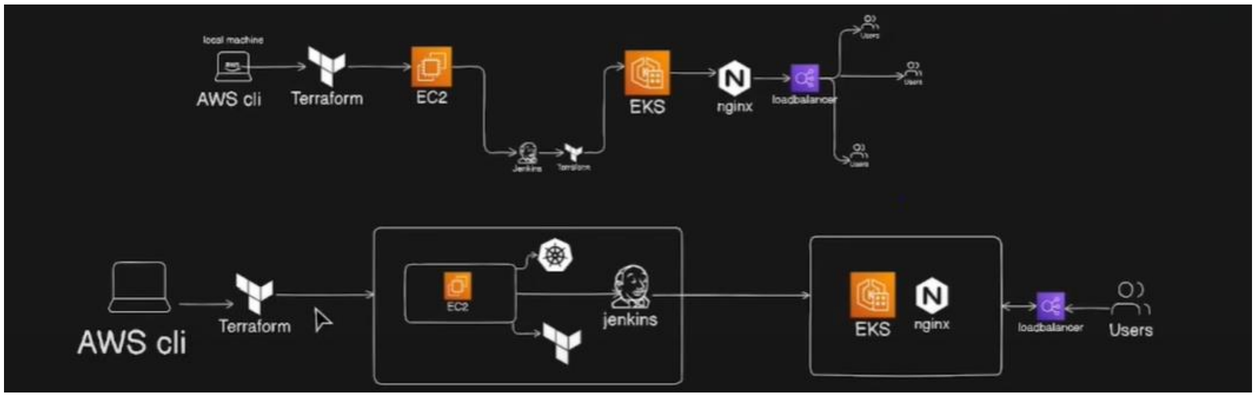
Deploying AWS EKS with Terraform and Jenkins

Project Created By: AMAN DUGGAL

https://www.linkedin.com/in/aman-duggal-4591bb146
amanduggal103@gmail.com

Introduction:



This project involves setting up an automated deployment pipeline using Terraform to provision infrastructure on AWS.

Jenkins is used to automate deployments to an EKS cluster, which uses Nginx.

An AWS Load Balancer handles incoming user traffic, ensuring it reaches the application running within the EKS cluster. This architecture provides a scalable and automated way to manage application deployments on AWS.

Tools & Technologies:

- **AWS CLI:** Command-line tool for interacting with AWS services.
- **Terraform:** Infrastructure as Code (IaC) tool for provisioning resources.
- **EC2:** Amazon Elastic Compute Cloud for virtual servers.
- **EKS:** Amazon Elastic Kubernetes Service for managing Kubernetes clusters.
- **Jenkins:** Automation server for CI/CD.
- **NGINX:** Web server used as a reverse proxy and load balancer.

Git Hub Repo For Scripts: https://github.com/amanduggal001/CICD-Terraform-EKS.git

Project Summary:

1. **Local Machine to AWS:**
    - The process starts from a local machine where the AWS CLI and Terraform are installed.
    - Using these tools, the user runs Terraform scripts to provision infrastructure on AWS.

2. **Terraform Provisioning:**
    - Terraform scripts create an EC2 instance.
    - Terraform scripts also create an EKS cluster.

3. **Deployment Tools on EC2:**
    - The EC2 instance is configured to run Jenkins, which is a Continuous Integration/Continuous Deployment (CI/CD) tool.

1. **Application Deployment:**
    - Jenkins automates the deployment process to the EKS cluster.
    - The EKS cluster is configured with Nginx as an ingress controller to manage incoming traffic.

5. **Traffic Management:**
    - An AWS Load Balancer is set up to handle incoming traffic from users.
    - The Load Balancer routes traffic to the Nginx ingress controller in the EKS cluster.
    - Nginx then directs the traffic to the appropriate services within the cluster, ensuring that users reach the correct application endpoints.

Project Structure:

```
terraform-project/
├── EKS/
│   ├── backend.tf
│   ├── data.tf
│   ├── main.tf
│   ├── provider.tf
│   ├── terraform.tfvars
│   ├── variables.tf
│
└── Jenkins-server/
    ├── backend.tf
    ├── data.tf
    ├── main.tf
    ├── provider.tf
    ├── terraform.tfvars
    ├── variables.tf
    ├── jenkins-install.sh
```

Detailed Steps and Explanation:

Step 1: Set Up AWS CLI

- Install AWS CLI:
```
pip install awscli
```

- Configure AWS CLI:
```
aws configure
```
Provide your AWS Access Key, Secret Key, region, and output format.

Step 2: Provision EC2 Instances with Terraform

- **Install Terraform:** Follow the installation guide from Terraform official site.

Setting up the Jenkins-server.

Step 3: Creating an EC2 instance + Deploying Jenkins on it using Terraform.

- Create a Terraform Configuration File **(main.tf).** It contains several components including an S3 bucket, a VPC (Virtual Private Cloud), a security group, and an EC2 instance configured for Jenkins. Let's go through each section in detail:

a. **S3 Bucket:**

1. **Random ID for Bucket Suffix.**

```
resource "random_id" "bucket_suffix" {
  byte_length = 4
}
```

This generates a random suffix of 4 bytes for the S3 bucket name to ensure it is unique.

