**Terraform**

**Introduction**

Terraform is an open source tool for managing the cloud infrastructure that developed and maintained by Hashicorp. It helps you to describe your infrastructure, using HCL (HashiCorp Configuration Language) which is human readable and simple to understand, and then deploy it . Terraform is used for building, changing and versioning infrastructure safely. It is Infrastructure as code. Terraform is used to create, manage, and update infrastructure resources such like as virtual machines, virtual networks, clusters. The Terraform CLI provides a simple mechanism to deploy and version the configuration files . The Terraform code is stored in configuration files that are files with a \*.tf extension.

**Provider**

Terraform configurations must declare which providers they require, so that Terraform can install and use them. Additionally, some providers require configuration before they can be used. A provider is responsible for understanding API interactions and exposing resources.

The Terraform Registry is the main directory of publicly available Terraform providers, and hosts providers for most major infrastructure platforms. Once you've found a provider you want to use, you can require it in your Terraform configuration and start using the resource types it provides. Terraform can automatically install providers from the Terraform Registry when you run terraform init.

**Backends**

A "backend" in Terraform determines how state is loaded and how an operation such as apply is executed. This abstraction enables non-local file state storage, remote execution, etc.

**Benefits of backends:**

**Working in a team**: Backends can store their state remotely and protect that state with locks to prevent corruption. Some backends such as Terraform Cloud even automatically store a history of all state revisions.

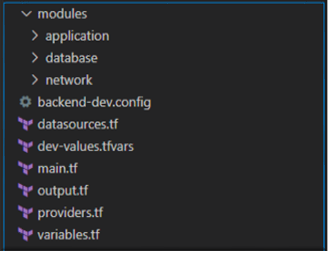
**Keeping sensitive information off disk**: State is retrieved from backends on demand and only stored in memory.

**Remote operations:** For larger infrastructures or certain changes, terraform apply can take a long, long time. Some backends support remote operations which enable the operation to execute remotely. You can then turn off your computer and your operation will still complete. Paired with remote state storage and locking above, this also helps in team environments.

Backends are completely optional. You can successfully use Terraform without ever having to learn or use backends. However, they do solve pain points that afflict teams at a certain scale. If you're an individual, you can likely get away with never using backends.

**Terraform Structure:**

The primary module structure requirement is that a root module must exist. The root module is the directory that holds the Terraform configuration files that are applied to build your desired infrastructure. Any module should include, a main.tf, a variables.tf, a providers.tf, a backend.config, and an outputs.tf file, modules, input-values.tfvars, datasources.tf.



**main.tf**

It contains data for creating resource groups, networks (resource groups, vnets, subnet, VM, Route tables, network security group etc).

It calls modules, locals and data-sources to create all resources. The file holds all your module blocks and any needed resources not contained within your nested modules. Nested modules to split up your infrastructure’s required resources.

**variables.tf**

The variables.tf files contain input variable and output variable declarations for all the resources. The configuration is declared as variables and the variable for individual resources are stored in “variables.tf” file.

**dev-values.tfvars**

To persist variable values, create a file and assign variables within this file. For all files which match terraform.tfvars or \*.auto.tfvars present in the current directory, terraform automatically loads them to populate variables.

**output.tf**

Outputs are a way to tell Terraform what data is important. This data is outputted when apply is called and can be queried using the terraform output command. Module may call other modules and connect them together by-passing output values from one to input values of another.

**datasource.tf**

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to build on information defined outside of Terraform or defined by another separate Terraform configuration.

**Modules:**

The Modules are sub directories with self-contained Terraform code. A module is a container for multiple resources that are used together. The root module is the directory that holds the Terraform configuration files that are applied to build your desired infrastructure. That module may call other modules and connect them together by-passing output values from one to input values of another.

Modules can also call other modules using a module block, but we recommend keeping the module tree relatively flat and using module composition as an alternative to a deeply nested tree of modules, because this makes the individual modules easier to re-use in different combinations.

**Terraform Workflow**:

There are steps to build infrastructure with terraform using CLI after created the resources.

* INIT
* Plan
* Apply
* Destroy

**INIT**

Initialize the Terraform configuration directory using terraform init.

Init will create a hidden directory ”.terraform” and download plugins as needed by the configuration. Init also configures the -backend-config option can be used for partial backend configuration.

**Command**:

terraform init -backend-config=”backend-dev.config”

**PLAN**

The terraform plan command is used to create an execution plan. The plan will be used to see what all the resources are gets created/updated/deleted, before getting applied. Actual creation will happen in “apply” command.

The var file given will define resources unique for each team.

Command:

terraform plan -var-file="input-values.tfvars"

**APPLY**

The terraform apply command is used to apply the changes in the configuration. You’ll notice that the apply command shows you the same plan output and asks you to confirm if you want to proceed with this plan.

The “-auto-approve” parameter will skip the confirmation for creating resources. It’s better not to have it when you want to apply directly, without “plan”.

Command:

terraform apply -var-file="input-values.tfvars" -auto-approve

**Terraform State Management:**

Terraform stores the resources and it manages into a state file. There are two types of state files: remote or local. Where local state is great for an isolated developer, remote state is quite indispensable for a team as each member will need to share the infrastructure state whenever there is a change.

Terraform compares those changes with the state file to determine what changes result in a new resource or resource modifications. Terraform must store state about our managed infrastructure and configuration. This state is used by Terraform to map real world resources to our configuration, keep track of metadata, and to improve performance for large infrastructures.

**Terraform Destroy:**

We get another plan output indicating which resources will be destroyed indicated by a - next to them.

**Conclusion:**

In this blog, we learned how to create Terraform configuration files. We reviewed the concepts of the init, plan, apply, and destroy commands that are most commonly used for deploying Terraform configuration files.