TERRAFORM VIA HCL BASICS

For managing and provisioning infra (creating and setting up IT infra, steps reqd to manage user and sys access to various resources) through code - IaC

Tools that can be used: Terraform, CloudFormation, Pulumi

-allows dev to define infra in a high level, declarative language, allowing version ctrl

TERRAFORM

1. Declarative Config- you specify desired state of infra and Tf figures out necessary actions to achieve that state
2. Infra Prov- resources like VMs, networks, storage etc are automatically provisioned i.e., created and managed
3. VCS - similar to Git

Need for Terraform:
Other than automation, provides Modules - self-contained units of Infra Config that can be used across different projects.
Ensures scalability and supports varied no of cloud providers to ensure consistency across diff env

TERMINOLOGY OF TERRAFORM

Provider- manages resources of specific cloud or infra platform

```
provider "aws" {
    region="us-east-1"
    }
```

Resource- is a specific infra obj that is managed eg VM, storage bucket or network

```
resource "aws_instance" "my_instance" {…}
```

Module- self-contained package of Terraform config, can be used as a reusable building block, encapsulates set of resources and their config

```
module "vpc" { … }
```

Variable- parameterisation of Tf configs, passing of values dynamically during runtime is enabled. Basically a placeholder for values received during runtime

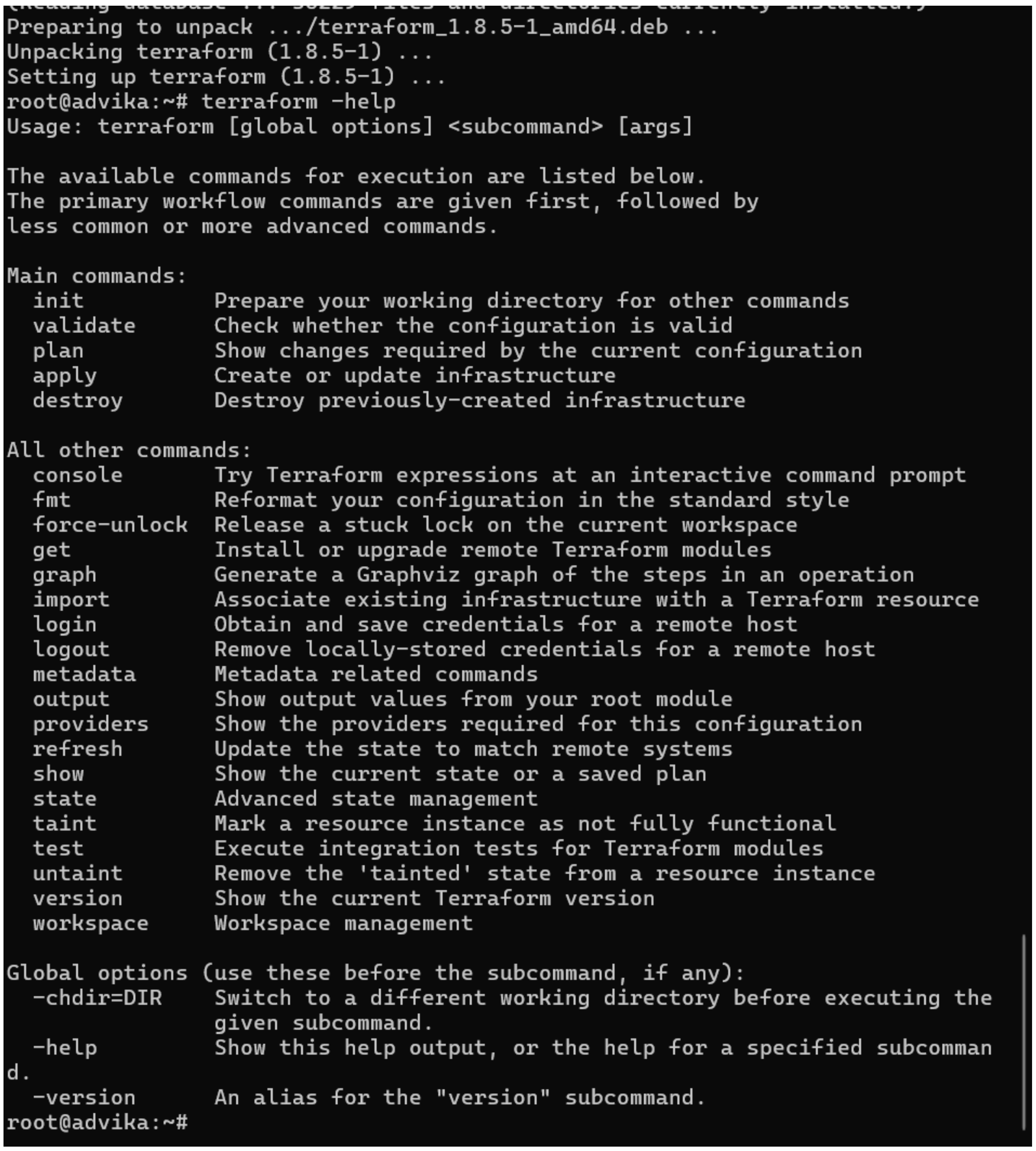
```
variable "identifier" { … }
```

Output- values that are computed, can be referenced after Tf applies changes. Can be displayed

```
output "private_ip" { …  }
