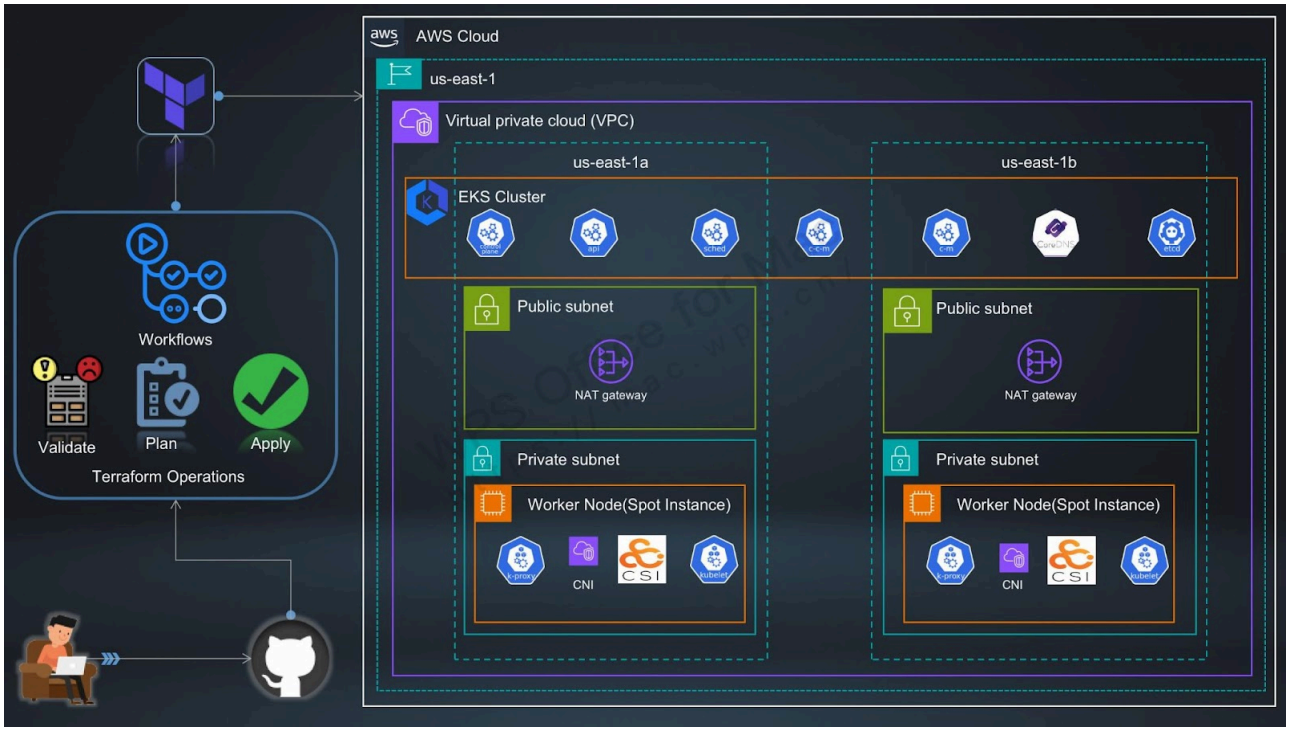
Configure Production-Ready EKS Clusters with Terraform and GitHub Actions



By Aman Pathak

Why are We using GitHub Actions?

The main reason behind this is **GitHub Actions** currently trending in the DevOps market. We already automated AWS Infrastructure using Jenkins. You can refer to the link if you want to learn it- https://medium.com/devops-dev/introduction-ab466a03c714

Let's now explore GitHub Actions and configure the Production Ready EKS Cluster.

If I talk about the EKS Cluster, So EKS is one of the important AWS services that helps to deploy our application to Kubernetes on the Cloud.

I always talk about industry-level practices instead of Hello World. So, we need to understand what kind of best practices we need to follow while configuring EKS Cluster.

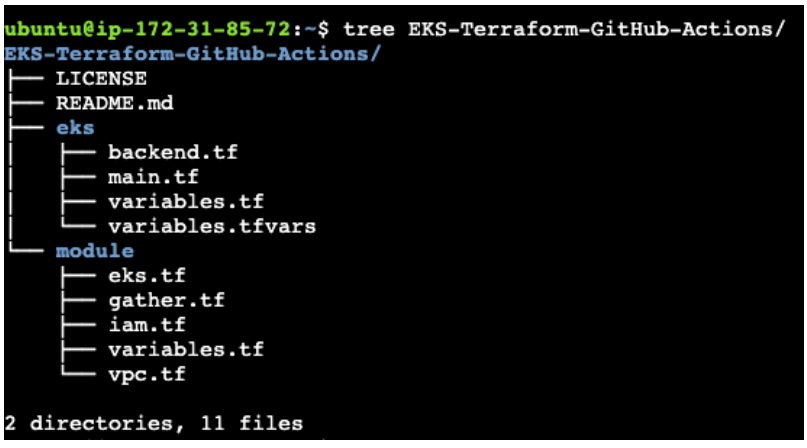
There are a few **Prerequisites** that need to be understood before starting this blog:

- Must have an AWS Account
- Must have AWS credentials including access key and secret key with Administrator Access(always avoid Administrator Access and follow the least privileged permissions)
- Must have a good understanding of Terraform
- Should have basic knowledge of YAML files for GitHub Actions workflow

To configure the EKS Cluster, we are going to use a modular approach. You can feel free to jump on the Terraform code by clicking on the repo link- https://github.com/AmanPathak-DevOps/EKS-Terraform-GitHub-Actions

But if you want to understand what configurations we are doing then you can continue in this blog and read the detailed information of each configuration.

Directory Structure-



module directory- Here all the resource scripts have been created related to the EKS cluster and it's required service(IAM, VPC, etc).

eks directory- Here, we will create the resources according to our requirements and we will call the resources module from the module directory.

First of all, we will start with module directory files:

gather.tf

In this file, we are fetching the TLS certificate for our EKS cluster with IAM policy which is dedicated to OIDC.

```
data "tls_certificate" "eks-certificate" {
 url = aws_eks_cluster.eks[0].identity[0].oidc[0].issuer
}

data "aws_iam_policy_document" "eks_oidc_assume_role_policy" {
 statement {
   actions = ["sts:AssumeRoleWithWebIdentity"]
   effect  = "Allow"

   condition {
     test     = "StringEquals"
     variable = "${replace(aws_iam_openid_connect_provider.eks-oidc.url,
"https://", "")}:sub"
     values   = ["system:serviceaccount:default:aws-test"]
   }

   principals {
     identifiers = [aws_iam_openid_connect_provider.eks-oidc.arn]
     type        = "Federated"
   }
 }
}
```

vpc.tf

In VPC file, we are creating AWS services related to VPC including VPC, internet gateway, public subnet and private subnets(with the desired number of subnets), public and private route tables respective to their type of subnets, NAT Gateway, elastic IP (required for NAT Gateway), security group for EKS cluster to restrict the access for specific people only

```
locals {
 cluster-name = var.cluster-name
}

resource "aws_vpc" "vpc" {
 cidr_block           = var.cidr-block
 instance_tenancy     = "default"
 enable_dns_hostnames = true
 enable_dns_support   = true

 tags = {
   Name = var.vpc-name
   Env  = var.env

 }
}

resource "aws_internet_gateway" "igw" {
 vpc_id = aws_vpc.vpc.id

 tags = {
   Name                                          = var.igw-name
   env                                           = var.env
   "kubernetes.io/cluster/${local.cluster-name}" = "owned"
 }

 depends_on = [aws_vpc.vpc]
}

resource "aws_subnet" "public-subnet" {
 count                   = var.pub-subnet-count
 vpc_id                  = aws_vpc.vpc.id
 cidr_block              = element(var.pub-cidr-block, count.index)
 availability_zone       = element(var.pub-availability-zone, count.index)
 map_public_ip_on_launch = true

 tags = {
   Name                                          =
"${var.pub-sub-name}-${count.index + 1}"
   Env                                           = var.env
   "kubernetes.io/cluster/${local.cluster-name}" = "owned"
   "kubernetes.io/role/elb"                      = "1"
 }
```

```
 depends_on = [aws_vpc.vpc,
 ]
}

resource "aws_subnet" "private-subnet" {
 count                   = var.pri-subnet-count
 vpc_id                  = aws_vpc.vpc.id
 cidr_block              = element(var.pri-cidr-block, count.index)
 availability_zone       = element(var.pri-availability-zone, count.index)
 map_public_ip_on_launch = false

 tags = {
   Name                                          =
"${var.pri-sub-name}-${count.index + 1}"
   Env                                           = var.env
   "kubernetes.io/cluster/${local.cluster-name}" = "owned"
   "kubernetes.io/role/internal-elb"             = "1"
 }

 depends_on = [aws_vpc.vpc,
 ]
}


resource "aws_route_table" "public-rt" {
 vpc_id = aws_vpc.vpc.id

 route {
   cidr_block = "0.0.0.0/0"
   gateway_id = aws_internet_gateway.igw.id
 }

 tags = {
   Name = var.public-rt-name
   env  = var.env
 }

 depends_on = [aws_vpc.vpc
 ]
}

resource "aws_route_table_association" "name" {
```

```
 count          = 3
 route_table_id = aws_route_table.public-rt.id
 subnet_id      = aws_subnet.public-subnet[count.index].id

 depends_on = [aws_vpc.vpc,
   aws_subnet.public-subnet
 ]
}

resource "aws_eip" "ngw-eip" {
 domain = "vpc"

 tags = {
   Name = var.eip-name
 }

 depends_on = [aws_vpc.vpc
 ]

}

resource "aws_nat_gateway" "ngw" {
 allocation_id = aws_eip.ngw-eip.id
 subnet_id     = aws_subnet.public-subnet[0].id

 tags = {
   Name = var.ngw-name
 }

 depends_on = [aws_vpc.vpc,
   aws_eip.ngw-eip
 ]
}

resource "aws_route_table" "private-rt" {
 vpc_id = aws_vpc.vpc.id

 route {
   cidr_block     = "0.0.0.0/0"
   nat_gateway_id = aws_nat_gateway.ngw.id
 }

 tags = {
```

```
   Name = var.private-rt-name
   env  = var.env
 }

 depends_on = [aws_vpc.vpc,
 ]
}

resource "aws_route_table_association" "private-rt-association" {
 count          = 3
 route_table_id = aws_route_table.private-rt.id
 subnet_id      = aws_subnet.private-subnet[count.index].id

 depends_on = [aws_vpc.vpc,
   aws_subnet.private-subnet
 ]
}

resource "aws_security_group" "eks-cluster-sg" {
 name        = var.eks-sg
 description = "Allow 443 from Jump Server only"

 vpc_id = aws_vpc.vpc.id

 ingress {
   from_port   = 443
   to_port     = 443
   protocol    = "tcp"
   cidr_blocks = ["0.0.0.0/0"] // It should be specific IP range
 }

 egress {
   from_port   = 0
   to_port     = 0
   protocol    = "-1"
   cidr_blocks = ["0.0.0.0/0"]
 }

 tags = {
   Name = var.eks-sg
 }
}
```

