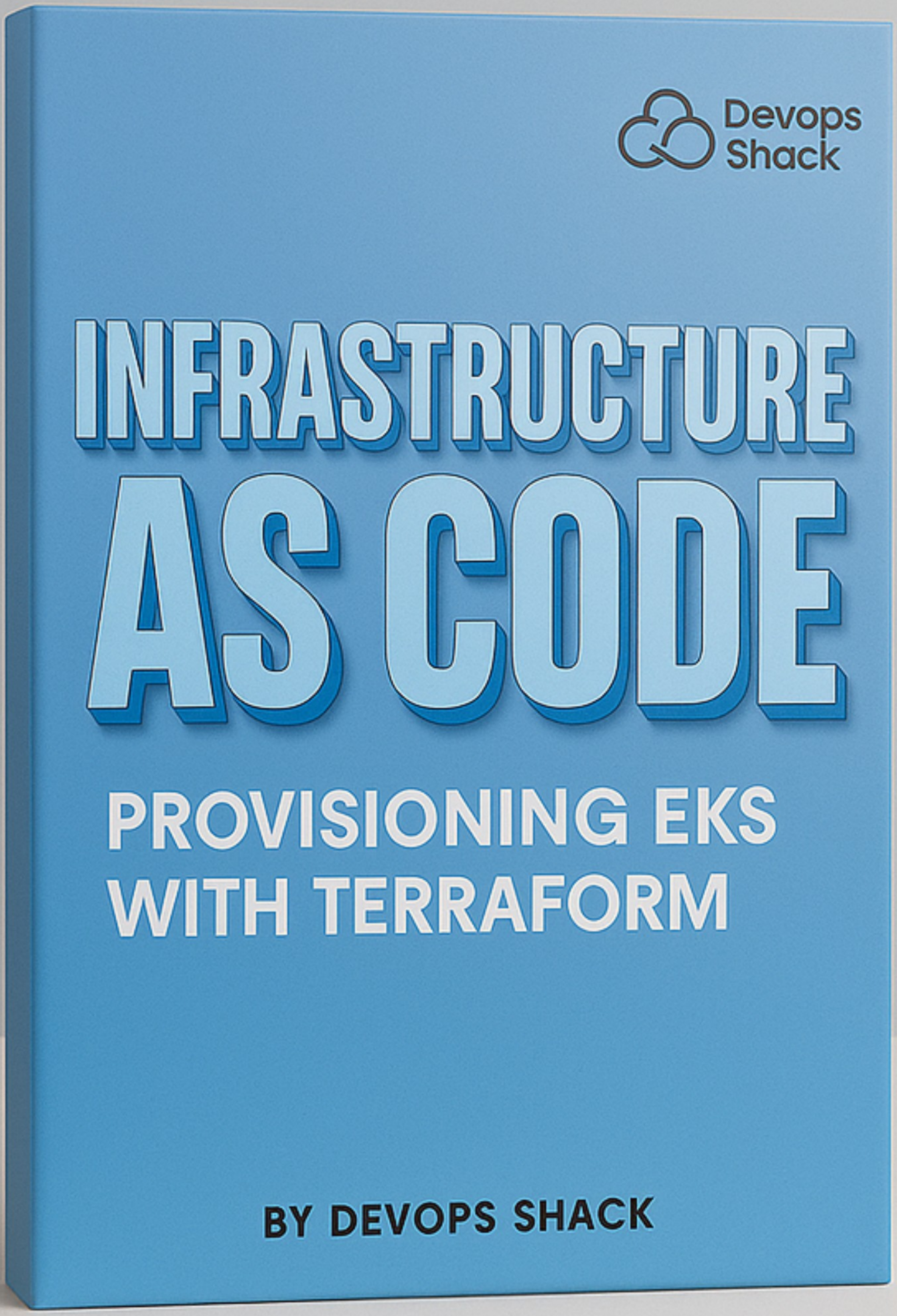
Devops Shack
INFRASTRUCTURE AS CODE
PROVISIONING EKS WITH TERRAFORM
BY DEVOPS SHACK

DevOps Shack

Infrastructure as Code: Provisioning EKS with Terraform

In today's cloud-native landscape, Kubernetes has emerged as the de facto standard for container orchestration. Amazon Elastic Kubernetes Service (EKS) offers a fully managed Kubernetes solution that simplifies cluster operations, scalability, and integration with the AWS ecosystem. However, manually provisioning EKS and its underlying infrastructure can be time-consuming and error-prone.

