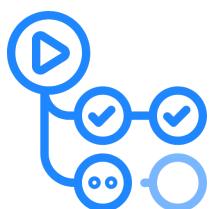
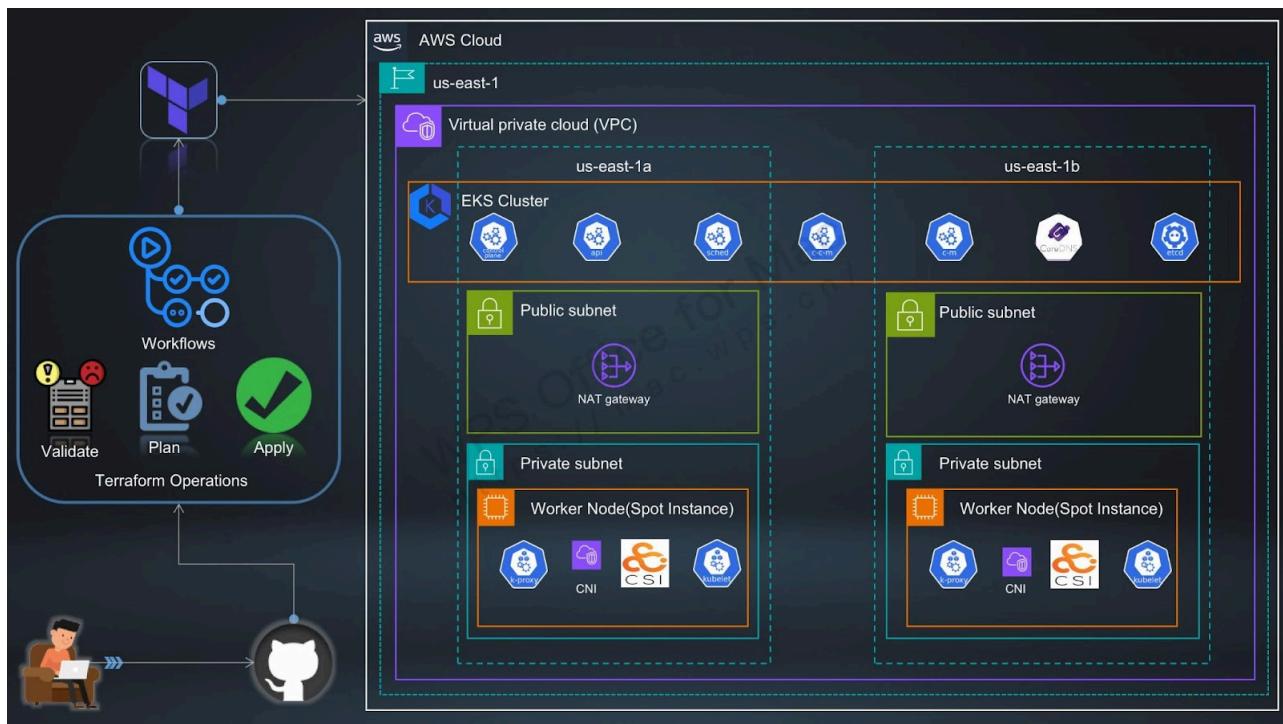


Configure Production-Ready EKS Clusters with Terraform and GitHub Actions



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-

```
ubuntu@ip-172-31-85-72:~$ tree EKS-Terraform-GitHub-Actions/
EKS-Terraform-GitHub-Actions/
├── LICENSE
├── README.md
└── eks
    ├── backend.tf
    ├── main.tf
    ├── variables.tf
    └── variables.tfvars
    └── module
        ├── eks.tf
        ├── gather.tf
        ├── iam.tf
        ├── variables.tf
        └── vpc.tf
2 directories, 11 files
```

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.on-demand_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.on-demand_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**

The screenshot shows the GitHub repository settings for 'AmanPathak-DevOps / EKS-Terraform-GitHub-Actions'. The 'General' tab is selected. On the left sidebar, under 'Code and automation', the 'Actions' option is highlighted with a blue border. In the main content area, there is a section titled 'Default branch' with 'master' selected. Below it is a 'Features' section containing a 'Wikis' option. A call-to-action button at the bottom right of this section says 'Upgrade or make this repository public to enable Wikis'.

