2025

Terraform Deployment

DEPLOYING HIGHLY AVAILABLE INFRASTRUCTURE FOR EKS USING TERRAFORM

VIRAJ ARIYASINGHE

Contents

1. Introduction

This document outlines the process of setting up a 3-tier architecture in AWS using Terraform, deploying an EKS (Elastic Kubernetes Service) cluster within the VPC, and managing applications on the cluster using Kubernetes. The guide covers the creation of VPC components, EKS cluster setup, and deploying applications using Kubernetes tools.

1.1 Current Architecture Overview

1.1.1 Architecture Description

Our current architecture consists of a three-tier model:

- Presentation Tier (Frontend): Angular application running on Apache server, serving as the user interface.
- Application Tier (Backend): Microservices built using Spring Boot, deployed on EC2 instances, handling business logic and application processing.
- Database Tier: Databases hosted on AWS RDS, handling persistent data storage.

1.1.2. Deployment and Components

- Frontend: Deployed on an Apache server in a public subnet of an AWS VPC.
- Backend Microservices: Deployed on EC2 instances in a private subnet, running nine different Spring Boot microservices.
- Database: AWS RDS instances in a private subnet, accessed by backend microservices through HAProxy and EC2.

1.2. Kubernetes Architecture Types

1.2.1. Microservices Architecture

- Definition: Microservices architecture decomposes an application into smaller, independent services, each responsible for a specific functionality. Each microservice can be developed, deployed, and scaled independently.

Deployment in Kubernetes:

- Namespaces: Create separate namespaces for different environments (e.g., dev, staging, prod) and teams.
- Pods: Deploy individual microservices as separate pods within the cluster.
- Services: Expose each microservice using Kubernetes services (Cluster IP, Node Port, or Load Balancer).
- Deployments: Use Kubernetes Deployments to manage the lifecycle of microservices, including scaling and rolling updates.

Pros:

- Scalability: Independent scaling of microservices based on demand.
- Flexibility: Different technologies or languages can be used for different microservices.
- Fault Isolation: Failure in one microservice does not impact others.

Cons:

- Complexity: Increased complexity in managing multiple services and their interactions.
- Overhead: Potential overhead in inter-service communication and data consistency.

Example Deployment:

- Frontend: Deploy Angular application as a pod in a namespace like frontend or prod.
- Backend: Deploy each microservice as a separate pod in a backend namespace.
- Database: Use managed database services or deploy database containers, if applicable.

1.2.2. Three-Tier Architecture

- Definition: A three-tier architecture separates the application into three distinct layers: presentation, application, and database. Each tier can be scaled and managed independently.

Deployment in Kubernetes:

- Namespaces: Create namespaces for each tier if needed (e.g., frontend, backend, database).
- Pods: Deploy frontend, backend services, and database components in their respective namespaces.
- Services: Expose frontend services using Load Balancer or Node Port, backend services using Cluster IP, and manage database access.

Pros:

- Clear Separation: Well-defined separation of concerns between presentation, application, and database layers.
- Manageability: Easier to manage and scale different tiers independently.

Cons:

- Less Granular Scaling: Less flexibility in scaling individual services compared to microservices architecture.
- Monolithic Tendencies: May lead to more monolithic deployments if not properly managed.

Example Deployment:

- Frontend: Deploy as a pod in a frontend namespace.
- Backend: Deploy as a set of pods in a backend namespace.
- Database: Deploy database components in a database namespace or use managed services.

1.3 Terraform Project Structure

1.3.1 Root Module (Main configuration Files)

- The root module is the main configuration entry point.
- Declares variables to accept user-defined inputs.
- Invokes resource modules using the module block.
- Passes values to and retrieves values from resource modules.

Example:

```
module "vpc" {
  source   = "./aws-modules/modules/vpc"
  vpc_name = var.vpc_name
  vpc_cidr = var.vpc_cidr
}

module "eks" {
  source  = "./aws-modules/modules/eks"
  eks_name = var.eks_name
  vpc_id   = module.vpc.vpc_id  # Retrieves output from the vpc module
}
```

1.3.2 Resource Modules

- Resource modules encapsulate reusable configurations for specific AWS resources, such as VPCs or EKS clusters. They:
- Declare **variables** to accept inputs from the root module.
- Create resources based on these inputs.
- Define **outputs** to expose values (e.g., resource IDs) to other modules or the root module.
-

Variables in a Resource Module:

- **Input Variables**: Accept values passed by the root module.

```
variable "vpc_name" {
  description = "Name of the VPC"
  type        = string
}
```

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

Outputs in a Resource Module:

- **Output Values**: Provide resource details to the root module.

```
output "vpc_id" {
  description = "The ID of the created VPC"
  value       = aws_vpc.main.id
}
```

1.3.3 Linking Between Modules

- Coordination happens through **variables** and **outputs**.

I. **Root Module → Resource Module**:
   - Pass input variables to a resource module.

```
module "vpc" {
  source   = "./aws-modules/modules/vpc"
  vpc_name = var.vpc_name
  vpc_cidr = var.vpc_cidr
}
```

II. **Resource Module → Root Module**:
   - Expose outputs from the resource module to the root module.

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

III. **Root Module → Another Resource Module**:
   - Use outputs from one resource module as inputs for another resource module.

```
module "eks" {
  source  = "./aws-modules/modules/eks"
  vpc_id  = module.vpc.vpc_id
}
```

1.3.4 Execution Flow

I. **Define Variables**:
   - The root module defines variables for user input, such as vpc_name or eks_name.
   - Resource modules define their own variables for resource-specific configurations.

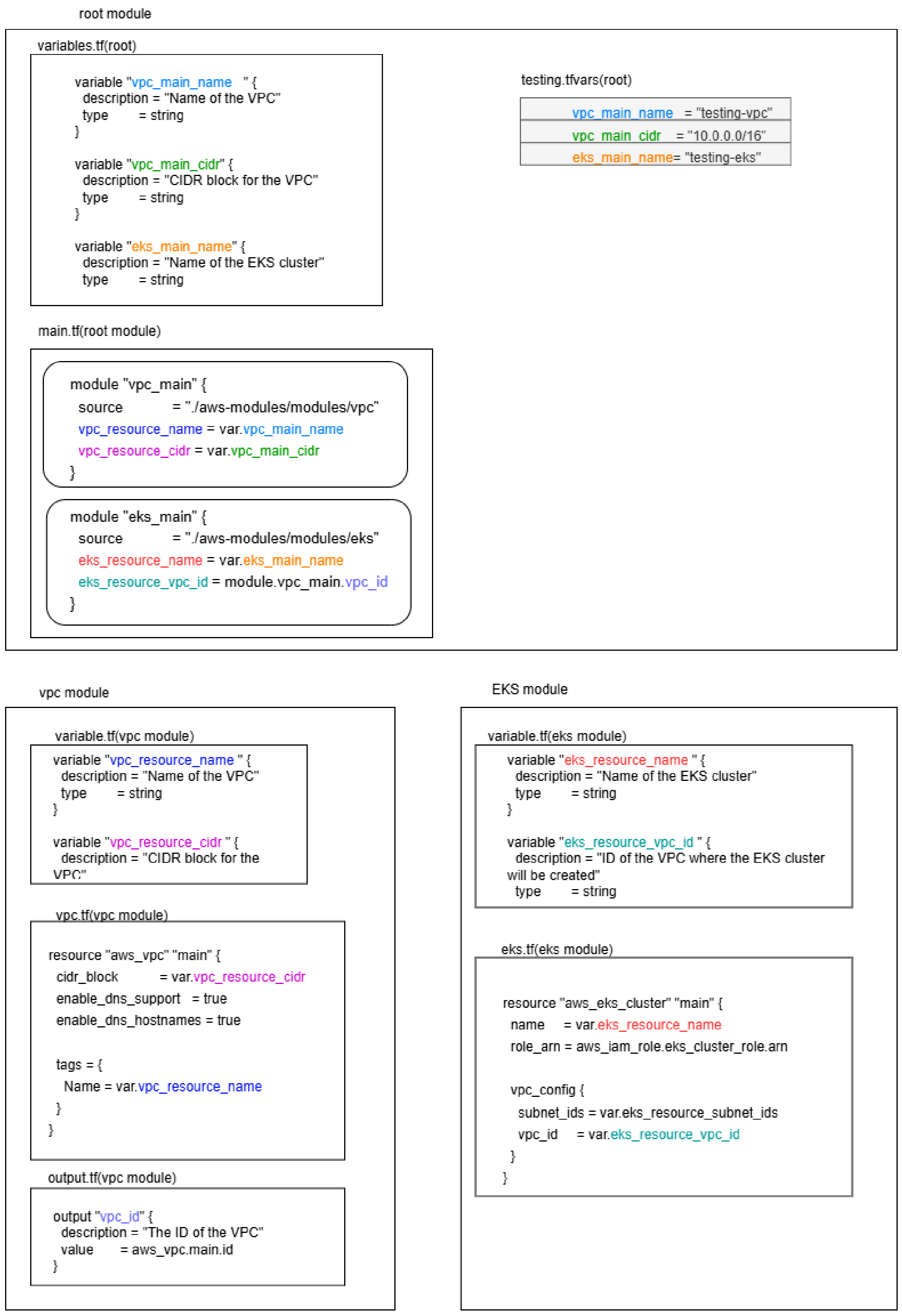
II. **Call Resource Modules**:
   - The root module invokes the vpc resource module to create a VPC.
   - The root module retrieves the vpc_id output from the vpc module.

III. **Pass Values Between Modules**:
   - The root module passes the vpc_id from the vpc module to the eks module to deploy the EKS cluster within the created VPC.

IV. **Outputs for Debugging/Consumption**:
   - The root module can expose key outputs (e.g., vpc_id, eks_endpoint) for consumption by other systems or users.

