Analysis Report

1. Why do we need provider?

Answer: The provider block in Terraform is used to define and configure the provider that Terraform will use to interact with a specific cloud platform or service. Providers are responsible for communicating with the API of the respective cloud platform to create, update, and delete resources.

The purpose of the provider block is to establish the connection and provide necessary authentication details to interact with the cloud provider's API. It includes information such as the provider type (e.g., aws, azure, google), the region to operate in, and the authentication credentials.

2. Types of writing provider code.

There are multiple ways to provide the access key and secret key for the AWS provider in Terraform.

- Directly Inserting the Access Key and Secret Key: In this method, you directly insert the access key and secret key within the provider block in your Terraform configuration file. While this approach is straightforward, it is generally not recommended for security reasons, as it can expose sensitive credentials in your codebase.

  Here's an example of directly inserting the access key and secret key:

```
provider "aws" {
  region     = "us-east-1"
  access_key = "YOUR_ACCESS_KEY"
  secret_key = "YOUR_SECRET_KEY"
}
```

- Using the AWS CLI: Terraform also allows you to use the AWS Command Line Interface (CLI) to authenticate. By configuring the AWS CLI, Terraform can automatically retrieve the access key and secret key from the underlying AWS CLI configuration.

  Here's an example using the AWS CLI:

```
provider "aws" {
  region = "us-east-1"
  }
```
For this we need to install aws cli in the local machine.

- Using Environment Variables: Another common approach is to use environment variables to provide the access key and secret key. This method allows for flexibility and keeps sensitive information separate from the Terraform configuration file.

  Here's an example using environment variables:

```
provider "aws" {
  region     = "us-east-1"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}

variable "aws_access_key" {
  description = "AWS access key"
  default     = ""
}

variable "aws_secret_key" {
  description = "AWS secret key"
  default     = ""
}
```
In this example, the access key and secret key are referenced from environment variables AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY, respectively. We can set these environment variables in our system.

To set environment variables, we can use commands such as export (in Unix-based systems) or setx (in Windows). Example how to use export:

```
export TF_VAR_aws_access_key="your_access_key"
```

```
export TF_VAR_aws_secret_key="your_secret_key"
```
Replace `your_access_key` and `your_secret_key` with the actual values of your AWS access key and secret key.

3. Analyzing ec2 resource

```
resource "aws_instance" "web" {
  ami           = "ami-08a52ddb321b32a8c"
  instance_type = "t3.micro"
  subnet_id = aws_subnet.Public_Sub.id
  count = 2
}
```

- AMI (Amazon Machine Image):
  - An Amazon Machine Image (AMI) is a pre-configured virtual machine image used to create EC2 instances. It includes the operating system, application software, and configuration.
  - In our configuration, ami is set to "ami-08a52ddb321b32a8c". This is a unique identifier for an AMI.
- Instance Type:
  - The code defines the instance type as "t3.micro".
  - An instance type determines the hardware capabilities of the EC2 instance, such as CPU, memory, and networking.
  - The "t3.micro" instance type is part of the T3 family, suitable for lightweight workloads.
  - AWS provides various instance types, optimized for different use cases, such as general-purpose, memory-optimized, and compute-optimized instances.

We can create an ec2 with ami id and instance type only. But in this case, default vpc didn't have subnet associated with it so subnet id falls into the minimum requirement also.

- Subnet_id: `subnet_id` refers to the ID of a specific subnet within a Virtual Private Cloud (VPC) in AWS.

A subnet is a segmented portion of a VPC's IP address range. It allows you to divide your VPC into smaller networks, each with its own IP address range. Subnets are used to control network traffic and provide isolation between resources.

The `aws_subnet.Public_Sub.id` expression is used to reference the ID of a specific subnet resource in your Terraform configuration. This ID is typically generated by AWS when you create a subnet.

By specifying the subnet_id in your EC2 instance resource, you are indicating that the EC2 instance should be launched within the specified subnet. This ensures that the instance is associated with the correct network and can communicate with other resources within that subnet.

When creating EC2 instances, it's important to select the appropriate subnet based on your networking requirements. Subnets can be public or private, and they can be associated with different routing configurations and access control settings.

- Creating a VPC:

```
resource "aws_vpc" "tf_vpc" {
  cidr_block = "192.168.0.0/24"
  tags = {
    Name = "tf_vpc"
  }
}
```

This code creates an AWS VPC (Virtual Private Cloud) resource named "tf_vpc". A VPC is a logically isolated section of the AWS cloud where you can launch AWS resources. Here's what each attribute does:

  - cidr_block: Specifies the IP address range for the VPC. In this case, the VPC will have the IP address range of "192.168.0.0/24", which allows for up to 256 IP addresses.
  - tags: Allows you to assign metadata to the VPC resource. In this case, a tag with the key "Name" and value "tf_vpc" is added to the VPC.