```

State- record book keeping track of resources created and their current status

Installation of Terraform:
https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli



```
(Reading database ... 36229 files and directories currently installed.)
Preparing to unpack .../terraform_1.8.5-1_amd64.deb ...
Unpacking terraform (1.8.5-1) ...
Setting up terraform (1.8.5-1) ...
root@advika:~# terraform -help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  metadata      Metadata related commands
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Execute integration tests for Terraform modules
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace management

Global options (use these before the subcommand, if any):
  -chdir=DIR    Switch to a different working directory before executing the
                given subcommand.
  -help         Show this help output, or the help for a specified subcomman
d.
  -version      An alias for the "version" subcommand.
root@advika:~#
```

TERRAFORM CONFIGURATION LANGUAGE (HCL)

HCL is the syntax used in Terraform, declarative language. It is a domain-specific lang (DSL)

{ DSL- source code is being generated from the domain, not being interpreted by it, therefore no runtime overhead: the generator like a compiler removes abstractions Independent of the target platform, dedicated to a particular problem domain, problem representation technique or particular solution technique }

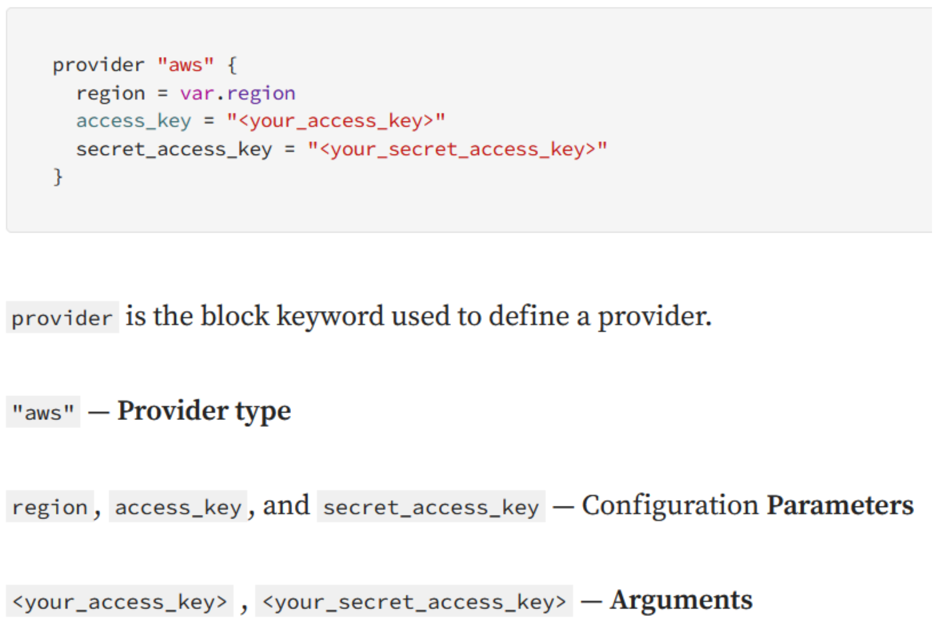
COMPONENTS OF HCL SYNTAX:

HCL Blocks - blocks are used to define the resources i.e., specify attributes and corresponding values to configure desired behaviour.