This is where Terraform, an open-source Infrastructure as Code (IaC) tool by HashiCorp, becomes a game-changer. By defining your infrastructure in declarative configuration files, Terraform allows you to automate, version, and manage your entire EKS environment with ease and consistency.

In this guide, we'll walk through a step-by-step process to create an EKS cluster on AWS using Terraform. Whether you're a DevOps engineer looking to streamline deployments or a developer exploring Kubernetes at scale, this tutorial will equip you with the foundational skills to provision a production-ready EKS setup—all through code.

Elastic Kubernetes Service (EKS)

Kubernetes (often abbreviated as **K8s**) is an **open-source container orchestration platform** designed to **automate the deployment, scaling, and management of containerized applications**. Setting up and self managing kubernetes clustera is a tedious job. AWS decided to simplify deployment, scaling, and operation of Kubernetes clusters by creating their own kubernetes cluster known as Elastic Kubernetes Service(EKS) managed by their team. All the customer need is to setup EKS and deploy their application.

Terraform

Terraform is an **open-source Infrastructure as Code (IaC) tool** developed by **HashiCorp**. It allows you to **define, provision, and manage cloud**

infrastructure using a **declarative configuration language** called **HCL (HashiCorp Configuration Language)**. Every cloud provider has their own IaC like aws has Cloud Formation but terraform is open source and can interact with different cloud providers

Prerequisite

- AWS account
- AWS cli
- Terraform
- Visual studio Code
- eksctl

Objective

- Setup aws credentials
- Create EKS with terraform
- Create Remote backend
- Modularize our terraform code
- Create State Locking using Dynamo DB
- Clean up

We will start with creating aws credentials.

The next stage is to configure aws credentials on our command line. Before you configure your credentials ensure that you have installed aws cli. you can refer to this article [AWS CLI](#) to install aws cli depending on your operating system.

To configure aws credentials

```
$ aws configure
AWS Access Key ID [****************JKX2]: skfjoiwefwf
AWS Secret Access Key [****************3Rdd]: ksjfoiwefkoij
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

Important extensions that can improve productivity

- **HashiCorp Terraform**

- **GitHub Copilot**

Most companies modularize their terraform code. The reason is to

- **Avoid Redundancy** – Prevent repetitive code and improve efficiency
- **Enhance Reusability** – Allow easy reuse of infrastructure components.
- **Improve Accessibility** – Enable seamless collaboration and management
- **Maintain Clean Code** – Ensure a structured and maintainable codebase

In order to modularize our terraform code we will create 2 folders inside the root directory

- backend
- modules

Then inside modules

- EKS
- VPC

File structure

```
.
|____.gitignore
|____backend
|  |____main.tf
|  |____outputs.tf
|____main.tf
|____modules
|  |____eks
|  |  |____main.tf
|  |  |____outputs.tf
|  |  |____variables.tf
|  |____vpc
|  |  |____main.tf
|  |  |____outputs.tf
|  |  |____variables.tf
|____outputs.tf
|____variables.tf
```

Next step is to create our remote backend.

Why Use AWS Remote Backend?

- **Collaboration** – Multiple engineers can access the same state file.
- **Disaster Recovery** – Remote backups ensure state is not lost.
- **State Locking** – Prevents simultaneous updates using **DynamoDB**.

AWS State Locking is a mechanism that prevents **simultaneous modifications** to the **Terraform state file** stored in **DynamoDB**. It ensures that only **one Terraform process** can modify the state at a time, preventing conflicts and corruption.

Prerequisite

- AWS Account
- AWS cli
- Terraform

To create remote backend and state lock we will create

- main.tf

```
provider "aws" {
  region = "us-east-1"
}


resource "aws_s3_bucket" "terraform_s3"
  { bucket = "remote-backend-s3"


  lifecycle {
    prevent_destroy = false
  }
}


resource "aws_s3_bucket_versioning" "terraform_s3" {
```

```
  bucket = aws_s3_bucket.terraform_state.id
  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration"
"terraform_s3" {
  bucket = aws_s3_bucket.terraform_state.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

resource "aws_dynamodb_table" "terraform_locks"
  { name         = "remote-backend-locks"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

- outputs.tf

```
output "s3_bucket_name" {
  value       = aws_s3_bucket.terraform_state.id
  description = "The name of the S3 bucket"
}

output "dynamodb_table_name" {
  value       = aws_dynamodb_table.terraform_locks.id
  description = "The name of the DynamoDB table"
}
```

To create the resource in aws

- You need to initialize

terraform terraform init

- To view resources before

creation terraform plan

- To create the

resources terraform apply

Work Flow

1. VPC & Networking Infrastructure

EKS needs a robust and well-designed network, because Kubernetes components and pods communicate over the network a lot.

a. Create a VPC

- Think of VPC as your **private, isolated network** inside AWS.
- You define a CIDR block (e.g., 10.0.0.0/16) which is the total range of IP addresses.

b. Create Subnets

- You split your VPC into **subnets**.
- Use **at least 2-3 Availability Zones** (AZs) to ensure **high availability**.

