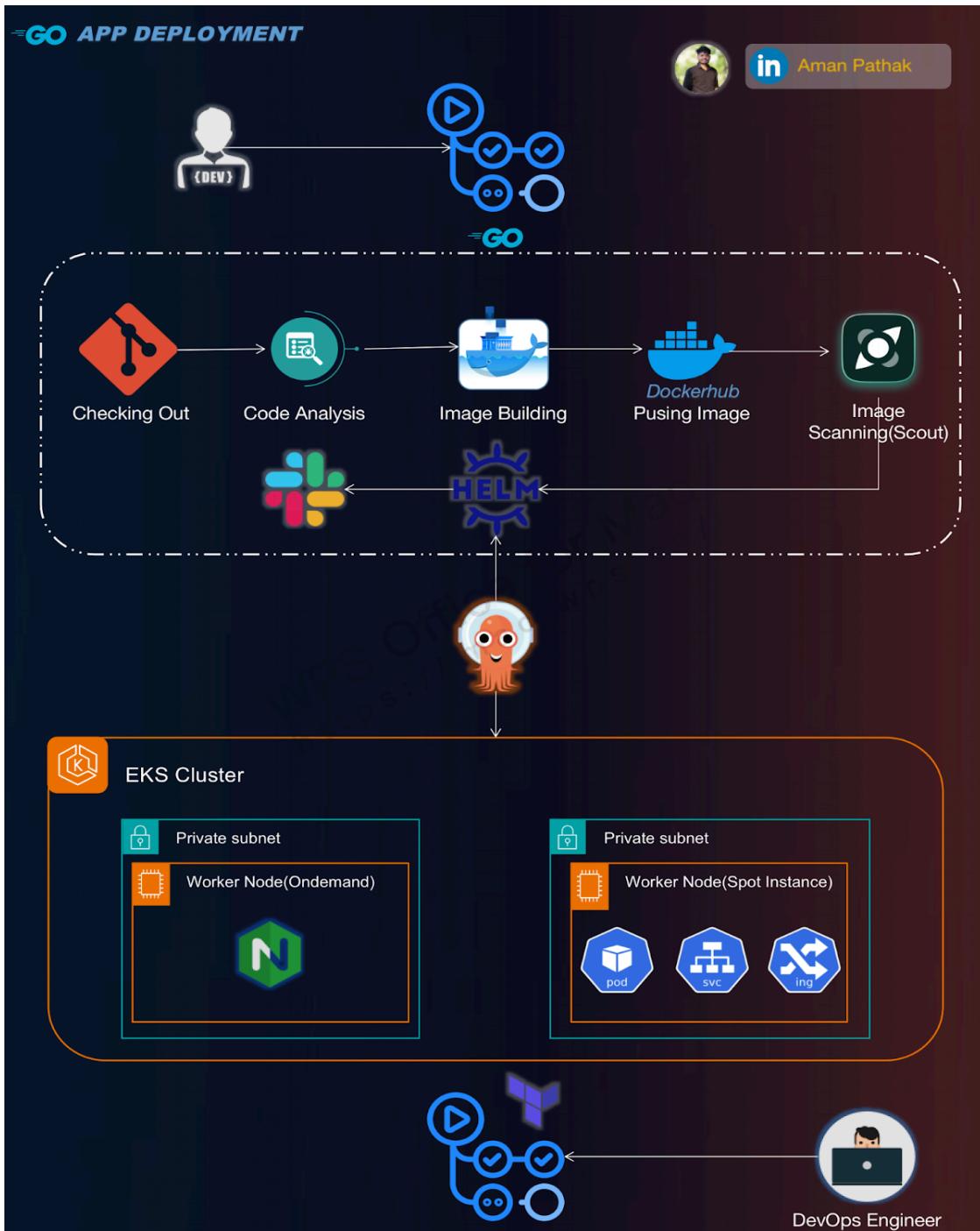


Automate Go App Deployment on Private EKS with GitHub Actions, Terraform, Helm, and ArgoCD