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

The screenshot shows the GitHub repository settings for 'AmanPathak-DevOps / EKS-Terraform-GitHub-Actions'. The 'General' tab is selected. On the left sidebar, under 'Secrets and variables', the 'Actions' option is highlighted with a blue border. In the main content area, there is a section titled 'Actions secrets and variables'. It contains information about secrets and variables, stating that anyone with collaborator access can use them for actions. Below this is a 'Repository secrets' table with two entries:

Name	Last updated	Action
AWS_ACCESS_KEY_ID	18 hours ago	
AWS_SECRET_ACCESS_KEY	18 hours ago	

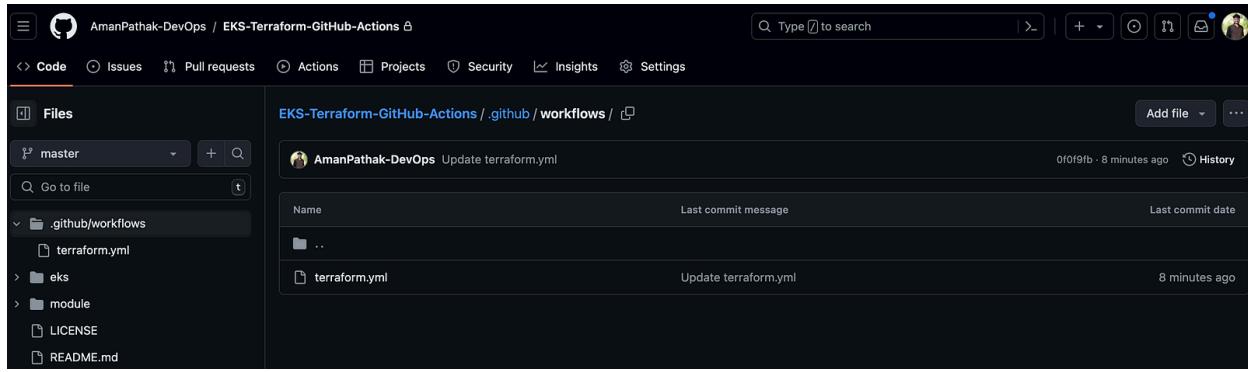
A blue 'New repository secret' button is located at the top right of the 'Repository secrets' table.

