**Terraform**

1. **What is Terraform?**

Terraform is an Infrastructure As a Code tool to build, change and version infrastructure of any cloud and on-prem resources. Terraform code is written in Hashicorp Configuration Language (HCL).

1. **Terraform Features?**

* *It manages any infrastructure:* it supports multiple cloud and on prem providers & services. Ex: AWS, Azure, GCP, Alibaba cloud, VMware, citrix
* *Track your infrastructure:* It keeps track of all real infrastructure in statefile.
* *Automated changes in Infrastructure:* Terraform configuration files are declarative; we do not need to write step-by-step instructions to create resources because Terraform handles the underlying logic.
* *Standardized configurations:* terraform supports reusable components called Modules. Most modules available publicly and we can write our own modules.
* *Collaborative:* Since your configuration is written in a file, you can commit it to a Version Control System (VCS) and use Terraform Cloud to efficiently manage Terraform workflows across teams.

1. **How does Terraform work?**

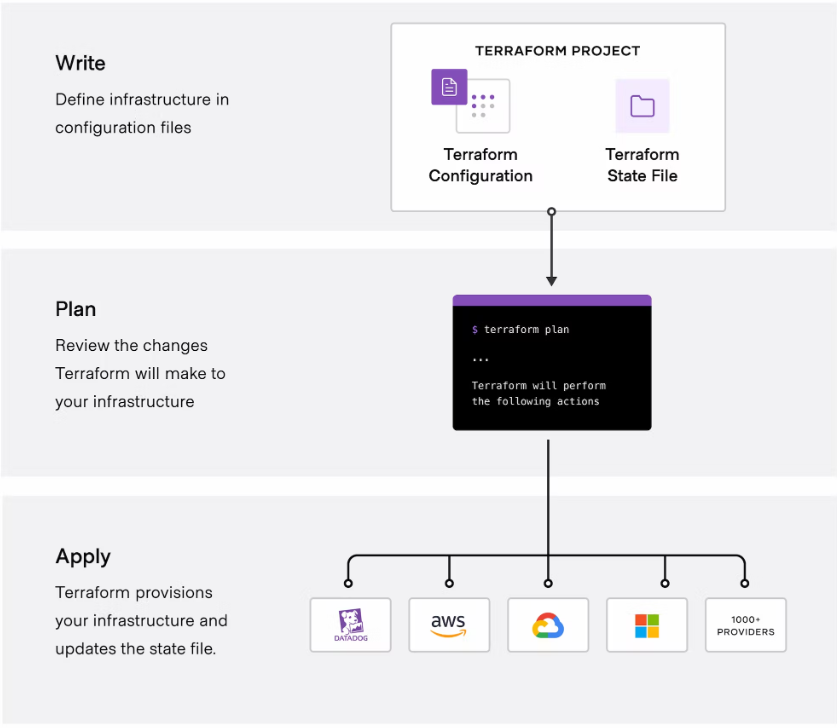
The core Terraform workflow consists of three stages:

***Write:*** Author infrastructure as code. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.

***Plan:*** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration. Preview changes before applying

***Apply:*** Provision reproducible infrastructure.. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.

*Reference url*: <https://developer.hashicorp.com/terraform/intro/core-workflow>



1. **Terraform important files?**

Here are the important files while working with terraform.

**main.tf** – is the configuration file of infrastructure

**variables.tf** – is the file where we can define the variables which are used in main.tf file.

**outputs.tf** – is the file where we can define the output values which are generated after infrastructure created and we can use these values for further.

**values.tfvars/values.auto.tfvars** – is the file where we can define the real values pass to variables.

**terraform.tfstate** – is the file where real infrastructure references are saved in JSON format.

**.terraform.lock.hcl** – is the lock file where we can mention specific, approved, verified versions of provider plugins if we don’t want to change the versions in future.

**.terraform.tfstate.lock.info** – is the file where we can see the statefile lock information. This file will be generated only when we execute terraform plan, apply, destroy and refresh commands and it will deleted after command exits.

1. **Terraform main commands?**

Here are the main commands and its usage. Before executing the terraform commands we must write/create one configuration file with at least one provider.