1.3.5 Benefits of This Approach

- **Reusability**: Resource modules are reusable across multiple environments.
- **Modularity**: Keeps configurations clean and manageable.
- **Flexibility**: Easily link resource dependencies by passing outputs as inputs.



variables.tf(root)

```
variable "vpc_main_name  " {
  description = "Name of the VPC"
  type        = string
}

variable "vpc_main_cidr" {
  description = "CIDR block for the VPC"
  type        = string
}

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

testing.tfvars(root)

```
vpc_main_name  = "testing-vpc"
vpc_main_cidr   = "10.0.0.0/16"
eks_main_name= "testing-eks"
```

main.tf(root module)

```
module "vpc_main" {
  source          = "./aws-modules/modules/vpc"
  vpc_resource_name = var.vpc_main_name
  vpc_resource_cidr = var.vpc_main_cidr
}

module "eks_main" {
  source          = "./aws-modules/modules/eks"
  eks_resource_name = var.eks_main_name
  eks_resource_vpc_id = module.vpc_main.vpc_id
}
```

variable.tf(vpc module)

```
variable "vpc_resource_name " {
  description = "Name of the VPC"
  type        = string
}

variable "vpc_resource_cidr " {
  description = "CIDR block for the
VPC"
```

vpc.tf(vpc module)

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

  tags = {
    Name = var.vpc_resource_name
  }
}
```

output.tf(vpc module)

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

variable.tf(eks module)

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

variable "eks_resource_vpc_id " {
  description = "ID of the VPC where the EKS cluster
will be created"
  type        = string
```

eks.tf(eks module)

```
resource "aws_eks_cluster" "main" {
  name     = var.eks_resource_name
  role_arn = aws_iam_role.eks_cluster_role.arn

  vpc_config {
    subnet_ids = var.eks_resource_subnet_ids
    vpc_id     = var.eks_resource_vpc_id
  }
}
```

1.3.6 Summary Table

Component	**Purpose**	**Example**
Root Module	Main entry point; calls resource modules and passes inputs and retrieves outputs.	module "vpc", module "eks"
Resource Module	Encapsulates logic for creating specific resources.	VPC and EKS modules
Input Variables	Allow resource modules to accept values from the root module.	variable "vpc_name"
Outputs	Expose resource details (e.g., vpc_id) to the root module for further use.	output "vpc_id"
Root → Resource	Passes inputs to resource modules.	vpc_name = var.vpc_name
Resource → Resource	Links outputs from one resource module as inputs to another module through the root module.	vpc_id = module.vpc.vpc_id

1.3.7 Security Enhancements

- **Restrict CIDR Ranges**: Avoid 0.0.0.0/0 in security groups unless necessary.
- **Enable IAM Roles**: Attach least-privilege policies to instances.
- **Use Network ACLs**: Implement strict rules for ingress and egress.

1.3.8 Versioning

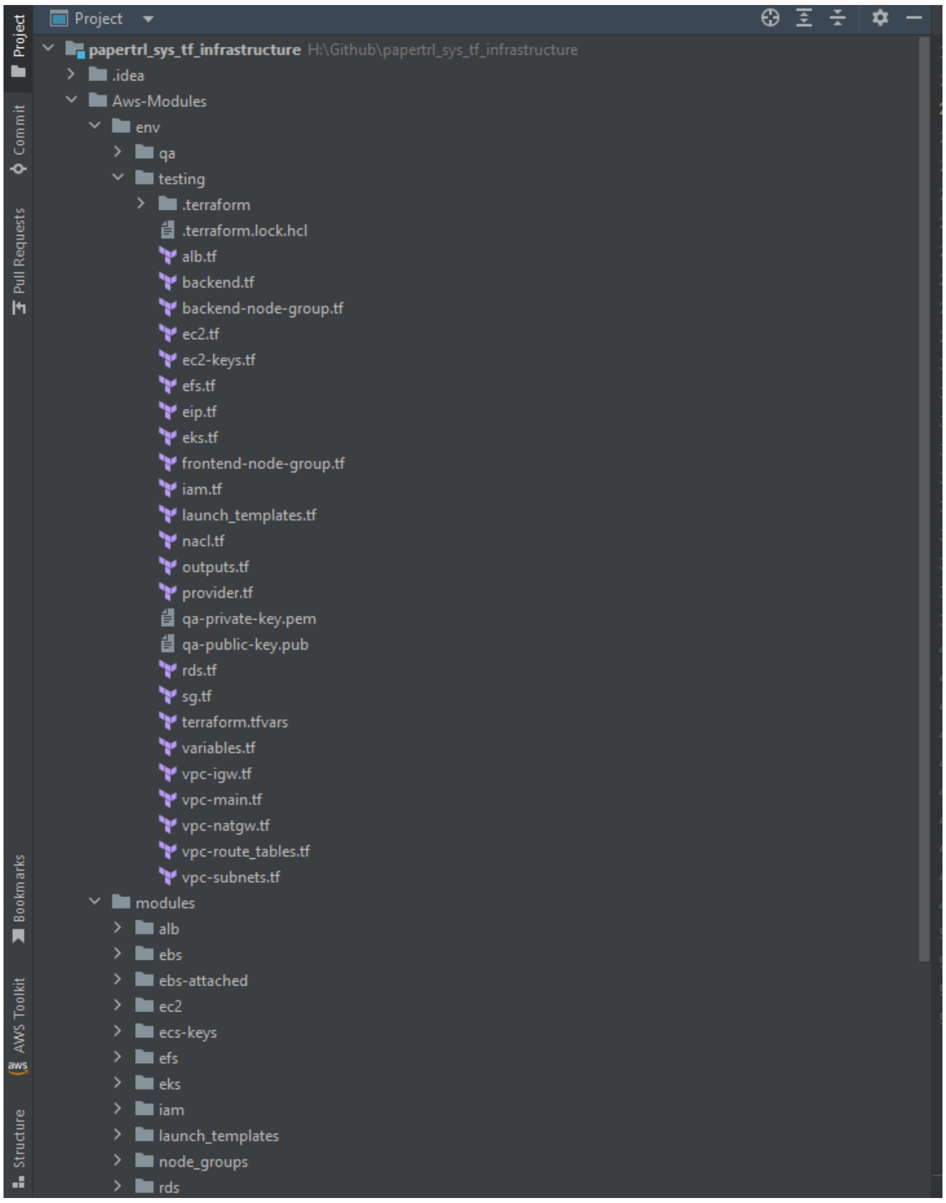
Recommended Versions:

- Terraform: 1.5.2
- AWS Provider: 5.0
- Kubernetes: 1.27

Location to Define Versions: Include version constraints in provider.tf:

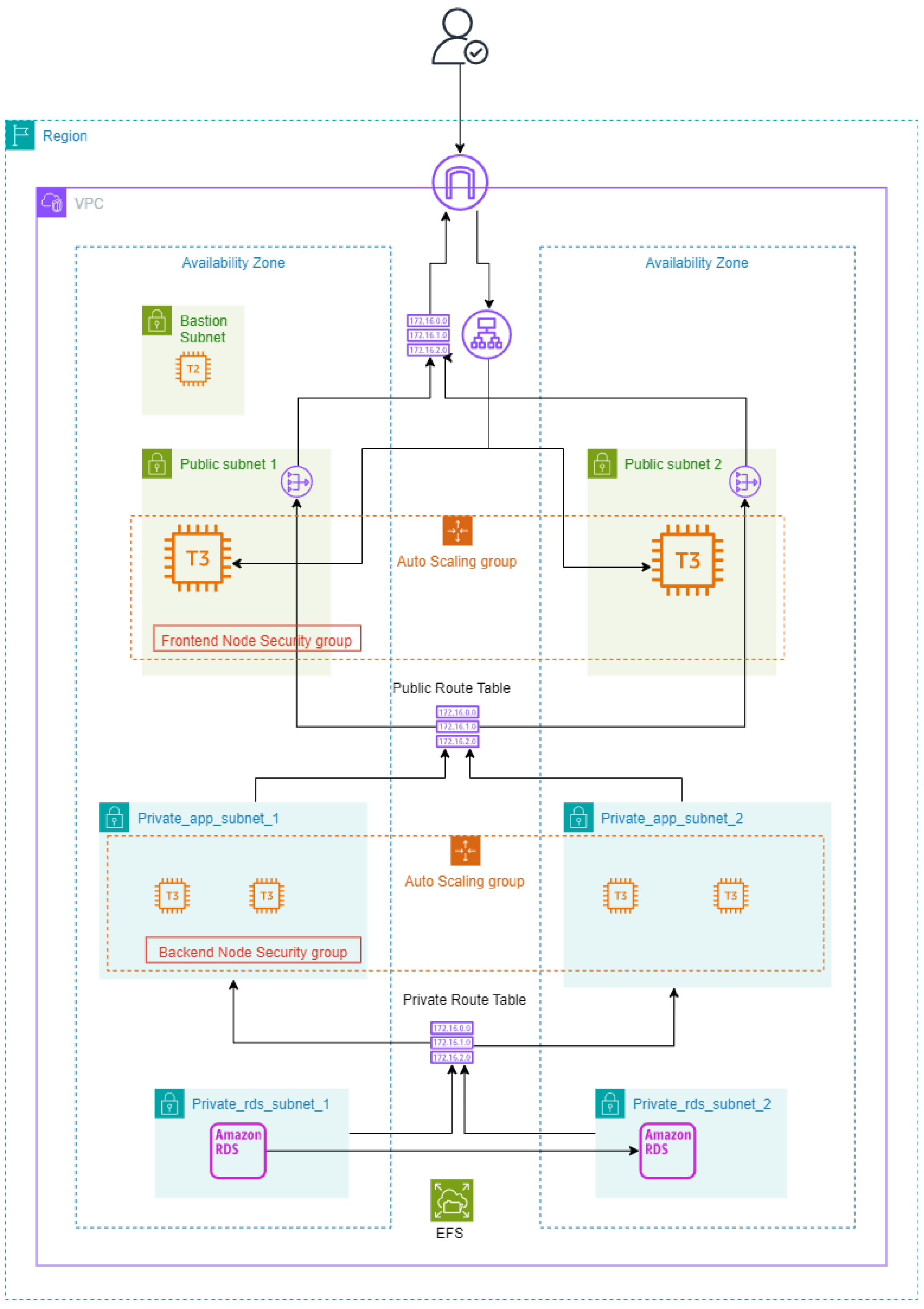
2. Creating a VPC for EKS Cluster Using Terraform

2.1 Terraform project structure



```
papertrl_sys_tf_infrastructure H:\Github\papertrl_sys_tf_infrastructure
  .idea
  Aws-Modules
    env
      qa
      testing
        .terraform
        .terraform.lock.hcl
        alb.tf
        backend.tf
        backend-node-group.tf
        ec2.tf
        ec2-keys.tf
        efs.tf
        eip.tf
        eks.tf
        frontend-node-group.tf
        iam.tf
        launch_templates.tf
        nacl.tf
        outputs.tf
        provider.tf
        qa-private-key.pem
        qa-public-key.pub
        rds.tf
        sg.tf
        terraform.tfvars
        variables.tf
        vpc-igw.tf
        vpc-main.tf
        vpc-natgw.tf
        vpc-route_tables.tf
        vpc-subnets.tf
    modules
      alb
      ebs
      ebs-attached
      ec2
      ecs-keys
      efs
      eks
      iam
      launch_templates
      node_groups
      rds
```

```
eks-project/
├── aws-modules/
│   ├── env/
│   │   ├── testing/
│   │       ├── main.tf
│   │       ├── variables.tf
│   │       ├── outputs.tf
│   │       ├── backend.tf
│   │       ├── provider.tf
│   │       ├── testing.tfvars
│   │   ├── qa/
│   │   ├── prod/
│   ├── modules/
│       ├── vpc/
│       │   ├── vpc.tf
│       │   ├── variables.tf
│       │   ├── outputs.tf
│       ├── eks/
│       │   ├── eks.tf
│       │   ├── variables.tf
│       │   ├── outputs.tf
│       ├── ec2/
│       ├── rds/
│       ├── iam/
│       ├── efs/
│       ├── s3/
│       ├── vpc-nacl/
│       ├── vpc-subnets/
│       ├── vpc-security-groups/
```


Region
VPC
Availability Zone
Availability Zone
Bastion Subnet
T2
172.16.0.0
172.16.1.0
172.16.2.0
Public subnet 1
Public subnet 2
T3
Auto Scaling group
T3
Frontend Node Security group
Public Route Table
172.16.0.0
172.16.1.0
172.16.2.0
Private_app_subnet_1
Private_app_subnet_2
Auto Scaling group
T3
T3
T3
T3
Backend Node Security group
Private Route Table
172.16.0.0
172.16.1.0
172.16.2.0
Private_rds_subnet_1
Private_rds_subnet_2
Amazon RDS
Amazon RDS
EFS

2.2 Project Modules

I. Main Configuration Files (Aws-Modules/env/testing/*.tf):

- This is the main entry point where you define the high-level configuration and use modules.
- It references variables and modules, allowing you to reuse the same configuration across different environments.