iam.tf

In the IAM file, we have created roles for our EKS cluster and node group. Along with that, we have created an OIDC role for our EKS Cluster.

Note: EKS cluster policy mostly remains the same which is AmazonEKSClusterPolicy but you can modify or attach other policies for node groups as they need other permissions according to your applications

```
locals {
 cluster_name = var.cluster-name
}

resource "random_integer" "random_suffix" {
 min = 1000
 max = 9999
}

resource "aws_iam_role" "eks-cluster-role" {
 count = var.is_eks_role_enabled ? 1 : 0
 name  =
"${local.cluster_name}-role-${random_integer.random_suffix.result}"

 assume_role_policy = jsonencode({
   Version = "2012-10-17"
   Statement = [{
     Effect = "Allow"
     Principal = {
       Service = "eks.amazonaws.com"
     }
     Action = "sts:AssumeRole"
   }]
 })
}

resource "aws_iam_role_policy_attachment" "AmazonEKSClusterPolicy" {
 count      = var.is_eks_role_enabled ? 1 : 0
 policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
 role       = aws_iam_role.eks-cluster-role[count.index].name
}
```

```
resource "aws_iam_role" "eks-nodegroup-role" {
 count = var.is_eks_nodegroup_role_enabled ? 1 : 0
 name  =
"${local.cluster_name}-nodegroup-role-${random_integer.random_suffix.result
}"

 assume_role_policy = jsonencode({
   Version = "2012-10-17"
   Statement = [{
     Action = "sts:AssumeRole"
     Effect = "Allow"
     Principal = {
       Service = "ec2.amazonaws.com"
     }
   }]
 })
}

resource "aws_iam_role_policy_attachment" "eks-AmazonWorkerNodePolicy" {
 count      = var.is_eks_nodegroup_role_enabled ? 1 : 0
 policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
 role       = aws_iam_role.eks-nodegroup-role[count.index].name
}

resource "aws_iam_role_policy_attachment" "eks-AmazonEKS_CNI_Policy" {
 count      = var.is_eks_nodegroup_role_enabled ? 1 : 0
 policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
 role       = aws_iam_role.eks-nodegroup-role[count.index].name
}
resource "aws_iam_role_policy_attachment"
"eks-AmazonEC2ContainerRegistryReadOnly" {
 count      = var.is_eks_nodegroup_role_enabled ? 1 : 0
 policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
 role       = aws_iam_role.eks-nodegroup-role[count.index].name
}

# OIDC
resource "aws_iam_role" "eks_oidc" {
 assume_role_policy =
data.aws_iam_policy_document.eks_oidc_assume_role_policy.json
 name               = "eks-oidc"
}
```

```
resource "aws_iam_policy" "eks-oidc-policy" {
 name = "test-policy"

 policy = jsonencode({
   Statement = [{
     Action = [
       "s3:ListAllMyBuckets",
       "s3:GetBucketLocation",
       "*"
     ]
     Effect   = "Allow"
     Resource = "*"
   }]
   Version = "2012-10-17"
 })
}

resource "aws_iam_role_policy_attachment" "eks-oidc-policy-attach" {
 role       = aws_iam_role.eks_oidc.name
 policy_arn = aws_iam_policy.eks-oidc-policy.arn
}
```

eks.tf

In the EKS file, we have configured private AWS EKS cluster and Private node groups.

If you read the configurations, we are creating two types of node groups. The first is ON-DEMAND type and the other is SPOT type to optimize the cloud.

```
resource "aws_eks_cluster" "eks" {

 count    = var.is-eks-cluster-enabled == true ? 1 : 0
 name     = var.cluster-name
 role_arn = aws_iam_role.eks-cluster-role[count.index].arn
 version  = var.cluster-version

 vpc_config {
   subnet_ids              = [aws_subnet.private-subnet[0].id,
aws_subnet.private-subnet[1].id, aws_subnet.private-subnet[2].id]
   endpoint_private_access = var.endpoint-private-access
   endpoint_public_access  = var.endpoint-public-access
   security_group_ids      = [aws_security_group.eks-cluster-sg.id]
```

```
 }


 access_config {
   authentication_mode                         = "CONFIG_MAP"
   bootstrap_cluster_creator_admin_permissions = true
 }

 tags = {
   Name = var.cluster-name
   Env  = var.env
 }
}

# OIDC Provider
resource "aws_iam_openid_connect_provider" "eks-oidc" {
 client_id_list  = ["sts.amazonaws.com"]
 thumbprint_list =
[data.tls_certificate.eks-certificate.certificates[0].sha1_fingerprint]
 url             = data.tls_certificate.eks-certificate.url
}


# AddOns for EKS Cluster
resource "aws_eks_addon" "eks-addons" {
 for_each      = { for idx, addon in var.addons : idx => addon }
 cluster_name  = aws_eks_cluster.eks[0].name
 addon_name    = each.value.name
 addon_version = each.value.version

 depends_on = [
   aws_eks_node_group.ondemand-node,
   aws_eks_node_group.spot-node
 ]
}

# NodeGroups
resource "aws_eks_node_group" "ondemand-node" {
 cluster_name    = aws_eks_cluster.eks[0].name
 node_group_name = "${var.cluster-name}-on-demand-nodes"

 node_role_arn = aws_iam_role.eks-nodegroup-role[0].arn
```

```
 scaling_config {
   desired_size = var.desired_capacity_on_demand
   min_size     = var.min_capacity_on_demand
   max_size     = var.max_capacity_on_demand
 }


 subnet_ids = [aws_subnet.private-subnet[0].id,
aws_subnet.private-subnet[1].id, aws_subnet.private-subnet[2].id]

 instance_types = var.ondemand_instance_types
 capacity_type  = "ON_DEMAND"
 labels = {
   type = "ondemand"
 }

 update_config {
   max_unavailable = 1
 }
 tags = {
   "Name" = "${var.cluster-name}-ondemand-nodes"
 }

 depends_on = [aws_eks_cluster.eks]
}

resource "aws_eks_node_group" "spot-node" {
 cluster_name    = aws_eks_cluster.eks[0].name
 node_group_name = "${var.cluster-name}-spot-nodes"

 node_role_arn = aws_iam_role.eks-nodegroup-role[0].arn

 scaling_config {
   desired_size = var.desired_capacity_spot
   min_size     = var.min_capacity_spot
   max_size     = var.max_capacity_spot
 }


 subnet_ids = [aws_subnet.private-subnet[0].id,
aws_subnet.private-subnet[1].id, aws_subnet.private-subnet[2].id]

 instance_types = var.spot_instance_types
```

```
 capacity_type  = "SPOT"

 update_config {
   max_unavailable = 1
 }
 tags = {
   "Name" = "${var.cluster-name}-spot-nodes"
 }
 labels = {
   type      = "spot"
   lifecycle = "spot"
 }
 disk_size = 50

 depends_on = [aws_eks_cluster.eks]
}
```

variables.tf

In the variables file, all the variables are defined which are required in the services files(iam, vpc, eks, etc)

```
variable "cluster-name" {}
variable "cidr-block" {}
variable "vpc-name" {}
variable "env" {}
variable "igw-name" {}
variable "pub-subnet-count" {}
variable "pub-cidr-block" {
 type = list(string)
}
variable "pub-availability-zone" {
 type = list(string)
}
variable "pub-sub-name" {}
variable "pri-subnet-count" {}
variable "pri-cidr-block" {
 type = list(string)
}
variable "pri-availability-zone" {
 type = list(string)
```

```
}
variable "pri-sub-name" {}
variable "public-rt-name" {}
variable "private-rt-name" {}
variable "eip-name" {}
variable "ngw-name" {}
variable "eks-sg" {}

#IAM
variable "is_eks_role_enabled" {
 type = bool
}
variable "is_eks_nodegroup_role_enabled" {
 type = bool
}

# EKS
variable "is-eks-cluster-enabled" {}
variable "cluster-version" {}
variable "endpoint-private-access" {}
variable "endpoint-public-access" {}
variable "addons" {
 type = list(object({
   name    = string
   version = string
 }))
}
variable "ondemand_instance_types" {}
variable "spot_instance_types" {}
variable "desired_capacity_on_demand" {}
variable "min_capacity_on_demand" {}
variable "max_capacity_on_demand" {}
variable "desired_capacity_spot" {}
variable "min_capacity_spot" {}
variable "max_capacity_spot" {}
```

Now, all the resources have been configured. To create them, we need to create a module of it.