2. **S3 Bucket Resource.**

```
resource "aws_s3_bucket" "s3bucket" {
  bucket = "jenkins-server-bucket-${random_id.bucket_suffix.hex}"
}
```

This creates an S3 bucket with a name that includes the random suffix, ensuring it's unique.

b. **VPC:**

1. **VPC Module.**

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = "jenkins-vpc"
  cidr = var.vpc_cidr

  azs                = data.aws_availability_zones.azs.names
  public_subnets     = var.public_subnets
  enable_dns_hostnames = true
  map_public_ip_on_launch = true

  tags = {
    Name = "Jenkins VPC"
    Terraform = "true"
    Environment = "dev"
  }

  public_subnet_tags = {
    Name = "Jenkins Subnet"
  }
}
```

- **source:** Specifies the module source, which is a publicly available VPC module.
- **name:** Name of the VPC.
- **cidr:** CIDR block for the VPC, provided via a variable.
- **azs:** List of availability zones for subnets.
- **public_subnets:** List of public subnets CIDRs.
- **enable_dns_hostnames:** Enables DNS hostnames in the VPC.
- **map_public_ip_on_launch:** Automatically assigns a public IP to instances in the public subnets.
- **tags:** Tags for the VPC.
- **public_subnet_tags:** Tags for the public subnets.

c. Security Group:

1. Security Group Resource.

```
resource "aws_security_group" "jenkins_sg" {
  name        = "Jenkins-SecurityGroup"
  description = "Security group for Jenkins"
  vpc_id      = module.vpc.vpc_id

  tags = {
    Name = "Jenkins-SecurityGroup"
  }
}
```

- **name:** Name of the security group.
- **description:** Description of the security group.
- **vpc_id: ID** of the VPC where the security group is created.
- **tags:** Tags for the security group.

2. Ingress Rule for HTTP (Port 8080).

```
resource "aws_security_group_rule" "ingress_http" {
  type              = "ingress"
  from_port         = 8080
  to_port           = 8080
  protocol          = "tcp"
  cidr_blocks       = ["0.0.0.0/0"]
  security_group_id = aws_security_group.jenkins_sg.id
  description       = "HTTP"
}
```

Allows incoming HTTP traffic on port 8080 from any IP.

3. **Ingress Rule for SSH (Port 22).**

```
resource "aws_security_group_rule" "ingress_ssh" {
  type              = "ingress"
  from_port         = 22
  to_port           = 22
  protocol          = "tcp"
  cidr_blocks       = ["0.0.0.0/0"]
  security_group_id = aws_security_group.jenkins_sg.id
  description       = "SSH"
}
```

Allows incoming SSH traffic on port 22 from any IP.

4. **Egress Rule.**

```
resource "aws_security_group_rule" "egress" {
  type              = "egress"
  from_port         = 0
  to_port           = 0
  protocol          = "-1"
  cidr_blocks       = ["0.0.0.0/0"]
  security_group_id = aws_security_group.jenkins_sg.id
}
```

Allows all outbound traffic to any IP.

d. EC2 Instance:

1. **EC2 Instance Resource.**

```
resource "aws_instance" "jenkins-server" {
  ami = var.ami
  instance_type = var.instance_type
  key_name = var.key_name
  vpc_security_group_ids = [aws_security_group.jenkins_sg.id]
  subnet_id                   = module.vpc.public_subnets[0]
  associate_public_ip_address = true
  availability_zone           =
data.aws_availability_zones.azs.names[0]
  user_data                   = file("jenkins-install.sh")

  tags = {
    Name        = "Jenkins-Server-New"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

- **ami:** AMI ID for the EC2 instance, provided via a variable.
- **instance_type:** Instance type, provided via a variable.
- **key_name:** Key pair name for SSH access, provided via a variable.
- **vpc_security_group_ids:** List of security groups to assign to the instance.
- **subnet_id:** Subnet ID for the instance, using the first public subnet.
- **associate_public_ip_address:** Assigns a public IP address.
- **availability_zone:** Availability zone for the instance.
- **user_data:** Script to run on instance launch, installs Jenkins.
- **tags:** Tags for the instance.

This configuration sets up the necessary infrastructure to run a Jenkins server on AWS, including an S3 bucket for storage, a VPC for networking, security groups for access control, and an EC2 instance for the Jenkins server itself.

e. Setup data.tf file:

This file is used to fetch information about the AWS availability zones.

The **data.tf** file is commonly used in Terraform to define data sources. Data sources are a way to query existing resources or information outside of Terraform's management to use in the Terraform configuration.

```
data "aws_availability_zones" "azs" {}
```

f. Setup variables.tf file:

This file defines the variables used in the Terraform configuration. Each variable includes a description and type.

```
variable "vpc_cidr" {
  description = "VPC CIDR"
  type = string
}

variable "public_subnets" {
  description = "public subnets"
  type = list(string)
}

variable "instance_type" {
  description = "Instance Type"
  type = string
}

variable "key_name" {
  description = "Key Pair name"
  type = string
}

variable "ami" {
  description = "ami id"
  type = string
}
```

g. Setting up terraform.tfvars file.
This file contains values for the variables defined in variables.tf.

```
vpc_cidr = "192.0.0.0/16"

public_subnets = [ "192.0.1.0/24", "192.0.2.0/24","192.0.3.0/24" ]

private_subnets = [ "192.0.4.0/24", "192.0.5.0/24","192.0.6.0/24" ]

instance_types = ["t2.medium"]
```

h. Setting up backend.tf file.
This file configures Terraform to use an S3 bucket to store the state file. This enables collaboration and ensures the state is safely stored.

```
terraform {
  backend "s3" {
    bucket = "jenkins-server-bucket-4e6aeaee"
    key = "jenkins/terraform.tfstate"
    region = "us-east-1"
  }

}
```

i. Setting up jenkins-install.sh file.
This script file installs essential tools for setting up a Jenkins server with the necessary dependencies for managing infrastructure. It includes installations for Java, Jenkins, Terraform, Kubernetes CLI, and AWS CLI.

