## vCloud Director Embraces Terraform - VMware Cloud Provider BlogTERRAFORM

Infrastructure as Code (IaC) is a widespread terminology among DevOps professionals. It is the process of managing and provisioning the complete IT infrastructure (comprises both physical and virtual machines) using machine-readable definition files. It is a software engineering approach toward operations. It helps in automating the complete data center by using programming scripts.

With all the features that Infrastructure as Code provides, it has multiple challenges:

* Need to learn to code
* Don’t know the change impact.
* Need to revert the change
* Can’t track changes
* Can’t automate a resource
* Multiple environments for infrastructure

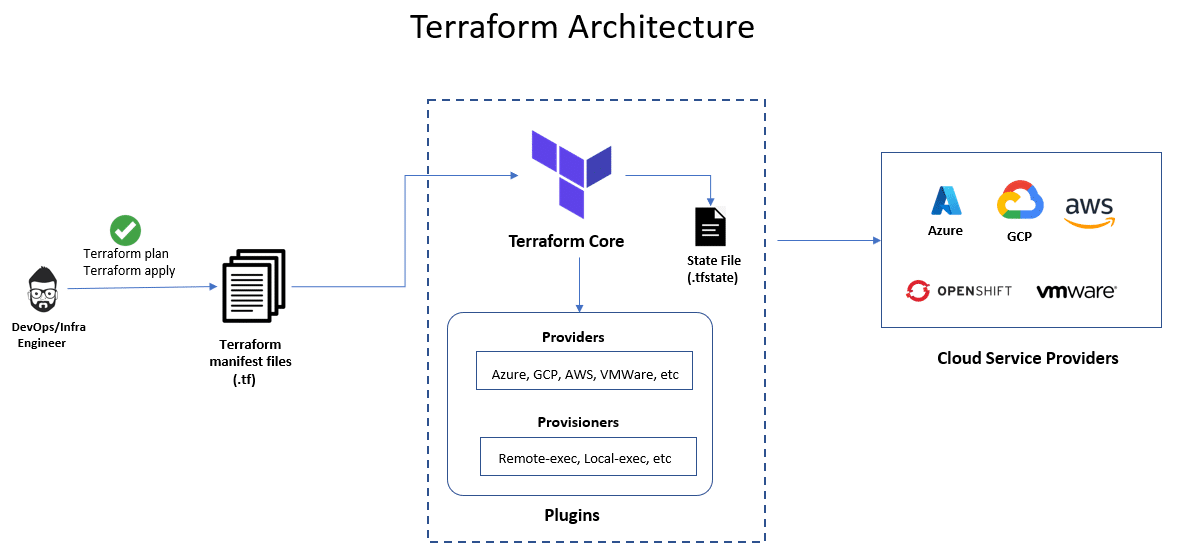
Terraform has been created to solve these challenges.

**What is Terraform?**

Terraform is an open-source infrastructure as Code tool developed by HashiCorp. It is used to define and provision the complete infrastructure using an easy-to-learn declarative language.  
  
It is an infrastructure provisioning tool where you can store your cloud infrastructure setup as codes. It’s very similar to tools such as CloudFormation, which you would use to automate your AWS infrastructure, but you can only use that on AWS. With Terraform, you can use it on other cloud platforms as well.

Below are some of the benefits of using Terraform.

* Does orchestration, not just configuration management
* Supports multiple providers such as AWS, Azure, GCP, Digital-Ocean and many more
* Provide immutable infrastructure were configuration changes smoothly
* Uses easy to understand language, HCL (Hash Corp configuration language)
* Easily portable to any other provider
* Supports Client only architecture, so no need for additional configuration management on a server