- 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>

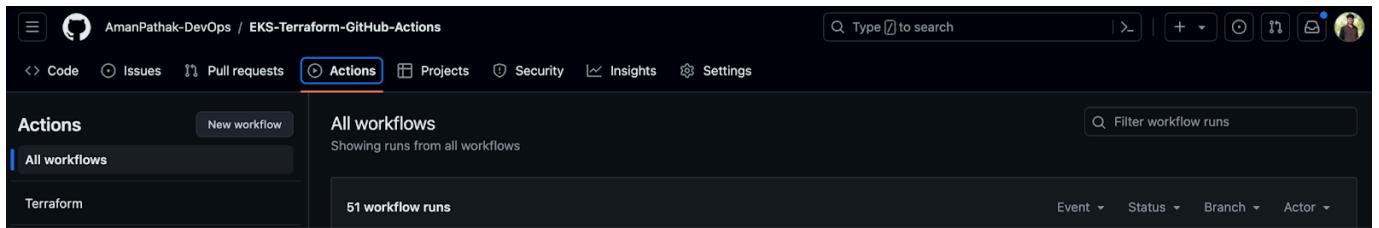
Deploy EKS using GitHub Actions

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

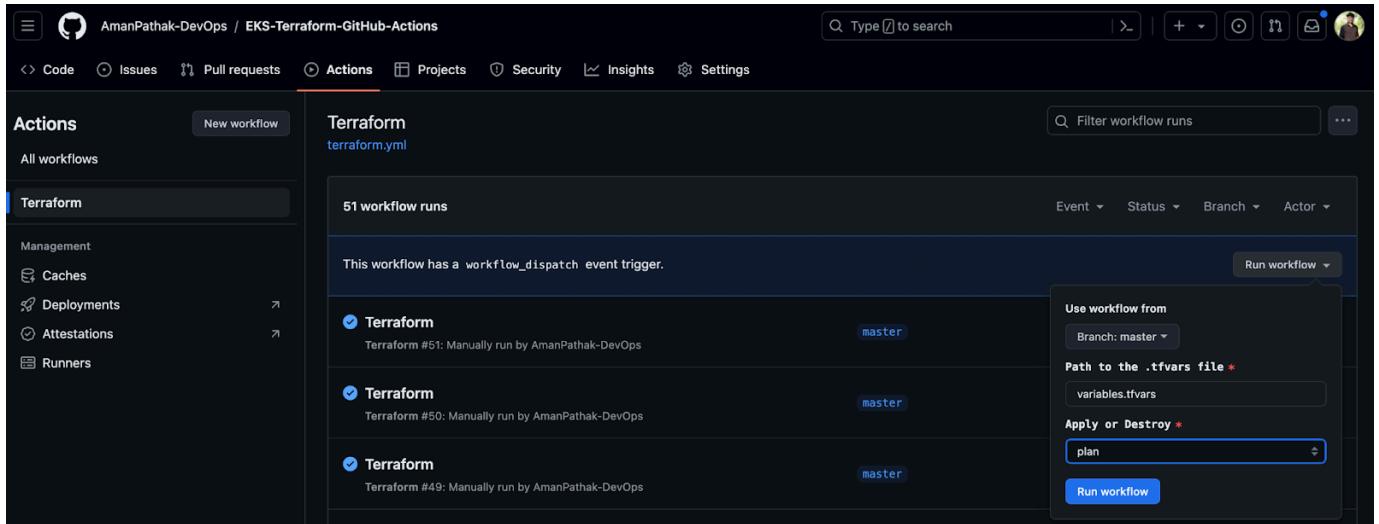
Click on Actions

The screenshot shows the GitHub repository page for 'EKS-Terraform-GitHub-Actions'. The repository is public and has 1 branch and 0 tags. The commit history shows several commits from 'AmanPathak-DevOps' over the past few weeks, including updates to README.md, .github/workflows, assets, eks, module, LICENSE, and README.md. The 'About' section describes the project as 'Configuring Production-Ready EKS Clusters with Terraform and GitHub Actions' and lists tags such as 'medium.com/p/c046e8d44865', 'github', 'kubernetes', 'aws', 'devops', 'terraform', 'eks', and 'github-actions'. The 'Readme' and 'Apache-2.0 license' files are also listed. The bottom of the page includes social links for LinkedIn, GitHub, and others, and sections for 'Releases' and 'Packages'.

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

Terraform-Action
succeeded 1 minute ago in 14s

Search logs

2s

Terraform Plan

```

780      + default_security_group_id      = (known after apply)
781      + dhcp_options_id              = (known after apply)
782      + enable_dns_hostnames        = true
783      + enable_dns_support          = true
784      + enable_network_address_usage_metrics = (known after apply)
785      + id                          = (known after apply)
786      + instance_tenancy            = "default"
787      + ipv6_association_id        = (known after apply)
788      + ipv6_cidr_block             = (known after apply)
789      + ipv6_cidr_block_network_border_group = (known after apply)
790      + main_route_table_id         = (known after apply)
791      + owner_id                    = (known after apply)
792      + tags                        = {
793          + "Env" = "dev"
794          + "Name" = "dev-medium-vpc"
795      }
796      + tags_all                   = {
797          + "Env" = "dev"
798          + "Name" = "dev-medium-vpc"
799      }
800  }
801
802 # module.eks.random_integer.random_suffix will be created
803 + resource "random_integer" "random_suffix" {
804     + id      = (known after apply)
805     + max    = 9999
806     + min    = 1000
807     + result = (known after apply)
808   }
809
810 Plan: 37 to add, 0 to change, 0 to destroy.
811

```

Now, we will run the apply

Go to Terraform workflow and click on **Run workflow**

The screenshot shows the Terraform workflow page with the following details:

- Actions** sidebar: Shows options like New workflow, All workflows, Management, Caches, Deployments, Attestations, and Runners.
- Terraform** section: Shows the file `terraform.tfvars`.
- Workflow Runs**: Displays 52 workflow runs. The first three are listed:
 - Terraform**: Terraform #52: Manually run by AmanPathak-DevOps (Branch: master)
 - Terraform**: Terraform #51: Manually run by AmanPathak-DevOps (Branch: master)
 - Terraform**: Terraform #50: Manually run by AmanPathak-DevOps (Branch: master)
- Run workflow** button: Located at the top right of the workflow list.
- Modal (open)**:
 - Use workflow from**: Branch: master
 - Path to the .tfvars file**: variables.tfvars
 - Apply or Destroy**: apply
 - Run workflow** button

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

The screenshot shows the GitHub Actions logs for the "Terraform-Action" step. The logs show the execution of the Terraform command and its output:

```

  Terraform Action
  succeeded 1 minute ago in 16m 45s
  1011 module.eks.aws.eks_node_group.spot-node: Still creating... [4m30s elapsed]
  1012 module.eks.aws.eks_node_group.spot-node: Still creating... [4m40s elapsed]
  1013 module.eks.aws.eks_node_group.on-demand-node: Still creating... [4m40s elapsed]
  1014 module.eks.aws.eks_node_group.spot-node: Creation complete after 4m49s [id=dev-medium-eks-cluster:dev-medium-eks-cluster-spot-nodes]
  1015 module.eks.aws.eks_node_group.on-demand-node: Still creating... [4m58s elapsed]
  1016 module.eks.aws.eks_node_group.on-demand-node: Creation complete after 5m0s [id=dev-medium-eks-cluster:dev-medium-eks-cluster-on-demand-nodes]
  1017 module.eks.aws.eks_addon.eks-addons["2"]: Creating...
  1018 module.eks.aws.eks_addon.eks-addons["3"]: Creating...
  1019 module.eks.aws.eks_addon.eks-addons["0"]: Creating...
  1020 module.eks.aws.eks_addon.eks-addons["1"]: Creating...
  1021 module.eks.aws.eks_addon.eks-addons["1"]: Still creating... [10s elapsed]
  1022 module.eks.aws.eks_addon.eks-addons["2"]: Still creating... [10s elapsed]
  1023 module.eks.aws.eks_addon.eks-addons["0"]: Still creating... [10s elapsed]
  1024 module.eks.aws.eks_addon.eks-addons["3"]: Still creating... [10s elapsed]
  1025 module.eks.aws.eks_addon.eks-addons["2"]: Still creating... [20s elapsed]
  1026 module.eks.aws.eks_addon.eks-addons["1"]: Still creating... [20s elapsed]
  1027 module.eks.aws.eks_addon.eks-addons["0"]: Still creating... [20s elapsed]
  1028 module.eks.aws.eks_addon.eks-addons["3"]: Still creating... [20s elapsed]
  1029 module.eks.aws.eks_addon.eks-addons["1"]: Creation complete after 25s [id=dev-medium-eks-cluster:coredns]
  1030 module.eks.aws.eks_addon.eks-addons["0"]: Still creating... [30s elapsed]
  1031 module.eks.aws.eks_addon.eks-addons["3"]: Still creating... [30s elapsed]
  1032 module.eks.aws.eks_addon.eks-addons["2"]: Still creating... [30s elapsed]
  1033 module.eks.aws.eks_addon.eks-addons["0"]: Still creating... [40s elapsed]
  1034 module.eks.aws.eks_addon.eks-addons["2"]: Still creating... [40s elapsed]
  1035 module.eks.aws.eks_addon.eks-addons["0"]: Still creating... [40s elapsed]
  1036 module.eks.aws.eks_addon.eks-addons["0"]: Creation complete after 45s [id=dev-medium-eks-cluster:vpc-cni]
  1037 module.eks.aws.eks_addon.eks-addons["2"]: Creation complete after 45s [id=dev-medium-eks-cluster:kube-proxy]
  1038 module.eks.aws.eks_addon.eks-addons["3"]: Still creating... [50s elapsed]
  1039 module.eks.aws.eks_addon.eks-addons["3"]: Creation complete after 55s [id=dev-medium-eks-cluster:aws-efs-csi-driver]
  1040
  1041 Apply complete! Resources: 37 added, 0 changed, 0 destroyed.