- Creating a Public Subnet:

```
resource "aws_subnet" "Public_Sub" {
  vpc_id     = aws_vpc.tf_vpc.id
  cidr_block = "192.168.0.0/28"
}
```

This code creates an AWS subnet resource named "Public_Sub" within the VPC created in the previous step. A subnet is a segmented portion of a VPC's IP address range. Here's what each attribute does:

- vpc_id: Specifies the ID of the VPC to which the subnet belongs. In this case, it references the ID of the VPC created earlier using aws_vpc.tf_vpc.id.
- cidr_block: Specifies the IP address range for the subnet. In this case, the subnet will have the IP address range of "192.168.0.0/28", which allows for up to 16 IP addresses.

By creating a VPC and a subnet within that VPC, we are setting up the basic networking infrastructure in AWS. The VPC provides the overall network environment, and the subnet defines a smaller network segment within the VPC.

- o Different ways to create multiple ec2:
  - Count: We can create multiple EC2 instances by using the count parameter in your aws_instance resource block. The count parameter tells Terraform how many instances of a resource to create.
  - Using For_each:
    The for_each construct in Terraform allows you to create multiple instances based on a set or map of values. Unlike count, for_each allows for creating resources with more meaningful identifiers that are tied to your data.
    If you have a set of values, for_each will create an instance for each item in the set. For a map, it will create an instance for each key-value pair.
    Here's an example of how you can create multiple EC2 instances using for_each:

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

// VPC and subnet omitted for brevity...

locals {
  instances = {
    instance1 = "ami-08a52ddb321b32a8c"
    instance2 = "ami-08a52ddb321b32a8c"
  }
}

resource "aws_instance" "web" {
  for_each      = local.instances
  ami           = each.value
  instance_type = "t3.micro"
  subnet_id     = aws_subnet.Public_Sub.id
  key_name      = "mahmuda-key-pair"