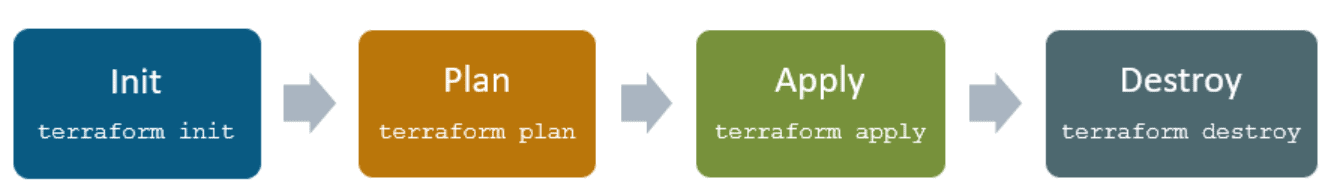
**Terraform Core concepts**

Below are the core concepts/terminologies used in Terraform:

* **Variables**: Also used as input-variables, it is key-value pair used by Terraform modules to allow customization.
* **Provider**: It is a plugin to interact with APIs of service and access its related resources. (
* **Module**: It is a folder with Terraform templates where all the configurations are defined
* **State**: It consists of cached information about the infrastructure managed by Terraform and the related configurations.
* **Resources**: It refers to a block of one or more infrastructure objects (compute instances, virtual networks, etc.), which are used in configuring and managing the infrastructure.
* **Data Source**: It is implemented by providers to return information on external objects to terraform.
* **Output Values**: These are return values of a terraform module that can be used by other configurations.
* **Plan**: It is one of the stages where it determines what needs to be created, updated, or destroyed to move from real/current state of the infrastructure to the desired state.
* **Apply**: It is one of the stages where it applies the changes real/current state of the infrastructure in order to move to the desired state.

**Terraform Lifecycle**

Terraform lifecycle consists of – **init**, **plan**, **apply**, and **destroy**.

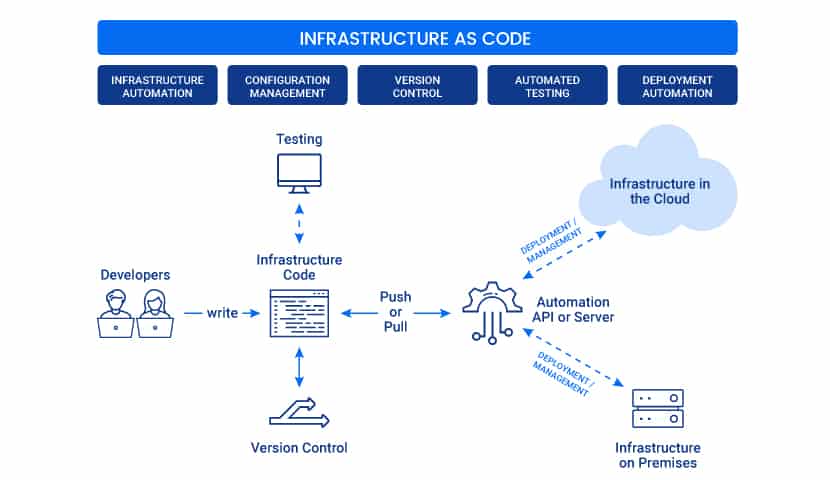


* Terraform init initializes the working directory which consists of all the configuration files
* Terraform plan is used to create an execution plan to reach a desired state of the infrastructure. Changes in the configuration files are done in order to achieve the desired state.
* Terraform apply then makes the changes in the infrastructure as defined in the plan, and the infrastructure comes to the desired state.
* Terraform destroy is used to delete all the old infrastructure resources, which are marked tainted after the apply phase.

**How Terraform Works?**

Terraform has two main components that make up its architecture:

* Terraform Core
* Providers



**Terraform Core**

Terraform core uses two input sources to do its job.

The **first** input source is a Terraform configuration that you, as a user, configure. Here, you define what needs to be created or provisioned. And the **second** input source is a state where terraform keeps the up-to-date state of how the current set up of the infrastructure looks like.

So, what terraform core does is it takes the input, and it figures out the plan of what needs to be done. It compares the state, what is the current state, and what is the configuration that you desire in the end result. It figures out what needs to be done to get to that desired state in the configuration file. It figures what needs to be created, what needs to be updated, what needs to be deleted to create and provision the infrastructure.