```

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

The screenshot shows the AWS EKS Cluster list. It displays one cluster named "dev-medium-eks-cluster" which is active and supports until March 23, 2025.

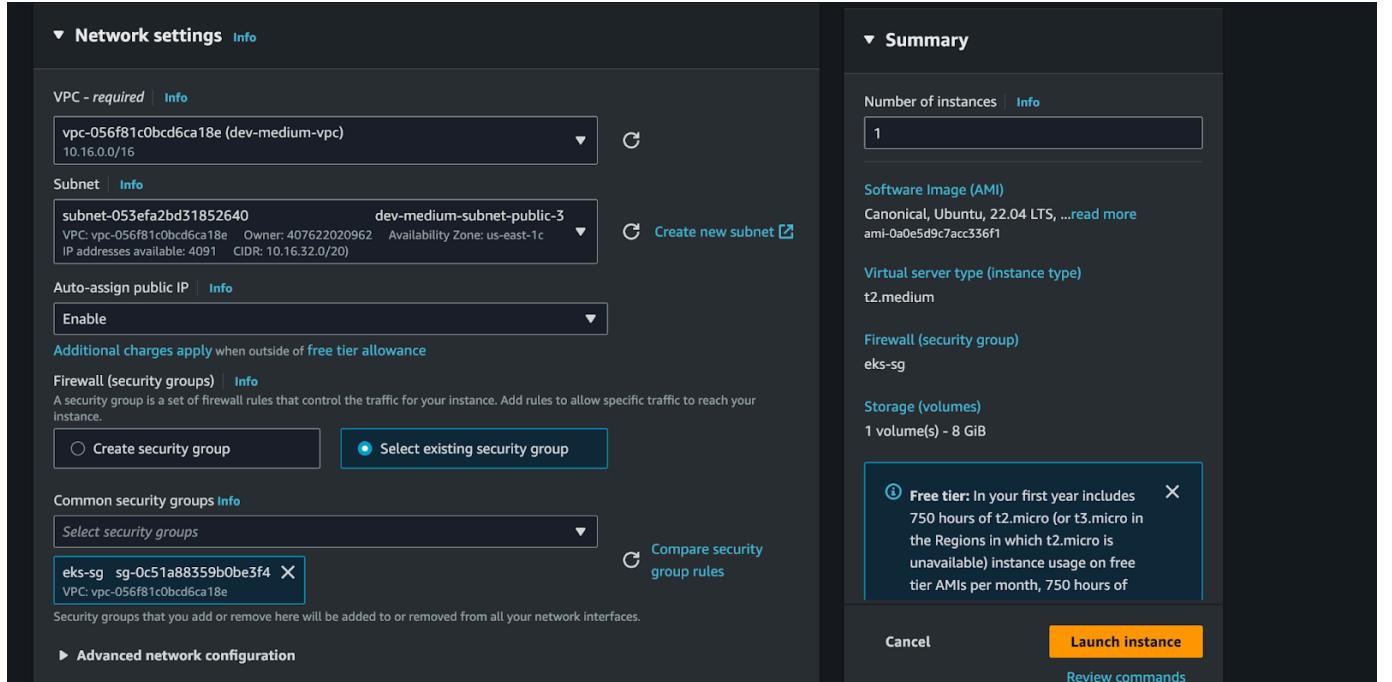
Cluster name	Status	Kubernetes version	Support period	Support type	Created	
dev-medium-eks-cluster	Active	1.29	Upgrade now	Standard support until March 23, 2025	Extended support	19 minutes