**terraform init** – it is used to initiate the current working directory and it will get required plugins, saved to **.terraform** directory, for providers which are defined in main.tf.

**terraform validate** – it will validate the syntax of configuration files.

**terraform fmt** – it will format the configuration.

**terraform plan** – It will show us the what are resources that will be create, update or delete in the target environment. We call it as execution plan. No actions will be performed on target environment.

**terraform apply** – it will create or update the resources in target environments which are defined in configuration file.

**terraform destroy** – it will delete the resources which are created previously or imported to terraform.

1. **What Terraform statefile and its management?**

Terraform would store the managed infrastructure state in statefile.This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.

1. **What Terraform lock files and state lock?**

State lock is to lock statefile while running terraform plan, apply, destroy and refresh commands.

.terraform.lock.hcl file is to lock the versions of providers.

1. **What are Terraform meta-arguments and its uses?**

**Depends\_on :** used to define the dependency. In built terraform has the capability to track the dependence of the resources, for some resources terraform cannot have dependency tracker.

**Count:** Used to create the same type of multiple resources.

**For\_each:** If we want to create the same type of resources with different configurations we can use for\_each. for\_each will work with only map or set of values.

We have two objects to identify the each member in map or set:

each.key: The map key or set member corresponding to each member.

each.value: The map value corresponding to each member.

**Provider:**

provider meta-argument specifies which provider to be used for a resource. This is useful when you are using multiple providers which is usually used when you are creating multi-region resources. For differentiating those providers, you use an alias field. Resource then reference the same alias field of the provider as **provider.alias** to tell which one to use.

**Lifecycle:** lifecycle is nested block of resource.it will useful if we don’t want change or delete the manual updates of resource.

There are three customization options available

**Create\_before\_destroy:** true/false – If we need to create resource before deleting it.

**Prevent\_destroy:** true/false – if we need to protect resource from accidental deletion of resource.

**Ignore\_changes:** list attributes – if we don’t want to change any existing configuration of resources, we can use this. It will be useful for DB resources and tags of resources.

1. **Different Provisioners used in Terraform?**

You can use provisioners to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

There three main provisioners are there.

1**. File:** used to create files and save some data into it anywhere, means either local or remote machines.

2. **Local-exec:** to execute the commands on local machine where terraform installed.

3. **Remote-exec**: to execute the commands or scripts on remote hosts which are created.

1. **Locals, dynamic block usages?**

***Local values*** can be helpful to avoid repeating the same values or expressions multiple times in a configuration, but if overused they can also make a configuration hard to read by future maintainers by hiding the actual values used.

Terraform ***dynamic blocks*** are a special Terraform block type that provide the functionality of a for expression by creating multiple nested blocks. For example if we need to create multiple subnets in single azure VNet.

1. **Input, output, environment variables?**

 Input variables are *created* by a variable block, but you *reference* them as attributes on an object named var.

Output values make information about your infrastructure available on the command line and can expose information for other Terraform configurations to use.

Environment Variables – we can use these

1. **Terraform target, import, taint, refresh?**

Terraform target: To create a plan or apply for some of the resources.

Ex: terraform plan -target=<resource-type>.<reference name of block>

terraform apply -target=<resource-type>.<reference name of block>

terraform taint: to delete the mis configured or corrupted resources in next terraform apply . untaint can be used to revert the tainted resources. Once we apply the taint on resource,it will marked as tainted in statefile and it will be deleted in next apply.

terraform import: used to bring the manually created resources to terraform management. It involves two steps. 1. First we need to write the resource in configuration file 2. Execute the terraform import command.

terraform import <resourcetype>.<reference name> <address or resource id>

it will update the data in statefile and we added the information in configuration file. It wont be deleted in next terraform apply.

terraform refresh : to update the manual changes of resource into statefile but it will be override with configuration files when run next terraform apply. To avoid this we should update the manual changes in the configuration file as well.

Reference: Terraform official tutorials URL - [Tutorials | Terraform | HashiCorp Developer](https://developer.hashicorp.com/terraform/tutorials)

**Terraform Scenarios/Use cases**