Terraform Configuration Documentation

File Breakdown

1. **cluster-addon.tf**
   This file defines the configuration for enabling various Kubernetes add-ons on the EKS cluster. It uses the aws-ia/eks-blueprints-addons/aws module.

Key Configurations:

AWS Load Balancer Controller: Enabled for managing AWS resources like Classic Load Balancers and Network Load Balancers.

Cluster Autoscaler: Enabled to allow automatic scaling of the Kubernetes cluster based on resource usage.

Metrics Server: Enabled to gather metrics for Kubernetes nodes and pods.

IRSA (IAM Roles for Service Accounts): Uses OIDC provider for IAM integration.

Tags: Tags the resources with an environment identifier (var.env).

```
module "eks_blueprints_addons" {
source = "aws-ia/eks-blueprints-addons/aws"
version = "~> 1.0"
cluster_name = module.eks.cluster_name
cluster_endpoint = module.eks.cluster_endpoint
cluster_version = module.eks.cluster_version
oidc_provider_arn = module.eks.oidc_provider_arn
enable_aws_load_balancer_controller = true
enable_cluster_autoscaler = true
enable_metrics_server = true
tags = {
Environment = "var.env"
    }
}
```

2. **eks.tf**

   This file sets up the main EKS cluster with the required IAM roles and managed node groups.

Key Configurations:

EKS Cluster: Defines the EKS cluster name, version, and VPC settings.

Managed Node Groups: Defines the configuration for EKS-managed EC2 node groups.

Cluster Addons: Specifies which Kubernetes addons (e.g., **CoreDNS, VPC CNI, EBS CSI driver**) to use in the cluster.

IAM Roles for Service Accounts (IRSA): Enabled to manage IAM permissions for services running inside the cluster.

Subnets and VPC Configuration: The cluster is deployed across public and private subnets in the specified VPC.

```
module "eks" {
source = "terraform-aws-modules/eks/aws"
version = "~> 20.0"
cluster_name = "${var.env}-${var.project-name}-cluster"
cluster_version = "1.30"
cluster_endpoint_public_access = true
cluster_endpoint_private_access = true
enable_irsa = true
cluster_addons = {
coredns = { most_recent = true }
eks-pod-identity-agent = { most_recent = true }
kube-proxy = { most_recent = true }
vpc-cni = { most_recent = true }
aws-ebs-csi-driver = { most_recent = true }
}
vpc_id = module.vpc.vpc_id
subnet_ids = module.vpc.private_subnets
control_plane_subnet_ids = module.vpc.public_subnets
eks_managed_node_group_defaults = {
instance_types = ["t3.medium", "t3a.medium"]
}
eks_managed_node_groups = {
"${var.env}-${var.project-name}-node" = {
ami_type = "AL2023_x86_64_STANDARD"
instance_types = ["t3a.medium"]
```

```
min_size = 1
max_size = 8
desired_size = 1
maxPodsPerNode = 110
  }
}
enable_cluster_creator_admin_permissions = true
bootstrap_self_managed_addons = true
tags = {
Environment = "dev"
Terraform = "true"
   }
}
```

3. **provider.tf**
   This file contains the AWS and Helm provider configuration to manage resources and Kubernetes deployments.

Key Configurations:
AWS Provider: Specifies the region to deploy resources in.
Helm Provider: Configures the Kubernetes cluster connection for Helm, using the aws eks CLI to authenticate.

```
provider "aws" {
region = var.region
}
provider "helm" {
kubernetes {
host = module.eks.cluster_endpoint
cluster_ca_certificate =
base64decode(module.eks.cluster_certificate_authority_data)
exec {
api_version = "client.authentication.k8s.io/v1beta1"
args = ["eks", "get-token", "-cluster-name", module.eks.cluster_name]
command = "aws"
    }
  }
}
```

4. **eks.tf (Main EKS Cluster Configuration)**
   This file consolidates the EKS cluster setup, including networking (subnets, VPC) and IAM integration.

5. **variables.tf**
   This file defines input variables for the Terraform configuration, including the environment name, project name, AWS region, and cluster version.

```
variable "project-name" {
type = string
}
variable "env" {
description = "For eg: dev,stage,prod"
type = string
}
variable "cluster_version" {
type = number
default = 1.30
}
variable "region" {
type = string
default = "us-west-2"
}
```

6. **vpc.tf**
   This file configures the Virtual Private Cloud (VPC) that the EKS cluster will reside in. It sets up the public and private subnets and enables necessary DNS and NAT Gateway functionality.

```
data "aws_region" "current" {}
data "aws_availability_zones" "available" {}
module "vpc" {
source = "terraform-aws-modules/vpc/aws"
version = "5.13.0"
name = "${var.project-name}-${var.env}-vpc"
cidr = "10.0.0.0/16"
azs = data.aws_availability_zones.available.names
private_subnets = ["10.0.0.0/19", "10.0.32.0/19"]
public_subnets = ["10.0.64.0/19", "10.0.96.0/19"]
public_subnet_tags = {
"kubernetes.io/role/elb" = "1"
}
private_subnet_tags = {
"kubernetes.io/role/internal-elb" = "1"
```

```
}
enable_nat_gateway = true
single_nat_gateway = true
one_nat_gateway_per_az = false
enable_dns_hostnames = true
enable_dns_support = true
tags = {
Environment = "var.env"
   }
}
```

To run the provided Terraform configuration you will need to follow these steps to **initialize, validate, plan,** and **apply** the configuration. Below is a step-by-step guide with the necessary commands:

1. **Navigate to the Terraform Directory**
   Open a terminal and navigate to the directory where your Terraform configuration files are located.

   cd /path/to/your/terraform/config

2. **Initialize the Terraform Working Directory**
   Initialize Terraform. This command installs the necessary provider plugins and prepares the working directory.

   terraform init

3. **Validate the Configuration**
   Ensure that your Terraform configuration files are syntactically correct and complete.

   terraform validate

4. **Create a Terraform Plan**
   Generate an execution plan. This shows what changes Terraform will make to your infrastructure, based on your configuration.

   terraform plan -out=tfplan

   If you need to specify variables such as environment or project name, you can use -var or provide a .tfvars file. For example:

terraform plan -var="env=dev" -var="project-name=my-project" -out=tfplan

5. **Apply the Terraform Plan**
   Apply the plan to create or update the infrastructure. You will be prompted to confirm the action before Terraform makes any changes.
   terraform apply tfplan
   Alternatively, you can run the following command if you want to apply the changes without saving the plan explicitly:

   terraform apply

   This command will prompt you to confirm the execution. Type **yes** to proceed.

6. Check the Status of the Infrastructure
   After applying, you can check the current state of your infrastructure:

   terraform show

7. Destroy Infrastructure (Optional)
   If you want to tear down the infrastructure created by Terraform, use the terraform destroy command. This will prompt you to confirm the destruction.
   terraform destroy

   Example of Full Command Flow

   cd /path/to/your/terraform/config

   terraform init
   terraform validate
   terraform plan -var="env=dev" -var="project-name=my-project" -out=tfplan
   terraform apply tfplan

   Additional Tips:

   Ensure your AWS credentials are properly configured for Terraform to access your AWS account. You can configure them via aws configure or by setting environment variables **AWS_ACCESS_KEY_ID** and **AWS_SECRET_ACCESS_KEY.**
   If you are using aws-profile, you can specify it using the AWS_PROFILE environment variable:

   export AWS_PROFILE=your-profile-name
   This will tell Terraform to use that specific AWS profile for your operations.