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)

```
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

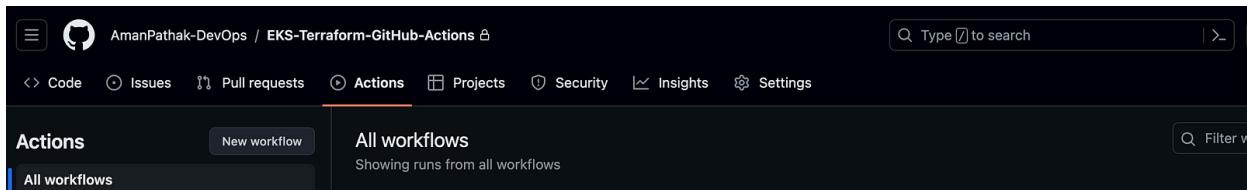
- name: Terraform Plan
  if: ${{ github.event.inputs.action == 'plan' }}
  working-directory: eks
  # Generates an execution plan for Terraform
  run: |
    terraform plan -var-file=${{ github.event.inputs.tfvars_file }}
    -input=false

    # 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

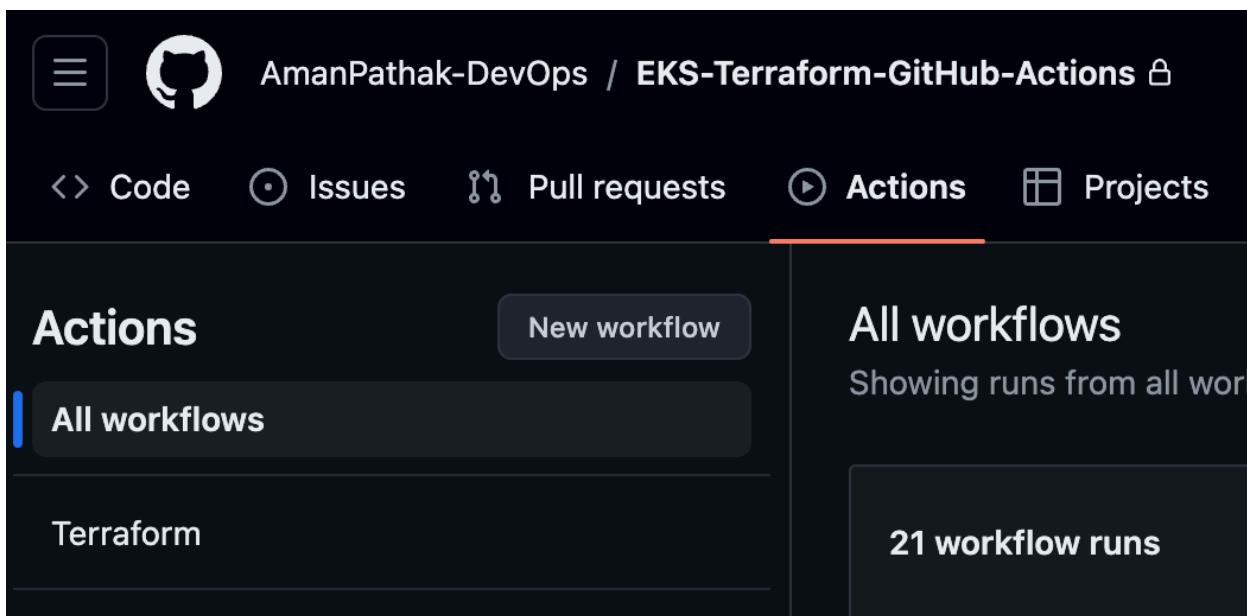
- name: Terraform Destroy
  if: ${{ github.event.inputs.action == 'destroy' }}
  working-directory: eks
```

```
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 screenshot shows the GitHub Actions interface for the repository 'AmanPathak-DevOps / EKS-Terraform-GitHub-Actions'. The 'Actions' tab is selected. On the left, a sidebar for 'Terraform' shows management options like Caches, Deployments, Attestations, and Runners. The main area displays '50 workflow runs' for the 'Terraform' workflow. A modal window is open, titled 'Run workflow', with the following configuration:

- Use workflow from: Branch: master
- Path to the .tfvars file: variables.tfvars
- Apply or Destroy: plan

A blue 'Run workflow' button is at the bottom of the modal.

The Plan is Successful.

The screenshot shows the GitHub Actions interface for the same repository, focusing on a specific workflow run labeled 'Terraform #49'. The 'Actions' tab is selected. The 'Summary' card for this run shows the following details:

Manually triggered 5 minutes ago	Status	Total duration	Billable time	Artifacts
3a03f28 master	Success	1m 40s	5m	-

The 'Jobs' section lists the steps of the workflow:

- CheckOut-Repo
- Setting-Up-Terraform
- Terraform-Initializing
- Terraform-Formatting-Validating
- Terraform-Action

Below the jobs, there are links for 'Run details', 'Usage', and 'Workflow file'. The 'terraform.yml' file is shown with its content: `on: workflow_dispatch`. A timeline diagram at the bottom illustrates the sequence of jobs and their durations.

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