Note: We defined this file in the user_data block in main.tf file.

```
#!/bin/bash
# Java installation for jenkins

sudo apt update
sudo apt install openjdk-11-jre -y

# Jenkins installation
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

```
sudo apt-get update
sudo apt-get install jenkins -y




# Terraform Installation

sudo apt-get update && sudo apt-get install -y gnupg software-properties-
common
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt-get install terraform -y

# Installing kubernetes

curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Install AWS CLI
sudo apt install unzip
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

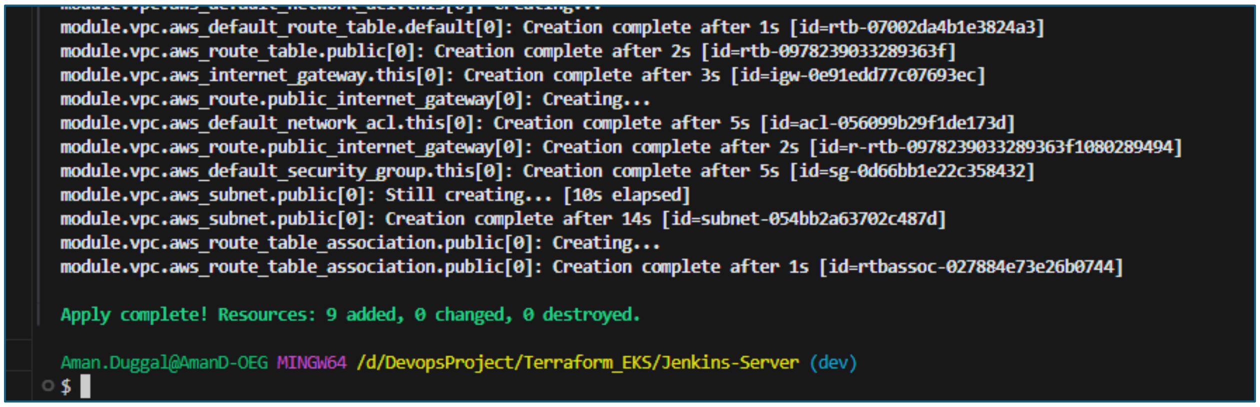
Step 4: Deploying the Terraform scripts to set up the jenkins server:

1. **Initialize Terraform:**
   terraform init

2. **Validate the terraform script:**
   terraform validate

3. **Plan the Deployment:**
   terraform plan

4. **Apply the Configuration:**
   terraform apply

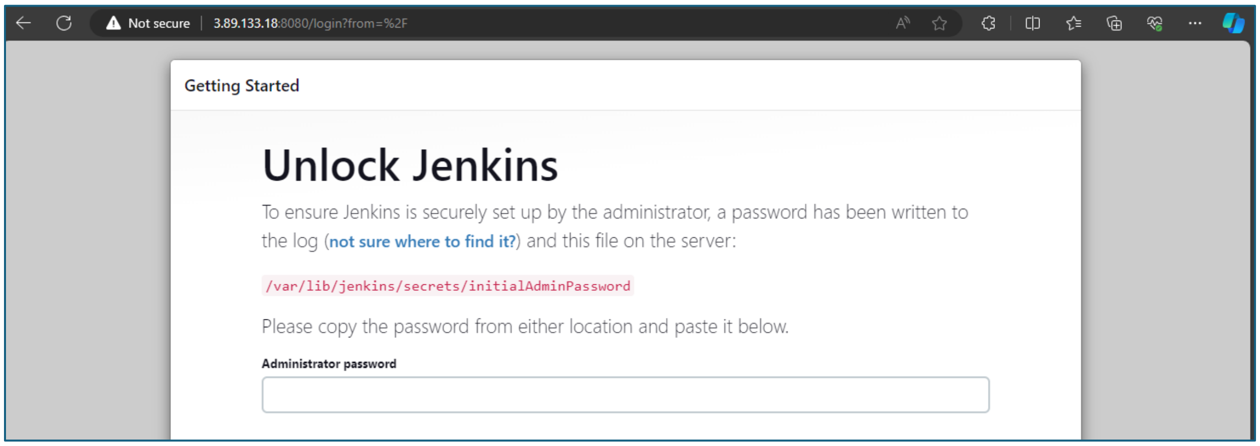


```
module.vpc.aws_default_route_table.default[0]: Creation complete after 1s [id=rtb-07002da4b1e3824a3]
module.vpc.aws_route_table.public[0]: Creation complete after 2s [id=rtb-0978239033289363f]
module.vpc.aws_internet_gateway.this[0]: Creation complete after 3s [id=igw-0e91edd77c07693ec]
module.vpc.aws_route.public_internet_gateway[0]: Creating...
module.vpc.aws_default_network_acl.this[0]: Creation complete after 5s [id=acl-056099b29f1de173d]
module.vpc.aws_route.public_internet_gateway[0]: Creation complete after 2s [id=r-rtb-0978239033289363f1080289494]
module.vpc.aws_default_security_group.this[0]: Creation complete after 5s [id=sg-0d66bb1e22c358432]
module.vpc.aws_subnet.public[0]: Still creating... [10s elapsed]
module.vpc.aws_subnet.public[0]: Creation complete after 14s [id=subnet-054bb2a63702c487d]
module.vpc.aws_route_table_association.public[0]: Creating...
module.vpc.aws_route_table_association.public[0]: Creation complete after 1s [id=rtbassoc-027884e73e26b0744]

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Aman.Duggal@AmanD-OEG MINGW64 /d/DevopsProject/Terraform_EKS/Jenkins-Server (dev)
$
```

