**­Terraform**

**Terraform** is an open-source infrastructure as code software tool created by HashiCorp. It enables users to create, change, and version the infrastructure.

The syntax of Terraform configurations is called [HashiCorp Configuration Language (HCL)](https://github.com/hashicorp/hcl)

Terraform configuration files can use either of two formats: Terraform domain-specific language (HashiCorp Configuration Language format [HCL]), which is the recommended approach, or JSON format if the files need to be machine-readable. Configuration files that use the HCL format end with the .tf file extension; those using JSON format end with the .tf.json file extension. The Terraform format is human-readable, while the JSON format is machine readable.

**Why we need terraform in devops?**

* Terraform automates the process of creating infrastructure and manages it efficiently. It will allow you to rebuild/change the infra and track the changes in infra with ease.
* It is platform independent and can deploy infra to multiple clouds.
* We can also preview infrastructure changes that we made before applying those changes to the actual infra. This allows you to validate changes more effectively. They can also catch unintended changes earlier, rather than too late.
* Instead of slower, manual processes that introduce the potential risk of human error, you can deploy and manage your infrastructure with reliable code that automates this process for you. You can use templates again and again to build and create identical environments, whether they’re test instances or production environments, without having to re-create settings the old fashioned way (the inefficient way).

**What is tfstate file ?**

This state file contains information about the provisioned infrastructure which terraform manage. Whenever we change the configuration file, it automatically determines which part of your configuration is already created and which needs to be changed with help of state file. State helps to provide idempotence to terraform as it already knows if one resource is present and prevent it to be created again when the same configuration executes.

**TERRAFORM**

Terraform is a infrastructure as code tool from hashicorp. It is a tool for building, changing & managing infra.

**What is infrastructure as code?**

Infrastructure as code as a concept, it is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface.

1. **Initialize** - Run terraform init in the project directory with the configuration files. This will download the correct provider plug-ins for the project.
2. **Plan & Apply** - Run terraform plan to verify creation process and then terraform apply to create real resources as well as state file that compares future changes in your configuration files to what actually exists in your deployment environment.

**Advantages of Terraform:**

* Platform agnostic:

In a modern datacenter, you may have several different clouds and platforms to support your various applications. With Terraform, you can manage a heterogenous environment with the same workflow by creating a configuration file to fit the needs of your project or organization.

* State management:

Terraform creates a state file when a project is first initialized. Terraform uses this local state to create plans and make changes to your infrastructure. Prior to any operation, Terraform does a refresh to update the state with the real infrastructure. This means that Terraform state is the source of truth by which configuration changes are measured. If a change is made or a resource is appended to a configuration, Terraform compares those changes with the state file to determine what changes result in a new resource or resource modifications.

* Operator Confidence:

The workflow built into Terraform aims to instill confidence in users by promoting easily repeatable operations and a planning phase to allow users to ensure the actions taken by Terraform will not cause disruption in their environment. Upon terraform apply, the user will be prompted to review the proposed changes and must affirm the changes or else Terraform will not apply the proposed plan.

**Terraform providers:**

Terraform relies on plugins called "providers" to interact with remote systems. Each provider adds a set of [resource types](https://www.terraform.io/docs/language/resources/index.html) and/or [data sources](https://www.terraform.io/docs/language/data-sources/index.html) that Terraform can manage.

Every resource type is implemented by a provider; without providers, Terraform can't manage any kind of infrastructure.

Most providers configure a specific infrastructure platform (either cloud or self-hosted)

A [provider](https://www.terraform.io/language/providers) is a Terraform plugin that allows users to manage an external API.

Providers **allow Terraform to interact with cloud providers**

**Variables & outputs:**

* Input variables : Input variables serve as parameters for a Terraform module.
* Output values : Output values are like the return values of a Terraform module.
* Local values : A local value assigns a name to an [expression](https://www.terraform.io/docs/language/expressions/index.html), so you can use it multiple times within a module without repeating it.

**Terraform commands:**

terraform init : Terraform uses a plugin-based architecture to support hundreds of infrastructure and service providers. Initializing a configuration directory downloads and installs providers used in the configuration, which in this case is the aws provider.

terraform fmt : The terraform fmt command automatically updates configurations in the current directory for easy readability and consistency.

terraform validate : To check If ur code is syntactically valid then use terraform validate.

terraform plan : to see the changes that are going to take place after running the script.

terraform apply : to run and create the actual infrastructure.

terraform show : Inspect the current state using terraform show. Terraform has a built in command called terraform state for advanced state management. For example, if you have a long state file, you may want a list of the resources in state, which you can get by using the list subcommand.

terraform state list

**PROVISIONERS :**

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

It cannot be shown in plan.

Built-in provisioners – local-exec, remote-exec, file

Alternatives – cloudinit, userdata, ansible

**Local exec :**

It will execute the command from local machine.

Provisioner “local-exec” {

command = “echo ${self.public\_ip} > publicip.txt”

}

When u run apply it will create a file in local machine and add public ip to it.

**File provisioner :**

We can create a file on ur remote machine.

Provisioner “file”{

Content = “anycontent” (or) Source = “localfile.txt”

Destination = “/home/ubuntu/file.txt”

}

**Adding connector to do ssh to remote machine :**

Connection {

type = “ssh”

host = self.public\_ip

user = “ubuntu”

private\_key = file(“/home/ubuntu/downloads/key.pem)

}

Provisioners runs only when the instance is created for the first time

**Remote-exec :**

This allows us to run commands on remote machine.

Provisioner “remote-exec” {

Script

Inline = [

“touch /home/ubuntu/file1.txt”

]

}

We can define a connection block inside provisioner. we can have multiple provisioners for dfrnt machines.

If provisioner fails it will make the resource tainted

We can mention parameter – on\_failure = “continue” or “fail”

**Terraform interview questions :**

1. What is terraform?
2. It works on client only architecture frame
3. Why is it used for devops ?

It can do cmplte orchestration not just configuration management. Supports multiple clouds. easily manages dynamic infra. configuration changes can be done smoothly. portable from 1 provider to another.

1. Why tf init ?

Plugin installation, child module installation and backend initialization

1. 4 stages – init,plan,apply,destroy
2. Init – creates operational directory , plan – execution strategy, apply – ensures plan is set on motion,destroy – destroys all applied resources
3. Define null resource in terraform - it implements standard resource lifecycle but takes no further action. Triggers argument will allow us to set arbitrary set of values that, when changed will cause the resource to be replaced.