What is Terraform?

Terraform can be defined as a tool for **versioning, changing,** and **building** infrastructure efficiently and safely. It can manage popular and existing service providers and custom in-house solutions also.

Configuration files explain to terraform that the elements required executing our entire data center or an individual application. Terraform produces a single execution plan explaining what it'll do for reaching the desired state after it runs for building the desired infrastructure. Terraform is capable of determining what will change and build execution plans that can be used as the configuration modifications.

The infrastructure terraform could handle low-level elements like networking, storage, compute instances, also high-level elements like **SaaS features, DNS entries**, etc.

Terraform can provide support with multi-cloud via having a single workflow for every cloud. Various manages of terraform infrastructure could be hosted over Google Cloud Platform, Microsoft Azure, and Amazon Web Services, or on-prem within the private clouds like CloudStack, OpenStack, or VMWare vSphere. Terraform considers **IaC** (Infrastructure as Code). So, we need not to be worried about our infrastructure drifting away through the desired configuration.

Architecture of Terraform

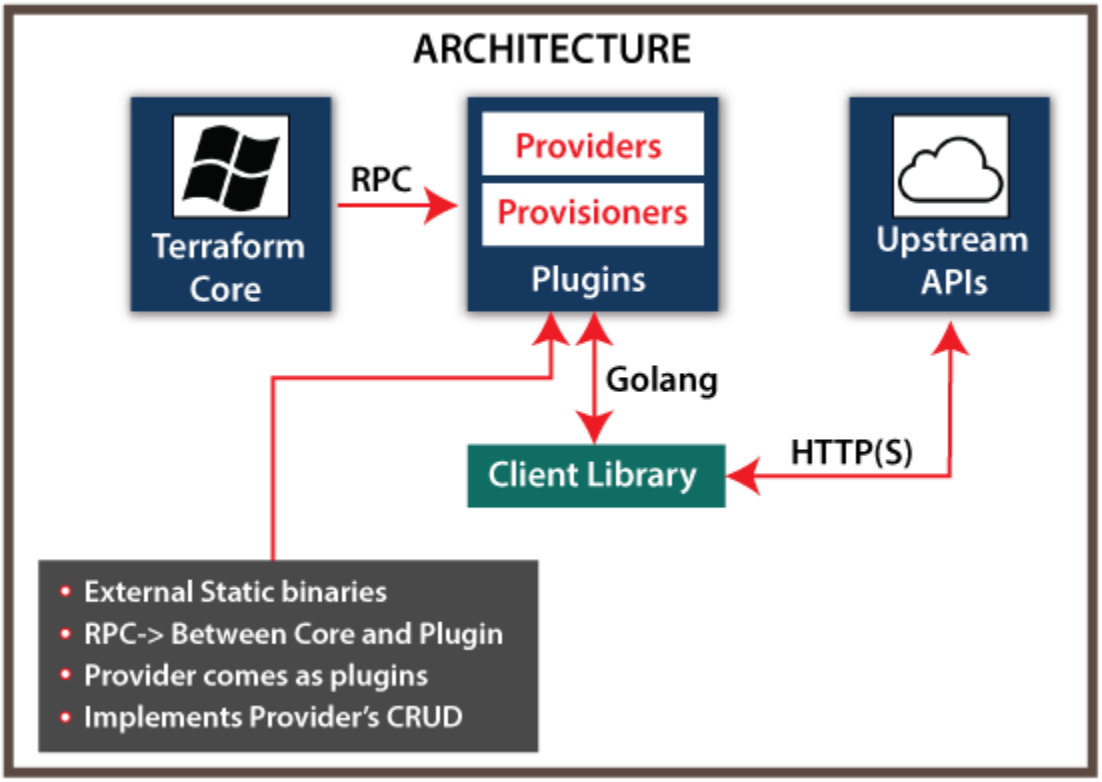
The architecture includes the given components:

Terraform Core: The core of Terraform is liable for creating the dependency graph and reading configuration. Terraform heavily relies on the current graph theory to manage dependencies. The core of Terraform is an actual project of Terraform on GitHub.

Plugins: Terraform plugins can be defined as the external individual static binaries. The core of Terraform communicates with the plugins by the RPC interface during the applying and planning phases. If we are unknown to RPC, assume that these plugins are the servers and the core of Terraform creates API calls for these servers. The most common plugin type is the

Terraform Provider Plugin. These plugins implement the resources along with any basic **create, read, update,** and **delete** (CRUD API) to communicate with various services of the third party.

