15 Days of Terraform – Curated by Shaik Hari Sadia Anjum

Providers in Terraform:

In Terraform, a "provider block" is a configuration block used to define the specific provider and its settings that Terraform will use to manage and interact with infrastructure resources. Providers are responsible for understanding API interactions and exposing resources. For example, AWS, Azure, Google Cloud, and many other services have their own providers.

Terraform Provider Workflow:

When you use Terraform, it handles the installation and management of providers automatically. Providers are the components that allow Terraform to interact with various services, like **AWS** or **Azure**. Here's how it works:

- **Initialize the Terraform Configuration**: First, you run the *terraform init* command in your project directory. This initializes your configuration and prepares Terraform to start working.
- **Provider Search**: Terraform will check your configuration files to see which providers are needed. If the required providers aren't already installed, Terraform will download them.
- **Downloading Provider Plugin:** If the provider isn't available locally, **Terraform** will fetch it from the Terraform registry or a local mirror, depending on your setup.
- **Storing Providers**: Once downloaded, Terraform stores the provider plugins in a .terraform folder within your working directory. This helps keep everything organized.
- **Checking Provider Versions**: Terraform checks the provider version specified in your configuration and ensures it's the correct one before proceeding.

This workflow ensures Terraform has all the right tools to interact with the services you want to manage, and it does so automatically, so you don't have to worry about manually managing providers.

```
provider <name of the provider> {

 # Configuration options

}
```

Define the provider in your .tf file using a provider block. For example, to configure AWS:

```
provider "aws" {

 region = "us-east-1"

}
```

You can replace "aws" with other providers like "azure" or "google" depending on your needs.

Authenticate with the Provider: Provide authentication details like API keys or secrets in the configuration or through environment variables. For instance:

```
provider "aws" {
 region     = "us-east-1"
 access_key = "your-access-key"
 secret_key = "your-secret-key"
}
```

Different ways to configure providers in terraform

There are three main ways to configure providers in Terraform:

In the root module

This is the most common way to configure providers. The provider configuration block is placed in the root module of the Terraform configuration. This makes the provider configuration available to all the resources in the configuration.

```
provider "aws" {

 region = "us-east-1"

}



resource "aws_instance" "example" {

 ami = "ami-0123456789abcdef0"

 instance_type = "t2.micro"

}
```

In a child module

You can also configure providers in a child module. This is useful if you want to reuse the same provider configuration in multiple resources.

```
module "aws_vpc" {
  source = "./aws_vpc"
  providers = {
    aws = aws.us-west-2
  }
}


resource "aws_instance" "example" {
  ami = "ami-0123456789abcdef0"
  instance_type = "t2.micro"
  depends_on = [module.aws_vpc]
}
```

In the required_providers block

You can also configure providers in the required_providers block. This is useful if you want to make sure that a specific provider version is used.

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.79"
    }
  }
}


resource "aws_instance" "example" {
```

```
 ami = "ami-0123456789abcdef0"

 instance_type = "t2.micro"

}
```

The best way to configure providers depends on your specific needs. If you are only using a single provider, then configuring it in the root module is the simplest option. If you are using multiple providers, or if you want to reuse the same provider configuration in multiple resources, then configuring it in a child module is a good option. And if you want to make sure that a specific provider version is used, then configuring it in the required_providers block is the best option.

Multiple Providers

You may need to use **multiple instances of the same provider** (e.g., multiple AWS accounts or regions).

✅ Use Case:

- Deploying in **multiple AWS accounts** or **regions**.
- Example: Production in us-east-1, Disaster Recovery in us-west-2.

✅ Syntax (Aliased Providers):

```
provider "aws" {

 alias  = "us_east"

 region = "us-east-1"

}


provider "aws" {

 alias  = "us_west"

 region = "us-west-2"

}
```

You can now associate a resource with a specific provider using provider = aws.alias_name:

```
resource "aws_instance" "east_instance" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
  provider      = aws.us_east
}

resource "aws_instance" "west_instance" {
  ami           = "ami-654321"
  instance_type = "t2.micro"
  provider      = aws.us_west
}
```

Multiple Region Implementation in Terraform

You can make use of `alias` keyword to implement multi region infrastructure setup in terraform.

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

provider "aws" {
  alias = "us-west-2"
  region = "us-west-2"
}

resource "aws_instance" "example" {
  ami = "ami-0123456789abcdef0"
  instance_type = "t2.micro"
  provider = "aws.us-east-1"
}

resource "aws_instance" "example2" {
  ami = "ami-0123456789abcdef0"
  instance_type = "t2.micro"
  provider = "aws.us-west-2"
}
```

Terraform Resources:

A **Terraform resource** is like a building block in a blueprint for your infrastructure. Think of it as a specific piece of the infrastructure that you want to create, manage, or update. For example, it could be a virtual machine, a database, a storage bucket or a load balancer.

When using Terraform, you write down what resources you need in a configuration file. For each resource, you describe its details such as the type of machine, how

much storage it needs or its location in the cloud. Terraform then creates or updates those resources to match your description.

Syntax of Terraform Resource Block

A **resource block** in Terraform is used to define individual pieces of infrastructure.To create an **AWS** EC2 instance, you would use a resource block like this:

```
resource “aws_instance” “example” {

  ami           = “ami-0c55b159cbfafe1f0”

  instance_type = “t2.micro”

}
```

Here’s how it works:

- **Resource keyword**:This indicates that you are defining a resource.
- **Type and Name**:
  - aws_instance is the type of resource you are creating—in this case, an EC2 instance.
  - "example" is a unique name you assign to this resource so you can reference it later in the configuration.
- **Arguments**:
  - **ami:** This specifies the Amazon Machine Image (AMI) ID, which is the template for the instance.
  - **instance_type:** This defines the size or type of the instance, like t2.micro for a small instance.

Resource blocks always start with the resource keyword, followed by the type and name, and then the arguments inside curly braces {}. Some blocks may also include additional nested configurations like network_interface for networking details.

Once you've written the resource block, you can apply the configuration to create the resource by running the following commands:

terraform init

terraform apply

These steps initialize Terraform and provision the EC2 instance using the settings in the resource block.