```

Terraform-Action
succeeded 5 minutes ago in 20s

Terraform Plan
700+ default_security_group_x0 = (known after apply)
701+ dhcp_options_id           = (known after apply)
702+ enable_dns_hostnames      = true
703+ enable_dns_support         = true
704+ enable_network_address_usage_metrics = (known after apply)
705+ id                         = (known after apply)
706+ instance_tenancy          = "default"
707+ ipv6_association_id       = (known after apply)
708+ ipv6_cidr_block           = (known after apply)
709+ ipv6_cidr_block_network_border_group = (known after apply)
710+ main_route_table_id       = (known after apply)
711+ owner_id                  = (known after apply)
712+ tags                      = {
713+   "Env" = "dev"
714+   "Name" = "dev-medium-vpc"
715}
716+ tags_all                  = {
717+   "Env" = "dev"
718+   "Name" = "dev-medium-vpc"
719}
720}
721
722# module.eks.random_integer.random_suffix will be created
723+ resource "random_integer" "random_suffix" {
724+   id      = (known after apply)
725+   max    = 9999
726+   min    = 1000
727+   result = (known after apply)
728}
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810 Plan: 37 to add, 0 to change, 0 to destroy.
811

```

Now, We are ready to run apply

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

The screenshot shows the GitHub Actions interface for a workflow named 'Terraform #50'. The workflow was triggered manually 19 minutes ago by 'AmanPathak-DevOps' on branch 'master'. The status is 'Success' with a total duration of 19m 44s and a billable time of 23m. There are no artifacts. The workflow consists of five jobs: 'CheckOut-Repo', 'Setting-Up-Terraform', 'Terraform-Initializing', 'Terraform-Formatting-Validating', and 'Terraform-Action'. The 'Terraform-Action' job is expanded, showing its sub-steps: Set up job, Checkout repository, Setup Terraform, Terraform Init, Terraform Plan, Terraform Action (which took 18m 19s), Terraform Destroy, Post Checkout repository, and Complete job. Each step has a timestamp next to it.

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

This screenshot shows the detailed logs for the 'Terraform-Action' job from the previous workflow. The job succeeded 2 minutes ago in 18m 31s. The logs are listed as follows:

- > Set up job (1s)
- > Checkout repository (1s)
- > Setup Terraform (1s)
- > Terraform Init (7s)
- > Terraform Plan (0s)
- > Terraform Action (18m 19s)
- > Terraform Destroy (0s)
- > Post Checkout repository (0s)
- > Complete job (0s)

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