◆ **Public subnets**:

- Used for things that need internet access (like NAT Gateways or bastion hosts).
- Can route traffic to/from the internet via an Internet Gateway.

◆ **Private subnets**:

- These are where your **EKS worker nodes live**.
- More secure because they don't have direct internet exposure.

c. Set Up Internet Gateway & Routing

- Attach an **Internet Gateway (IGW)** to your VPC so public subnets can talk to the internet.
- Set up **Route Tables**:
  - Public subnets route traffic to the IGW.
  - Private subnets route internet-bound traffic through a **NAT Gateway** (located in a public subnet).

d. NAT Gateway

- Ensures **private subnet resources can reach out to the internet** (e.g., pull Docker images) without being directly exposed.

2. IAM Roles & Permissions

EKS interacts with many AWS services, so it needs permission through IAM roles.

a. EKS Cluster Role

- This IAM role is assumed by the EKS **control plane**.
- It needs policies that let it:
  - Manage networking (ENIs, security groups)
  - Create and attach load balancers
  - Work with EC2 and other AWS services

b. EKS Node IAM Role

- This role is assumed by **worker nodes** (EC2 instances).
- Needs permissions for:
  - Pulling container images from Amazon ECR
  - Pushing logs to CloudWatch
  - Communicating with the EKS control plane

3. Create the EKS Cluster

Now you set up the actual EKS cluster.

a. Control Plane

- Managed by AWS (you don't manage the master nodes).
- You specify:
  - The VPC and subnets to use
  - Security groups
  - Logging options (e.g., audit logs, API logs)
- The control plane is deployed across **multiple AZs** automatically for HA.

b. API Endpoint Access

- You decide whether the Kubernetes API is accessible:
  - Public (over the internet)
  - Private (internal VPC only)
  - Or both

4. EKS Node Group Setup

These are the EC2 instances that act as Kubernetes worker nodes.

a. Managed Node Groups

- AWS provisions and manages the EC2 instances for you.
- You specify:
  - Instance type (e.g., t3.medium)
  - Min, max, and desired capacity
  - AMI type and disk size
  - Subnets (should be private)

b. Node Auto-scaling

- Nodes can scale in/out based on cluster load.
- Use **Cluster Autoscaler** later for dynamic pod-based scaling.

5. Set Up Remote State for Terraform

In teams or production environments, you need to store Terraform's state securely.

a. S3 for Backend

- Stores Terraform state files.
- Ensures the current infrastructure state is not lost between runs.

b. DynamoDB for Locking

- Prevents two people from running terraform apply at the same time.

Before we can create EKS we need to create VPC because EKS does not work well with the default VPC

What we need to create VPC

- Public and private subnet
- Internet gateway
- NAT Gateway
- Route Table

What we need to create EKS

- iam role for eks cluster
- iam role for node group

Terraform has 4 important areas

- Provider - Determines the cloud provider (aws, azure, gcp etc)
- Resources - Creates and manages infrastructure on the cloud platform.
- Variable - Makes values reusable and configurable
- output - Displays the results of created resources