```
module "vpc" {
  source = "../../modules/vpc" # Adjust this path if your VPC module is in a
different directory

  region       = var.region
  vpc_cidr     = var.vpc_cidr
  vpc_name     = var.vpc_name
  vpc_tenancy = var.vpc_tenancy
  env_name     = var.env_name

}
```

II. Variables Definition (variables.tf):

- This file contains the variable definitions used in your Terraform configuration. Variables can be strings, numbers, lists, maps, or other complex types.

```
variable "bastion_ami_id" {
  type        = string
  description = "AMI  of the bastion instance"
}

variable "bastion_instance_type" {
  type        = string
  description = "Instance type"
}
```

III. Outputs (outputs.tf):

- Outputs provide information after the resources are created, such as IP addresses or DNS names, which can be useful for further configuration or integration.

```
output "Testing_SSH_Instance_Public_Ip" {
  value = module.ec2.ssh_Instance_public_IP
}

output "Testing_EFS_Filesystem_ID" {
  value = module.efs.efs_file_system_id
}
```

IV. Environment-Specific Variables (*.tfvars):

- These files contain values for variables specific to each environment. For example, different instance types, CIDR blocks, or AMI IDs.

```
cluster_name    = "testing-vpc-eks-cluster"
vpc_name        = "020_TESTING_VPC"
env_name        = "testing-vpc"
vpc_cidr        = "10.0.0.0/16"
vpc_tenancy     = "default"

public_bastion_subnet_az_01_cidr = "10.0.10.0/24"

web_public_subnet_az1_cidr = "10.0.11.0/24"
web_public_subnet_az2_cidr = "10.0.12.0/24"

private_app_subnet_az1_cidr = "10.0.21.0/24"
private_app_subnet_az2_cidr = "10.0.22.0/24"

private_rds_subnet_az1_cidr = "10.0.31.0/24"
private_rds_subnet_az2_cidr = "10.0.32.0/24"
```

To successfully integrate VPC resources with the EKS cluster, ensure the following tags are applied:

- `kubernetes.io/cluster/<cluster-name>` = `shared` for better tracking and cluster association.

These tags are critical for EKS-managed services like Load Balancers and Auto Scaling Groups to function seamlessly with VPC resources.

V. Modules (modules/):

- Modules are reusable components that define specific resources. They encapsulate configurations for resources like VPCs, EC2 instances, etc., so they can be reused across environments.

VI. Backend (Backend.tf)

- The `backend.tf` file is used to configure the backend where Terraform stores its state file. The state file keeps track of the current state of your infrastructure and is critical for Terraform to manage resources effectively.

```
terraform {
  backend "s3" {
    bucket = "papertrl-terraform"
    key    = "testing-env/teraform.tfstate"
    region = "us-east-2"
    #dynamodb_table = "terraform-state-lock"
    encrypt = true
    profile = "Terraform_User"
  }
}
```

VII. Provider.tf

- The `provider.tf` file is used to define the providers that Terraform will use to manage and interact with your infrastructure

```
# Configure the AWS Provider
provider "aws" {
  #  access_key = var.AWS_ACCESS_KEY
  #  secret_key = var.AWS_SECRET_KEY
  region  = var.region
  profile = "Terraform_User"
}


#provider "kubernetes" {
#  host                     = module.vpc.eks_cluster_endpoint
#  cluster_ca_certificate =
base64decode(module.vpc.eks_cluster_certificate_authority)
#  token                    = data.aws_eks_cluster_auth.example.token
#}
#
#data "aws_eks_cluster_auth" "example" {
#  name = module.vpc.eks_cluster_name
#}
```

2.3 VPC Configuration

The first step is to create a Virtual Private Cloud (VPC) that will host all the resources. The VPC will have public and private subnets distributed across availability zones.

Create the resources in the vpc.tf file and set variables in vpc modules variable.tf file.

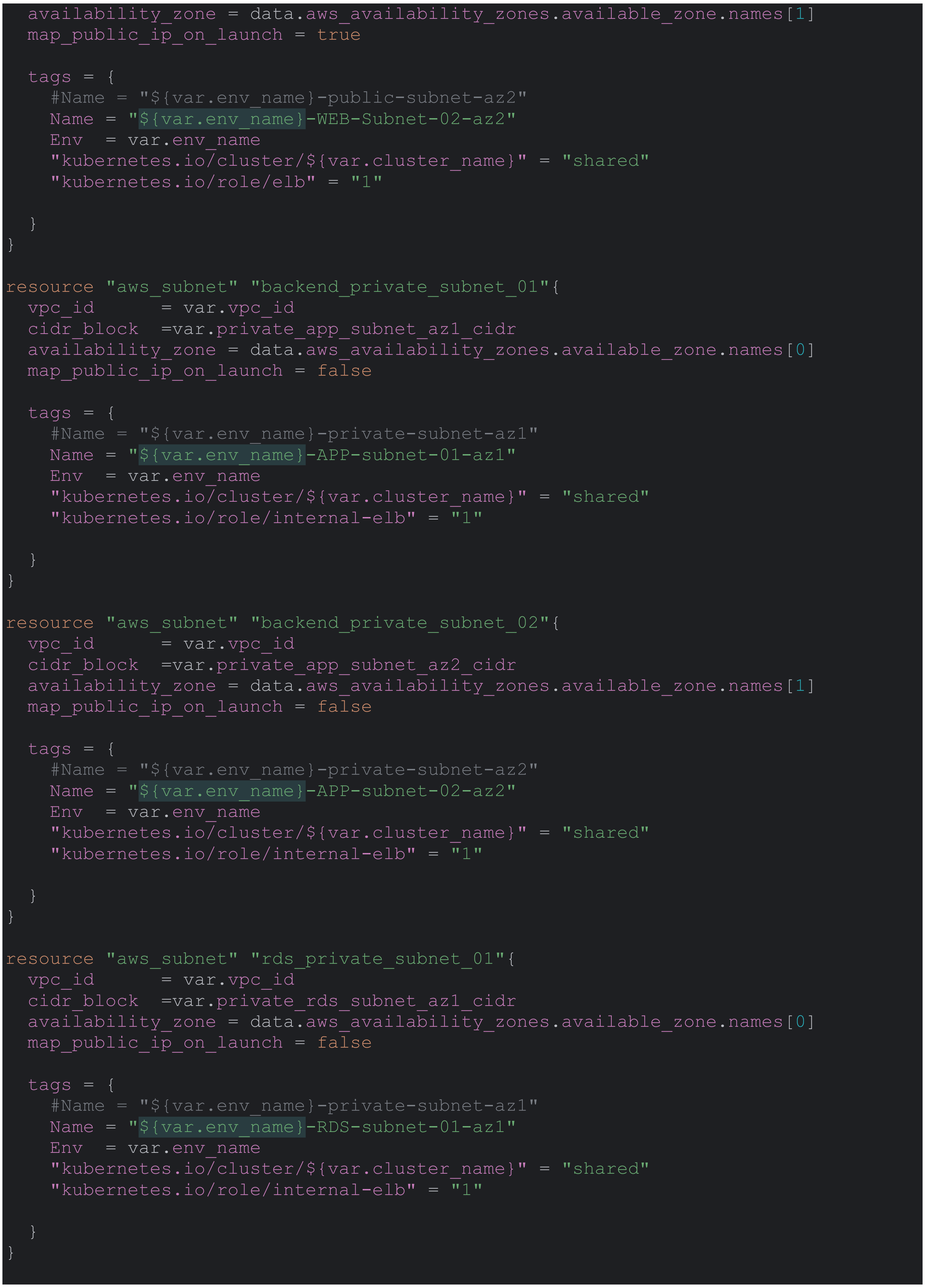
I. VPC: Create a VPC with CIDR block, typically /16.

```
## source = "../Aws-Modules/modules/vpc/vpc.tf"
## VPC Declaration

#=========================================================================

resource "aws_vpc" "main_vpc" {
  cidr_block             = var.vpc_cidr
  enable_dns_support     = true
  enable_dns_hostnames   = true
  instance_tenancy       = var.vpc_tenancy

  tags = {
    Name = var.vpc_name
    Env  = var.env_name
  }
}
```

II. Subnets:

- Public Subnets: Three public subnets in different availability zones for load balancers and Bastion host.
- Private Subnets: Two private subnets for application servers.
- Private Subnets (DB Layer): Two private subnets dedicated to databases.
- Add the tags to subnets, These tags are critical to identify subnets by EKS managed services such as ELB and ASG
  - `kubernetes.io/cluster/<cluster-name>` = `shared` or `owned` (based on the cluster's control over the resource).
  - `kubernetes.io/role/elb` = `1` for public subnets used by Load Balancers.
  - `kubernetes.io/role/internal-elb` = `1` for private subnets.

```
## source = "../Aws-Modules/modules/vpc-subnets/subnets.tf"
## Subnets

#========================================================================

data "aws_availability_zones" "available_zone" {}

resource "aws_subnet" "public_subnet_bastion_az_01" {

  vpc_id     = var.vpc_id
  cidr_block = var.public_bastion_subnet_az_01_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[0]
  map_public_ip_on_launch = true

  tags = {
    #Name = "${var.env_name}-public-subnet-bastian-az1"
    Name = "${var.env_name}-SSH-subnet"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}

resource "aws_subnet" "web_public_subnet_az1"{
  vpc_id      = var.vpc_id
  cidr_block  =var.web_public_subnet_az1_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[0]
  map_public_ip_on_launch = true

  tags = {
    #Name = "${var.env_name}-public-subnet-az1"
    Name = "${var.env_name}-WEB-Subnet-01-az1"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/elb" = "1"

  }
}

resource "aws_subnet" "web_public_subnet_az2"{
  vpc_id      = var.vpc_id
  cidr_block  =var.web_public_subnet_az2_cidr
```



```
  availability_zone = data.aws_availability_zones.available_zone.names[1]
  map_public_ip_on_launch = true

  tags = {
    #Name = "${var.env_name}-public-subnet-az2"
    Name = "${var.env_name}-WEB-Subnet-02-az2"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/elb" = "1"

  }
}

resource "aws_subnet" "backend_private_subnet_01"{
  vpc_id      = var.vpc_id
  cidr_block  =var.private_app_subnet_az1_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[0]
  map_public_ip_on_launch = false

  tags = {
    #Name = "${var.env_name}-private-subnet-az1"
    Name = "${var.env_name}-APP-subnet-01-az1"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb" = "1"

  }
}

resource "aws_subnet" "backend_private_subnet_02"{
  vpc_id      = var.vpc_id
  cidr_block  =var.private_app_subnet_az2_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[1]
  map_public_ip_on_launch = false

  tags = {
    #Name = "${var.env_name}-private-subnet-az2"
    Name = "${var.env_name}-APP-subnet-02-az2"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb" = "1"

  }
}

resource "aws_subnet" "rds_private_subnet_01"{
  vpc_id      = var.vpc_id
  cidr_block  =var.private_rds_subnet_az1_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[0]
  map_public_ip_on_launch = false

  tags = {
    #Name = "${var.env_name}-private-subnet-az1"
    Name = "${var.env_name}-RDS-subnet-01-az1"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb" = "1"

  }
}
```

```
resource "aws_subnet" "rds_private_subnet_02"{
  vpc_id      = var.vpc_id
  cidr_block  =var.private_rds_subnet_az2_cidr
  availability_zone = data.aws_availability_zones.available_zone.names[1]
  map_public_ip_on_launch = false

  tags = {
    #Name = "${var.env_name}-private-subnet-az2"
    Name = "${var.env_name}-RDS-subnet-02-az2"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb" = "1"

  }
}
```

III. Network Components

- **Internet Gateway**: Attach an Internet Gateway to the VPC for internet access.

