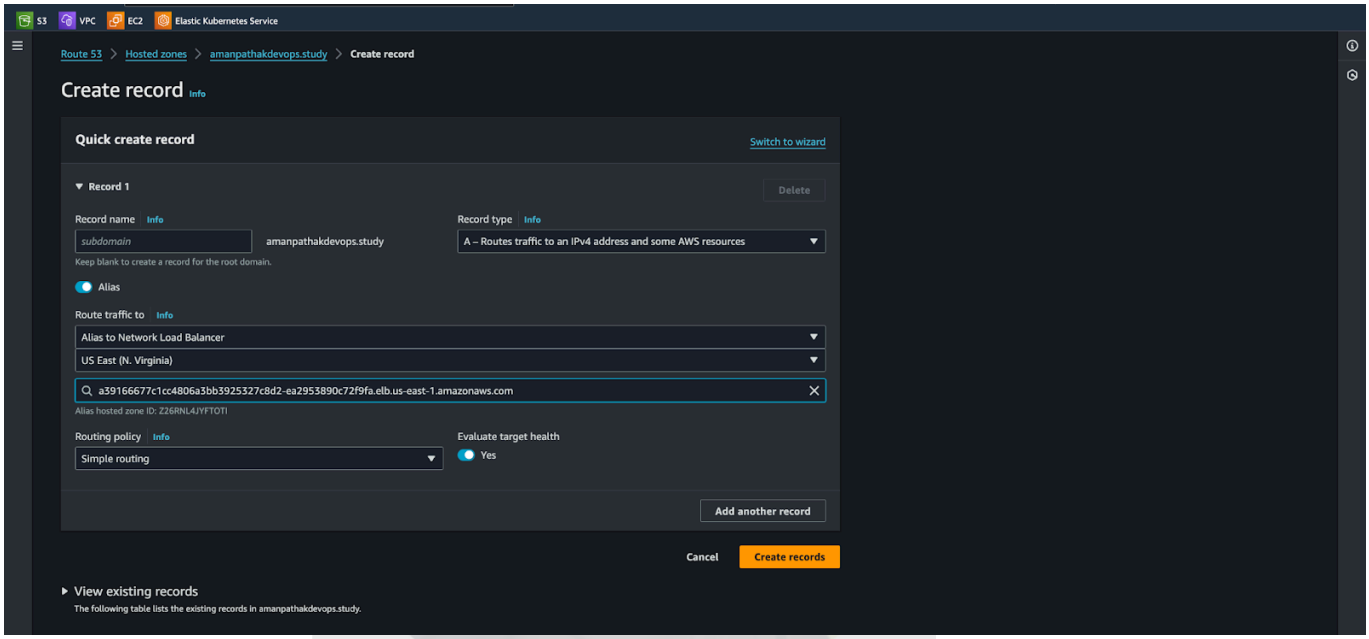
```
kubectl get ing -n go-app
```

```
ubuntu@ip-10-16-45-215:~$ kubectl get ing -n go-app
NAME    CLASS   HOSTS                    ADDRESS                                                                   PORTS   AGE
go-lb   nginx   amanpathakdevops.study   a39166677c1cc4806a3bb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com   80      2m23s
ubuntu@ip-10-16-45-215:~$
```