Inside the VPC folder

- Create file with name

main.tf resource "aws_vpc" "main"

```
{ cidr_block       = var.vpc_cidr
enable_dns_hostnames = true
enable_dns_support  = true


  tags = {
    Name                                   = "${var.cluster_name}-vpc"
    "kubernetes.io/cluster/${var.cluster_name}"    = "shared"
  }
```

```
}

resource "aws_subnet" "private" {
  count          = length(var.private_subnet_cidrs)
  vpc_id         = aws_vpc.main.id
  cidr_block      = var.private_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index]

  tags = {
    Name                              = "${var.cluster_name}-private-${count.index +
1}"
    "kubernetes.io/cluster/${var.cluster_name}"    =
    "shared" "kubernetes.io/role/internal-elb"       = "1"
  }
}

resource "aws_subnet" "public" {
  count          = length(var.public_subnet_cidrs)
  vpc_id         = aws_vpc.main.id
  cidr_block      = var.public_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index]

  map_public_ip_on_launch = true

  tags = {
    Name                              = "${var.cluster_name}-public-${count.index +
1}"
```

```
    "kubernetes.io/cluster/${var.cluster_name}"    =
    "shared" "kubernetes.io/role/elb"      = "1"
  }
}

resource "aws_internet_gateway" "main"
  { vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.cluster_name}-igw"
  }
}

resource "aws_eip" "nat" {
  count = length(var.public_subnet_cidrs)
  domain = "vpc"

  tags = {
    Name = "${var.cluster_name}-nat-${count.index + 1}"
  }
}

resource "aws_nat_gateway" "main" {
  count        = length(var.public_subnet_cidrs)
  allocation_id = aws_eip.nat[count.index].id
  subnet_id     =
  aws_subnet.public[count.index].id
```

```
  tags = {
    Name = "${var.cluster_name}-nat-${count.index + 1}"
  }
}

resource "aws_route_table" "public"
  { vpc_id = aws_vpc.main.id

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

  tags = {
    Name = "${var.cluster_name}-public"
  }
}

resource "aws_route_table" "private"
  { count =
  length(var.private_subnet_cidrs) vpc_id
  = aws_vpc.main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.main[count.index].id
```

```
  }

  tags = {
    Name = "${var.cluster_name}-private-${count.index + 1}"
  }
}

resource "aws_route_table_association" "private"
  { count        = length(var.private_subnet_cidrs)
  subnet_id      = aws_subnet.private[count.index].id
  route_table_id = aws_route_table.private[count.index].id
}

resource "aws_route_table_association" "public" {
  count          = length(var.public_subnet_cidrs)
  subnet_id      =
  aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}
```

Create another file output.tf

```
output "vpc_id" {
  description = "VPC ID"
  value       =
  aws_vpc.main.id
}

output "private_subnet_ids" {
```

```
description = "Private subnet IDs"
```

```
  value      = aws_subnet.private[*].id
}


output "public_subnet_ids" {
  description = "Public subnet IDs"
  value      =
  aws_subnet.public[*].id
}
```

Create another file variables.tf

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


variable "availability_zones" {
  description = "Availability
  zones" type        = list(string)
}


variable "private_subnet_cidrs" {
  description = "CIDR blocks for private
  subnets" type     = list(string)
}


variable "public_subnet_cidrs" {
  description = "CIDR blocks for public
  subnets" type     = list(string)
```

```
}

variable "cluster_name" {
  description = "Name of the EKS
  cluster" type       = string
}
```

Inside the EKS folder

Create main.tf

```
resource "aws_vpc" "main" {
  cidr_block         =
  var.vpc_cidr
  enable_dns_hostnames =
  true enable_dns_support  =
  true

  tags = {
    Name                                  = "${var.cluster_name}-vpc"
    "kubernetes.io/cluster/${var.cluster_name}"    = "shared"
  }
}

resource "aws_subnet" "private" {
  count          = length(var.private_subnet_cidrs)
  vpc_id         = aws_vpc.main.id
  cidr_block       = var.private_subnet_cidrs[count.index]
```

```
}
  availability_zone = var.availability_zones[count.index]
```

```
  tags = {
    Name                                  = "${var.cluster_name}-private-${count.index +
1}"
    "kubernetes.io/cluster/${var.cluster_name}"    =
    "shared" "kubernetes.io/role/internal-elb"       = "1"
  }
}

resource "aws_subnet" "public" {
  count            = length(var.public_subnet_cidrs)
  vpc_id           = aws_vpc.main.id
  cidr_block       = var.public_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index]

  map_public_ip_on_launch = true

  tags = {
    Name                                  = "${var.cluster_name}-public-${count.index +
1}"
    "kubernetes.io/cluster/${var.cluster_name}"    =
    "shared" "kubernetes.io/role/elb"      = "1"
  }
}

resource "aws_internet_gateway" "main"
  { vpc_id = aws_vpc.main.id
```

```
  tags = {
    Name = "${var.cluster_name}-igw"
  }
}

resource "aws_eip" "nat" {
  count = length(var.public_subnet_cidrs)
  domain = "vpc"