```


## source = "../Aws-Modules/modules/vpc-igw/igw.tf"
## Internet Gateway Declaration

#========================================================================

resource "aws_internet_gateway" "igw" {
  vpc_id = var.vpc_id

  tags = {
    Name   = "${var.env_name}-igw"
    vpc    = var.vpc_name
    Env    = var.env_name
  }

}
```

- **NAT Gateways**: Create two NAT Gateways, one in each public subnet, for outbound internet access from private subnets.

```


## source = "../Aws-Modules/modules/vpc-eip/v.tf"
## Elastic IP`s For Nat Gateways

#========================================================================

resource "aws_eip" "eip_for_nat_az1" {
  domain = "vpc"

  tags = {
    Name = "${var.env_name}-nat-eip-az1"
    Env  = var.env_name
  }
}
```

```
resource "aws_eip" "eip_for_nat_az2" {
  domain = "vpc"

  tags = {
    Name = "${var.env_name}-nat-eip-az2"
    Env  = var.env_name
  }
}
```

```
## source = "../Aws-Modules/modules/vpc-natgw/natgw.tf"
## Nat Gateways

#=========================================================================

resource "aws_nat_gateway" "nat_gw_az1" {
  allocation_id = var.eip_nat_az1
  subnet_id     = var.public_subnet_az1_id

  tags = {
    Name = "${var.env_name}-nat-gateway-az1"
    Env  = var.env_name
  }

}

resource "aws_nat_gateway" "nat_gw_az2" {
  allocation_id = var.eip_nat_az2
  subnet_id     = var.public_subnet_az2_id

  tags = {
    Name = "${var.env_name}-nat-gateway-az2"
    Env  = var.env_name
  }
}
```

IV. **Route Tables**:
- Public Route Table: Configure routes to send internet-bound traffic to the Internet Gateway.
- Private Route Table: Configure routes to send internet-bound traffic to the NAT Gateway.
- After creating Route Tables We should assign them to the subnets.

- Public Route Table

```
## source = "../Aws-Modules/modules/vpc-route_tables/public_rt.tf"
## Public Route Tables For public subnets

#=========================================================================

resource "aws_route_table" "main_public_rt" {

  vpc_id = var.vpc_id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = var.igw_id
  }
  route {
    cidr_block = var.vpc_cidr_block
    gateway_id = "local"
  }

  tags = {
    Name = "${var.env_name}_public_rt"
    Env  = var.env_name
#     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}
```

- Private Route Table

```
## source = "../Aws-Modules/modules/vpc-route_tables/private_rt.tf"
## Private Route Tables For private subnets

#========================================================================

resource "aws_route_table" "main_private_rt_1" {

  vpc_id = var.vpc_id

  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = var.nat_gw_az1_id
  }

  route {
    cidr_block = var.vpc_cidr_block
    gateway_id = "local"
  }

  tags = {
    Name = "${var.env_name}_private_rt_1"
    Env  = var.env_name
#     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}

resource "aws_route_table" "main_private_rt_2" {

  vpc_id = var.vpc_id

  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = var.nat_gw_az2_id
  }

  route {
    cidr_block = var.vpc_cidr_block
    gateway_id = "local"
  }

  tags = {
    Name = "${var.env_name}_private_rt_2"
    Env  = var.env_name
#     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
```

- Route Table Associations

```
resource "aws_route_table_association" "main_bastion_subnet_association" {
  subnet_id      = var.bastion_subnet_id
  route_table_id = aws_route_table.main_public_rt.id
}

resource "aws_route_table_association" "main_public_subnet_1_association" {
  subnet_id      = var.public_subnet_az1_id
  route_table_id = aws_route_table.main_public_rt.id
}

resource "aws_route_table_association" "main_public_subnet_2_association" {
  subnet_id      = var.public_subnet_az2_id
  route_table_id = aws_route_table.main_public_rt.id
}

resource "aws_route_table_association" "main_private_app_subnet_1_association" {
  subnet_id      = var.app_subnet_az1_id
  route_table_id = aws_route_table.main_private_rt_1.id
}

resource "aws_route_table_association" "main_private_app_subnet_2_association" {
  subnet_id      = var.app_subnet_az2_id
  route_table_id = aws_route_table.main_private_rt_2.id
}

resource "aws_route_table_association" "main_private_rds_subnet_1_association" {
  subnet_id      = var.rds_subnet_az1_id
  route_table_id = aws_route_table.main_private_rt_1.id
}

resource "aws_route_table_association" "main_private_rds_subnet_2_association" {
  subnet_id      = var.rds_subnet_az2_id
  route_table_id = aws_route_table.main_private_rt_2.id
}
```

V. **Network ACLs**: Create Network Access Control Lists (NACLs) for controlling traffic in and out of your subnets.

- Bastion Subnet Network ACL

```
## source = "../Aws-Modules/modules/vpc-nacl/Bastion_nacl.tf"
## Network Acl-Bastion subnet

#==========================================================================

resource "aws_network_acl" "main_bastion_nacl" {

  vpc_id = var.vpc_id

  ingress {
    protocol   = "tcp"
    rule_no    = 100
    action     = "allow"
    cidr_block = var.vpn_ip
    from_port  = 22
    to_port    = 22
  }
  ingress {
    protocol   = "tcp"
    rule_no    = 101
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 1024
    to_port    = 65535
  }

  egress {
    protocol   = "-1"
    rule_no    = 200
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 0
  }



  tags = {
    Name                                            = "${var.env_name}_SSH_Net_acl"
    Env                                             = var.env_name
#    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}

resource "aws_network_acl_association" "bastion_nacl_association" {
  subnet_id      = var.bastion_subnet_id
  network_acl_id = aws_network_acl.main_bastion_nacl.id
}
```

- Public Subnet Network ACL

```
## source = "../Aws-Modules/modules/vpc-nacl/Public_subnet_nacl.tf"
## Network Acl-Public subnet

#=========================================================================

resource "aws_network_acl" "main_public_nacl" {
  vpc_id = var.vpc_id

  #description = "Allow http web traffic to the subnet"
  ingress {
    protocol   = "tcp"
    rule_no    = 101
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 80
    to_port    = 80
  }

  #description = "Allow https web traffic to the subnet"
  ingress {
    protocol   = "tcp"
    rule_no    = 102
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port   = 443
    to_port     = 443
  }

  #description = "Allow ssh traffic to the public subnet from the bastion subnet"
  ingress {
    protocol    = "tcp"
    rule_no     = 103
    action      = "allow"
    cidr_block = var.public-subnet-bastion-az_01_cidr
    from_port   = 22
    to_port     = 22
  }

  #description = "Allow EKS control plane to communicate with nodes"
  ingress {

    rule_no     = 104
    action      = "allow"
    from_port    = 10250
    to_port      = 10250
    protocol     = "tcp"
    cidr_block = var.vpc_cidr_block # Only allow traffic from within the VPC
  }

  #description = "Allow ingress tcp traffic on ephemarel ports for responses"
  ingress {
    protocol    = "tcp"
    rule_no     = 105
    action      = "allow"
    cidr_block = "0.0.0.0/0"
    from_port   = 1024
    to_port     = 65535
  }

  #description = "Allow ingress udp traffic on ephemarel ports for responses"
  ingress {
    protocol    = "udp"
    rule_no     = 106
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port   = 1024
    to_port     = 65535
  }


  #description = "Allow UDP DNS resolutions traffic "
  ingress {
    protocol    = "udp"
    rule_no     = 107
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port   = 53
    to_port     = 53
  }

  #description = "Allow TCP DNS resolutions traffic "
  ingress {
    protocol    = "tcp"
    rule_no     = 108
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port  = 53
    to_port    = 53
  }


  #description = "Allow all egress traffic "
  egress {
    protocol   = "-1"
    rule_no    = 201
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 0
  }


  tags = {
    Name                                                 = "${var.env_name}_WEB_Net_acl"
    Env                                                  = var.env_name
#    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}

resource "aws_network_acl_association" "public_nacl_association_1" {
  subnet_id      = var.public_subnet_az_01_id
  network_acl_id = aws_network_acl.main_public_nacl.id
}

resource "aws_network_acl_association" "public_nacl_association_2" {
  subnet_id      = var.public_subnet_az_02_id
  network_acl_id = aws_network_acl.main_public_nacl.id
}
```

- App Subnet (Private) Network ACl

```
## source = "../Aws-Modules/modules/vpc-nacl/App_subnet_nacl.tf"
## Network Acl-App subnet

#=========================================================================

resource "aws_network_acl" "main_private_app_nacl" {
  vpc_id = var.vpc_id

  #description = "Allow SSH traffic to the public subnet from the bastion subnet"
  ingress {
    protocol   = "tcp"
    rule_no    = 101
    action     = "allow"
    cidr_block = var.public-subnet-bastion-az_01_cidr
    from_port  = 22
    to_port    = 22
  }

  #description = "Allow ingress TCP traffic on ephemarel ports for responses"
  ingress {
    protocol   = "tcp"
    rule_no    = 103
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 1024
    to_port     = 65535
  }

  #description = "Allow ingress UDP traffic on ephemarel ports for responses"
  ingress {
    protocol    = "udp"
    rule_no     = 109
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port   = 1024
    to_port     = 65535
  }


  #description = "Allow UDP DNS resolutions traffic "
  ingress {
    protocol    = "udp"
    rule_no     = 110
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port   = 53
    to_port     = 53
  }

  #description = "Allow TCP DNS resolutions traffic "
  ingress {
    protocol    = "tcp"
    rule_no     = 111
    action      = "allow"
    cidr_block = var.vpc_cidr_block
    from_port   = 53
    to_port     = 53
  }

  #description = "Allow EKS control plane to communicate with nodes"
  ingress {
    protocol    = "tcp"
    rule_no     = 112
    action      = "allow"
    #cidr_block = var.vpc_cidr_block
    cidr_block = "0.0.0.0/0"
    from_port   = 10250
    to_port     = 10250
  }

  #description = "Allow EKS API server to communicate with nodes"
  ingress {
    rule_no     = 107
    action      = "allow"
    from_port    = 443
    to_port      = 443
    protocol     = "tcp"
    cidr_block   = var.vpc_cidr_block # Only allow traffic from within the VPC
  }


  egress {
    protocol    = "-1"
    rule_no     = 201
    action      = "allow"
    cidr_block = "0.0.0.0/0"
    from_port   = 0
    to_port     = 0
  }


  tags = {
    Name    = "${var.env_name}_APP_Net_acl"
    Env     = var.env_name
#     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}


resource "aws_network_acl_association" "private_app_nacl_association_1" {
  subnet_id      = var.app_subnet_az_01_id
  network_acl_id = aws_network_acl.main_private_app_nacl.id
}

resource "aws_network_acl_association" "private_app_nacl_association_2" {
  subnet_id      = var.app_subnet_az_02_id
  network_acl_id = aws_network_acl.main_private_app_nacl.id
}
```

- DB Subnet (RDS) Network ACL