After executing and applying the terraform scripts the jenkins is installed and running. Copy the public IP of the EC2 instance and search on the browser followed by port 8080.
<Public IP>:8080



Not secure | 3.89.133.18:8080/login?from=%2F

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

/var/lib/jenkins/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

Step 5: Writing the Terraform Code for creation for EKS Cluster and pushing that code to the GitHub repository.

- For the EKS Cluster, create a new folder named EKS and add the following files: **backend.tf, data.tf, main.tf, provider.tf, terraform.tfvars,** and **variables.tf.**

- Copy the **backend.tf, provider.tf,** and **data.tf files** from **the Jenkins server**. Modify the backend file name accordingly.

- To set up the EKS cluster, we need a VPC. Copy the VPC configuration from the Jenkins main.tf file and rename the VPC.

1. **Setting up main.tf file for EKS:**

   a. **s3 bucket:**

```
resource "random_id" "bucket_suffix" {
  byte_length = 4
}

resource "aws_s3_bucket" "s3bucket" {
  bucket = "eks-cluster-bucket-${random_id.bucket_suffix.hex}"
}
```

   b. **VPC:**

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name = "eks-cluster-vpc"
  cidr = var.vpc_cidr

  azs             = data.aws_availability_zones.azs.names
  public_subnets  = var.public_subnets
  private_subnets = var.private_subnets
  enable_dns_hostnames = true
  enable_nat_gateway = true
  single_nat_gateway = true

  tags = {
    "kubernetes.io/cluster/my-eks-cluster" = "shared"
  }
  public_subnet_tags = {
    "kubernetes.io/cluster/my-eks-cluster" = "shared"
    "kubernetes.io/role/elb"               = 1

  }
  private_subnet_tags = {
    "kubernetes.io/cluster/my-eks-cluster" = "shared"
    "kubernetes.io/role/private_elb"       = 1

  }
}
```

c. EKS:

```
module "eks" {
  source                          = "terraform-aws-modules/eks/aws"
  cluster_name                    = "my-eks-cluster"
  cluster_version                 = "1.29"
  cluster_endpoint_public_access = true
  vpc_id                          = module.vpc.vpc_id
  subnet_ids                      = module.vpc.private_subnets

  eks_managed_node_groups = {
    nodes = {
      min_size       = 1
      max_size       = 3
      desired_size   = 2
      instance_types = var.instance_types
    }
  }
  tags = {
    Environment = "dev"
    Terraform   = "true"
  }
}
```

module "eks": Defines a Terraform module named eks.

- **source:** Specifies the source of the module, in this case, the official terraform-aws-modules/eks/aws module from the Terraform Registry.
- **cluster_name:** Sets the name of the EKS cluster to "my-eks-cluster".
- **cluster_version:** Specifies the version of the Kubernetes cluster, here set to "1.29".
- **cluster_endpoint_public_access:** Enables public access to the EKS cluster endpoint.
- **vpc_id:** References the VPC ID from the vpc module, indicating which VPC the EKS cluster will be deployed into.
- **subnet_ids:** References the private subnets from the vpc module, specifying the subnets where the EKS nodes will be deployed.

eks_managed_node_groups: Configures the managed node groups for the EKS cluster.

- **nodes:** Defines a node group named nodes with the following properties:
- **min_size:** Minimum number of nodes in the node group, set to 1.
- **max_size:** Maximum number of nodes in the node group, set to 3.
- **desired_size:** Desired number of nodes in the node group, set to 2.
- **instance_types:** Specifies the instance types for the nodes, sourced from a variable instance_types.

tags: Adds metadata tags to the EKS resources for easier management and identification.

- **Environment:** Tag to identify the environment, set to "dev".
- **Terraform**: Tag to indicate that the resource is managed by Terraform, set to "true".

2. **Push terraform scripts to Git Hub repository.**

- Push the terraform scripts to the Git hub repository.

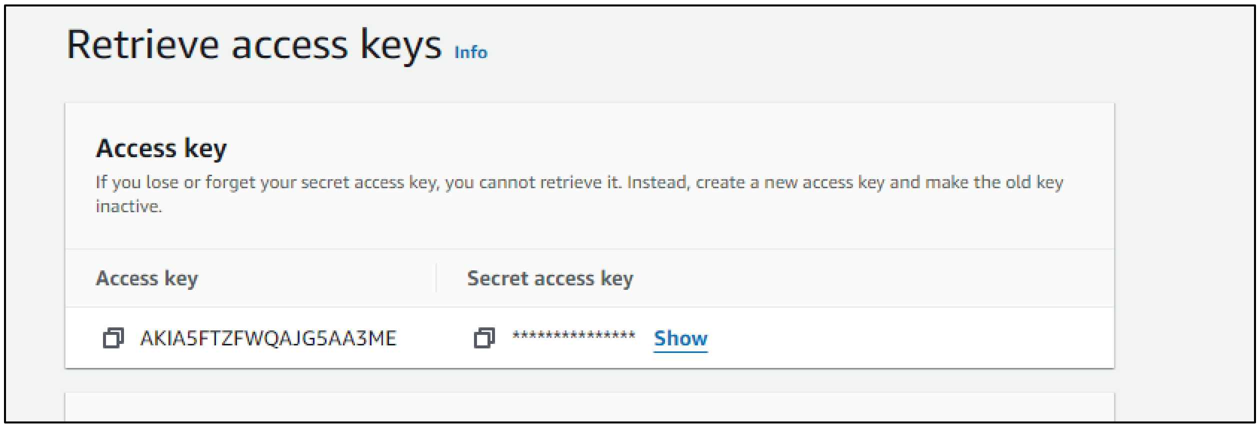
  Commands:

```
git init
git add .
git commit -m 'new commit'
git push -u origin main
```

We have set up the configuration files but have not applied them yet. The EKS cluster will be created through the pipeline. Now, push these files to the GitHub repository.

Step 6: Creating a jenkins pipeline - EKS cluster then Implementing a deployment files – kubectl and Deploying changes to aws:

- We need to provide security credentials to Jenkins to enable access to our AWS services.
- To do this, create an IAM user and obtain its secret access key and access key ID.



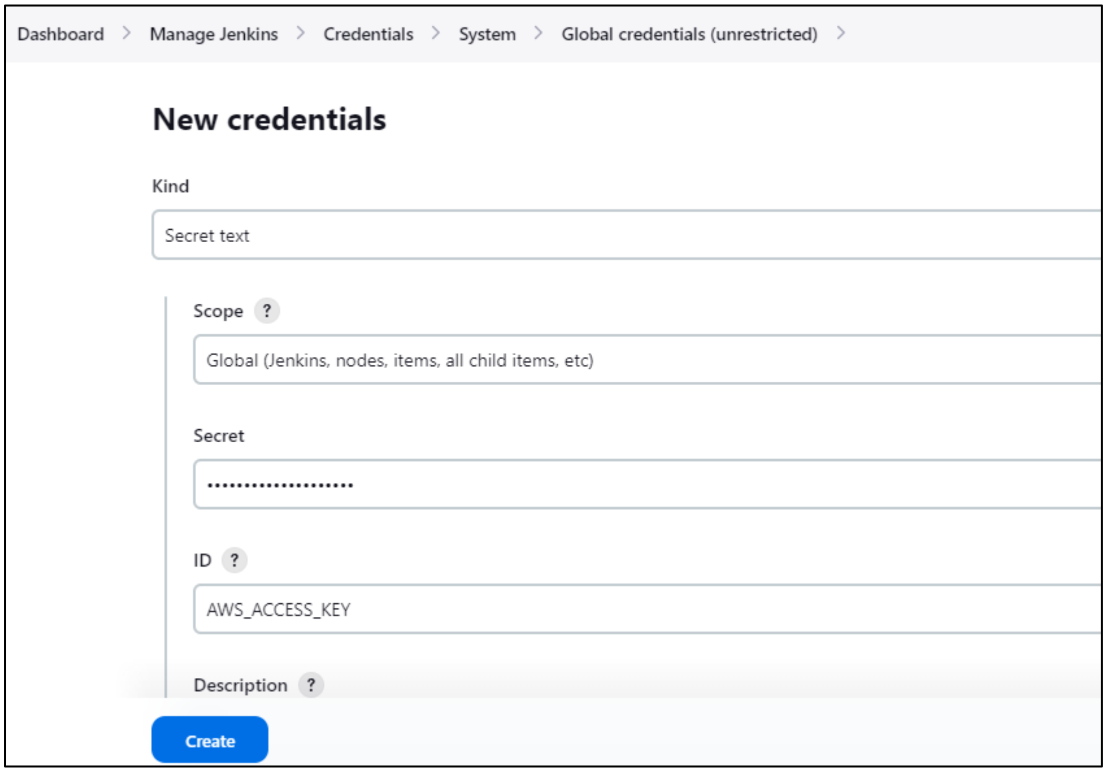
Retrieve access keys Info

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key and make the old key inactive.

Access key	Secret access key
AKIA5FTZFWQAJG5AA3ME	*************** Show

- Next, create credentials by navigating to: **Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) > Add credentials.**


Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

New credentials

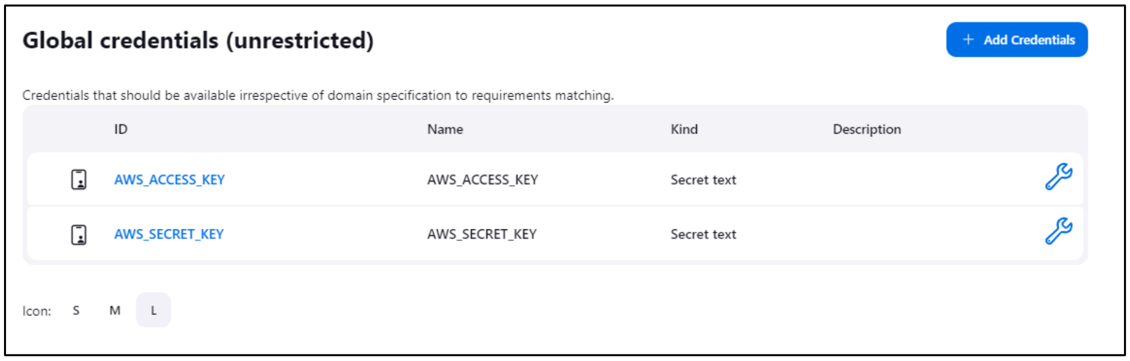
Kind: Secret text

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

Secret: ••••••••••••••••••••

ID: AWS_ACCESS_KEY

Description


Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
AWS_ACCESS_KEY	AWS_ACCESS_KEY	Secret text	
AWS_SECRET_KEY	AWS_SECRET_KEY	Secret text	

- Now create the pipeline job using the below script.

Complete Script:

```
pipeline {
    agent any
    environment {
        AWS_ACCESS_KEY_ID = credentials('AWS_ACCESS_KEY_ID')
        AWS_SECRET_ACCESS_KEY = credentials('AWS_SECRET_ACCESS_KEY')
        AWS_DEFAULT_REGION = 'us-east-2'
    }
    stages{
        stage('Checkout SCM'){
            steps{
                script{
                    //Generate  using pipeline syntax
                }
            }
        }
         stage('Initializing Terraform'){
            steps{
                script{
                    dir('EKS'){
                         sh 'terraform init'
                    }
                }
            }
        }
        stage('Formating terraform code'){
            steps{
                script{
                    dir('EKS'){
                         sh 'terraform fmt'
                    }
                }
            }
        }
        stage('Validating Terraform'){
            steps{
                script{
                    dir('EKS'){
                         sh 'terraform validate'
                    }
                }
            }
        }
        stage('Previewing the infrastructure'){
            steps{
                script{
                    dir('EKS'){
                         sh 'terraform plan'
                    }
                    input(message: "Are you sure to proceed?", ok:
"proceed")
                }
            }
        }
        stage('Creating/Destroying an EKS cluster'){
            steps{
                script{
                    dir('EKS'){
                         sh 'terraform $action --auto-approve'
                    }
                }
            }
        }
        stage("Deploying Nginx"){
            steps{
                script{
                    dir('EKS/configuration-files'){
                        sh 'aws eks update-kubeconfig --name my-eks-
cluster'
                        sh 'kubectl apply -f deployment.yml'
                        sh 'kubectl apply -f service.yml'
                    }
                }
            }
        }
    }
}
```

Explanation:

1. **Checkout SCM**