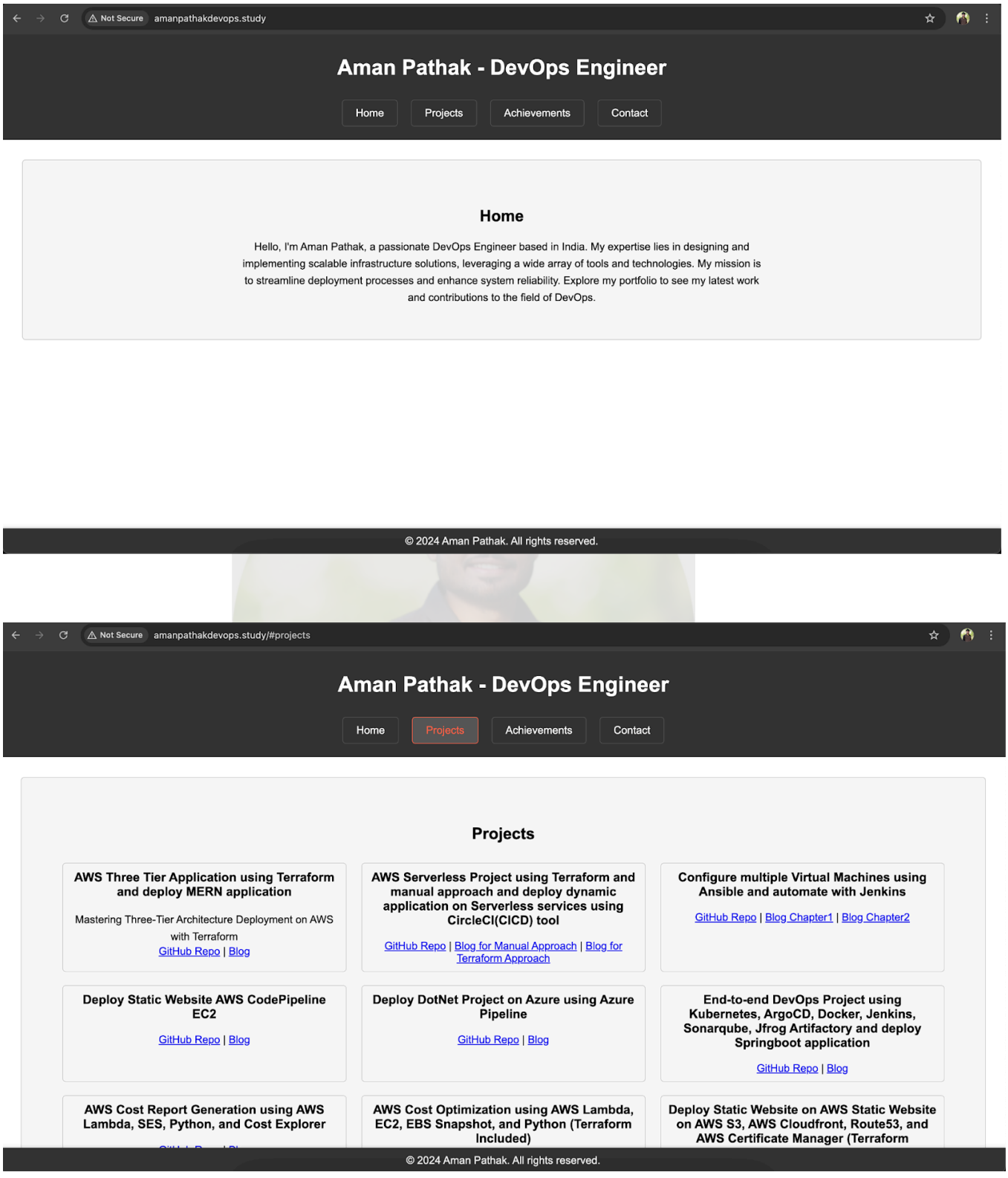
Go to your AWS console, navigate to the Load balancer, and copy the DNS name



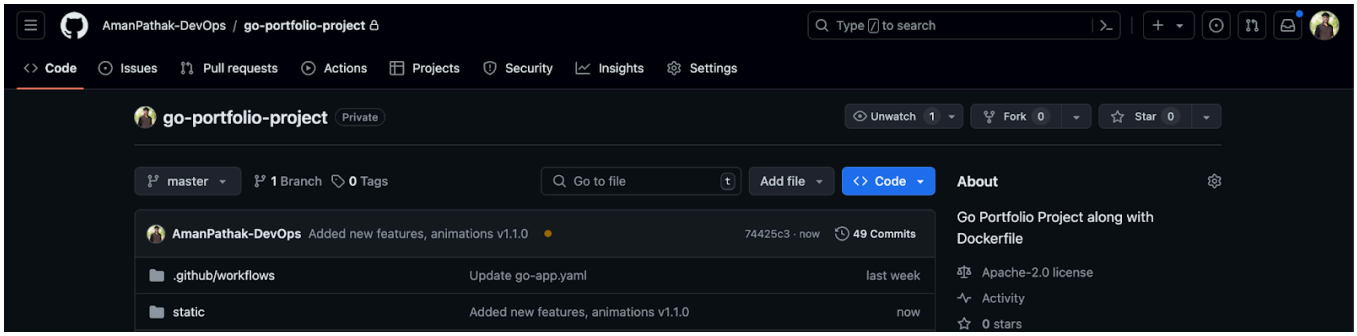
Go to your domain provider and add A type record.
I am using AWS Route53 to add the record