```
## source = "../Aws-Modules/modules/vpc-nacl/RDS_subnet_nacl.tf"
## Network Acl-RDS subnet

#========================================================================

resource "aws_network_acl" "main_private_rds_nacl" {
  vpc_id = var.vpc_id


  #description = "Allow SSH traffic to the public subnet from the bastion subnet"
  ingress {
    protocol   = "tcp"
    rule_no    = 101
    action     = "allow"
    cidr_block = var.vpc_cidr_block
    from_port  = 22
    to_port    = 22
  }

  #description = "Allow DB traffic to the public subnet from the bastion subnet"
  ingress {
    protocol   = "tcp"
    rule_no    = 102
    action     = "allow"
    cidr_block = var.vpc_cidr_block
    from_port  = 3306
    to_port    = 3306
  }

  #description = "Allow ingress TCP traffic on ephemarel ports for responses"
  ingress {
    protocol   = "tcp"
    rule_no    = 103
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port   = 1024
    to_port     = 65535
  }

  egress {
    protocol   = "-1"
    rule_no    = 202
    action     = "allow"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 0
  }

  tags = {
    Name                                          =
"${var.env_name}_private_rds_nacl"
    Env                                           = var.env_name
#     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
}

resource "aws_network_acl_association" "private_rds_nacl_association_1" {
  subnet_id      = var.rds_subnet_az_01_id
  network_acl_id = aws_network_acl.main_private_rds_nacl.id
}

resource "aws_network_acl_association" "private_rds_nacl_association_2" {
  subnet_id      = var.rds_subnet_az_02_id
  network_acl_id = aws_network_acl.main_private_rds_nacl.id
}
```

VI. **Security Groups**: Create security groups to control access to EC2 instances and other services.

- Bastion Instances Security Groups

```
## source = "../Aws-Modules/modules/vpc-security_groups/bastion_sg.tf"
## Bastion Security Group

#=======================================================================

# Define the security group without inline rules
resource "aws_security_group" "bastion_sg" {
  name   = "${var.env_name}_bastion_sg"
  vpc_id = var.vpc_id

  tags = {
    Name  = "${var.env_name}_bastion_sg"
    Env   = var.env_name
  }
}


# Ingress rule: Allow SSH from VPN IP
resource "aws_security_group_rule" "bastion_ingress_ssh" {
  type                        = "ingress"
  from_port                   = 22
  to_port                     = 22
  protocol                    = "tcp"
  security_group_id           = aws_security_group.bastion_sg.id
  cidr_blocks                 = [var.vpn_ip]
  description                = "Allow VPN connection for SSH"
}

# Egress rule: Allow all outbound traffic
resource "aws_security_group_rule" "bastion_egress_all" {
  type                       = "egress"
  from_port                  = 0
  to_port                    = 0
  protocol                   = "-1"
  security_group_id          = aws_security_group.bastion_sg.id
  cidr_blocks                = ["0.0.0.0/0"]
  ipv6_cidr_blocks           = ["::/0"]
}
```

- ALB Security Groups

```
## source = "../Aws-Modules/modules/vpc-security_groups/alb_sg.tf"
## ALB Security Group

#======================================================================

resource "aws_security_group" "main_alb_sg" {
  name        = "${var.env_name}-alb-sg"
  description = "Security group for the Application Load Balancer"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}-alb-sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}


#===========ingress rules==============================================

resource "aws_security_group_rule" "ingress_http" {
  type                = "ingress"
  security_group_id = aws_security_group.main_alb_sg.id

  from_port           = 80
  to_port             = 80
  protocol            = "tcp"
  cidr_blocks         = ["0.0.0.0/0"]  # Allow HTTP traffic from anywhere

  description         = "Allow HTTP traffic on port 80"
}

resource "aws_security_group_rule" "ingress_https" {
  type                = "ingress"
  security_group_id = aws_security_group.main_alb_sg.id

  from_port           = 443
  to_port             = 443
  protocol            = "tcp"
  cidr_blocks         = ["0.0.0.0/0"]  # Allow HTTPS traffic from anywhere

  description         = "Allow HTTPS traffic on port 443"
}
```

```
#===============Egress rules===========================================

resource "aws_security_group_rule" "egress_allow_all" {
  type              = "egress"
  security_group_id = aws_security_group.main_alb_sg.id

  from_port         = 0
  to_port           = 0
  protocol          = "-1"  # Allow all outbound traffic
  cidr_blocks       = ["0.0.0.0/0"]  # Modify this to restrict egress as needed

  description       = "Allow all outbound traffic"
}


#=====================================================================
```

- Frontend Node group Security Groups

```
## source = "../Aws-Modules/modules/vpc-security_groups/frontkend_nodegroup_sg.tf"
## Frontend nodes Security Group

#=====================================================================
resource "aws_security_group" "eks_frontend_node_group_sg" {
  name        = "${var.env_name}-eks-frontend-nodegroup-sg"
  description = "Security group for the Frontend EKS nodes"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}_eks_frontend_node-sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}


#==========ingress rules====================================

# SSH access from the bastion security group
resource "aws_security_group_rule" "frontend_nodegroup_ssh" {
  type              = "ingress"
  from_port         = 22
  to_port           = 22
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = aws_security_group.bastion_sg.id
  description       = "Allow SSH communication from Bastion SG"
}

# EKS API server to communicate with nodes on port 443
resource "aws_security_group_rule" "frontend_nodegroup_https" {
  type              = "ingress"
  from_port         = 443
  to_port           = 443
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = aws_security_group.main_alb_sg.id
#  cidr_blocks       = ["0.0.0.0/0"]  # Replace with specific CIDR if required
```

```
  description        = "Allow EKS ALB to communicate with frontend nodes on HTTPS"
}

# EKS metrics server to communicate with nodes on port 443
resource "aws_security_group_rule" "frontend_nodegroup_metrics_server_from_front"
{
  type               = "ingress"
  from_port          = 10250
  to_port            = 10250
  protocol           = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  #  cidr_blocks        = ["0.0.0.0/0"]  # Replace with specific CIDR if required
  description        = "Allow EKS metrics server to communicate with frontend nodes
on HTTPS"
}

# EKS metrics server to communicate with nodes on port 443
resource "aws_security_group_rule" "frontend_nodegroup_metrics_server_from_back" {
  type               = "ingress"
  from_port          = 10250
  to_port            = 10250
  protocol           = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = aws_security_group.eks_backend_node_group_sg.id
  #  cidr_blocks        = ["0.0.0.0/0"]  # Replace with specific CIDR required
  description        = "Allow EKS metrics server to communicate with backend nodes
on HTTPS"
}

# EKS API server to communicate with nodes on port 80
resource "aws_security_group_rule" "frontend_nodegroup_http" {
  type               = "ingress"
  from_port          = 80
  to_port            = 80
  protocol           = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = aws_security_group.main_alb_sg.id
#  cidr_blocks        = ["0.0.0.0/0"]  # Replace with specific CIDR if required
  description        = "Allow EKS ALB to communicate with frontend nodes on HTTP"
}

# CoreDNS communication within the cluster (UDP port 53)
resource "aws_security_group_rule" "frontend_nodegroup_dns_udp" {
  type               = "ingress"
  from_port          = 53
  to_port            = 53
  protocol           = "udp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  cidr_blocks        = [var.vpc_cidr_block]
  description        = "Allow CoreDNS communication within the cluster (UDP)"
}

# CoreDNS communication within the cluster (TCP port 53)
resource "aws_security_group_rule" "frontend_nodegroup_dns_tcp" {
  type               = "ingress"
  from_port          = 53
  to_port            = 53
  protocol           = "tcp"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  cidr_blocks        = [var.vpc_cidr_block]
  description       = "Allow CoreDNS communication within the cluster (TCP)"
}

# # NodePort communication within cluster (TCP ports 30000-32762)
# resource "aws_security_group_rule" "frontend_nodegroup_nodeport" {
#   type              = "ingress"
#   from_port         = 30000
#   to_port           = 32762
#   protocol          = "tcp"
#   security_group_id = aws_security_group.eks_frontend_node_group_sg.id
#   cidr_blocks       = [var.vpc_cidr_block]
#   description       = "Allow NodePort communication within the cluster"
# }

resource "aws_security_group_rule" "frontend_allow_control_plane" {
  type                     = "ingress"
  from_port                = 10250
  to_port                  = 10250
  protocol                 = "tcp"
  security_group_id        = aws_security_group.eks_frontend_node_group_sg.id
  source_security_group_id = var.control_plane_security_group_id
#  cidr_blocks       = ["0.0.0.0/0"]
  description              = "Allow control plane to communicate with frontend
nodes"
}



#===============Egress rules==========================================

resource "aws_security_group_rule" "frontend_nodegroup_egress" {
  type              = "egress"
  from_port         = 0
  to_port           = 0
  protocol          = "-1"
  security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  cidr_blocks       = ["0.0.0.0/0"]
  description       = "Allow all outbound traffic"
}


#=========================================================================
```

- Backend Node group Security Group

```
## source = "../Aws-Modules/modules/vpc-security_groups/backend_nodegroup_sg.tf"
## Backend nodes Security Group

#========================================================================

resource "aws_security_group" "eks_backend_node_group_sg" {
  name        = "${var.env_name}_eks_backend_node-sg"
  description = "Security group for the EKS backend nodes"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}_eks_backend_node-sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}


#===========ingress rules==============================================

# SSH access from Bastion security group
resource "aws_security_group_rule" "ssh_from_bastion" {
  type                     = "ingress"
  from_port                = 22
  to_port                  = 22
  protocol                 = "tcp"
  security_group_id        = aws_security_group.eks_backend_node_group_sg.id
  source_security_group_id = aws_security_group.bastion_sg.id
  description              = "Allow SSH from Bastion security group"
}



# Allow control plane access from specific CIDR
resource "aws_security_group_rule" "eks_control_plane_access_for_lb_controller" {
  type              = "ingress"
  from_port         = 9443
  to_port           = 9443
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  source_security_group_id = var.control_plane_security_group_id
  #  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow EKS lb controller to communicate with the Kubernetes
API server."
}

# EKS metrics server to communicate with nodes on port 443
resource "aws_security_group_rule" "backend_nodegroup_metrics_server_from_front" {
  type              = "ingress"
  from_port         = 10250
  to_port           = 10250
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  source_security_group_id = aws_security_group.eks_frontend_node_group_sg.id
  #  cidr_blocks       = ["0.0.0.0/0"]  # Replace with specific CIDR if required
  description       = "Allow EKS metrics server to communicate with  front end
nodes on HTTPS"
}

# EKS metrics server to communicate with nodes on port 443
resource "aws_security_group_rule" "backend_nodegroup_metrics_server_from_back" {
  type              = "ingress"
  from_port         = 10250
  to_port           = 10250
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  source_security_group_id = aws_security_group.eks_backend_node_group_sg.id
  #  cidr_blocks       = ["0.0.0.0/0"]  # Replace with specific CIDR if required
  description       = "Allow EKS metrics server to communicate with backend nodes
on HTTPS"
}