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-1:1487622020962:cluster/dev-medium-eks-cluster to /Users/aman.pathak/.kube/config
aman.pathak@Aman-Pathak ~ % kubectl get nodes
E0801 12:06:51.347798 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 is installed
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-dkhwx 1/1 Running 0 33s  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)  
AGE  
service/ingress-nginx-controller LoadBalancer 172.20.134.63 a39166677clcc4806a3bb3925327c8d2-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/approjects.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-fb6447b8-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:~$
```

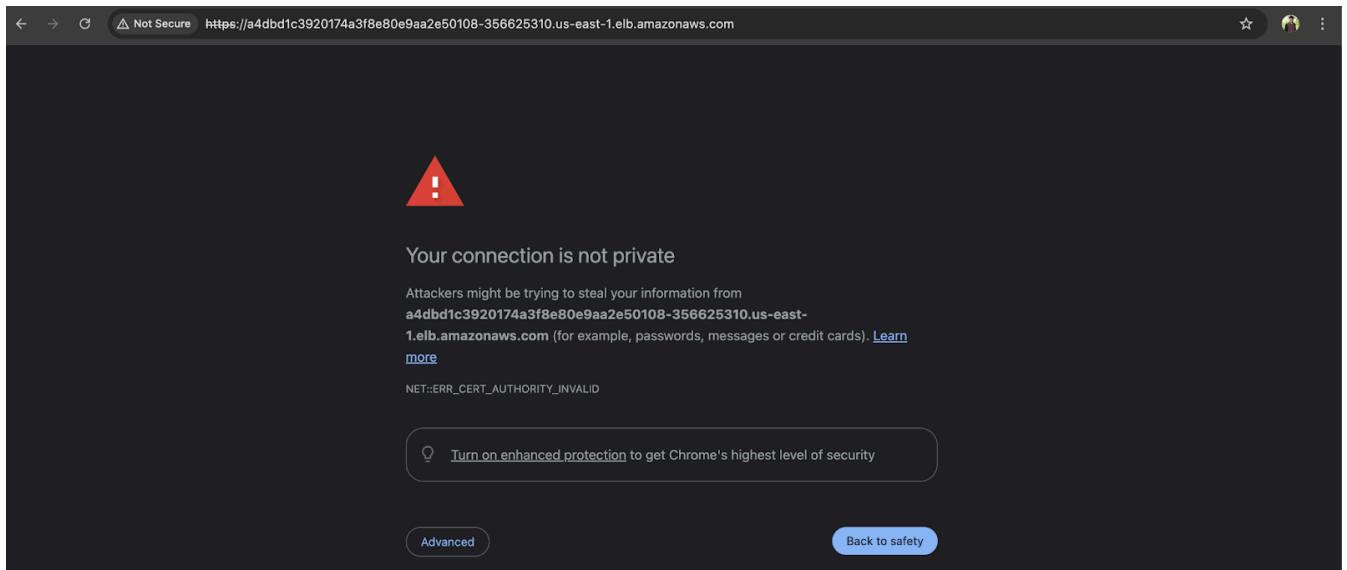
Validate whether the load balancer has been created or not by going to the AWS console

The screenshot shows the AWS EC2 Load Balancers page with the following details:

Load balancers (2)		Actions			Create load balancer	
Name		DNS name	State	VPC ID	Availability Zone	
<input type="checkbox"/>	a4dbd1c3920174a3f8e80e9aa2e50108	a4dbd1c3920174a3f8e80e9aa2e50108-356625310.us-east-1.elb.amazonaws.com	-	vpc-056f81c0bcd6ca18e	3 Availability Zones	
<input type="checkbox"/>	a39166677c1cc4806a3hb3925327c8d2	a39166677c1cc4806a3hb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com	Active	vpc-056f81c0bcd6ca18e	3 Availability Zones	

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

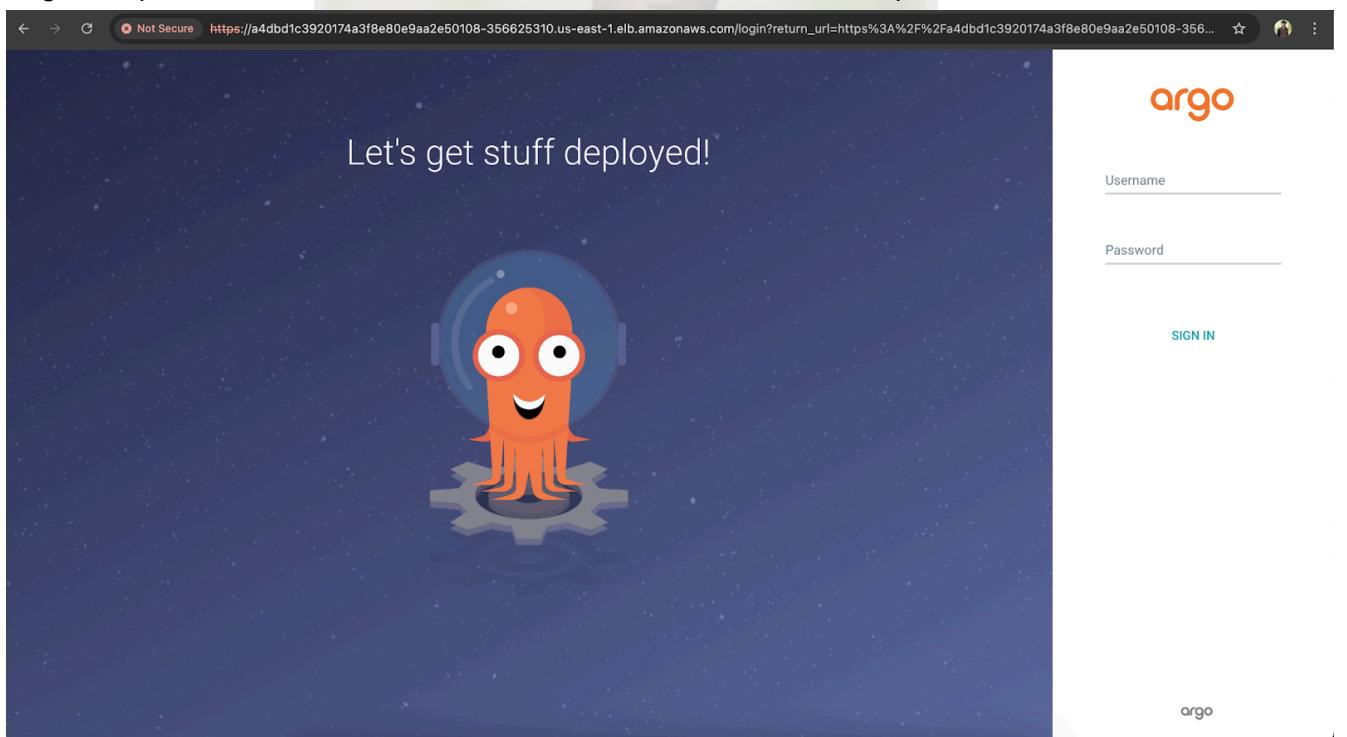
Click on Advanced as SSL is not configured on argoCD



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
```

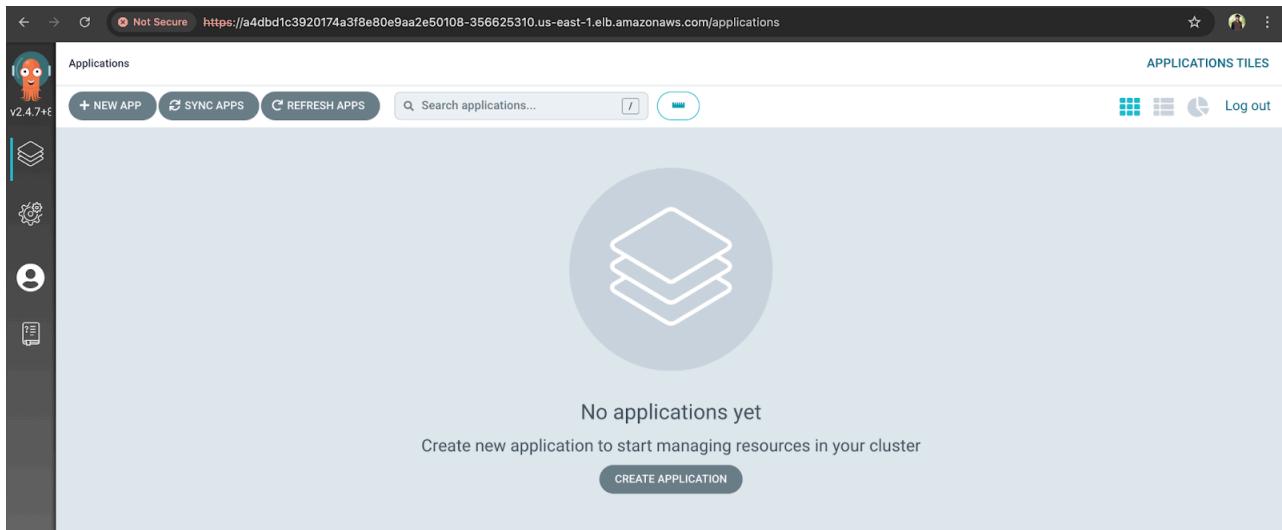
```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  password: VmNMbW9pejYwRmhmajhYQw==
kind: Secret
metadata:
  creationTimestamp: "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:~$ echo VmNMbW9pejYwRmhmajhYQw== | base64 --decode
VcLmoiz60Fhfj8XCubuntu@ip-10-16-45-215:~$
```