Upstream API: Upstream API can be defined as a third party. It is an external API or service. It is necessary to remember that the core of Terraform never interacts with the APIs directly. Rather than, the core of Terraform asks the plugin of Terraform Provider for performing any operation. After that, these plugins will communicate with upstream API. It facilitates a clear corner separation. The core of Terraform doesn't require understanding API nuances. Terraform Provider Plugin doesn't require understanding the graph theory.



aC (Infrastructure as Code)

IaC or Infrastructure as Code has attained enough momentum because it provides support in solving issues that previously bothered management of Infrastructure:

- **Reproducible environments:** The similar environment can be built again and again by applied code for generating infrastructure. An environment may drift away through the desired state and complex to recognize problems that may creep into our release pipeline. New environments can be destroyed and created easily and no environment will get specific treatment with IaC.

- **Convergence and Idempotence:** Idempotence can be defined as a trait that means no matter we use the configuration defines by our IaC. There will no side effects over the environments. Besides, convergence can be defined as a trait which means actions are taken when they require. Only those actions that are required to fetch the environment towards the desired condition are run. In case, an environment is in the desired condition already then no actions are required.
- **Easing collaboration:** If we have code within the version control system such as Git, it will permit teams for collaborating on infrastructure. Various members of the team can get the code's specific version and build their environments to test and other purposes.
- **Self-service infrastructure:** Cloud elasticity permits resources to be built on-demand. Various developers can plan any infrastructure they require when they require it. Further, IaC improves the condition by permitting developers to apply modules of the infrastructure for creating identical environments in the application development lifecycle. The modules of the infrastructure can be shared with many developers and built by operations.

Some other options of IaC

Usually, cloud providers facilitate the native infrastructure as the solutions of code. For example:

- Microsoft Azure contains Azure Resource Manager templates
- Google Cloud Platform contains deployment manager
- Amazon Web Services contain Cloud Formation

All come with the specific form of defining Infrastructure as Code (IaC). Also, that holds for various private clouds such as OpenStack, which includes heat for defining Infrastructure within the code.

Hybrid-cloud is more powerful to have a common workflow and one tool for managing infrastructure. Even if we are using a single cloud only, it can be worth futureproofing in case we do leverage more than one cloud later on.

Supported Infrastructure of Terraform

Terraform infrastructure assimilation permits to manage services and software, including databases such as **MySQL**, configuration management tools such as **Chef**, and source control systems such as **GitHub**, and many others.

Use Cases of Terraform

Let's consider a few example use cases of terraform:

- **Disposable environments:** It is general to have QA or staging and production environments. These types of environments are the production counterpart's smaller clones. But these environments are used for testing newer applications before publishing in production. As the environment of production is more complex and grows larger, it becomes harsh for maintaining the up-to-date environment.

  The environment of production could be codified and begun with dev, QA, or staging using terraform. These configurations are used for spinning-up newer

environments rapidly to test in and be disposed of easily. Terraform can support reduce the complexity of controlling parallel environments. Also, it can destroy and create environments elastically.

- **Multi-cloud Deployment:** It can be creative to spread the infrastructure across more than one cloud for increasing fault-tolerance. Fault-tolerance is bound by the provider availability by using a single cloud provider or region. Multi-cloud deployment permits for more loss recovery of the entire provider or a region. Also, increasing fault-tolerance uses for hybrid clouds where we have our private cloud executing on-prem. However, leverage any public cloud for disaster recovery and business continuity. For example, executing a secondary application or database, tiring access to information, backing up data when our servers of primary on-prem fails. Also, we may encounter some situations where a single public cloud provider contains the services that are not available on our approved public provider. Accomplishing multi-cloud deployments may be very asserting as various tools for management of Infrastructure are cloud-specific. Also, terraform is cloud-agnostic and permits one configuration to be applied for managing more than one provider. It simplifies orchestration and management and supporting operators create multi-cloud large-scale infrastructures.
- **Multi-tier applications:** A common arrangement is N-tier architecture. The web server pool is the most general 2-tier architecture that uses the database tier. Other additional tiers included for **API servers, routing meshes, caching servers**, etc. This arrangement is used due to tiers could be independently scaled and facilitate the corners separation. Terraform is useful for managing and building these infrastructures. All tiers can be defined as the resources collection. The dependencies among all tiers are automatically managed. Terraform will make sure that the tier of the database is present before all the web servers are begun. Then all the tiers could be scaled efficiently with terraform by changing a single value of count configuration because the provisioning and creation of any resource are automated and codified.
- **Resource Schedulers:** The application's static assignment for machines becomes challenging increasingly with large-scale infrastructures. To solve this issue, there are a lot of schedulers such as **Kubernetes, YARN, Mesos,** and **Borg**. These schedulers can be used for scheduling the Docker containers, Spark, Hadoop, and many more. Terraform is not bound to physical providers such as AWS. The

resource schedulers may be treated like a provider, allowing terraform to claim resources through them. It also permits terraform to be applied inside the layers. It is further used for setting up various physical infrastructures executing the schedulers and also provisioning on the scheduled grid.