```

1032 module.eks.aws_eks_node_group.ondemand-node: Still creating... [6m0s elapsed]
1033 module.eks.aws_eks_node_group.spot-node: Creation complete after 6m1s [id=dev-medium-eks-cluster:dev-medium-eks-cluster-spot-nodes]
1034 module.eks.aws_eks_node_group.on-demand-node: Creation complete after 6m1s [id=dev-medium-eks-cluster:dev-medium-eks-cluster-on-demand-nodes]
1035 module.eks.aws_eksAddon.eks-addons["3"]: Creating...
1036 module.eks.aws_eksAddon.eks-addons["0"]: Creating...
1037 module.eks.aws_eksAddon.eks-addons["2"]: Creating...
1038 module.eks.aws_eksAddon.eks-addons["1"]: Creating...
1039 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [10s elapsed]
1040 module.eks.aws_eksAddon.eks-addons["3"]: Still creating... [10s elapsed]
1041 module.eks.aws_eksAddon.eks-addons["2"]: Still creating... [10s elapsed]
1042 module.eks.aws_eksAddon.eks-addons["1"]: Still creating... [10s elapsed]
1043 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [20s elapsed]
1044 module.eks.aws_eksAddon.eks-addons["2"]: Still creating... [20s elapsed]
1045 module.eks.aws_eksAddon.eks-addons["1"]: Still creating... [20s elapsed]
1046 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [20s elapsed]
1047 module.eks.aws_eksAddon.eks-addons["3"]: Still creating... [30s elapsed]
1048 module.eks.aws_eksAddon.eks-addons["1"]: Still creating... [30s elapsed]
1049 module.eks.aws_eksAddon.eks-addons["2"]: Still creating... [30s elapsed]
1050 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [30s elapsed]
1051 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [40s elapsed]
1052 module.eks.aws_eksAddon.eks-addons["2"]: Still creating... [40s elapsed]
1053 module.eks.aws_eksAddon.eks-addons["1"]: Still creating... [40s elapsed]
1054 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [40s elapsed]
1055 module.eks.aws_eksAddon.eks-addons["2"]: Creation complete after 45s [id=dev-medium-eks-cluster:kube-proxy]
1056 module.eks.aws_eksAddon.eks-addons["0"]: Creation complete after 45s [id=dev-medium-eks-cluster:vpc-cni]
1057 module.eks.aws_eksAddon.eks-addons["1"]: Still creating... [50s elapsed]
1058 module.eks.aws_eksAddon.eks-addons["0"]: Still creating... [50s elapsed]
1059 module.eks.aws_eksAddon.eks-addons["1"]: Creation complete after 55s [id=dev-medium-eks-cluster:coredns]
1060 module.eks.aws_eksAddon.eks-addons["2"]: Creation complete after 55s [id=dev-medium-eks-cluster:aws-ebs-csi-driver]
1061

```

Let's validate them by viewing them on Console

VPC

Your VPCs (1) Info							Last updated	Actions	Create VPC
							less than a minute ago		
EC2 Global View									
Filter by VPC									
vpc-02583d5096ff374e9	x	VPC ID : vpc-02583d5096ff374e9	x	Clear filters					
Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP option set	Main route table			
dev-medium-vpc	vpc-02583d5096ff374e9	Available	10.16.0.0/16	-	dopt-0aa819f5cac391ae	rte-0846bb7607f710e3a			

Public & Private Subnets

Subnets (6) Info							Last updated	Actions	Create subnet
							1 minute ago		
EC2 Global View									
Filter by VPC									
vpc-02583d5096ff374e9	x	VPC ID : vpc-02583d5096ff374e9	x	Clear filters					
Name	Subnet ID	State	VPC	IPv4 CIDR	IPv6 CIDR	Available IPv4 address			
dev-medium-subnet-public-1	subnet-047f71ce245e39f91	Available	vpc-02583d5096ff374e9 dev...	10.16.0.0/20	-	4090			
dev-medium-subnet-public-3	subnet-0a63aac96172a1b53	Available	vpc-02583d5096ff374e9 dev...	10.16.32.0/20	-	4091			
dev-medium-subnet-private-1	subnet-09fe35ce0f0fafaf4	Available	vpc-02583d5096ff374e9 dev...	10.16.128.0/20	-	4091			
dev-medium-subnet-private-2	subnet-0faee2c3aa36f3509	Available	vpc-02583d5096ff374e9 dev...	10.16.144.0/20	-	4058			
dev-medium-subnet-private-3	subnet-0219cd770be9552b	Available	vpc-02583d5096ff374e9 dev...	10.16.160.0/20	-	4090			
dev-medium-subnet-public-2	subnet-0dca60d2c2e6858bd	Available	vpc-02583d5096ff374e9 dev...	10.16.16.0/20	-	4091			

Internet Gateway

VPC dashboard

Internet gateways (1) Info

Actions ▾ Create internet gateway						
Search						
VPC ID : vpc-02583d5096ff374e9 X		Clear filters		< 1 >		
Filter by VPC	Name	Internet gateway ID	State	VPC ID	Owner	Actions
vpc-02583d5096ff374e9 dev-medium-vpc Owner: 407622020962	dev-medium-igw	igw-0df23a55e76dbf334	Attached	vpc-02583d5096ff374e9 dev-medium...	407622020962	

Virtual private cloud

- Your VPCs
- Subnets
- Route tables
- Internet gateways

Route Table

VPC dashboard

Route tables (3) Info

Actions ▾ Create route table						
Last updated 1 minute ago						