Let’s do it

backend.tf

In the backend file, we configured our tfstate file to store it on an AWS S3 bucket. Also, we added tfstate file locking using DynamoDB to avoid multiple deployments/terraform <command> at one time.

Note: You need to create the S3 bucket and DynamoDB table manually from your console. Then, change the name of both services in the below configuration file accordingly. While creating the DynamoDB table you need to provide the Partition key which will be LockID with type default(string).

```
terraform {
 required_version = "~> 1.8.4"
 required_providers {
   aws = {
     source  = "hashicorp/aws"
     version = "~> 5.49.0"
   }
 }
 backend "s3" {
   bucket         = "my-ews-baket1"
   region         = "us-east-1"
   key            = "eks/terraform.tfstate"
   dynamodb_table = "Lock-Files"
   encrypt        = true
 }
}

provider "aws" {
 region  = var.aws-region
}
```

main.tf

In the main file, we are calling the services from the ../module directory. Still, we are using variables because we are going to use the tfvars file to initialize the variable. Try to follow the same approach for your multiple environments.

```
locals {
 org = "medium"
 env = var.env
}
```

```
module "eks" {
 source = "../module"

 env                    = var.env
 cluster-name           = "${local.env}-${local.org}-${var.cluster-name}"
 cidr-block             = var.vpc-cidr-block
 vpc-name               = "${local.env}-${local.org}-${var.vpc-name}"
 igw-name               = "${local.env}-${local.org}-${var.igw-name}"
 pub-subnet-count       = var.pub-subnet-count
 pub-cidr-block         = var.pub-cidr-block
 pub-availability-zone  = var.pub-availability-zone
 pub-sub-name           = "${local.env}-${local.org}-${var.pub-sub-name}"
 pri-subnet-count       = var.pri-subnet-count
 pri-cidr-block         = var.pri-cidr-block
 pri-availability-zone  = var.pri-availability-zone
 pri-sub-name           = "${local.env}-${local.org}-${var.pri-sub-name}"
 public-rt-name         = "${local.env}-${local.org}-${var.public-rt-name}"
 private-rt-name        = "${local.env}-${local.org}-${var.private-rt-name}"
 eip-name               = "${local.env}-${local.org}-${var.eip-name}"
 ngw-name               = "${local.env}-${local.org}-${var.ngw-name}"
 eks-sg                 = var.eks-sg

 is_eks_role_enabled           = true
 is_eks_nodegroup_role_enabled = true
 ondemand_instance_types       = var.ondemand_instance_types
 spot_instance_types           = var.spot_instance_types
 desired_capacity_on_demand    = var.desired_capacity_on_demand
 min_capacity_on_demand        = var.min_capacity_on_demand
 max_capacity_on_demand        = var.max_capacity_on_demand
 desired_capacity_spot         = var.desired_capacity_spot
 min_capacity_spot             = var.min_capacity_spot
 max_capacity_spot             = var.max_capacity_spot
 is-eks-cluster-enabled        = var.is-eks-cluster-enabled
 cluster-version               = var.cluster-version
 endpoint-private-access       = var.endpoint-private-access
 endpoint-public-access        = var.endpoint-public-access

 addons = var.addons
}
```

variables.tfvars

In the variables.tfvars file, we have initialized the value of each variable. So, if you want to check the number of services you can do that accordingly. Also, if you want to deploy the same services in multiple environments then you can create a new tfvars file as dev.tfvars. Then, create a different backend.tf file and apply the changes with different tfvars files.

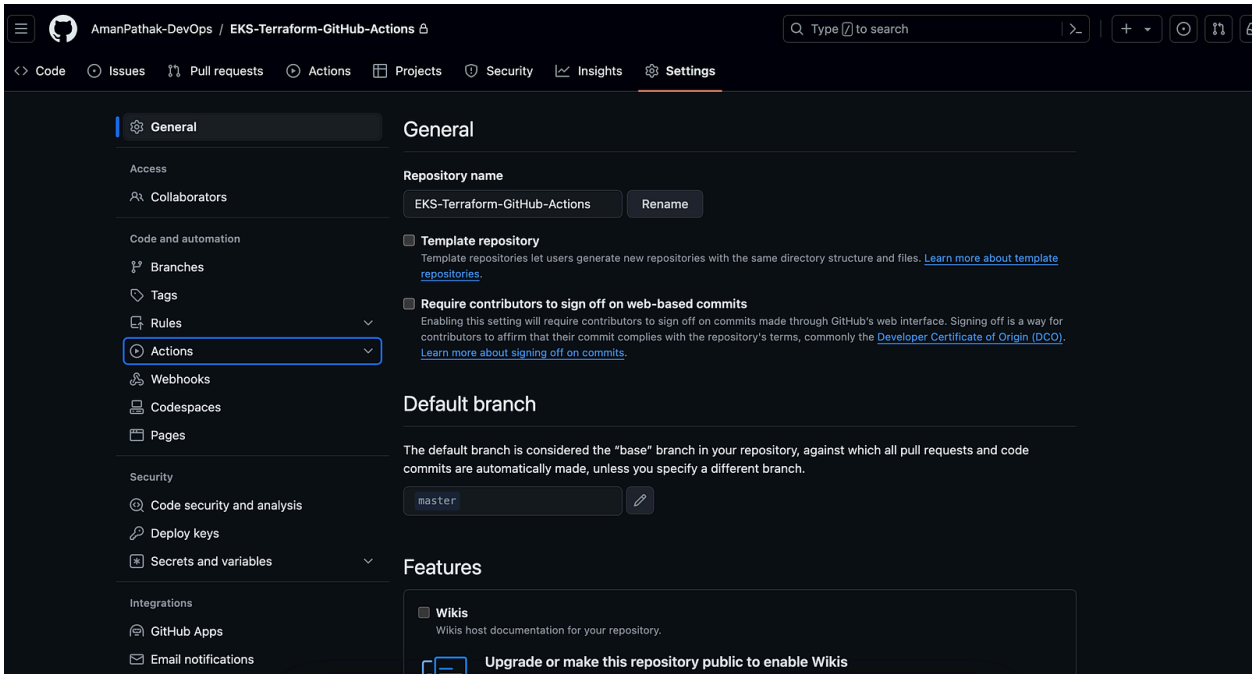
Note: Don't be confused about multiple environments if you don't have any idea about that.

```
env                  = "dev"
aws-region           = "us-east-1"
vpc-cidr-block       = "10.16.0.0/16"
vpc-name             = "vpc"
igw-name             = "igw"
pub-subnet-count     = 3
pub-cidr-block       = ["10.16.0.0/20", "10.16.16.0/20", "10.16.32.0/20"]
pub-availability-zone = ["us-east-1a", "us-east-1b", "us-east-1c"]
pub-sub-name         = "subnet-public"
pri-subnet-count     = 3
pri-cidr-block       = ["10.16.128.0/20", "10.16.144.0/20",
"10.16.160.0/20"]
pri-availability-zone = ["us-east-1a", "us-east-1b", "us-east-1c"]
pri-sub-name         = "subnet-private"
public-rt-name       = "public-route-table"
private-rt-name      = "private-route-table"
eip-name             = "elasticip-ngw"
ngw-name             = "ngw"
eks-sg               = "eks-sg"

# EKS
is-eks-cluster-enabled     = true
cluster-version            = "1.29"
cluster-name               = "eks-cluster"
endpoint-private-access    = true
endpoint-public-access     = false
ondemand_instance_types    = ["t3a.medium"]
spot_instance_types        = ["c5a.large", "c5a.xlarge", "m5a.large",
"m5a.xlarge", "c5.large", "m5.large", "t3a.large", "t3a.xlarge",
"t3a.medium"]
desired_capacity_on_demand = "1"
min_capacity_on_demand     = "1"
max_capacity_on_demand     = "5"
desired_capacity_spot      = "1"
min_capacity_spot          = "1"
```

```
max_capacity_spot          = "10"
addons = [
 {
   name    = "vpc-cni",
   version = "v1.18.1-eksbuild.1"
 },
 {
   name    = "coredns"
   version = "v1.11.1-eksbuild.9"
 },
 {
   name    = "kube-proxy"
   version = "v1.29.3-eksbuild.2"
 },
 {
   name    = "aws-ebs-csi-driver"
   version = "v1.30.0-eksbuild.1"
 }
 # Add more addons as needed
]
```