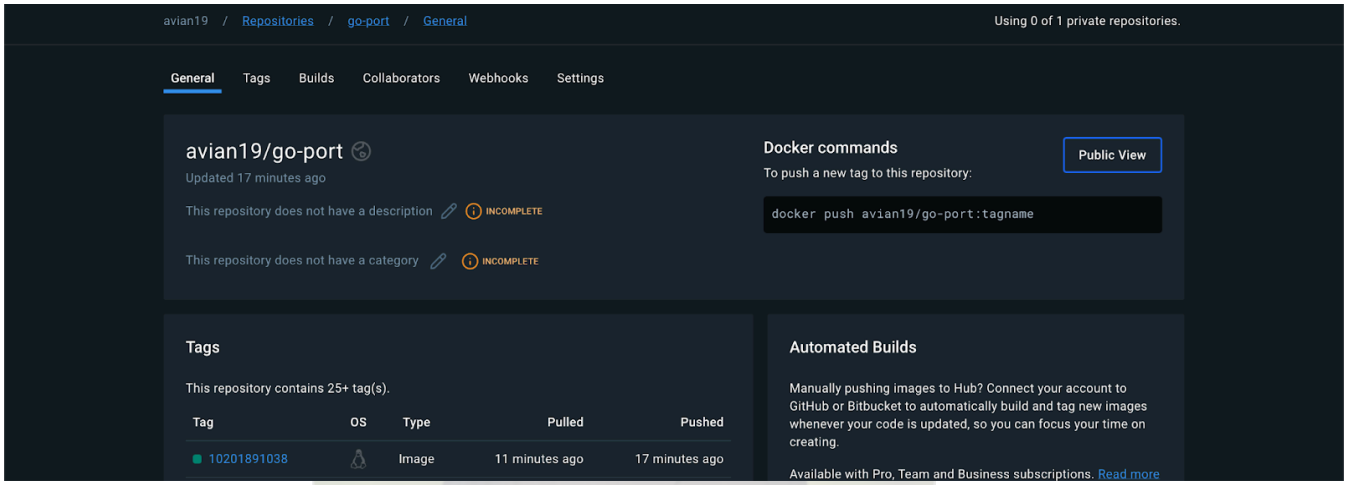
Once you add the record you can hit your domain on your favorite browser and see the magic



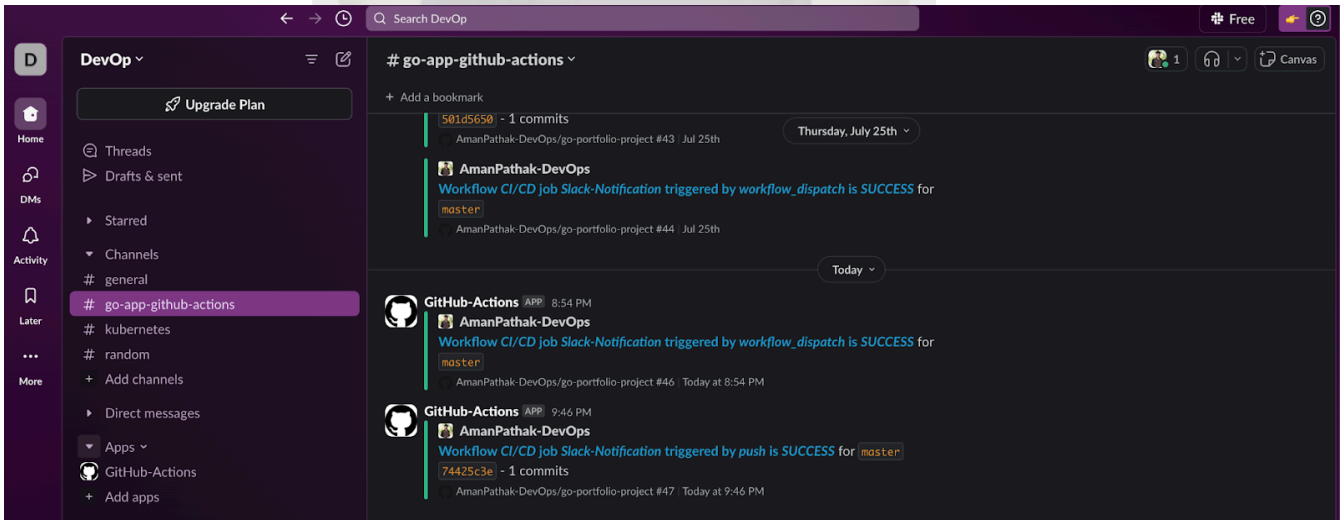
Now, I will make a few changes to my portfolio website which will trigger the workflow and the entire application will be deployed without any human intervention