Now, log in to your argoCD console



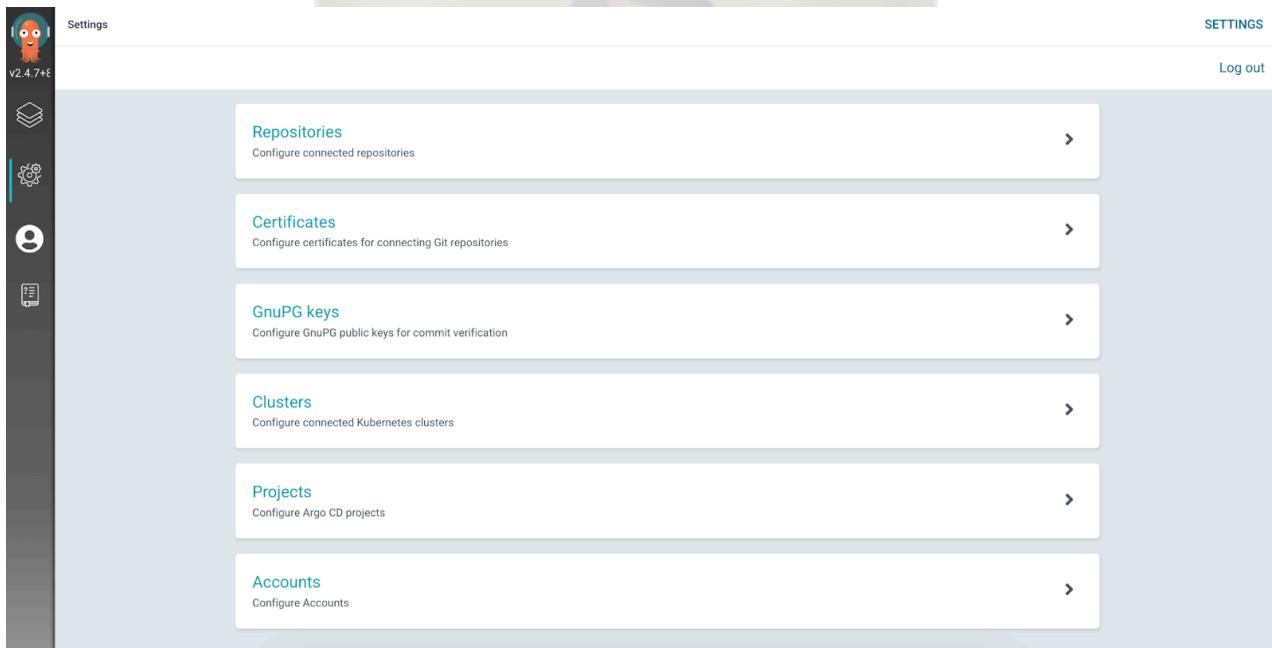
We have set everything to deploy our application

But there is one more thing that is connected to the repository in argoCD

Why are we doing that?

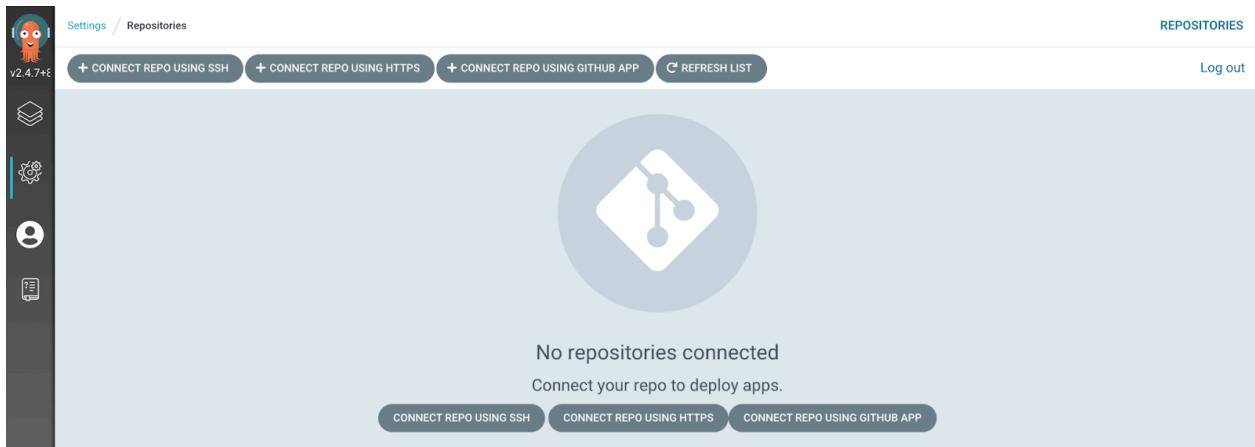
So, our helm chart or manifest file is always stored in a private repository. To clone that repo, we need to set up our repository in argoCD.