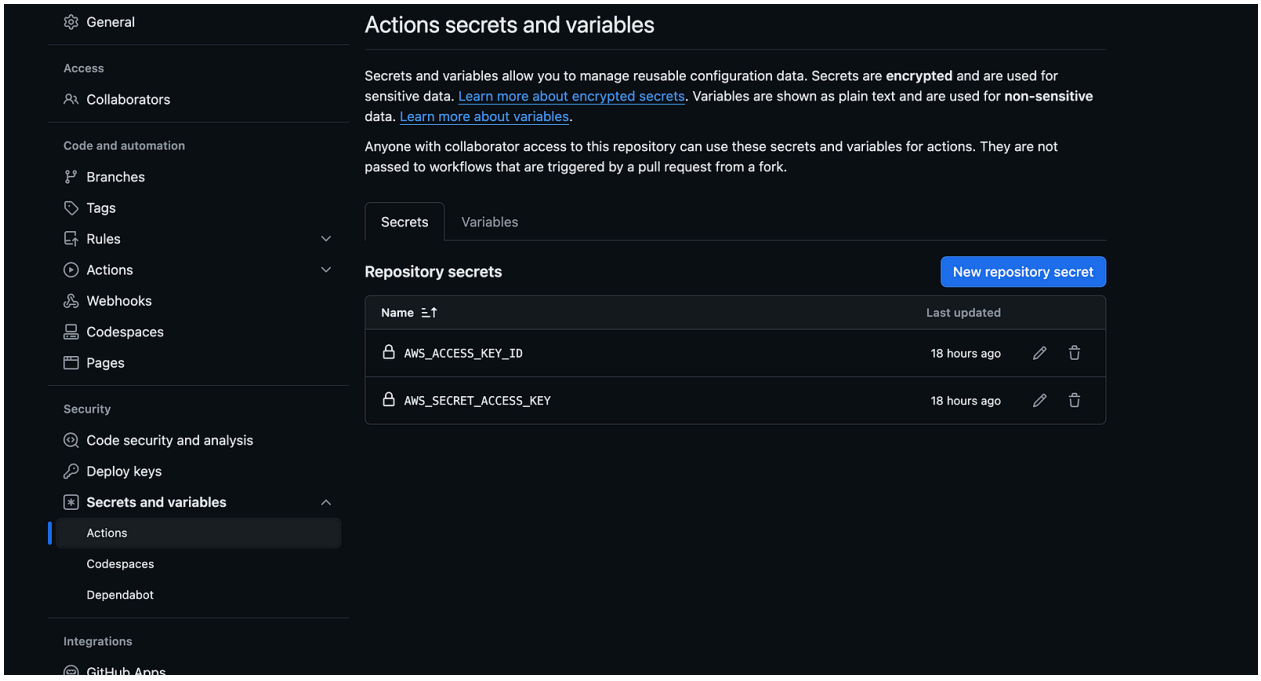
We are ready with our Terraform configurations for our EKS Cluster. Now, we need to automate the deployment using GitHub Actions. To do that, the first step is to add AWS secret key and secret access key in GitHub repository.

Navigate to the settings of your GitHub repository and click on **Secrets and variables**



Click on **Actions** and Add your keys in the secrets section.

Note: The name of the keys will remain as it is shown in the below snippet

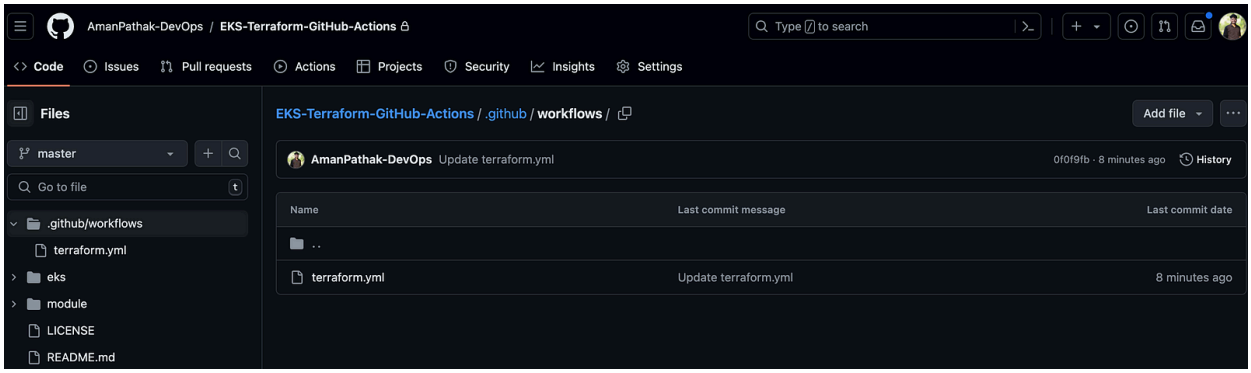


Once you add the AWS access key and secret access key. Our next aim is to create a workflow to deploy the infrastructure on AWS Cloud.

To do that, you need to create a directory which is .github/workflows inside your repository.

Inside the workflows directory, you can create workflows. Currently, we are creating terraform.yaml file

Note: Workflows will be written in YAML format



Below is the Workflow YAML script

The Workflow will trigger manually and add two parameters (tfvars file name and apply or destroy action)

```yaml
name: 'Terraform'

on:
 workflow_dispatch:
   inputs:
     tfvars_file:
       description: 'Path to the .tfvars file'
       required: true
       default: 'variables.tfvars'
     action:
       type: choice
       description: 'Apply or Destroy'
       options:
       - plan
       - apply
       - destroy
       required: true
       default: 'apply'

env:
     AWS_REGION: us-east-1
```

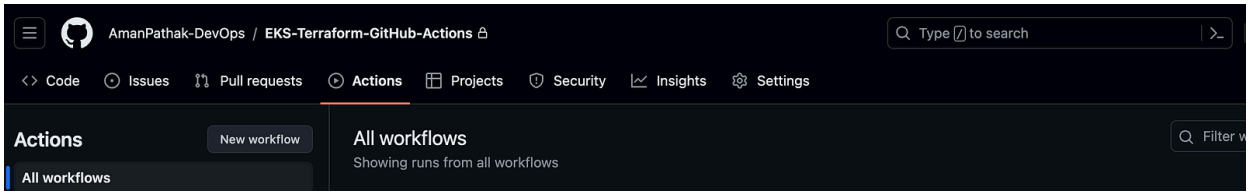
```
    AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
    AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

permissions:
 contents: read

jobs:


 CheckOut-Repo:
   runs-on: ubuntu-latest
   environment: production

   defaults:
     run:
       shell: bash
       working-directory: eks

   env:
     AWS_REGION: us-east-1
     AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
     AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}


   steps:
   # Checkout the repository to the GitHub Actions runner
   - name: Checkout
     uses: actions/checkout@v4

 Setting-Up-Terraform:
   runs-on: ubuntu-latest

   needs: CheckOut-Repo

   steps:
   # Install the latest version of Terraform CLI and configure the
Terraform CLI configuration file with a Terraform Cloud user API token
   - name: Setup Terraform
     uses: hashicorp/setup-terraform@v1
     with:
       terraform_version: 1.8.4

 Terraform-Initializing:
```

```
    runs-on: ubuntu-latest

    needs: Setting-Up-Terraform

    steps:

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v1
      with:
        terraform_version: 1.8.4

    - name: Checkout repository
      uses: actions/checkout@v4

    # Initialize a new or existing Terraform working directory by creating
initial files, loading any remote state, downloading modules, etc.
    - name: Terraform Init
      working-directory: eks
      run: terraform init

  Terraform-Formatting-Validating:
    runs-on: ubuntu-latest
    needs: Terraform-Initializing

    steps:

    - name: Checkout repository
      uses: actions/checkout@v4

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v1
      with:
        terraform_version: 1.8.4

    - name: Terraform Init
      working-directory: eks
      run: terraform init

    # Checks that all Terraform configuration files adhere to a canonical
format
    - name: Terraform Format
      working-directory: eks
      run: terraform fmt
```