VPC ID : vpc-02583d5096ff374e9 X		Clear filters		< 1 >		
Filter by VPC	Name	Route table ID	Explicit subnet associ...	Edge associations	Main	VPC
vpc-02583d5096ff374e9 dev-medium-vpc Owner: 407622020962	dev-medium-public-route-table	rtb-052ad891356096a00	3 subnets	-	No	vpc-02583d5096ff374e9 dev-...
	dev-medium-private-route-table	rtb-0000977bf31a505a7	3 subnets	-	No	vpc-02583d5096ff374e9 dev-...
	-	rtb-08465b7607f710e3a	-	-	Yes	vpc-02583d5096ff374e9 dev-...

NAT Gateway

VPC dashboard

NAT gateways (1) Info

Actions ▾ Create NAT gateway						
Last updated 1 minute ago						
VPC ID : vpc-02583d5096ff374e9 X		Clear filters		< 1 >		
Filter by VPC	Name	NAT gateway ID	Connectivity...	State	State message	Primary public I...
vpc-02583d5096ff374e9 dev-medium-vpc Owner: 407622020962	dev-medium-ngw	nat-03a1d31cdd25cd5e0	Public	Available	-	44.208.19.122 10.16.8.25 eni-024f594bf95e85...

Elastic IP

VPC dashboard

Elastic IP addresses (1)

Actions ▾ Allocate Elastic IP address						
Last updated 1 minute ago						
VPC ID : vpc-02583d5096ff374e9 X		Clear filters		< 1 >		
Filter by VPC	Name	Allocated IPv4 addr...	Type	Allocation ID	Reverse DNS record	Associated instance ID
vpc-02583d5096ff374e9 dev-medium-vpc Owner: 407622020962	dev-medium-elasticip-ngw	44.208.19.122	Public IP	eipalloc-0c6925fb340384823	-	- 10.16.8.25

Security Group

VPC dashboard

Security Groups (3) Info

Actions ▾ Export security groups to CSV ▾ Create security group						
Last updated 1 minute ago						
VPC ID : vpc-02583d5096ff374e9 X		Clear filters		< 1 >		
Filter by VPC	Name	Security group ID	Security group name	VPC ID	Description	Actions
vpc-02583d5096ff374e9 dev-medium-vpc Owner: 407622020962	eks-cluster-sg-dev-medium-eks-cluster-103253222	sg-01c010af96c7d043a	eks-cluster-sg-dev-medium-eks-cluste...	vpc-02583d5096ff374e9	EKS created security group applied to ..	
	-	sg-037c0d240198e175c	default	vpc-02583d5096ff374e9	default VPC security group	
	eks-sg	sg-009b810ac40b3e0cd	eks-sg	vpc-02583d5096ff374e9	Allow 443 from Jump Server only	

EKS Cluster

Clusters (1) Info

Cluster name	Status	Kubernetes version	Support period	Created	Provider	
dev-medium-eks-cluster	Active	1.29	Upgrade now	Standard support until March 23, 2025	an hour ago	EKS

NodeGroups

Nodes (2) Info

Node name	Instance type	Node group	Created	Status
ip-10-16-144-183.ec2.internal	c5a.large	dev-medium-eks-cluster-spot-nodes	an hour ago	Ready
ip-10-16-151-119.ec2.internal	t3a.medium	dev-medium-eks-cluster-on-demand-nodes	an hour ago	Ready

Node groups (2) Info

Group name	Desired size	AMI release version	Launch template	Status
dev-medium-eks-cluster-on-demand-nodes	1	1.29.3-20240703	-	Active
dev-medium-eks-cluster-spot-nodes	1	1.29.3-20240703	-	Active

OIDC Connector

Details

API server endpoint https://5FC9062C133DCB37178911AE588392B4.us-east-1.amazonaws.com	OpenID Connect provider URL https://oidc.eks.us-east-1.amazonaws.com/id/5FC9062C133DCB37178911AE588392B4	Created an hour ago
Certificate authority LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0hLS0tCk1JSURCVENDQWUyZ0F3SUh20LYh5UHbq5VNEcg3RFzSktwVklodmNOQVFTEJRQXdGVEVUTUJFR0ExVUUQXhNS2EZvmlaWEp1WhSbGNG/	Cluster IAM role ARN arn:aws:iam::407622020962:role/dev-medium-eks-cluster-role-9982 View in IAM	Cluster ARN arn:aws:eks:us-east-1:407622020962:cluster/dev-medium-eks-cluster
		Platform version Info eks.10

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

The screenshot shows the GitHub Actions interface for a repository named 'AmanPathak-DevOps / EKS-Terraform-GitHub-Actions'. The 'Actions' tab is selected. On the left, a sidebar for 'Terraform' shows three runs: 'Terraform #22', 'Terraform #21', and 'Terraform #20', all manually run by the user. Each run is listed with its status as 'master'. A modal window is open for the first run, titled 'Run workflow', with options to 'Use workflow from Branch: master', 'Path to the .tfvars file * variables.tfvars', 'Apply or Destroy * destroy', and a 'Run workflow' button.

Here our workflow succeeded and services have been deleted.

The screenshot shows the logs for the Terraform workflow run. The logs indicate that the workflow succeeded 1 minute ago in 6m 34s. The log output shows the execution of various Terraform commands to destroy resources, such as modules for EKS cluster, security groups, subnets, IAM roles, and VPCs. The logs conclude with 'Destroy complete! Resources: 37 destroyed.' and 'module.eks.aws_vpc.vpc: Destruction complete after 1s'.