  tags = {
    Name = each.key
  }
}
```

`for_each` is better suited when there's a more complex relationship between your instances.
In this case, the instances created would be named `instance1` and `instance2`.

4. **Why do we need region in aws provider block?**

   The `region` attribute in the `provider` block in AWS is important because it specifies the AWS region where the resources created by Terraform will be deployed. AWS has a number of regions around the world, and each region has its own unique set of Availability Zones. By specifying the region in the `provider` block, you can ensure that your resources are created in the correct region and Availability Zone.

5. **Resource block:** *Resources* are the most important element in the Terraform language. Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records[1].
6. **AMI**: An Amazon Machine Image, or AMI, is a template for Amazon EC2 instances, created from an Amazon EC2 instance.[2]
7. **Instance Type:** Amazon EC2 provides a variety of instance types optimized to fit different use cases. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications. [3]
8. **Subnet ID:** A subnet, or subnetwork, is a logical subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting.[4]

- Difference between public subnet and private subnet:

A subnet is a range of IP addresses in your VPC (Virtual Private Cloud) where you can place resources such as EC2 instances, RDS databases, and more. Subnets allow you to logically segment your VPC into smaller networks, which can aid in organizing and securing your infrastructure.

Here's the difference between a public subnet and a private subnet:

Public Subnet:

A public subnet is a subnet that has a route to an Internet Gateway (IGW). This enables resources within the public subnet to communicate directly with the internet.

Typically, resources placed in a public subnet can have public IP addresses assigned to them, allowing them to be directly accessible from the internet.

Public subnets are often used for resources that need direct internet connectivity, such as web servers, load balancers, and instances that need to download software updates.

Private Subnet:

A private subnet, on the other hand, does not have a direct route to the internet via an IGW. Instead, it routes traffic through a Network Address Translation (NAT) gateway or NAT instance to access resources on the internet.

Resources in a private subnet do not have public IP addresses by default and cannot be accessed directly from the internet.

Private subnets are commonly used for databases, application servers, and other resources that need to communicate with each other but don't require direct internet access.

- Why we use VPC and subnet

- VPC (Virtual Private Cloud):
  - VPC provides a logically isolated section of the AWS cloud where you can launch AWS resources.
  - It allows you to define your own virtual network environment, including IP address ranges, subnets, route tables, and network gateways.

  - With VPC, you have full control over your network configuration, including the ability to create and manage subnets, define network access control policies, and establish connectivity options.
  - VPC enables you to create a secure and private network environment for your AWS resources, isolating them from other networks and providing control over inbound and outbound traffic.

- Subnet:
  - A subnet is a segmented portion of a VPC's IP address range.
  - It allows you to divide your VPC into smaller networks, each with its own IP address range.
  - Subnets provide network isolation and help in organizing resources within a VPC.
  - By dividing a VPC into subnets, you can control network traffic, apply different security policies, and deploy resources in a more granular manner.
  - Subnets are associated with availability zones (AZs), which are physically separate data centers within an AWS region. This allows you to distribute your resources across multiple AZs for high availability and fault tolerance.

- **CIDR block:**

Based on the CIDR block "192.168.0.0/28" that you provided, the available IP addresses within the subnet are as follows:

- Network address: 192.168.0.0
- Usable IP addresses: 192.168.0.1 to 192.168.0.14
- Broadcast address: 192.168.0.15

Here's a breakdown of the IP addresses:

- Network address: 192.168.0.0 is the first IP address in the range and represents the network itself.
- Usable IP addresses: From 192.168.0.1 to 192.168.0.14, there are 14 usable IP addresses that can be assigned to resources within the subnet.

- Broadcast address: 192.168.0.15 is the last IP address in the range and is used for broadcasting messages to all devices within the subnet.

It's important to note that the network address and broadcast address are reserved and cannot be assigned to individual resources. The usable IP addresses (192.168.0.1 to 192.168.0.14) are available for assigning to your resources within the subnet

- **Why we need an Internet Gateway (IGW) and a route table when setting up your VPC:**

Internet Gateway: An Internet Gateway is a scalable, redundant, and highly available VPC component that allows communication between instances in your VPC and the internet. It performs two main functions:

1. It provides a target in your VPC's route tables for internet-routable traffic.
2. It does the necessary network address translation for instances that have been assigned public IPv4 addresses.

In a flowchart-like manner, you can consider the Internet Gateway as an "exit door" for your VPC leading to the internet.

Here's how it's used when your EC2 instance wants to communicate with the internet:

EC2 Instance --> Subnet --> Internet Gateway --> Internet

Route Table: A route table contains a set of rules, called routes, that determines how packets are routed between subnets within your VPC, to the internet or to other AWS services. Each subnet in your VPC must be associated with a route table; the table controls the traffic out of the subnet.

In a flowchart-like manner, you can consider the Route Table as a "directions map" that guides traffic inside your VPC and to outside networks, including the internet.

When traffic is meant to go to the internet, it looks like:

EC2 instance --> Subnet --(guided by route table)--> Internet Gateway --> Internet

For an instance in your VPC to communicate with the internet:

1. The subnet it resides in should have a route table that has a rule (route) allowing it to reach the internet. This route directs the traffic to the Internet Gateway.
2. You need an Internet Gateway attached to your VPC to provide a way for the traffic to leave AWS and reach the internet network.

Thus, both the Internet Gateway and the Route Table play crucial roles in defining and enabling internet access for your instances in the VPC.

Code:

```
//Create my VPC

resource "aws_vpc" "tf_vpc" {

  cidr_block = "192.168.0.0/28"

  tags = {

    Name = "tf_vpc"

  }

}

//Create a Public Subnet

resource "aws_subnet" "Public_Sub" {
  vpc_id     = aws_vpc.tf_vpc.id

  cidr_block = "192.168.0.0/28"

}

# resource "aws_security_group" "tf_sg" {

#   name   = "terraform_SG_EC2"

#   vpc_id = aws_vpc.tf_vpc.id



#   ingress {

#     description      = "SSH"

#     from_port        = 22

#     to_port          = 22

#     protocol         = "tcp"

#     cidr_blocks      = ["0.0.0.0/0"]

#     ipv6_cidr_blocks = ["::/0"]

#   }

# }

resource "aws_instance" "web" {

  ami           = "ami-08a52ddb321b32a8c"

  instance_type = "t3.micro"

  # vpc_id        = aws_vpc.tf_vpc.id

  subnet_id = aws_subnet.Public_Sub.id

  count = 2

  # security_groups             = [aws_security_group.tf_sg.id]

  # associate_public_ip_address = true
  # key_name = "mahmuda-key-pair"
}
# provider "aws" {
#   region     = "us-east-1"
#   # access_key = "AKIAZITYUD2S35DXGENA"
#   # secret_key = "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"
# }


provider "aws" {
  region     = "us-east-1"
  access_key = var.aws_access_key
  secret_key = var.aws_secret_key
}

variable "aws_access_key" {
  description = "AWS access key"
  default     = ""
}

variable "aws_secret_key" {
  description = "AWS secret key"
  default     = ""
}
```

[redacted]