# Allow kubelet API access from the control plane
resource "aws_security_group_rule" "kubelet_api_access" {
  type              = "ingress"
  from_port         = 10250
  to_port           = 10250
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  source_security_group_id = var.control_plane_security_group_id
#  cidr_blocks       = ["0.0.0.0/0"]
  description       = "Allow kubelet API access from control plane"
}


# Allow core dns access from the vpc
resource "aws_security_group_rule" "core_dns_access_udp" {
  type              = "ingress"
  from_port         = 53
  to_port           = 53
  protocol          = "udp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
#  source_security_group_id = var.control_plane_security_group_id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow coredns access from vpc"
}

# Allow core dns access from the vpc
resource "aws_security_group_rule" "core_dns_access_tcp" {
  type              = "ingress"
  from_port         = 53
  to_port           = 53
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  #  source_security_group_id = var.control_plane_security_group_id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow coredns access from vpc"
}

# Proxy service communication
resource "aws_security_group_rule" "proxy_service_ingress" {
  type              = "ingress"
  from_port         = var.proxy_service_port
  to_port           = var.proxy_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow proxy service communication"
}

# Repeat for each specific service
resource "aws_security_group_rule" "auth_service_ingress" {
  type              = "ingress"
  from_port         = var.auth_service_port
  to_port           = var.auth_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow auth service communication"
}

# VP service communication
resource "aws_security_group_rule" "vp_service_ingress" {
  type              = "ingress"
  from_port         = var.vp_service_port
  to_port           = var.vp_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow VP service communication"
}

# UMM service communication
resource "aws_security_group_rule" "umm_service_ingress" {
  type              = "ingress"
  from_port         = var.umm_service_port
  to_port           = var.umm_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow UMM service communication"
}

# Tenant service communication
resource "aws_security_group_rule" "tenant_service_ingress" {
  type              = "ingress"
  from_port         = var.tenant_service_port
  to_port           = var.tenant_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow Tenant service communication"
}

# Integration service communication
resource "aws_security_group_rule" "integration_service_ingress" {
  type              = "ingress"
  from_port         = var.integration_service_port
  to_port           = var.integration_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow Integration service communication"
}

# Common service communication
resource "aws_security_group_rule" "common_service_ingress" {
  type              = "ingress"
  from_port         = var.common_service_port
  to_port           = var.common_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow Common service communication"
}

# Message service communication
resource "aws_security_group_rule" "message_service_ingress" {
  type              = "ingress"
  from_port         = var.message_service_port
  to_port           = var.message_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow Message service communication"
}
```

```
# Payment service communication
resource "aws_security_group_rule" "payment_service_ingress" {
  type              = "ingress"
  from_port         = var.payment_service_port
  to_port           = var.payment_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow Payment service communication"
}

#-----------------erp-connectors-----------------------------------

# bcc service communication
resource "aws_security_group_rule" "bcc_service_ingress" {
  type              = "ingress"
  from_port         = var.bcc_service_port
  to_port           = var.bcc_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow bcc service communication"
}


#qbo service communication
resource "aws_security_group_rule" "qbo_service_ingress" {
  type              = "ingress"
  from_port         = var.qbo_service_port
  to_port           = var.qbo_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow qbo service communication"
}


#-----------------payment-connectors-----------------------------------


#usb service communication
resource "aws_security_group_rule" "usb_service_ingress" {
  type              = "ingress"
  from_port         = var.usb_service_port
  to_port           = var.usb_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow usb service communication"
}

#notifiction service communication
resource "aws_security_group_rule" "notification_service_ingress" {
  type              = "ingress"
  from_port         = var.notification_service_port
  to_port           = var.notification_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow notification service communication"
}
```

```
#checkeeper service communication
resource "aws_security_group_rule" "checkeeper_service_ingress" {
  type              = "ingress"
  from_port         = var.checkeeper_service_port
  to_port           = var.checkeeper_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow checkeeper service communication"
}

#papertrl-db-lb service communication
resource "aws_security_group_rule" "papertrl-db-lb_service_ingress" {
  type              = "ingress"
  from_port         = var.papertrl-db-lb_service_port
  to_port           = var.papertrl-db-lb_service_port
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow papertrl-db-lb service communication"
}


# Example for the EFS NFS communication rule
resource "aws_security_group_rule" "nfs_efs_ingress" {
  type              = "ingress"
  from_port         = 2049
  to_port           = 2049
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = [var.vpc_cidr_block]
  description       = "Allow EKS worker nodes to communicate with EFS via NFS"
}


#===============Egress rules==========================================

resource "aws_security_group_rule" "all_outbound_traffic" {
  type              = "egress"
  from_port         = 0
  to_port           = 0
  protocol          = "-1"
  security_group_id = aws_security_group.eks_backend_node_group_sg.id
  cidr_blocks       = ["0.0.0.0/0"]
  description       = "Allow all outbound traffic"
}


#=======================================================================
```

- RDS Security Group

```
## source = "../Aws-Modules/modules/vpc-security_groups/rds_sg.tf"
## RDS Security Group

#=======================================================================
resource "aws_security_group" "main_rds_sg" {
  name = "${var.env_name}_rds_sg"
  description = "Security group for the EKS nodes to connect to RDS"
  vpc_id      = var.vpc_id
  tags = {
    Name = "${var.env_name}_rds_sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}


resource "aws_security_group_rule" "main_rds_ingress" {
  type              = "ingress"
  from_port         = var.rds_db_port
  to_port           = var.rds_db_port
  protocol          = "tcp"
  security_group_id = aws_security_group.main_rds_sg.id
  cidr_blocks       = [var.app_subnet_1_cidr_block, var.app_subnet_2_cidr_block]
  description       = "Allow traffic to RDS from application subnets"
}
```

- Backend Node group Egress Security Group (RDS)

```
## source = "../Aws-Modules/modules/vpc-security_groups/backend_node_rds_sg.tf"
## Backend to RDS  Security Group

#=======================================================================

resource "aws_security_group" "main_backend_node_rds_sg" {
  name        = "${var.env_name}_backend_node_rds_sg"
  description = "Security group for the EKS nodes to connect to RDS"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}_backend_node_rds_sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"

  }
}


#===============Egress rules============================================

resource "aws_security_group_rule" "egress_rds_to_subnets" {
  type              = "egress"
  security_group_id = aws_security_group.main_backend_node_rds_sg.id

  from_port         = var.rds_db_port
  to_port           = var.rds_db_port
  protocol          = "tcp"
  cidr_blocks       = [var.rds_subnet_1_cidr_block, var.rds_subnet_2_cidr_block]

  description       = "Allow outbound RDS traffic to specific RDS subnets"
}

#=======================================================================
```

- EKS Cluster Security Group

```
## source = "../Aws-Modules/modules/vpc-security_groups/eks_sg.tf"
## EKS Cluster(control plane) Security Group

resource "aws_security_group" "eks_cluster_sg" {
  name        = "${var.env_name}-eks-cluster-sg"
  description = "Security group for the EKS cluster"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}-eks-cluster-sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}

#===========ingress rules======================================

# Allow worker nodes to communicate with EKS API server on port 443
resource "aws_security_group_rule" "eks_cluster_api_ingress" {
  type              = "ingress"
  from_port         = 443
  to_port           = 443
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_cluster_sg.id
  cidr_blocks       = [var.vpc_cidr_block]  # Update with the correct CIDR block
  description       = "Allow worker nodes to communicate with EKS API server"
}

# Allow worker nodes to communicate with the kubelet on port 10250
resource "aws_security_group_rule" "eks_cluster_kubelet_ingress" {
  type              = "ingress"
  from_port         = 10250
  to_port           = 10250
  protocol          = "tcp"
  security_group_id = aws_security_group.eks_cluster_sg.id
  cidr_blocks       = [var.vpc_cidr_block]  # Update with the correct CIDR block
  description       = "Allow worker nodes to communicate with the kubelet"
}

#===============Egress rules============================================

# Allow all egress traffic
resource "aws_security_group_rule" "eks_cluster_egress" {
  type              = "egress"
  from_port         = 0
  to_port           = 0
  protocol          = "-1"
  security_group_id = aws_security_group.eks_cluster_sg.id
  cidr_blocks       = ["0.0.0.0/0"]
  description       = "Allow all outbound traffic"
}


#========================================================================
```

- EFS Volume Security Group

```
## source = "../Aws-Modules/modules/vpc-security_groups/efs_sg.tf"
## EFS drive Security Group

#========================================================================

resource "aws_security_group" "main_efs_sg" {
  name = "${var.env_name}-efs-sg"
  description = "Security group for EFS mount targets"
  vpc_id      = var.vpc_id

  tags = {
    Name = "${var.env_name}-efs-sg"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }
}

#===========ingress rules==============================================

resource "aws_security_group_rule" "main_efs_nfs_ingress" {
  type                = "ingress"
  from_port           = 2049
  to_port             = 2049
  protocol            = "tcp"
  security_group_id = aws_security_group.main_efs_sg.id
  source_security_group_id = aws_security_group.eks_backend_node_group_sg.id
  description         = "Allow NFS traffic from backend node group"
}


#===============Egress rules===========================================

resource "aws_security_group_rule" "main_efs_egress" {
  type                = "egress"
  from_port           = 0
  to_port             = 0
  protocol            = "-1"
  security_group_id = aws_security_group.main_efs_sg.id
  cidr_blocks         = [var.vpc_cidr_block]   # Update with the correct CIDR block
if necessary
  description         = "Allow all outbound traffic"
}

#=======================================================================
```

2.4 Bastion Host

- **Bastion Subnet**: Deploy a Bastion host in a separate public subnet for secure SSH access to resources within private subnets.

```
resource "aws_instance" "bastion_instance" {

  ami                          = var.bastion_ami_id              # Replace with
the AMI ID you want to use
  instance_type                = var.bastion_instance_type       # Replace with
your desired instance type (e.g., "t2.micro")
  subnet_id                    = var.bastion_subnet_id           # This refers to
the Bastion subnet you defined earlier
```

```
  key_name                    = var.Bastion_key_name             # The name of the
key pair to use for SSH access
  associate_public_ip_address = true                             # Ensure the
instance has a public IP
  vpc_security_group_ids      = [var.bastion_sg_id]                # Reference the
security group here

  tags = {
    Name                      = "${var.env_name}-SSH-Instance"  # Replace with the
desired instance name
    Env                       = var.env_name
#    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }
  user_data = base64encode(file("${path.module}/user.sh"))

  # Ensure the instance is created after the subnet and security groups
  #depends_on = [aws_subnet.main_bastion_subnet, aws_security_group.bastion_sg]
}
```

- GitHub Runner : Deploy a instance for GitHub runner services inside the VPC

```
resource "aws_instance" "gh_runner_instance" {

  ami = var.gh_runner_ami_id   # Replace with the AMI ID you want to use
  instance_type  = var.gh_runner_instance_type  # Replace with your desired
instance type (e.g., "t2.micro")
  subnet_id  = var.gh_runner_subnet_id  # This refers to the Bastion subnet you
defined earlier
  key_name  = var.gh_runner_key_name # The name of the key pair to use for SSH
access
  associate_public_ip_address = false # Ensure the instance has a public IP
  vpc_security_group_ids      =
[var.backend_node_sg_id,var.backend_node_rds_sg_id]                # Reference the
security group here

  tags = {
    Name                      = "${var.env_name}-gh_runner-Instance"  # Replace
with the desired instance name
    Env                       = var.env_name
    #    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
  }