  tags = {
    Name = "${var.cluster_name}-nat-${count.index + 1}"
  }
}

resource "aws_nat_gateway" "main" {
  count         = length(var.public_subnet_cidrs)
  allocation_id = aws_eip.nat[count.index].id
  subnet_id     =
  aws_subnet.public[count.index].id

  tags = {
    Name = "${var.cluster_name}-nat-${count.index + 1}"
  }
}

resource "aws_route_table" "public"
  { vpc_id = aws_vpc.main.id
```

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

  tags = {
    Name = "${var.cluster_name}-public"
  }
}

resource "aws_route_table" "private"
  { count =
  length(var.private_subnet_cidrs) vpc_id
  = aws_vpc.main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.main[count.index].id
  }

  tags = {
    Name = "${var.cluster_name}-private-${count.index + 1}"
  }
}

resource "aws_route_table_association" "private" {
```

```
  count          = length(var.private_subnet_cidrs)
  subnet_id      = aws_subnet.private[count.index].id
  route_table_id = aws_route_table.private[count.index].id
}

resource "aws_route_table_association" "public" {
  count          = length(var.public_subnet_cidrs)
  subnet_id      =
  aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}
```

output.tf

```
output "vpc_id" {
  description = "VPC ID"
  value       =
  aws_vpc.main.id
}

output "private_subnet_ids" {
  description = "Private subnet IDs"
  value       = aws_subnet.private[*].id
}

output "public_subnet_ids" {
  description = "Public subnet IDs"
  value       =
```

```
  aws_subnet.public[*].id
}
```

variables.tf

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

variable "availability_zones" {
  description = "Availability
  zones" type        = list(string)
}

variable "private_subnet_cidrs" {
  description = "CIDR blocks for private
  subnets" type     = list(string)
}

variable "public_subnet_cidrs" {
  description = "CIDR blocks for public
  subnets" type     = list(string)
}

variable "cluster_name" {
  description = "Name of the EKS
  cluster" type       = string
}
```

Terraform modules can not execute unless they are invoked. In order to invoke VPC and EKS module. On the root folder we will create the following files

main.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket         = "demo-terraform-eks-state-s3-bucket"
    key            = "terraform.tfstate"
    region         = "us-west-2"
    dynamodb_table = "terraform-eks-state-locks"
    encrypt        = true
  }
}

provider "aws" {
  region = var.region
}

module "vpc" {
```

```
  source = "./modules/vpc"

  vpc_cidr           = var.vpc_cidr
  availability_zones  = var.availability_zones
  private_subnet_cidrs = var.private_subnet_cidrs
  public_subnet_cidrs  = var.public_subnet_cidrs
  cluster_name        = var.cluster_name
}

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

  cluster_name    = var.cluster_name
  cluster_version = var.cluster_version
  vpc_id          = module.vpc.vpc_id
  subnet_ids      = module.vpc.private_subnet_ids
  node_groups     = var.node_groups
}
```

- outputs.tf

```
output "cluster_endpoint" {
  description = "EKS cluster endpoint"
  value       =
  module.eks.cluster_endpoint
}

output "cluster_name" {
  description = "EKS cluster name"
```

```
  value       = module.eks.cluster_name
}

output "vpc_id" {
  description = "VPC ID"
  value       = module.vpc.vpc_id
}
```

- variables.tf

```
variable "region" {
  description = "AWS
  region" type        = string
  default     = "us-west-2"
}

variable "vpc_cidr" {
  description = "CIDR block for
  VPC" type = string
  default     = "10.0.0.0/16"
}

variable "availability_zones" {
  description = "Availability
  zones" type        = list(string)
  default     = ["us-west-2a", "us-west-2b", "us-west-2c"]
}
```

```
variable "private_subnet_cidrs" {
  description = "CIDR blocks for private
  subnets" type     = list(string)
  default     = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
}

variable "public_subnet_cidrs" {
  description = "CIDR blocks for public
  subnets" type     = list(string)
  default     = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
}

variable "cluster_name" {
  description = "Name of the EKS
  cluster" type      = string
  default     = "my-eks-cluster"
}

variable "cluster_version" {
  description = "Kubernetes
  version" type      = string
  default     = "1.30"
}

variable "node_groups" {
  description = "EKS node group configuration"
```

```
  type = map(object({
    instance_types = list(string)
    capacity_type  = string
    scaling_config = object({
      desired_size =
      number max_size
                   = number
      min_size     = number
    })
  }))
  default =
    { general = {
      instance_types = ["t3.medium"]
      capacity_type  = "ON_DEMAND"
      scaling_config = {
        desired_size = 2
        max_size     = 4
        min_size     = 1
      }
    }
  }
}
```