**Providers**

The second component of the architecture are providers for specific technologies. This could be cloud providers like AWS, Azure, GCP, or other infrastructure as a service platform. It is also a provider for more high-level components like Kubernetes or other platform-as-a-service tools, even some software as a self-service tool.

It gives you the possibility to create infrastructure on different levels.

|  |  |
| --- | --- |
| **Terraform** | **Ansible** |
| Terraform is a provisioning tool. | Ansible is a configuration management tool. |
| It follows a declarative Infrastructure as a Code approach. | It follows both declarative & procedural approach. |
| It is the best fit for orchestrating cloud services and setup cloud infrastructure from scratch. | It is mainly used for configuring servers with the right software and updating already configured resources. |
| Terraform does not support bare metal provisioning by default. | Ansible supports the provisioning of bare metal servers. |
| It does not provide better support in terms of packaging and templating. | It provides full support for packaging and templating. |
| It highly depends on lifecycle or state management. | It does not have lifecycle management at all. |

**Terraform Cheat Sheet**

Terraform command accepts a variety of subcommands such as terraform init or terraform plan. Some of the widely used commands and subcommands I have listed below.

|  |  |
| --- | --- |
| **Show version** | |
| terraform –version | Shows terraform version installed |
| **Initialize infrastructure** | |
| terraform init | Initialize a working directory |
| terraform init -input=true | Ask for input if necessary |
| terraform init -lock=false | Disable locking of state files during state-related operations |
| **Get** | |
| terraform get | downloads and update modules mentioned in the root module |
| terraform get -update=true | modules already downloaded will be checked for updates and updated |
| **Provision infrastructure** | |
| terraform plan | Creates an execution plan (dry run) |
| terraform plan -out=path | save generated plan output as a file |
| terraform plan -destroy | Outputs a destroy plan |
| terraform apply | Executes changes to the actual environment |
| terraform apply –auto-approve | Apply changes without being prompted to  enter” yes” |
| terraform apply -refresh=true | Update the state for each resource prior to planning and applying |
| terraform apply -input=false | Ask for input for variables if not directly set |
| terraform apply -var ‘foo=bar’ | Set a variable in the Terraform configuration, can be used multiple times |
| terraform apply -var-file=foo | Specify a file that contains key/value pairs for variable values |
| terraform apply -target | Only apply/deploy changes to the targeted resource |
| terraform destroy –auto-approve | Destroy/cleanup without being prompted to enter ”yes” |
| terraform destroy -target | Only destroy the targeted resource and its dependencies |
| **Terraform Workspaces** | |
| terraform workspace new | Create a new workspace and select it |
| terraform workspace select | Select an existing workspace |
| terraform workspace list | List the existing workspaces |
| terraform workspace show | Show the name of the current workspace |
| terraform workspace delete | Delete an empty workspace |
| **Format and validate Terraform Code** | |
| terraform fmt | Format code as per HCL canonical standard |
| terraform validate | validate configuration files for syntax |
| **Inspect Infrastructure** | |
| terraform graph | creates a resource graph listing all resources in your configuration and their dependencies. |
| terraform output | List all the outputs for the root module |
| terraform output instance\_public\_ip | List only the specified output |
| terraform output -json | List all the outputs in JSON format |
| terraform show | provide human-readable output from a state or plan file |
| **Terraform Import** | |
| terraform import aws\_instance.foo i-abcd1234 | import an AWS instance with ID i-abcd1234 into aws\_instance resource named “foo” |
| **Terraform State Manipulation** | |
| terraform state list | list all resources in the state file |
| terraform state list aws\_instance.my\_ec2 | Only list resource with the given name |
| terraform state mv | move an item in the state file |
| terraform state rm | Remove items from the state file |
| terraform state pull | Pull current state and output to stdout |
| terraform state push | Update remote state from a local state file |
| terraform state show aws\_instance.my\_ec2 | Show the attributes of a single resource |
| terraform state refresh |  |