  # Ensure the instance is created after the subnet and security groups
  #depends_on = [aws_subnet.main_bastion_subnet, aws_security_group.bastion_sg]
}
```

2.5 RDS : Create RDS in the RDS private subnet

```
resource "aws_db_subnet_group" "main_rds_dbsubnet_group" {
  name        = "${var.env_name}-db-subnet-group"
  subnet_ids = var.private_rds_subnet_ids

  tags = {
    Name         = "${var.env_name}-db-subnet-group"
    Env          = var.env_name
#    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }

  description = "Database subnet group for ${var.env_name} environment."
}

## source = "../Aws-Modules/modules/rds/rds.tf"
## RDS

#========================================================================

resource "aws_db_instance" "rds" {
  identifier                 = "${var.env_name}-rds-instance"
  allocated_storage          = var.allocated_storage
  engine                     = var.engine
  engine_version             = var.engine_version
  instance_class             = var.instance_class
  username                   = var.username
  password                   = var.password
  port                       = var.rds_db_port
  db_subnet_group_name       = aws_db_subnet_group.main_rds_dbsubnet_group.name
  vpc_security_group_ids     = [var.rds_sg_id]

  skip_final_snapshot        = var.skip_final_snapshot
  publicly_accessible        = false
  auto_minor_version_upgrade = false
  # Storage options
  storage_type               = var.storage_type
  multi_az                   = var.multi_az
  backup_retention_period    = var.backup_retention
  storage_encrypted          = true

  lifecycle {
    prevent_destroy = true
    ignore_changes  = [engine_version]
  }

  tags = {
    Name                     = "${var.env_name}-rds-instance"
    Env                      = var.env_name
  }

  depends_on = [aws_db_subnet_group.main_rds_dbsubnet_group]
}
```

2.6 EFS

```
## source = "../Aws-Modules/modules/efs/efs.tf"
## EFS File System

#========================================================================

# Create the EFS file system
resource "aws_efs_file_system" "main_efs" {

  creation_token = "${var.env_name}-efs"

  lifecycle_policy {
    transition_to_ia = "AFTER_30_DAYS"  # Moves data to Infrequent Access after 30
days
  }
  encrypted = true  # Encrypts the file system at rest
  # Backup policy
  # enable_backup_policy = true

  # Replication configuration
  #  #create_replication_configuration = true
  #  replication_configuration_destination = {
  #    region = "eu-west-2"
  #  }

  tags = {
    Name                = "${var.env_name}-efs"
    Env                 = var.env_name
  }
}

# Create Mount Targets in each AZ for the VPC
# Assume that subnet_ids corresponds to subnets in different availability zones

resource "aws_efs_mount_target" "main_mount_target" {

  count          = length(var.efs_subnet_ids)  # Create mount target in each subnet
  file_system_id = aws_efs_file_system.main_efs.id
  subnet_id      = var.efs_subnet_ids[count.index]  # Attach mount target to each
subnet
  security_groups = [var.efs_sg_id]  # Security group for EFS

}


# Step 5: Create EFS Access Point
resource "aws_efs_access_point" "efs_access_point" {
  file_system_id = aws_efs_file_system.main_efs.id

  # Define the root directory for this access point
  root_directory {
    path = "/"  # Root directory for this access point
    creation_info {
      owner_gid   = 1000  # Group ID of the owner
      owner_uid   = 1000  # User ID of the owner
      permissions = 755   # Permissions of the root directory
    }
  }

  tags = {
    Name = "${var.env_name}-efs-access-point"
    Env  = var.env_name
  }
}
```

2.7 ALB

```
## source = "../Aws-Modules/modules/alb/alb.tf"
## Application Load Balancer

#=========================================================================

resource "aws_lb" "application_lb" {
  name                     = "${var.env_name}-alb"
  internal                 = false
  load_balancer_type = "application"
  security_groups    = [var.alb_sg_id]
  subnets            = [var.public_subnet_az_01_id,var.public_subnet_az_02_id]

  enable_deletion_protection = false

#  access_logs {
#    bucket  = aws_s3_bucket.lb_logs.id
#    prefix  = "test-lb"
#    enabled = true
#  }

  tags = {
    name                 = "${var.env_name}-alb"
    Env                  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "ingress.k8s.aws/stack" = "alb"                              #Identifies the
ALB as managed by the AWS Load Balancer Controller.
    "ingress.k8s.aws/resource" = "load-balancer"
    "elbv2.k8s.aws/cluster"  = var.cluster_name
  }
}



resource "aws_lb_target_group" "app_target_group" {
  name     = "${var.env_name}-frontend-service-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = var.vpc_id

  health_check {
    enabled              = true
    path                 = "/" # Change to your service's health check endpoint
    matcher              = "200"
    interval             = 30
    timeout              = 5
    healthy_threshold    = 2
    unhealthy_threshold = 2
  }

  tags = {
    Name = "${var.env_name}-frontend-service-tg"
    Env                  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
    "ingress.k8s.aws/resource" = "targetgroup"
  }
}


resource "aws_lb_listener" "app_http_listener" {
  load_balancer_arn = aws_lb.application_lb.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.app_target_group.arn
  }
}
```

```
#resource "aws_lb_listener" "app_https_listener" {
#  load_balancer_arn = aws_lb.application_lb.arn
#  port              = 443
#  protocol          = "HTTPS"
#  ssl_policy        = "ELBSecurityPolicy-2016-08"
#  certificate_arn   = var.acm_certificate_arn
#
#  default_action {
#    type             = "forward"
#    target_group_arn = aws_lb_target_group.app_target_group.arn
#  }
#}
```

3. Deploying an EKS Cluster Using Terraform

3.1 EKS Cluster Configuration

Deploy an EKS cluster within the VPC created earlier.

- EKS Cluster: Use Terraform to create the EKS cluster, specifying the desired Kubernetes version.
- Define the resources in the eks.tf file
- Add the oidc resource creation for the cluster to assign for IRSA

```
## source = "../Aws-Modules/modules/eks/eks.tf"
## EKS Cluster

#=========================================================================

resource "aws_eks_cluster" "main_eks_cluster" {
  name     = var.cluster_name
  role_arn = var.eks_cluster_role_arn

  vpc_config {
    subnet_ids =
[var.public_subnet_az_01,var.public_subnet_az_02,var.app_subnet_az_01,var.app_subn
et_az_02]  # Public subnets for frontend
    #subnet_ids =
[module.vpc.main_public_subnet_1,module.vpc.main_public_subnet_2,module.vpc.main_p
rivate_app_subnet_1,module.vpc.main_private_app_subnet_2]

    endpoint_private_access = true
    endpoint_public_access  = false

    # Cluster security group
    security_group_ids = [var.eks_cluster_sg_id]


  }

  tags = {
    Name = "${var.env_name}-eks-cluster"
    Env  = var.env_name
  }
}


# Data source to find the control plane security group
data "aws_security_group" "eks_control_plane" {
  filter {
    name   = "tag:aws:eks:cluster-name"
    values = [aws_eks_cluster.main_eks_cluster.name]
  }
}


# resource "kubernetes_config_map" "aws_auth" {
#   metadata {
#     name      = "aws-auth"
#     namespace = "kube-system"
#   }
#
#   data = {
#     mapRoles = yamlencode([
#       {
#         rolearn = var.eks_node_role_arn
#         username = "system:node:{{EC2PrivateDNSName}}"
#         groups   = ["system:bootstrappers", "system:nodes"]
#       }
#     ])
#
#     mapUsers = yamlencode([
#       for user in var.additional_users : {
#         userarn  = "arn:aws:iam::${var.account_id}:user/${user}"
#         username = user
#         groups   = ["system:masters"]
#       }
#     ])
#   }
# }

data "aws_eks_cluster" "my_eks_cluster" {
  name = aws_eks_cluster.main_eks_cluster.name
}

data "aws_eks_cluster_auth" "my_eks_cluster" {
  name = aws_eks_cluster.main_eks_cluster.name
}

# provider "kubernetes" {
#   host                   = data.aws_eks_cluster.my_eks_cluster.endpoint
#   token                  = data.aws_eks_cluster_auth.my_eks_cluster.token
#   cluster_ca_certificate =
base64decode(data.aws_eks_cluster.my_eks_cluster.certificate_authority[0].data)
#   config_path = "~/.kube/config"
# }

#resource "aws_eks_addon" "efs_csi_driver" {
#  cluster_name = aws_eks_cluster.main_eks_cluster.name
#  addon_name   = "efs_csi_driver"
#}


# data "aws_iam_openid_connect_provider" "existing" {
#   url = data.aws_eks_cluster.my_eks_cluster.identity[0].oidc[0].issuer
# }
#
data "tls_certificate" "oidc_thumbprint" {
  url = data.aws_eks_cluster.my_eks_cluster.identity[0].oidc[0].issuer
}

resource "aws_iam_openid_connect_provider" "eks_oidc" {
  url             = data.aws_eks_cluster.my_eks_cluster.identity[0].oidc[0].issuer
  client_id_list  = ["sts.amazonaws.com"]
  thumbprint_list =
[data.tls_certificate.oidc_thumbprint.certificates[0].sha1_fingerprint]
}
```

- IAM Roles and Policies:
    - IAM roles for EKS cluster.
    - IAM policies for node groups, allowing them to interact with the cluster.
- Cluster Role

```
## source = "../Aws-Modules/modules/iam/cluster_role.tf"
## Cluster_Role

#=======================================================================

# IAM Role for EKS Cluster
resource "aws_iam_role" "eks_cluster_role" {
  name = "eks-cluster-role"

  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" "eks_cluster_policy_attachment" {
  role       = aws_iam_role.eks_cluster_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}

#=======================================================================
```

- Node Role

```
## source = "../Aws-Modules/modules/iam/node_role.tf"
## Node_Role

#=======================================================================

# IAM Role for EKS Node Groups
resource "aws_iam_role" "eks_node_role" {
  name = "${var.env_name}-eks-node-role"

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

}
```

```
resource "aws_iam_role_policy_attachment" "worker_node_policy" {
  role       = aws_iam_role.eks_node_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
}

resource "aws_iam_role_policy_attachment" "container_registry_read_only" {
  role       = aws_iam_role.eks_node_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
}

resource "aws_iam_role_policy_attachment" "cni_policy" {
  role       = aws_iam_role.eks_node_role.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
}



# Additional Inline Policy for EFS Access
resource "aws_iam_policy" "efs_access_policy" {
  name        = "${var.env_name}-EFSAccessPolicy"
  description = "Policy to allow EKS nodes to access EFS"
  policy      = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Action = [
        "elasticfilesystem:DescribeAccessPoints",
        "elasticfilesystem:DescribeFileSystems",
        "elasticfilesystem:DescribeMountTargets",
        "elasticfilesystem:CreateAccessPoint",
        "elasticfilesystem:CreateFileSystem",
        "elasticfilesystem:DeleteAccessPoint",
        "elasticfilesystem:DeleteFileSystem",
        "elasticfilesystem:ListTagsForResource",
        "elasticfilesystem:TagResource",
        "elasticfilesystem:UntagResource",
        "elasticfilesystem:UpdateFileSystem",
      ]
      Resource = "*"
    }]
  })
}

# Attach the EFS Access Policy to the Node Role
resource "aws_iam_role_policy_attachment" "attach_efs_policy" {
  policy_arn = aws_iam_policy.efs_access_policy.arn
  role       = aws_iam_role.eks_node_role.name
}