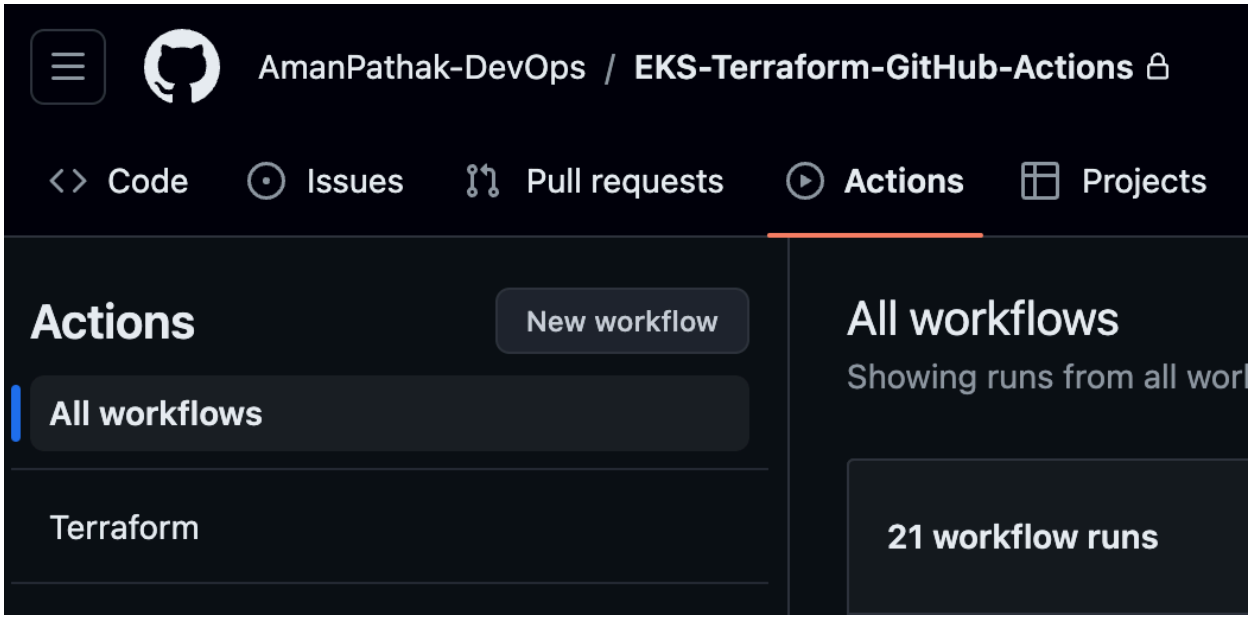
```
    - name: Terraform Validate
      working-directory: eks
      run: terraform validate

  Terraform-Action:
    runs-on: ubuntu-latest
    needs: Terraform-Formatting-Validating

    steps:

    - name: Checkout repository
      uses: actions/checkout@v4

    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v1
      with:
        terraform_version: 1.8.4

    - name: Terraform Init
      working-directory: eks
      run: terraform init


    - name: Terraform Plan
      if: ${{ github.event.inputs.action  == 'plan' }}
      working-directory: eks
      # Generates an execution plan for Terraform
      run: |
        terraform plan -var-file=${{ github.event.inputs.tfvars_file }}
-input=false


      # Apply the Terraform Configuration according to the parameter
    - name: Terraform Action
      if: ${{ github.event.inputs.action  == 'apply' }}
      working-directory: eks
      run: terraform ${{ github.event.inputs.action }} -auto-approve
-var-file=${{ github.event.inputs.tfvars_file }} -input=false

    - name: Terraform Destroy
      if: ${{ github.event.inputs.action == 'destroy' }}
      working-directory: eks
```

```
    run: terraform ${{ github.event.inputs.action }} -auto-approve
-var-file=${{ github.event.inputs.tfvars_file }} -input=false
```

To do that click on Actions

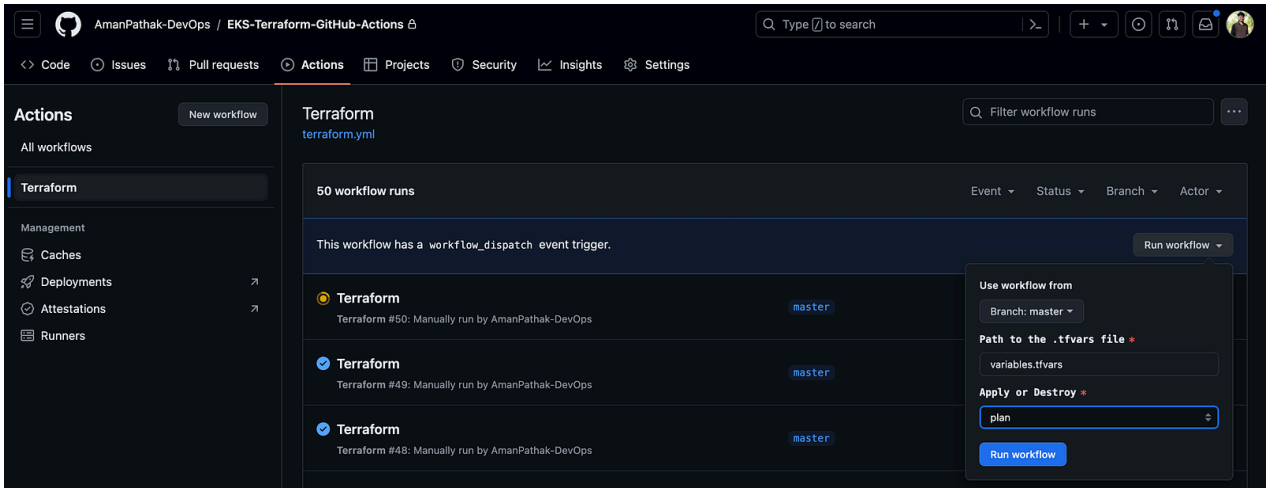


Click on **Terraform**

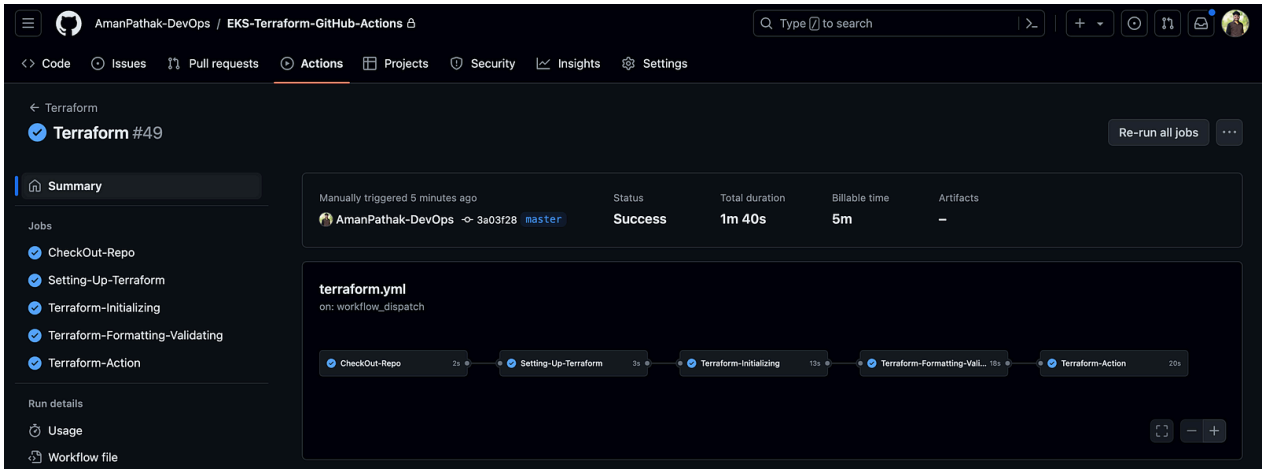


To run the workflow, you need to provide a parameter.

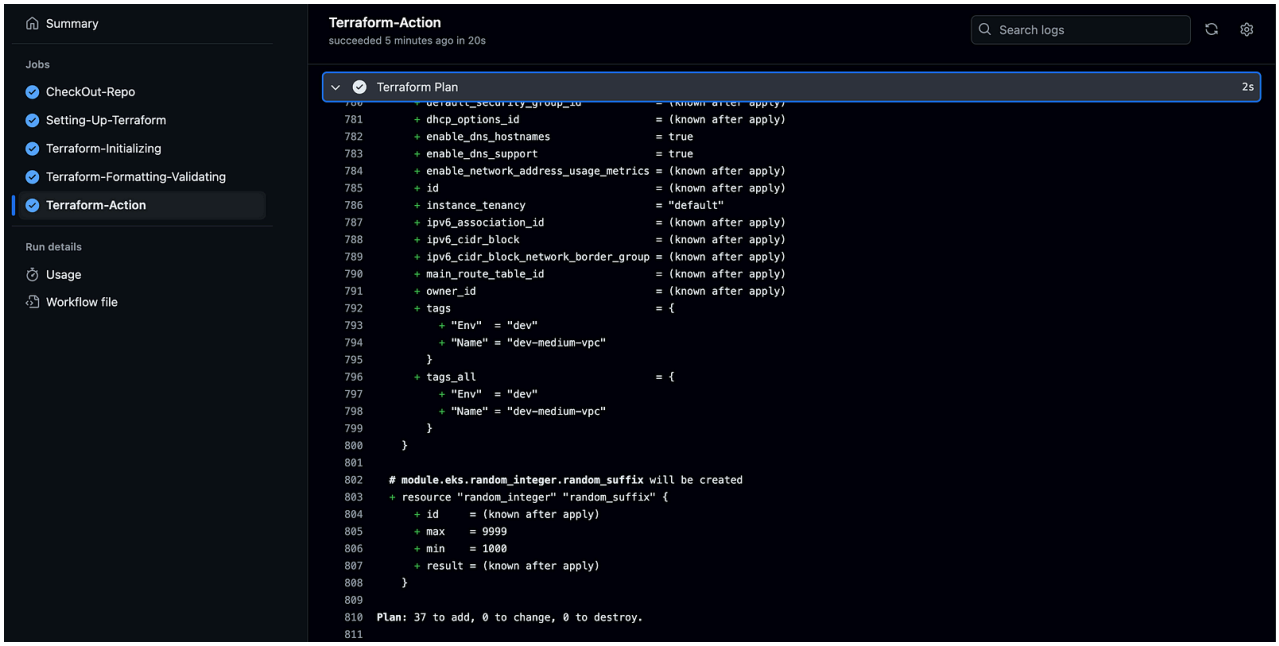
Click on **Run workflow** after providing the arguments(Initially we will run plan only).