Has a type that can have 0 or more labels followed by { } brackets

1. Provider Block: specifies the cloud/service provider that Tf will interact with, has config settings within it (access credentials and API endpoints)
2. Resource Block: defines the resources, their behaviour and properties that Tf will manage
3. Data Block: allows query and fetching of info to use in resource config from external source
4. Variable Block: specifies values relating to variables, enables parameterisation and dynamic config of infra
5. Output Block: defines values that are displayed as outputs after successful resource provisioning
6. Module Block: code reuse, groups related resources and configs together
7. Provisioner Block: allows scripts to run and commands to be executed on a resource
8. Locals Block: enables creation of intermediate variables, local scope vars that can be reused
9. Terraform Block: contains global config settings, describes behaviour of tf during execution of config

Parameters and Args



```
provider "aws" {
  region = var.region
  access_key = "<your_access_key>"
  secret_access_key = "<your_secret_access_key>"
}
```

`provider` is the block keyword used to define a provider.

`"aws"` — **Provider type**

`region`, `access_key`, and `secret_access_key` — Configuration **Parameters**

`<your_access_key>` , `<your_secret_access_key>` — **Arguments**

TYPES OF RESOURCES:

Compute- VMs or server instances, Containers (ECS), Serverless Computing Functions

Networking- VPCs, subnets, routing tables, Traffic Load Balancers, DNS

Storage- Block level storage (eBS), Object storage (S3), file storage

DB- RDBs, NoSQL DBs

Security - IAM, encryption and key management, network security groups

TYPE OF DATA SOURCES:

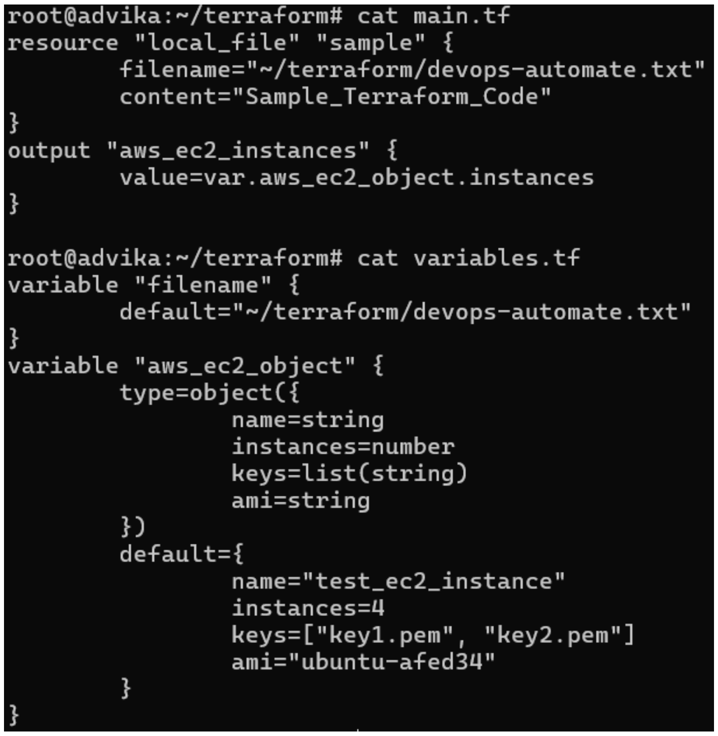
Retrieve info from external sources such as APIs, config files, existing infra