```

1041 module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m40s elapsed]
1042 module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 1m50s elapsed]
1043 module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m0s elapsed]
1044 module.eks.aws_eks_cluster.eks[0]: Still destroying... [id=dev-medium-eks-cluster, 2m10s elapsed]
1045 module.eks.aws_eks_cluster.eks[0]: Destruction complete after 2m15s
1046 module.eks.aws_security_group.eks-cluster-sg: Destroying... [id=sg-009b810ac40b3e0cd]
1047 module.eks.aws_subnet.private-subnet[2]: Destroying... [id=subnet-0219cd3770e93532b]
1048 module.eks.aws_subnet.private-subnet[0]: Destroying... [id=subnet-003fe35cef08faaf4]
1049 module.eks.aws_iam_role.eks-cluster-role[0]: Destroying... [id=dev-medium-eks-cluster-role-9982]
1050 module.eks.aws_subnet.private-subnet[1]: Destroying... [id=subnet-0faea2c3aa36f3509]
1051 module.eks.aws_iam_role.eks-cluster-role[0]: Destruction complete after 0s
1052 module.eks.random_integer.random_suffix: Destroying... [id=9982]
1053 module.eks.random_integer.random_suffix: Destruction complete after 0s
1054 module.eks.aws_subnet.private-subnet[2]: Destruction complete after 0s
1055 module.eks.aws_subnet.private-subnet[0]: Destruction complete after 0s
1056 module.eks.aws_subnet.private-subnet[1]: Destruction complete after 0s
1057 module.eks.aws_security_group.eks-cluster-sg: Destruction complete after 1s
1058 module.eks.aws_vpc.vpc: Destroying... [id=vpc-02583d5096ff374e9]
1059 module.eks.aws_vpc.vpc: Destruction complete after 1s
1060
1061 Destroy complete! Resources: 37 destroyed.
1062 :debug::Terraform exited with code 0.
1063

```

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!