    - steps -> script: Placeholder for code to check out the source code from the source control management (SCM) system.

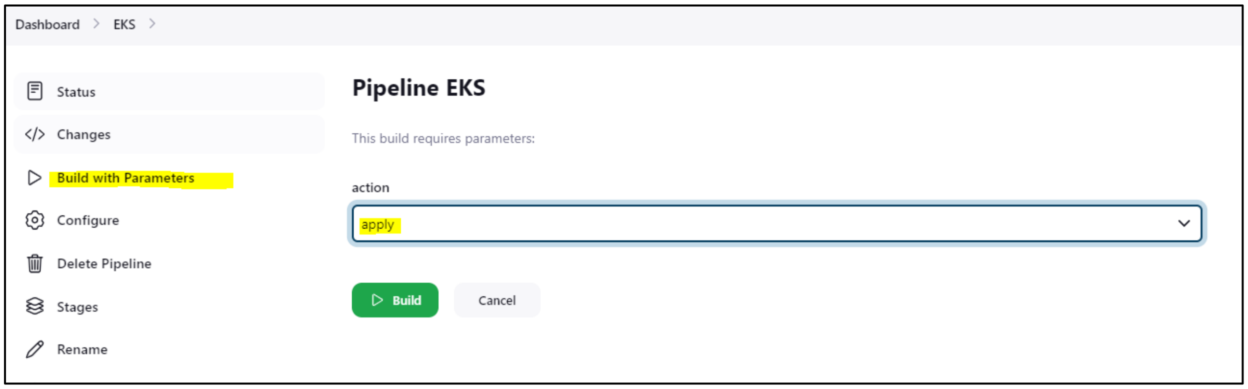
2. **Initializing Terraform**
    - dir('EKS'): Changes the directory to `EKS`.
    - sh 'terraform init': Initializes the Terraform configuration.

3. **Formatting Terraform code**
- dir('EKS'): Changes the directory to `EKS`.
- sh 'terraform fmt': Formats the Terraform configuration files to canonical format.

4. **Validating Terraform**
- dir('EKS'): Changes the directory to `EKS`.
- sh 'terraform validate': Validates the Terraform configuration.

5. **Previewing the infrastructure**
- dir('EKS'): Changes the directory to `EKS`.
- sh 'terraform plan': Creates an execution plan for Terraform, showing what actions will be taken.
- input: Prompts the user for manual approval to proceed with the next steps.

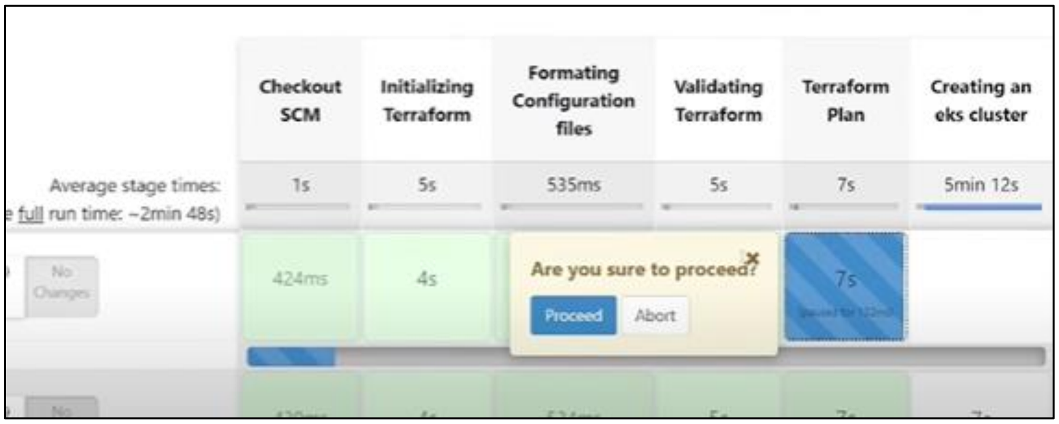
If we want that the jenkins will ask after the terraform plan if we want to proceed further or not so add this parameter so that it will ask to proceed further or not.



Pipeline EKS

This build requires parameters:

action: apply



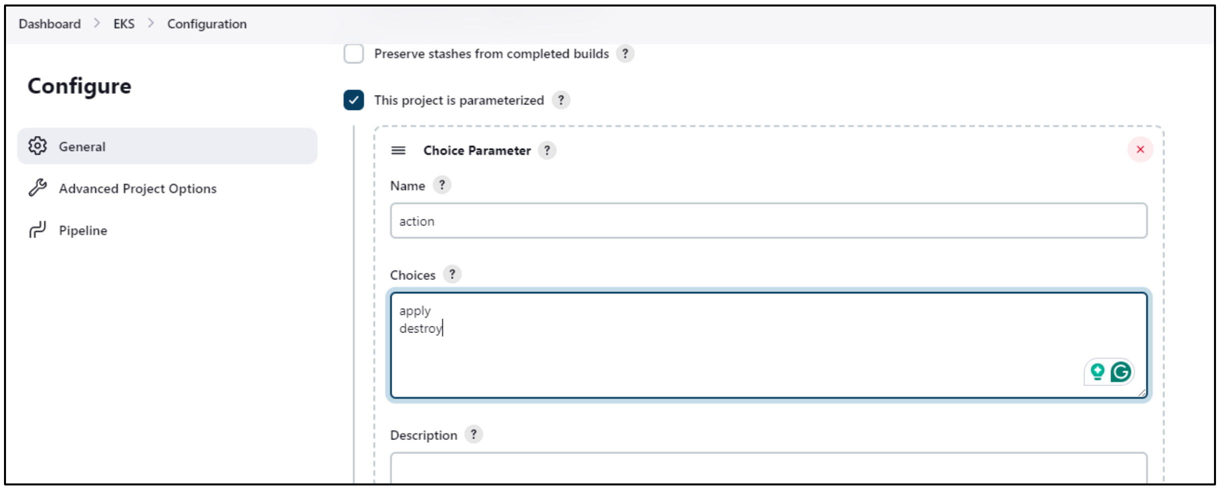
	Checkout SCM	Initializing Terraform	Formating Configuration files	Validating Terraform	Terraform Plan	Creating an eks cluster
Average stage times: full run time: ~2min 48s)	1s	5s	535ms	5s	7s	5min 12s
	424ms	4s			7s	

6. **Creating/Destroying an EKS cluster**

- dir('EKS'): Changes the directory to `EKS`.
- sh 'terraform $action --auto-approve': Executes the Terraform command (`apply` or `destroy`) stored in the `$action` variable with automatic approval of changes.

- To destroy the EKS cluster, we need to provide options for that in the configuration. Go to "Configure" and enable the "This project is parameterized" option. Add a parameter of type "Choice" to determine whether to apply or destroy the cluster. Name this parameter "action" with choices "apply" or "destroy", and then save it. When building, it will prompt us to choose whether to apply or destroy. We need to specify this action in our steps.