#========================================================================
```

- IRSA (Roles for Service Accounts)

```
resource "aws_iam_role" "irsa_role_lb_controller" {
  name = "${var.env_name}-irsa-role-LB-controller"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = var.eks_oidc_arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "${replace(var.eks_oidc_url, "https://", "")}:sub" =
"system:serviceaccount:kube-system:aws-load-balancer-controller"
          }
        }
      }
    ]
  })

  tags = {
    Name = "${var.env_name}-irsa-role-LB-controller"
    Env  = var.env_name
  }
}

resource "aws_iam_role_policy_attachment" "aws_lb_controller_policy" {
  role       = aws_iam_role.irsa_role_lb_controller.name
  policy_arn =
"arn:aws:iam::194978441177:policy/Custom_AWS_LoadBalancer_Controller_IAM_Policy"
}

resource "aws_iam_role" "irsa_role_ebs_csi_controller" {
  name = "${var.env_name}-irsa-role-ebs-csi-controller"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = var.eks_oidc_arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "${replace(var.eks_oidc_url, "https://", "")}:sub" =
"system:serviceaccount:kube-system:ebs-csi-controller-sa"
          }
        }
      }
    ]
  })

  tags = {
    Name = "${var.env_name}-irsa-role-ebs-csi-controller"
    Env  = var.env_name
  }
}

resource "aws_iam_role_policy_attachment" "ebs_csi_policy" {
  role       = aws_iam_role.irsa_role_ebs_csi_controller.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
}

resource "aws_iam_role" "irsa_role_efs_csi_controller" {
  name = "${var.env_name}-irsa-role-efs-csi-controller"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Federated = var.eks_oidc_arn
        }
        Action = "sts:AssumeRoleWithWebIdentity"
        Condition = {
          StringEquals = {
            "${replace(var.eks_oidc_url, "https://", "")}:sub" =
"system:serviceaccount:kube-system:efs-csi-controller-sa"
          }
        }
      }
    ]
  })

  tags = {
    Name = "${var.env_name}-irsa-role-efs-csi-controller"
    Env  = var.env_name
  }
}

resource "aws_iam_role_policy_attachment" "efs_csi_policy" {
  role       = aws_iam_role.irsa_role_efs_csi_controller.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy"
}
```

- Security Groups:
    - Cluster Security Group: Control traffic to and from the EKS control plane.
    - Node Security Groups: Control traffic between nodes and other components within the cluster.

3.2 Node Groups & Launch templates

Create two managed node groups within the EKS cluster, each associated with specific subnets and node roles.

- We have to create launch templates without instance types .If so nodes will seamlessly join with the EKS cluster because EKS managed node groups will automatically launched with EKS optimized AMI's. IF not we have to manually configure nodes by installing Kubernetes agents in the nodes to add to the cluster
- ASG will automatically created by node groups when deployed

- **Frontend Launch Template**

```
## source = "../Aws-Modules/modules/launch_templates/frontend_launchtemplate.tf"
## Frontend Launch Template For Frontend Nodes

#==========================================================================

resource "aws_launch_template" "frontend_launch_template" {
  name      = "frontend-node-launch-template"
  # image_id = var.node_group_ami  # Use the appropriate AMI
  # instance_type = "t3.medium"
  # No image_id specified here, default optimized AMI will be used
  key_name = var.node_ssh_key_name

  network_interfaces {
    security_groups = [var.frontend_node_sg_id]
    # Attach security group here
  }

  # Propagate tags to EC2 instances
  tag_specifications {
    resource_type = "instance"

    tags = {
      Name = "${var.env_name}-frontend-node"
      Env  = var.env_name
      "kubernetes.io/cluster/${var.cluster_name}" = "owned"
      "k8s.io/cluster-autoscaler/enabled" = "true"
      "k8s.io/cluster-autoscaler/${var.cluster_name}" = "owned"
    }
  }

#  user_data = base64encode(<<-EOT
##!/bin/bash
## Create papertrl user
#useradd papertrl
#
## Add papertrl user to sudoers file with NOPASSWD for passwordless sudo
#echo 'papertrl ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
#EOT
#  )
#  user_data = base64encode(file("${path.module}/user.sh"))
}
```

- **Frontend Node Group**: Nodes that will host the frontend services.

```
## source = "../Aws-
Modules/modules/node_groups/frontend_node_groups/frontend_nodegroup.tf"
## Frontend Node Group

#==========================================================================

# Frontend Node Group - Managed Node Group
resource "aws_eks_node_group" "frontend_node_group" {
  cluster_name    = var.cluster_name
  node_group_name = "frontend_node_group"
  node_role_arn   = var.node_role_arn
  subnet_ids      = [var.public_subnet_az_01_id,var.public_subnet_az_02_id]

  launch_template {
    id      = var.frontend_launch_template_id
    version = "$Latest"

  }

  scaling_config {
    desired_size = 2
    min_size     = 2
    max_size     = 4
  }

  instance_types = ["t3.medium"]

  tags = {
    Name = "${var.env_name}-frontend_node_group"
    Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "owned"
    "k8s.io/cluster-autoscaler/enabled" = "true"
    "k8s.io/cluster-autoscaler/${var.cluster_name}" = "owned"
  }

  labels = {
    Name = "${var.env_name}_frontend_node"
  }

  capacity_type = "ON_DEMAND" # Can also be "SPOT" for spot instances

  # Optional: Disk size for nodes (in GiB)
  #disk_size = 20

  update_config {
    max_unavailable = 1
  }

  depends_on = [var.cluster_name]

}
```

I. Backend Launch Template

```
## source = "../Aws-Modules/modules/launch_templates/backend_launchtemplate.tf"
## Backend Launch Template For Backend Nodes

#==========================================================================

resource "aws_launch_template" "backend_launch_template" {
  name      = "backend-node-launch-template"
  # image_id = var.node_group_ami  # Use the appropriate AMI
  # instance_type = "t3.medium"
  # No image_id specified here, default optimized AMI will be used
  key_name = var.node_ssh_key_name

  network_interfaces {
    security_groups = [var.backend_node_sg_id, var.backend_node_rds_sg_id]
    # Attach security group here
  }

  # Propagate tags to EC2 instances
  tag_specifications {
    resource_type = "instance"
    tags = {
      Name                                              = "${var.env_name}-backend-
node"
      Env                                               = var.env_name
      "kubernetes.io/cluster/${var.cluster_name}"      = "owned"
      "k8s.io/cluster-autoscaler/enabled"              = "true"
      "k8s.io/cluster-autoscaler/${var.cluster_name}" = "owned"
    }
  }
  user_data = base64encode(file("${path.module}/user.sh"))
}


#  # Optional: Install Kubernetes tools
#  yum install -y kubelet containerd aws-cli
#
#  # Start kubelet service
#  systemctl enable kubelet
#  systemctl start kubelet
#
#  # Optional: Join the instance to the EKS cluster (make sure bootstrap script is
included)
#  /etc/eks/bootstrap.sh ${var.cluster_name} --apiserver-endpoint
${var.cluster_endpoint} --b64-cluster-ca ${var.cluster_ca}
```

II. **Backend Node Group**: Nodes that will host the backend services.

```
## source = "../Aws-
Modules/modules/node_groups/backend_node_groups/backend_nodegroup.tf"
## Backeend Node Group

#==========================================================================

# Backend Node Group - Managed Node Group
resource "aws_eks_node_group" "backend_node_group" {
  cluster_name    = var.cluster_name
  node_group_name = "backend-node-group"
  node_role_arn   = var.node_role_arn
  subnet_ids      =
[var.backend_private_subnet_az_01_id,var.backend_private_subnet_az_02_id]



  launch_template {
    id      = var.backend_launch_template_id
    version = "$Latest"

  }

  scaling_config {
    desired_size = 4
    min_size     = 2
    max_size     = 8
  }

  instance_types = ["t3.large"]

  tags = {
    Name = "${var.env_name}-backend-node-group"
    Env  = var.env_name
#     Env  = var.env_name
    "kubernetes.io/cluster/${var.cluster_name}" = "owned"
    "k8s.io/cluster-autoscaler/enabled" = "true"
    "k8s.io/cluster-autoscaler/${var.cluster_name}" = "owned"
  }

  capacity_type = "ON_DEMAND" # Can also be "SPOT" for spot instances

  # Optional: Disk size for nodes (in GiB)
  #disk_size = 20

  update_config {
    max_unavailable = 1
  }

  depends_on = [var.cluster_name]

}
```

4. Common Errors and Troubleshooting

Common Errors

I. **Invalid Security Group Rules:**
   - *Error*: "Security group rule mismatch"
   - *Solution*: Validate CIDR blocks and ensure no conflicts.

II. **Terraform State Lock:**
   - *Error*: "State locked by another process"
   - *Solution*: Use terraform force-unlock to resolve.

III. **IAM Role Issues:**
   - *Error*: "Access denied for EKS cluster creation"
   - *Solution*: Ensure the role has eks.amazonaws.com trust.

IV. **Circular Reference:**
   - *Error*: "Resource dependencies create a circular reference"
   - *Solution*: Review and remove any cyclic dependencies in module inputs and outputs. Use explicit depends_on only where required.

V. **Custom AMI Node Group Issues:**
   - *Error*: "Node groups created but do not join the cluster"
   - *Solution*: Ensure the AMI is preconfigured with the correct EKS worker node requirements, including:
     - Proper kubelet and AWS authenticator configurations.

        - Valid IAM role attached.

VI. **Manual Resource Additions Block Destroy:**

  - *Error*: "Terraform cannot destroy the project due to manually added resources."
  - *Solution*: Avoid manual modifications. Use Terraform to manage all resources. For existing manual changes:
    - Import the resources into Terraform using terraform import.
    - Ensure state is updated with the changes.

Debugging Steps

- Run terraform plan to identify potential issues.
- Check logs in /var/log/terraform.
- Use AWS CloudTrail for API error debugging.
- For circular references:
  - Analyze terraform graph output to understand dependencies.
- For node group AMI issues:
  - Verify logs in /var/log/messages or /var/log/cloud-init.log on the instance.
  - Ensure AMI user data is correctly initializing worker nodes.

Special Notes

- If we Use terraform in CICD pipeline, For create a EKS environment .we need to configure Kubernetes provider as well to keep the flow of pipeline or else we need to configure the kubectl client in the CICD server using a script to engage with the Kubernetes environment.
- Load Balancer is Commented out Because In a Kubernetes environment load balancers are created automatically using ingress resource
- We have created IRSA roles with respect to the default Service account names respectively for the helm installations of the AWS load balancer controller,EFS,EBS controllers. If we manually create we need to create SA in Kubernetes cluster with respect to the IRSA roles defined SA names.
- To successfully integrate VPC resources with the EKS cluster, ensure the following tags are applied:

  - \`kubernetes.io/cluster/<cluster-name>\` = \`shared\` for better tracking and cluster association.

  These tags are critical for EKS-managed services like Load Balancers and Auto Scaling Groups to function seamlessly with VPC resources.

5. Improvements

- Set Kubernetes clusters public endpoint to false to improve the cluster security so it can only be accessed within VPC for admin purposes
- We can create front end node groups also in the private subnets for security improvements and can be load balanced using ALB, In current architecture front end node group is installed in public subnets.

END OF DOCUMENT