Click on the settings icon on the left and then, click on Repositories

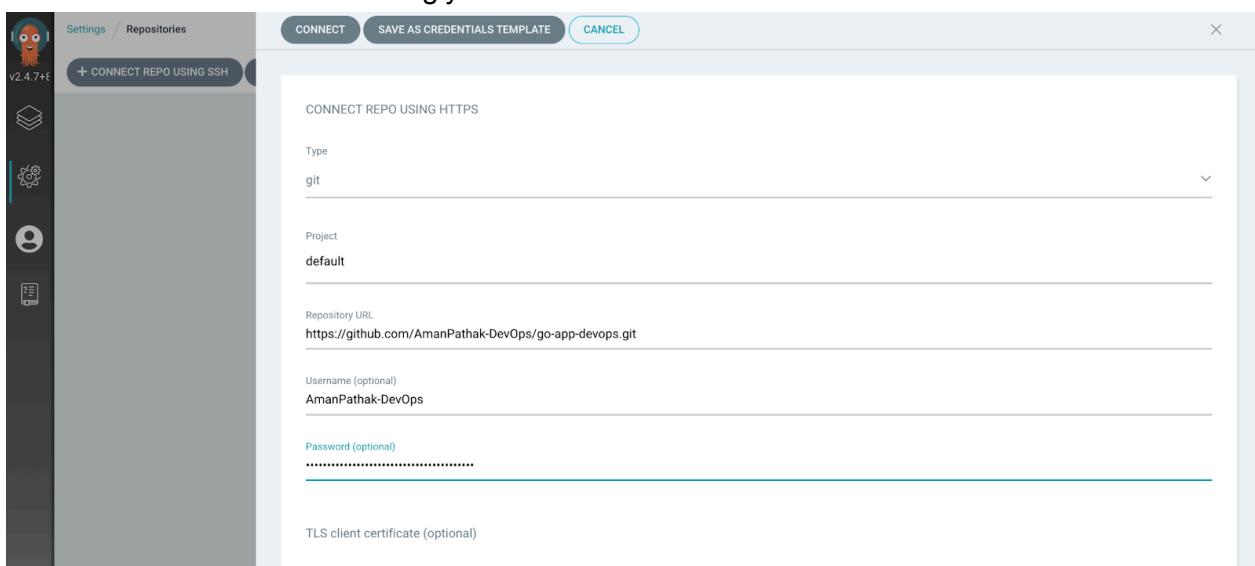


For now, we are going to connect the repo using HTTPS which means we need a Personal Access token(PAT)

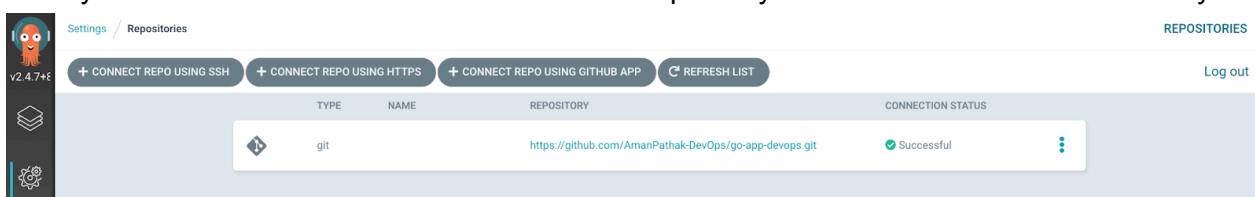
So, kindly generate the Personal Access token(PAT) of your GitHub account



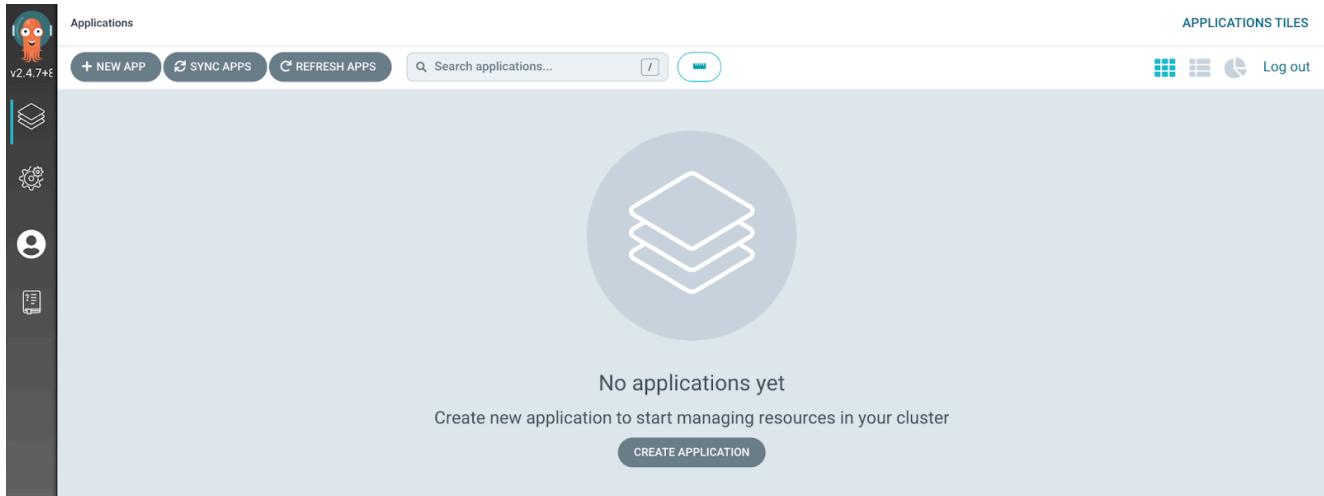
Provide the information accordingly



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.

The screenshot shows the GitHub repository settings for 'AmanPathak-DevOps / go-portfolio-project'. The left sidebar has sections like General, Access, Collaborators, Code and automation (Branches, Tags, Rules, Actions, Webhooks, Codespaces, Pages), and Security (Code security and analysis, Deploy keys, Secrets and variables). The right pane is titled 'Actions secrets and variables' and shows a table of repository secrets. It includes four entries: DOCKER_PASSWORD, DOCKER_USERNAME, REPO_PAT, and SLACK_WEBHOOK_URL, each with a last updated timestamp (last week or 26 minutes ago) and edit/delete icons.

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>

The screenshot shows the GitHub repository files page for 'go-portfolio-project'. The left sidebar lists files like .gitignore, Dockerfile, LICENSE, README.md, go.mod, go.sum, and main.go. The right pane displays the contents of the 'go-app.yaml' file under '.github/workflows'. The file defines a CI/CD workflow with an 'on' trigger for 'push' to the 'master' branch, ignoring changes to 'README.md'. It uses environment variables from secrets and runs on an Ubuntu 20.04 LTS container. The workflow includes steps for checking out the repository, setting up Go 1.22, building the application, and running tests.

```
name: CI/CD
on:
  workflow_dispatch:
  push:
    branches:
      - master
    paths-ignore:
      - 'README.md'
env:
  REGISTRY: docker.io
  SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK_URL }}
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4.1.7
      - name: Setting Up Go 1.22
        uses: actions/setup-go@v2
        with:
          go-version: 1.22
      - name: Build
        run: go build -o go-app
```

Now, we are set to run our workflow/pipeline

The screenshot shows the GitHub Actions CI/CD interface for the repository 'AmanPathak-DevOps / go-portfolio-project'. The left sidebar has 'Actions' selected under 'CI/CD'. The main area displays '46 workflow runs' for the 'go-app.yaml' file. A tooltip indicates that this workflow has a 'workflow_dispatch' event trigger. Two specific runs are highlighted: one successful run ('#46') and one failed run ('#45'). Both runs show they were manually triggered by 'AmanPathak-DevOps' and were run from the 'master' branch. A button labeled 'Run workflow' is visible for each run.