- For example, instead of using "apply" directly, use a variable named "action" which will take the selected option. Additionally, if we want to control whether to proceed after the planning stage, we can add an input for that. After running `terraform plan`, it should ask us whether to proceed. If we choose to proceed, it will start creating the resources; otherwise, it will abort.



Dashboard > EKS > Configuration

Configure

General

Advanced Project Options

Pipeline

☐ Preserve stashes from completed builds

☑ This project is parameterized

Choice Parameter

Name: action

Choices:
```
apply
destroy
```

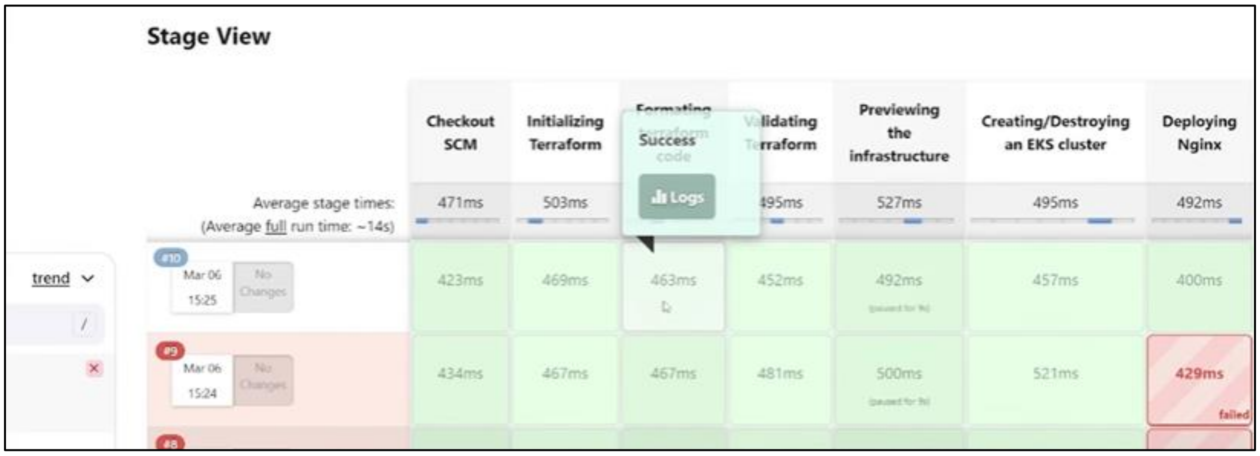
Description

7. **Deploying Nginx**

- dir('EKS/configuration-files'): Changes the directory to ` EKS/configuration-files` .
- sh 'aws eks update-kubeconfig --name my-eks-cluster': Updates the kubeconfig file to use the specified EKS cluster.
- sh 'kubectl apply -f deployment.yml': Applies the Kubernetes deployment configuration for Nginx.
- sh 'kubectl apply -f service.yml': Applies the Kubernetes service configuration for Nginx.

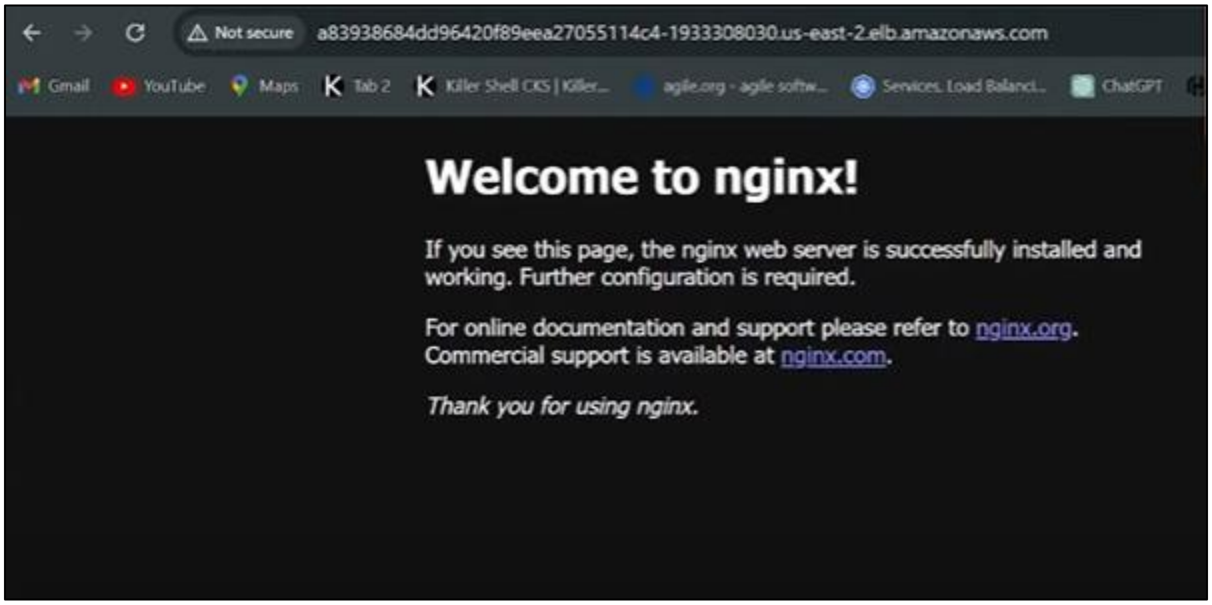
This completes the EKS cluster creation.

Run the build and, if successful, check the load balancer. Copy the DNS from the load balancer, paste it into your browser, and you should see the Nginx welcome page. This demonstrates the deployment of our Nginx application on an EKS cluster on AWS using Terraform, automated via a Jenkins CI/CD pipeline.



Stage View

	Checkout SCM	Initializing Terraform	Formating terraform code	Validating Terraform	Previewing the infrastructure	Creating/Destroying an EKS cluster	Deploying Nginx
Average stage times: (Average full run time: ~14s)	471ms	503ms	[illegible]	495ms	527ms	495ms	492ms
#10 Mar 06 15:25 No Changes	423ms	469ms	463ms	452ms	492ms	457ms	400ms
#9 Mar 06 15:24 No Changes	434ms	467ms	467ms	481ms	500ms	521ms	429ms failed


Not secure
a83938684dd96420f89eea27055114c4-1933308030.us-east-2.elb.amazonaws.com
Gmail
YouTube
Maps
Tab 2
Killer Shell CKS | Killer...
agile.org - agile softw...
Services, Load Balanci...
ChatGPT

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.