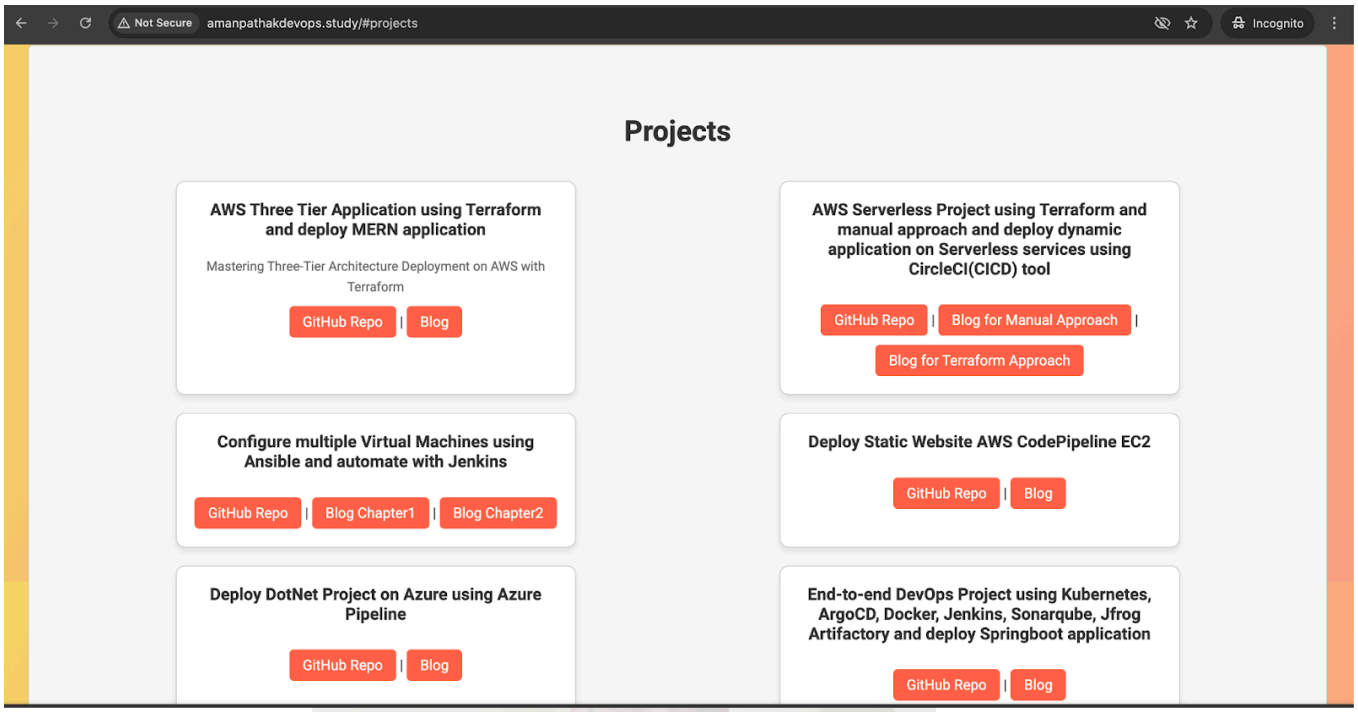
The updated image has been pushed to the docker hub



Received the Slack notification



Our v1.1.0 of go-portfolio has been deployed without any human intervention which means we achieved complete automation of CICD


Not Secure
amanpathakdevops.study/#projects
Incognito
Projects
AWS Three Tier Application using Terraform and deploy MERN application
Mastering Three-Tier Architecture Deployment on AWS with Terraform
GitHub Repo | Blog
AWS Serverless Project using Terraform and manual approach and deploy dynamic application on Serverless services using CircleCI(CICD) tool
GitHub Repo | Blog for Manual Approach |
Blog for Terraform Approach
Configure multiple Virtual Machines using Ansible and automate with Jenkins
GitHub Repo | Blog Chapter1 | Blog Chapter2
Deploy Static Website AWS CodePipeline EC2
GitHub Repo | Blog
Deploy DotNet Project on Azure using Azure Pipeline
GitHub Repo | Blog
End-to-end DevOps Project using Kubernetes, ArgoCD, Docker, Jenkins, Sonarqube, Jfrog Artifactory and deploy Springboot application
GitHub Repo | Blog