Workflow got successful

The screenshot shows the GitHub Actions CI/CD interface for the repository 'AmanPathak-DevOps / go-portfolio-project'. The left sidebar has 'Actions' selected under 'CI/CD', with 'CI/CD #46' selected. The main area shows a summary of the successful run: it was triggered 20 minutes ago by 'AmanPathak-DevOps' on the 'master' branch, with a status of 'Success', a total duration of '3m 31s', billable time of '6m', and one artifact produced. Below this, the workflow graph for 'go-app.yaml' is shown, starting with 'build' and followed by 'Code-Quality', 'Docker-Image-Push', 'Image-Scanning', 'Updating-helm-tags', and 'Slack-Notification'. Each step is represented by a bar indicating its duration.

The image has been pushed

The screenshot shows the Docker Hub repository page for 'avian19/go-port'. The repository was updated 5 minutes ago and contains 25+ tags. One tag, '10201111330', was pushed 5 minutes ago. The Docker commands section shows the command to push a new tag: 'docker push avian19/go-port:tagname'. The Automated Builds section explains that manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

The image tag has been updated in values.yaml file

```
Code Blame 22 lines (19 loc) · 512 Bytes Code 55% faster with GitHub Copilot
1 # Default values for go-portfolio-chart.
2 # This is a YAML-formatted file.
3 # Declare variables to be passed into your templates.
4
5 replicaCount: 1
6
7 image:
8   repository: avian19/go-port
9   pullPolicy: IfNotPresent
10  # Overrides the image tag whose default is the chart appVersion.
11  tag: "10201111330"
12
```

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

Applications

+ NEW APP

CREATE CANCEL

GENERAL

Application Name: my-go-portfolio

Project Name: default

SYNC POLICY

Automatic

PRUNE RESOURCES

SELF HEAL

Applications

+ NEW APP

CREATE CANCEL

SOURCE

Repository URL: https://github.com/AmanPathak-DevOps/go-app-devops.git

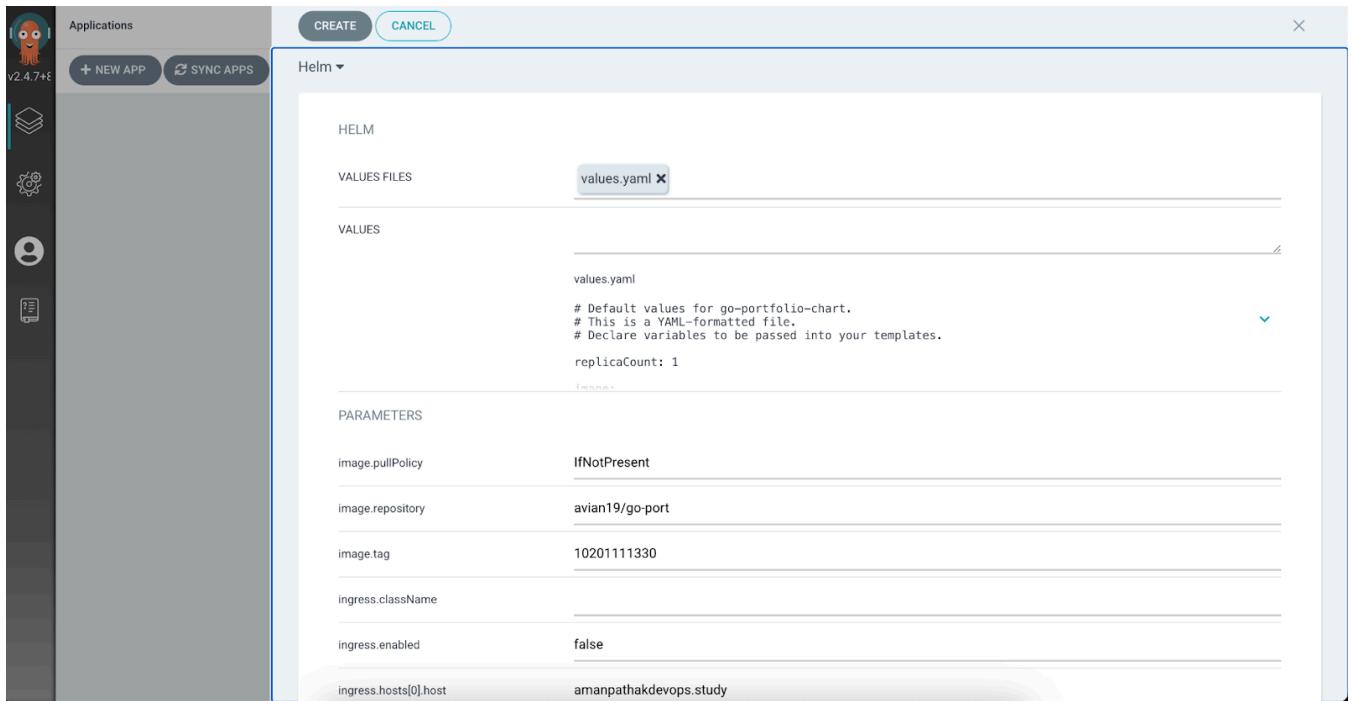
Revision: HEAD

Path: .

DESTINATION

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

Namespace: go-app



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

my-go-portfolio

Project: default

Labels:

Status: Healthy Synced

Repository: https://github.com/AmanPathak-DevOps...