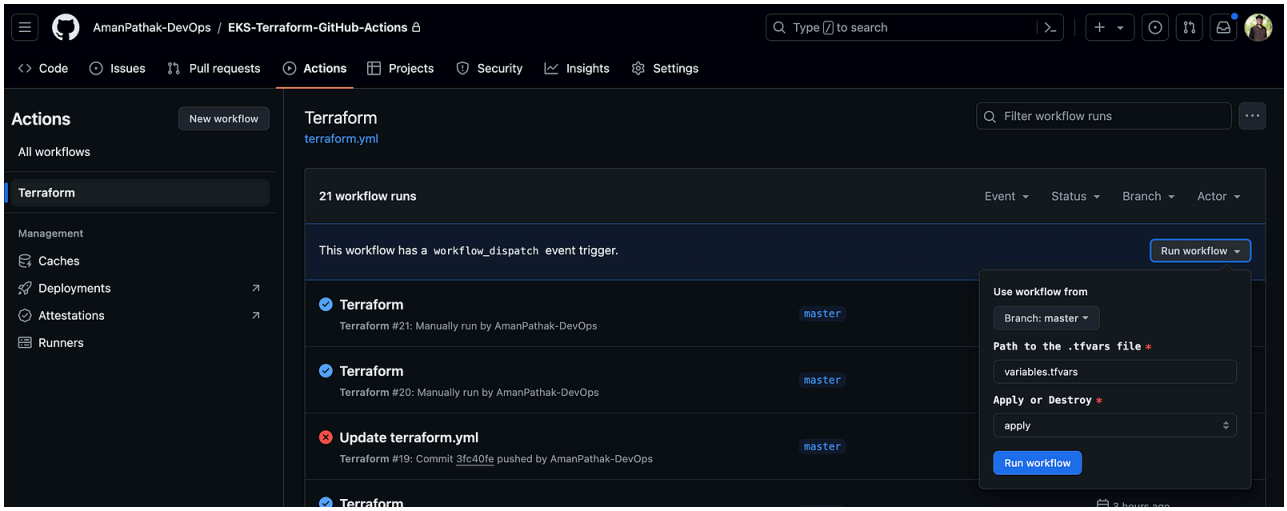
The Plan is Successful.



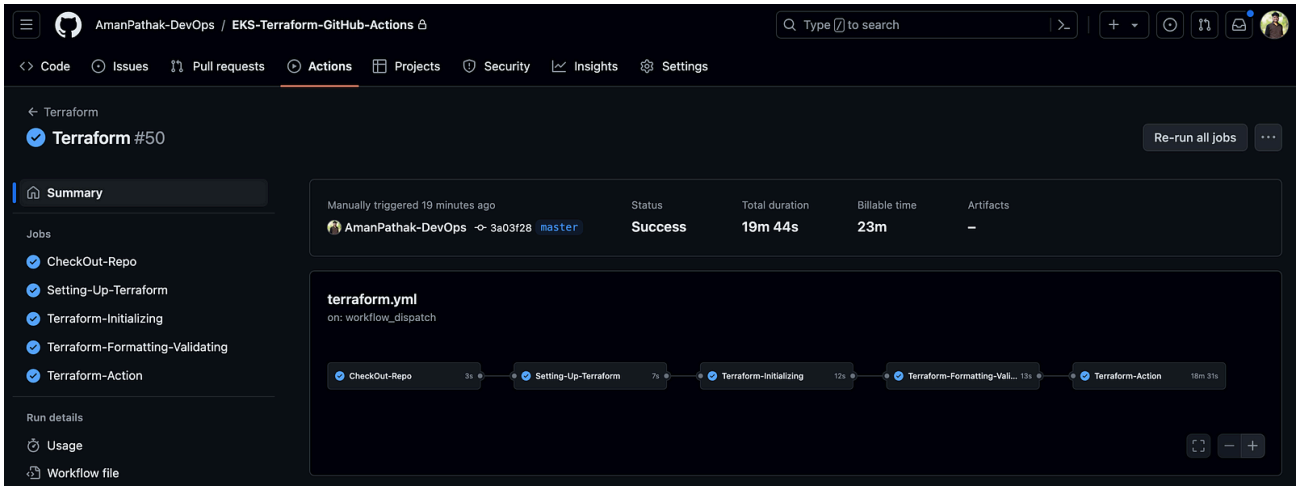
You can click on **Terraform-Action** to check the plan



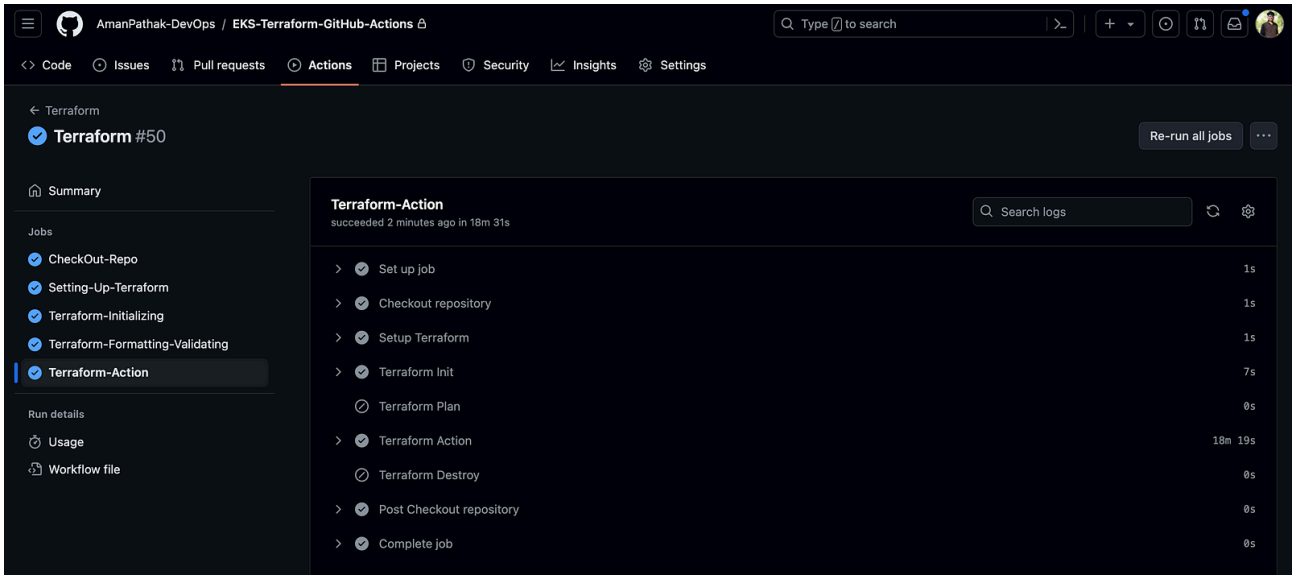
Now, We are ready to run apply



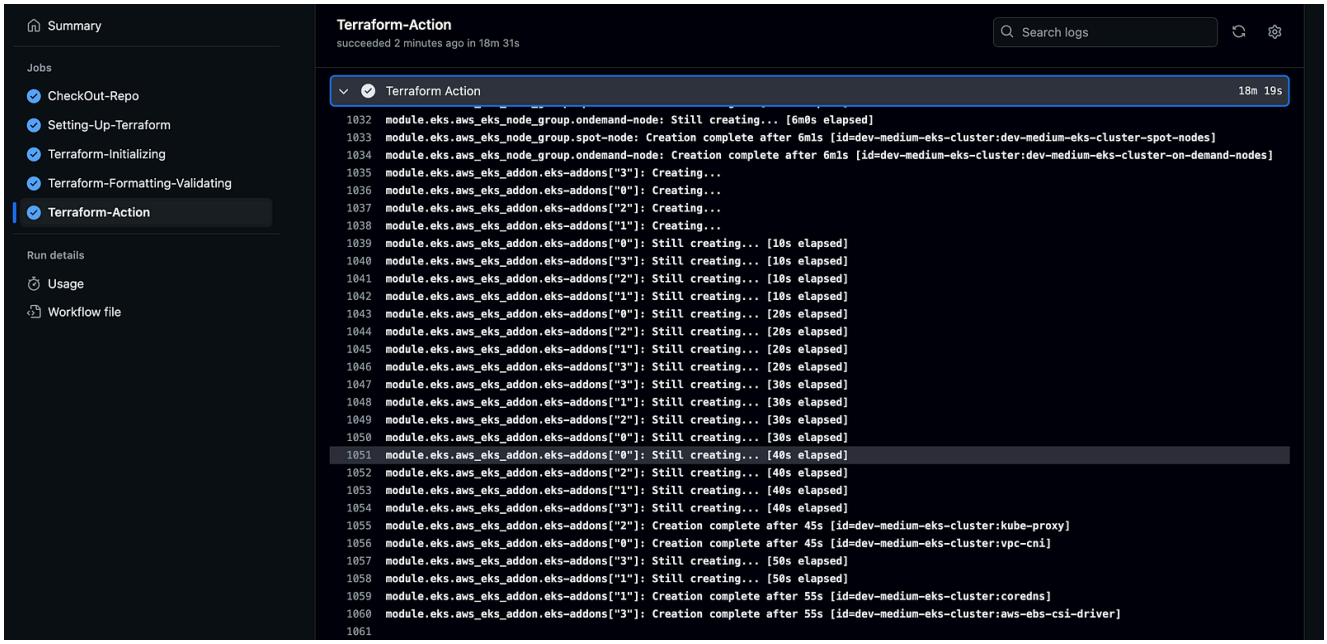
Once your workflow has been completed then, Click on **Terraform-Action** to view the logs



You can see each step in the below snippet properly. Click on the greater than (**>**) symbol to view the logs

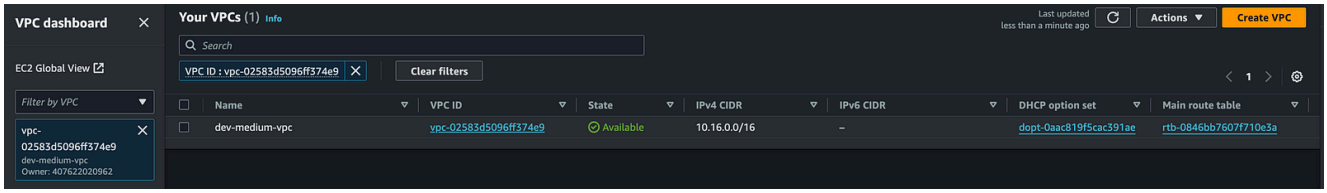


Workflow completed and it has created our EKS Cluster with nodegroups.