On the root directory create

- main.tf

```
terraform {
  required_providers {
```

```
type = map(object({
  aws = {
```

```
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket         = "remote-backend-
    s3" key       = "terraform.tfstate"
    region         = "us-west-2"
    dynamodb_table = "remote-backend-
    locks" encrypt  = true
  }
}

provider "aws" {
  region = var.region
}

module "vpc" {
  source = "./modules/vpc"

  vpc_cidr             = var.vpc_cidr
  availability_zones  = var.availability_zones
  private_subnet_cidrs = var.private_subnet_cidrs
  public_subnet_cidrs  = var.public_subnet_cidrs
  cluster_name         = var.cluster_name
```

```
}

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

  cluster_name    = var.cluster_name
  cluster_version = var.cluster_version
  vpc_id          = module.vpc.vpc_id
  subnet_ids      = module.vpc.private_subnet_ids
  node_groups     = var.node_groups
}
```

outputs.tf

```
output "cluster_endpoint" {
  description = "EKS cluster endpoint"
  value       =
  module.eks.cluster_endpoint
}

output "cluster_name" {
  description = "EKS cluster name"
  value       =
  module.eks.cluster_name
}

output "vpc_id" {
  description = "VPC ID"
  value       = module.vpc.vpc_id
```

```
    }
    }
```

variables.tf

```
variable "region" {
  description = "AWS
  region" type       = string
  default    = "us-west-2"
}

variable "vpc_cidr" {
  description = "CIDR block for
  VPC" type = string
  default    = "10.0.0.0/16"
}

variable "availability_zones" {
  description = "Availability
  zones" type       = list(string)
  default    = ["us-west-2a", "us-west-2b", "us-west-2c"]
}

variable "private_subnet_cidrs" {
  description = "CIDR blocks for private
  subnets" type    = list(string)
  default    = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
}

variable "public_subnet_cidrs" {
```

```
  description = "CIDR blocks for public
  subnets" type     = list(string)
  default     = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
}

variable "cluster_name" {
  description = "Name of the EKS
  cluster" type       = string
  default     = "my-eks-cluster"
}

variable "cluster_version" {
  description = "Kubernetes
  version" type       = string
  default     = "1.30"
}

variable "node_groups" {
  description = "EKS node group
  configuration" type = map(object({
    instance_types = list(string)
    capacity_type  = string
    scaling_config = object({
      desired_size =
      number max_size
                   = number
      min_size     = number
```

```
    })
  }))
  default =
   { general = {
     instance_types = ["t3.medium"]
     capacity_type  = "ON_DEMAND"
     scaling_config = {
      desired_size = 2
      max_size     = 4
      min_size     = 1
     }
    }
  }
}
```

Go to the terminal the change directory into the root folder

- Initialize terraform

terraform init

- Validate the resources that will be

created terraform plan

- Create the resources in

aws terraform apply

- To connect to kubernetes cluster

aws eks update-kubeconfig --region region --name cluster-name

- To view kubernetes

onfig kubectl config view

- To get current kubernetes config

kubectl config current-context

- To switch between kubernetes

cluster kubectl config use-context

- Clean up the

resources terraform destroy

Conclusion:

Setting up a **Highly Available Amazon EKS Cluster using Terraform** is a robust, scalable, and production-grade way to run Kubernetes workloads on AWS. The process involves:

- Designing a secure and redundant **network infrastructure** (VPC, subnets, NAT, routing).
- Provisioning **IAM roles** that allow secure communication between AWS and Kubernetes components.
- Deploying a **multi-AZ EKS control plane** with **managed node groups** for high availability.
- Leveraging **Terraform's infrastructure-as-code** capabilities to automate, manage, and version the entire setup reliably.
- Securing access via **RBAC and IAM mappings**, and enabling observability through logs and monitoring tools.
- Managing state and team collaboration using **remote state backends** like S3 and DynamoDB.

By following this structured approach, your EKS cluster will be:

- **Highly Available** across multiple AZs
- **Secure** through proper IAM and network controls
- **Scalable** with autoscaling node groups
- **Maintainable and repeatable** through Terraform

This setup forms the **foundation for a modern, cloud-native DevOps environment**, enabling smooth CI/CD, microservices orchestration, and future growth.

Let me know if you want this turned into a **document, deck, or infographic** for your team or clients!