Target R...: HEAD

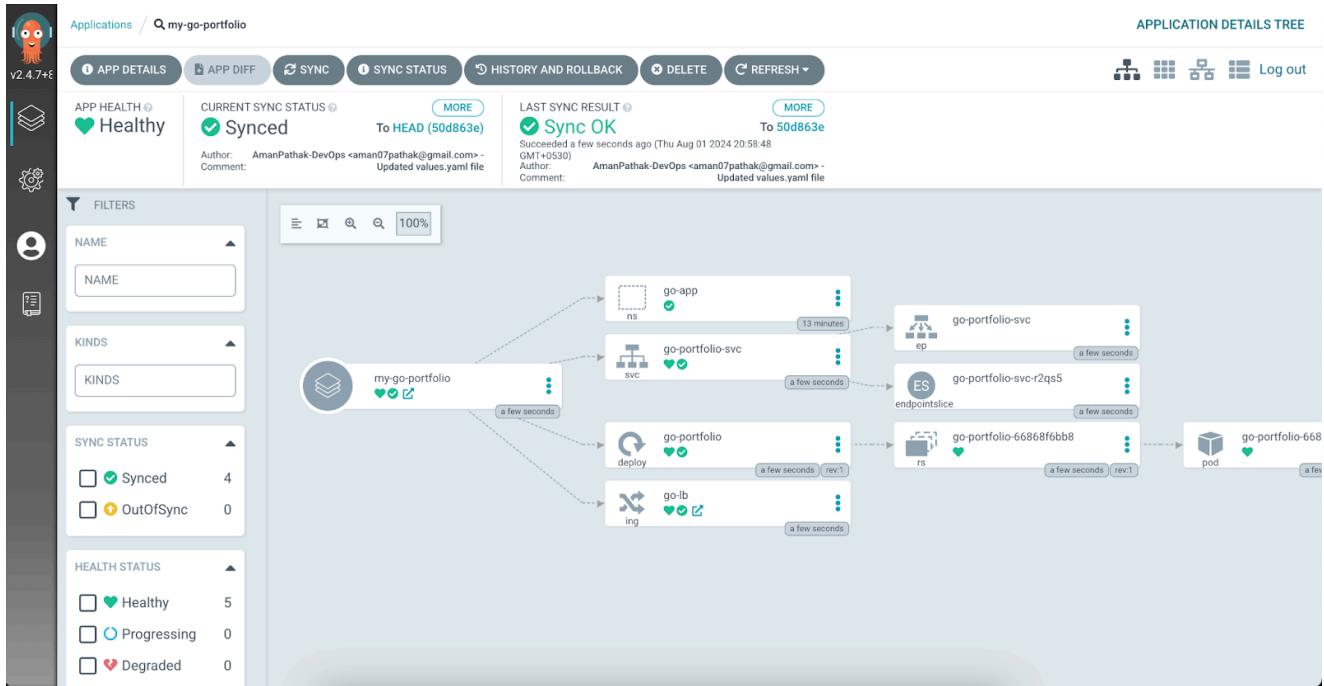
Path: .

Destination: in-cluster

Namespace: go-app

SYNC **REFRESH** **DELETE**

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

The screenshot shows the AWS EC2 Load balancers page. At the top, there is a search bar labeled "Filter load balancers". Below it is a table with columns: Name, DNS name, State, VPC ID, and Availability Zones. There are two entries:

Name	DNS name	State	VPC ID	Availability Zones
a4dbd1c3920174a3f8e80e9aa2e5...	a4dbd1c3920174a3f8e80e9aa2e50108-356625310.us-east-1.elb.amazonaws.com	-	vpc-056f81c0bcd6ca18e	3 Availability Zones
<input checked="" type="checkbox"/> a39166677c1cc4806a3bb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com	a39166677c1cc4806a3bb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com	Active	vpc-056f81c0bcd6ca18e	3 Availability Zones

Go to your domain provider and add A type record.

I am using AWS Route53 to add the record

The screenshot shows the AWS Route 53 Create record page. The subdomain "subdomain" is being aliased to the Network Load Balancer "a39166677c1cc4806a3bb3925327c8d2-ea2953890c72f9fa.elb.us-east-1.amazonaws.com" in the "US East (N. Virginia)" region. The "Evaluate target health" option is set to "Yes".

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

The screenshot shows a dark-themed web browser window. The address bar says "Not Secure amanpathakdevops.study". The main header is "Aman Pathak - DevOps Engineer". Below it are four buttons: "Home", "Projects", "Achievements", and "Contact".

The content area has a light gray background. The title "Home" is centered at the top. Below it is a text block: "Hello, I'm Aman Pathak, a passionate DevOps Engineer based in India. My expertise lies in designing and implementing scalable infrastructure solutions, leveraging a wide array of tools and technologies. My mission is to streamline deployment processes and enhance system reliability. Explore my portfolio to see my latest work and contributions to the field of DevOps."



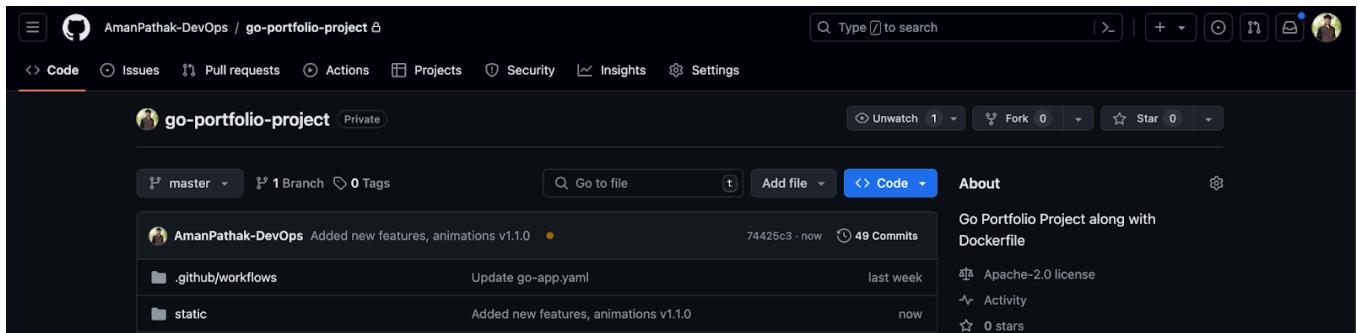
The screenshot shows a dark-themed web browser window. The address bar says "Not Secure amanpathakdevops.study/#projects". The main header is "Aman Pathak - DevOps Engineer". Below it are four buttons: "Home", "Projects" (which is highlighted with a red border), "Achievements", and "Contact".

The content area has a light gray background. It features a grid of project cards:

- 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](#)
- AWS Cost Report Generation using AWS Lambda, SES, Python, and Cost Explorer**
[GitHub Repo](#) | [Blog](#)
- AWS Cost Optimization using AWS Lambda, EC2, EBS Snapshot, and Python (Terraform Included)**
- Deploy Static Website on AWS Static Website on AWS S3, AWS Cloudfront, Route53, and AWS Certificate Manager (Terraform)**

At the bottom, there is a dark footer bar with the text "© 2024 Aman Pathak. All rights reserved."

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

The screenshot shows the Docker Hub repository 'avian19/go-port'. It displays a single tag '10201891038' which was pushed 17 minutes ago. The repository has 25+ tags. A 'Docker commands' section shows the command 'docker push avian19/go-port:tagname'. The 'Automated Builds' section indicates manual pushing is required to trigger automated builds.

Received the Slack notification

The screenshot shows a Slack channel '# go-app-github-actions'. It displays messages from the 'GitHub Actions' app indicating successful CI/CD jobs triggered by workflow_dispatch and push events. The messages show the job status as 'SUCCESS' and provide links to the GitHub repository and commit details.

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

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(CI_CD) 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](#)