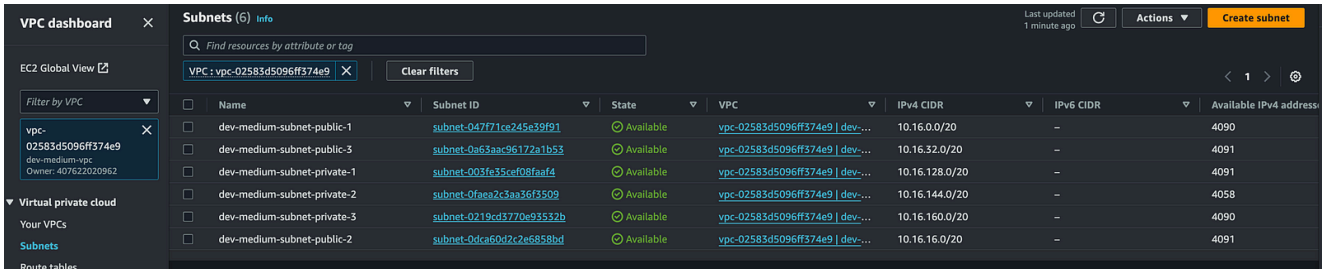


Let’s validate them by viewing them on Console

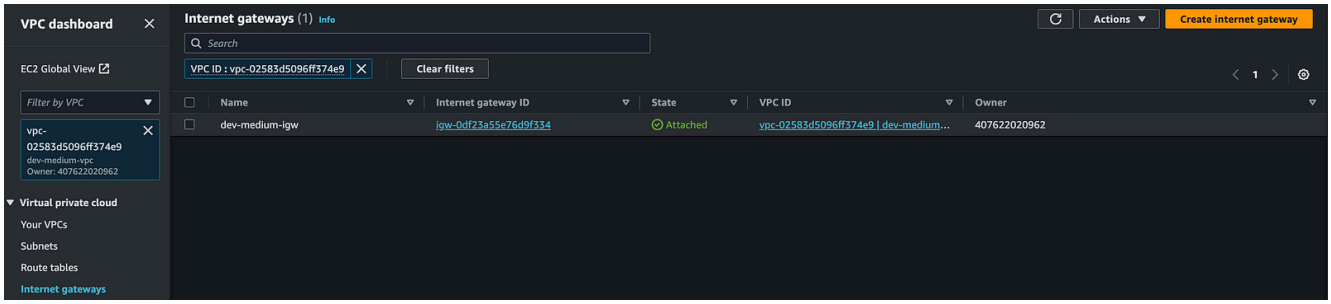
VPC



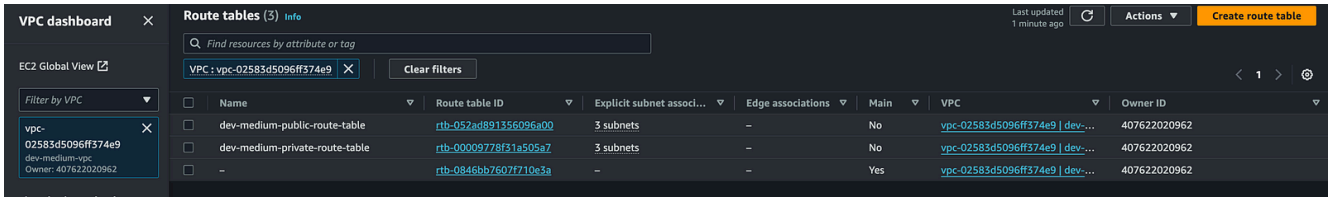
Public & Private Subnets



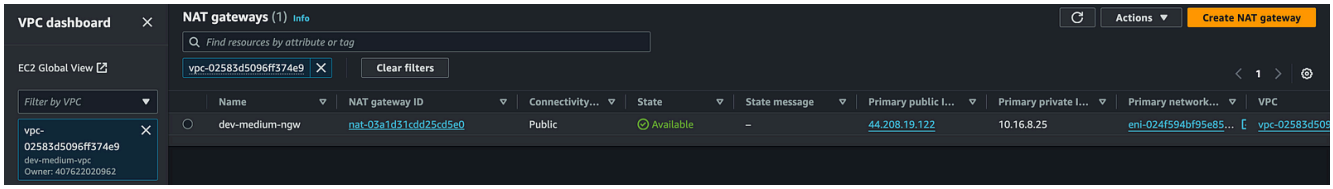
Internet Gateway



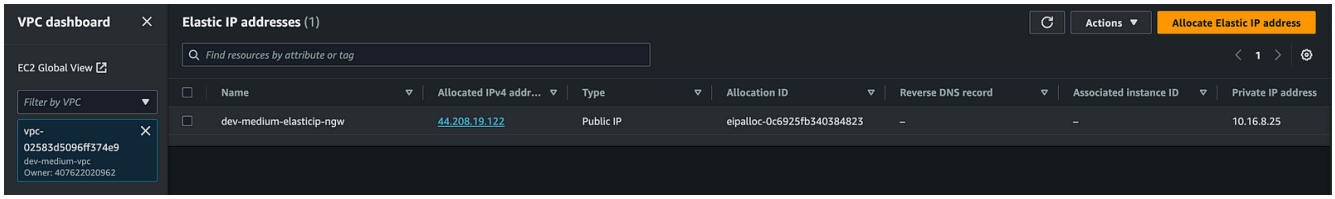
Route Table



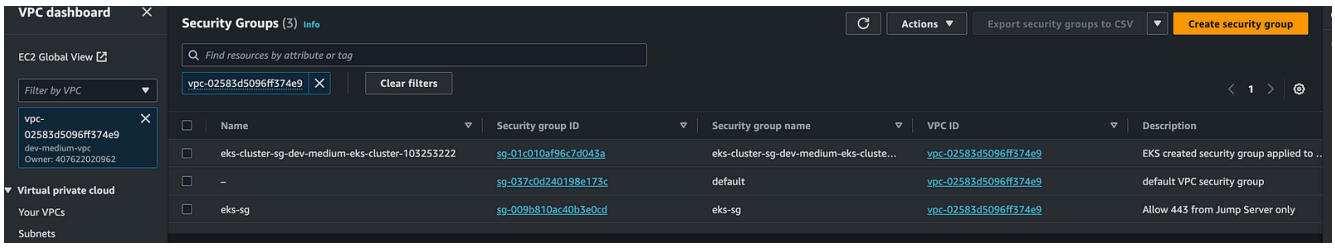
NAT Gateway



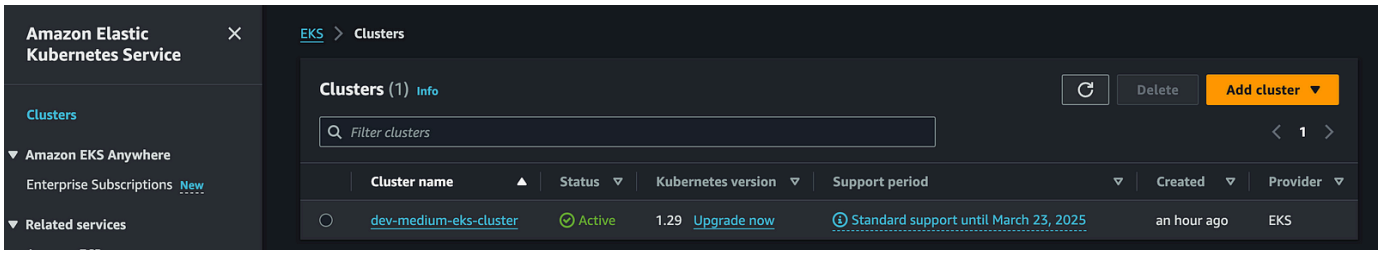
Elastic IP



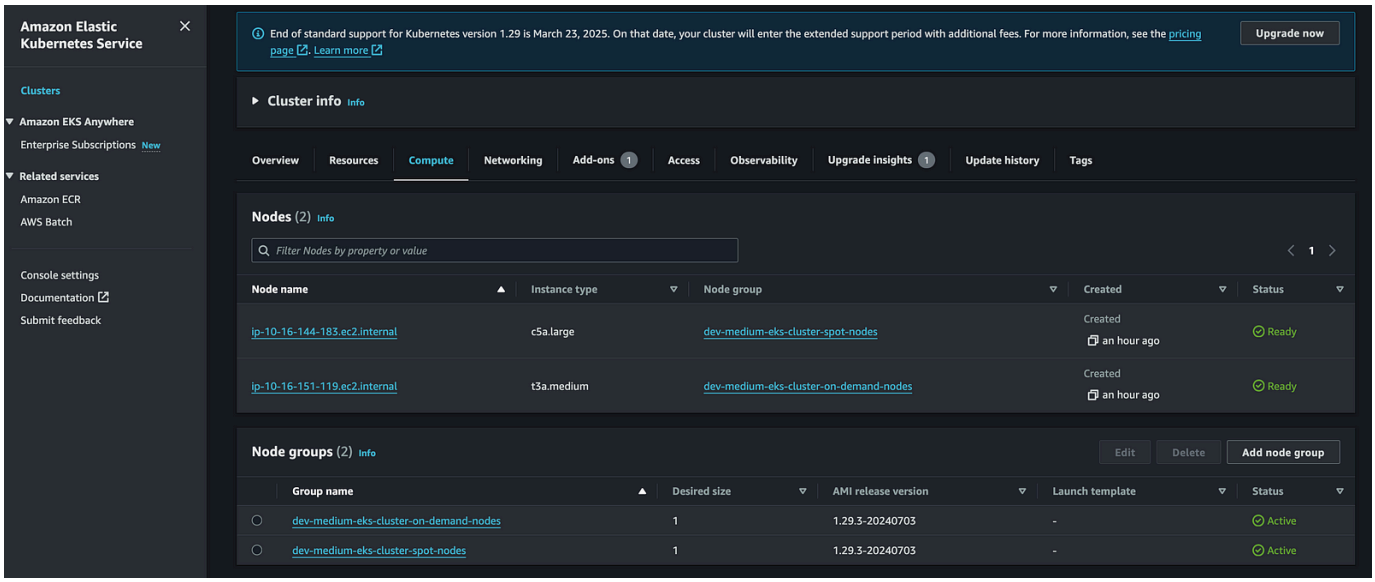
Security Group



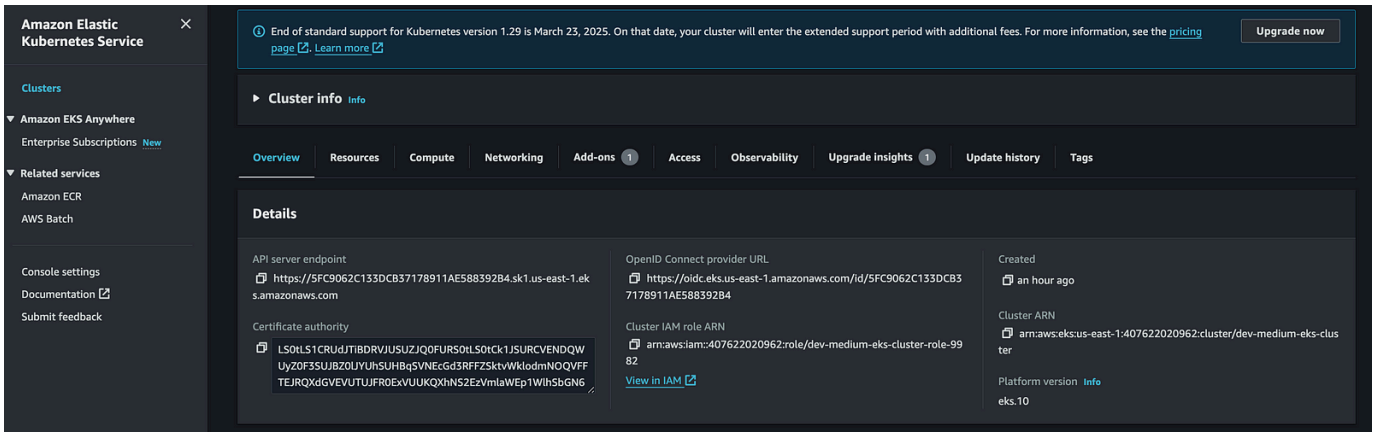
EKS Cluster



NodeGroups

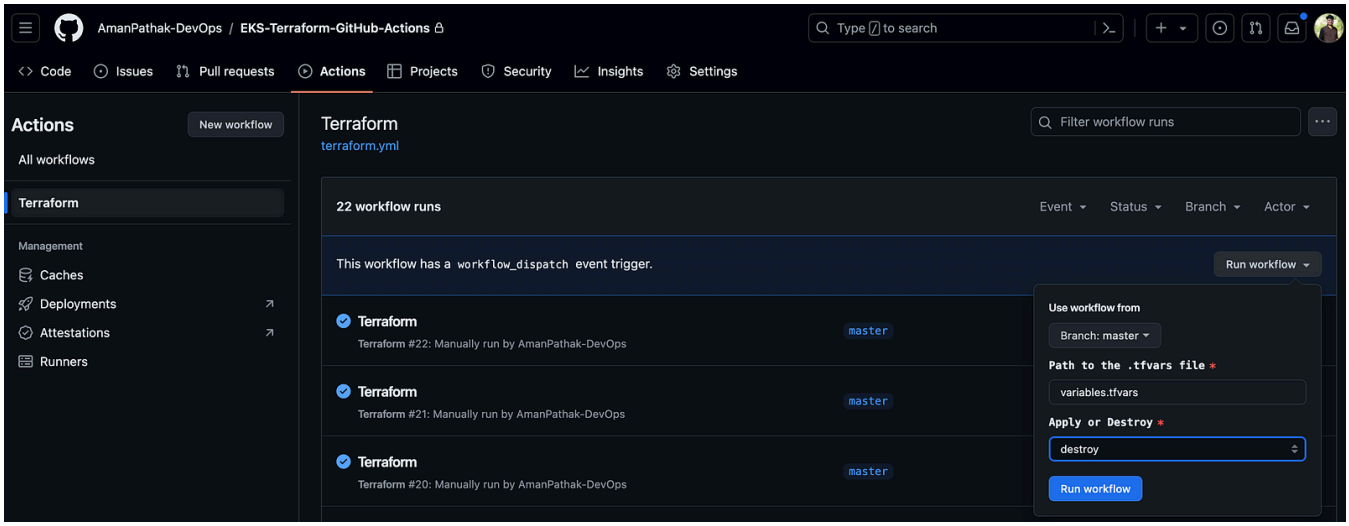


OIDC Connector

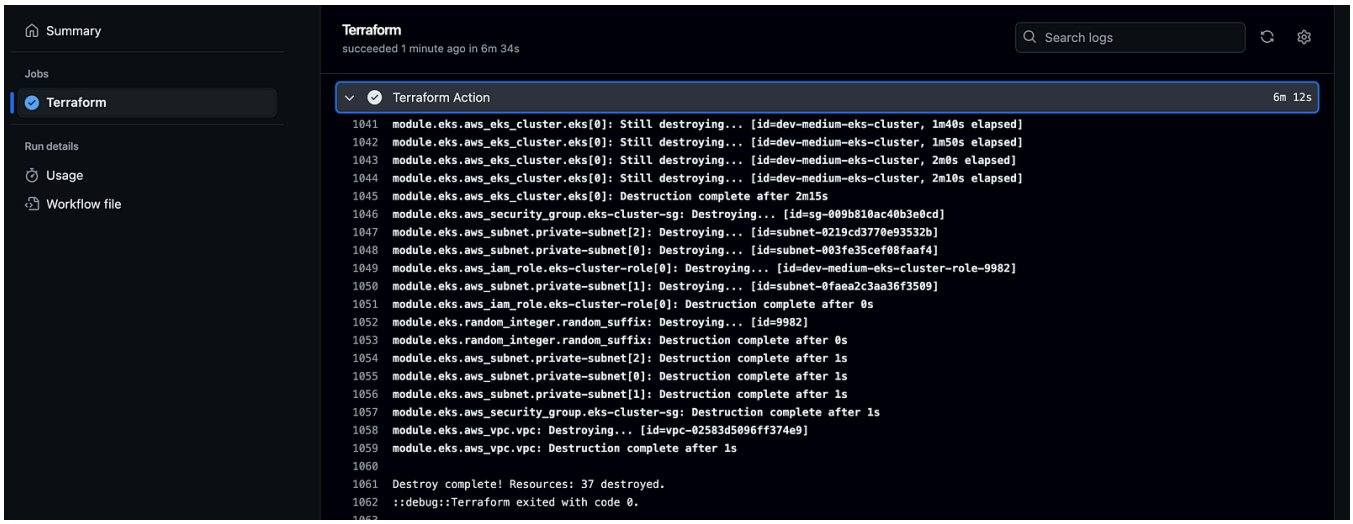


If you want to destroy the infrastructure, you can simply go to the **Actions** Section then, click on **Terraform** workflow

Note: To destroy, select the **destroy** option from the parameters



Here our workflow succeeded and services have been deleted.



That's it for now.

Hope you learned something new from this. I would recommend you create an EKS Cluster with required services from basics instead of using modules that are available on the internet. If you create each service yourself from scratch then, you get to know every service needed to create a Production-ready EKS Cluster.

Stay connected on **LinkedIn**: LinkedIn Profile

Stay up-to-date with **GitHub**: GitHub Profile

Want to discuss trending technologies in DevOps & Cloud
Join the **Discord Server**- https://discord.gg/jdzF8kTtw2

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

Happy Learning!