Terraform configuration files are composed of one or multiple blocks

Each block is a container of code with {}

Sample Code blocks:

Resource / Data Source / Terraform / Provider / Variable / Module block

Sample block code:

```
block_type "block_label" {
  # Block body with attributes and nested blocks
  attribute_name = "value"
  nested_block {
    nested_attribute = "nested_value"
  }
}
```

Block_type →resource / variable / provider / output

Block_label → Just a block label for provider we can name it as “azure” / “aws”

Resource block:

```
Resource “resource_type” “local_name” {
Identifier = expression (argument)
}
```

TF configuration files uses the .TF as an extension

Module == Collection of TF Files

TF Files can be saved in a Version Control System

Installation

Terraform core can be installed on the following OS:
Linux / Windows / MacOS / Solaris / FreeBSD / OpenBSD

Terraform Versioning:

Terraform core is versioned using 0.X.Y numbering system.

Current latest version of Terraform is 1.7 (as of 17th Jan)

Terraform Block:

```
terraform {
  required_version = ">= 1.6"

  # Set the Azure Provider source and version
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.0.0"
    }
  }
}

# Configure the Microsoft Azure Provider
provider "azurerm" {
  # Skip provider registration (only needed if permissions are lacking)
  skip_provider_registration = true

  # Other features can be configured here if needed
}
```

Example 2:

1. **Equal To (=)**:
   - The = operator specifies an exact version number.
   - Example:
   - required_version = "1.2.0"
   - This ensures that only version 1.2.0 is acceptable.

2. **Greater Than (>) and Greater Than or Equal To (>=)**:
   - The > operator allows versions greater than the specified version.
   - The >= operator allows versions equal to or greater than the specified version.
   - Examples:
   - required_version = "> 2.0"
   - required_version = ">= 2.0"
   - The first example allows any version greater than 2.0, while the second allows 2.0 and newer versions.
3. **Less Than (<) and Less Than or Equal To (<=)**:
   - The < operator allows versions less than the specified version.
   - The <= operator allows versions equal to or less than the specified version.
   - Examples:
   - required_version = "<= 2.9"
   - required_version = "< 3.0"
   - The first example allows versions up to 2.9, and the second allows any version less than 3.0.
4. **Pessimistic Constraint (~>)**:
   - The ~> operator (pessimistic constraint) allows only the rightmost version component to increment.
   - It ensures compatibility with new patch releases within a specific minor release.
   - Examples:
   - required_version = "~> 1.0.4" # Allows 1.0.5 and 1.0.10 but not 1.1.0
   - required_version = "~> 1.1"   # Allows 1.2 and 1.10 but not 2.0
   - Use this format to specify a range of compatible patch releases.

Terraform Providers:

By Default it takes the latest version

```
provider "azurerm" {
  features = {}


  subscription_id = "your_subscription_id"
  client_id       = "your_client_id"
  client_secret   = "your_client_secret"
  tenant_id       = "your_tenant_id"
```

  # Optional: You can specify the version of the AzureRM provider

  # version = "=2.68.0"

  # Optional: Specify the environment (AzureCloud, AzureChinaCloud, AzureUSGovernment, AzureGermanCloud)

  # environment = "AzureCloud"

}

Functionalities of Terraform Providers:

- CRUD Operations
- API Interaction
- Resource Provisioning
- Authentication & Authorization
- State Management
- Data Retrieval
- Dependency Resolution
- Error Handling

- Terraform do not install any provider plugin directly
- Terraform will download the provider during the first step terraform init during initialization phase
- Download and store the provider plugin inside .terraform folder
- For 3rd party needs to be manually installed

Terraform Alias

Alias support multiple regions

Best use-case to consume for different environments

One default provider and additional provider with alias argument

Example:

```
Define two aliases for the azurerm provider
provider "azurerm" {
  alias = "dev"
  subscription_id = "12345678-12234-5678-9012-123456789012"
  client_id = "..."
  client_secret = "..."
  tenant_id = "..."
  features {}
}

provider "azurerm" {
  alias = "prod"
  subscription_id = "87654321-4321-8765-2109-876543210987"
  client_id = "..."
  client_secret = "..."
  tenant_id = "..."
  features {}
}
```

Terraform State:

Terraform state file == Real world Infrastructure provider configuration

It is created after terraform apply command is issued to provision infrastructure

All the provisioned objects in the infra provider with all their attributes are captured in the state file

What if there is no Terraform file?

Terraform state files:

Terraform state and backup are automatically created

Create your own by passing terraform – state flag

Terraform backend stores the state file in root module and its not encrypted at rest.

State should not be edited manually it can be edited only through terraform state commands

Terraform show to view the terraform state file

Terraform refresh to determine the actual state and automatically update the state file but do not provision anything