Infrastructure, Cloud Service, DNS, Security, Database

WRITING TERRAFORM CONFIG USING HCL SYNTAX

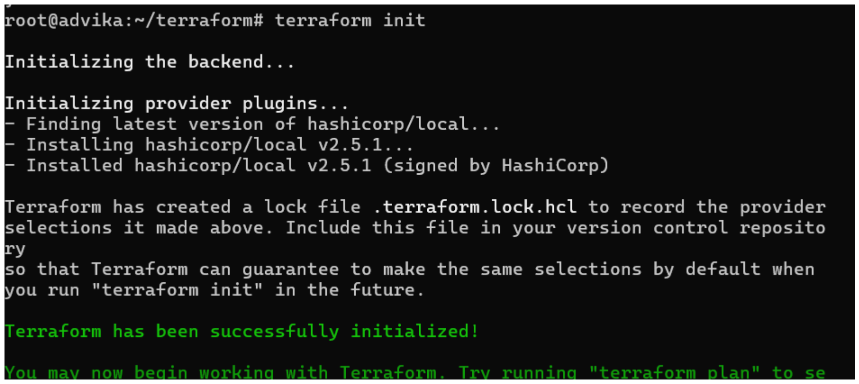
Referenced Blog

Variables are defined in the *variables.tf* config file and resources were defined in *main.tf,* since it is a local file, we specified an output



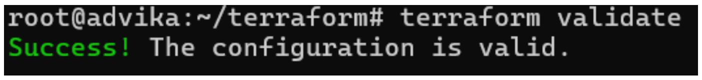
```
root@advika:~/terraform# cat main.tf
resource "local_file" "sample" {
        filename="~/terraform/devops-automate.txt"
        content="Sample_Terraform_Code"
}
output "aws_ec2_instances" {
        value=var.aws_ec2_object.instances
}

root@advika:~/terraform# cat variables.tf
variable "filename" {
        default="~/terraform/devops-automate.txt"
}
variable "aws_ec2_object" {
        type=object({
                name=string
                instances=number
                keys=list(string)
                ami=string
        })
        default={
                name="test_ec2_instance"
                instances=4
                keys=["key1.pem", "key2.pem"]
                ami="ubuntu-afed34"
        }
}
```

- terraform init : initialised the WD to download the necessary provider plugins



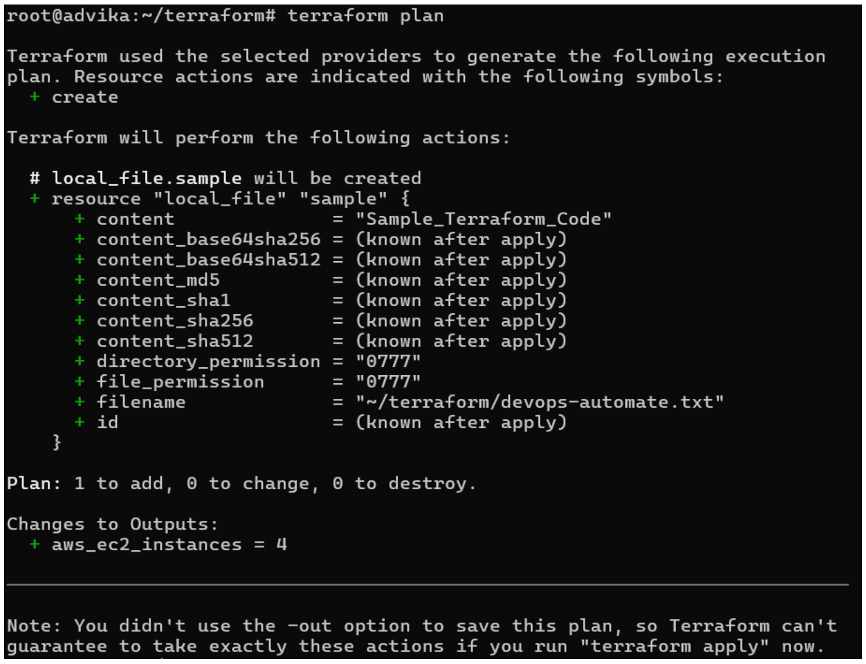
```
root@advika:~/terraform# terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.5.1...
- Installed hashicorp/local v2.5.1 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control reposito
ry
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to se
```

- terraform validate: validates the config files that were just initialised, debugs the syntax



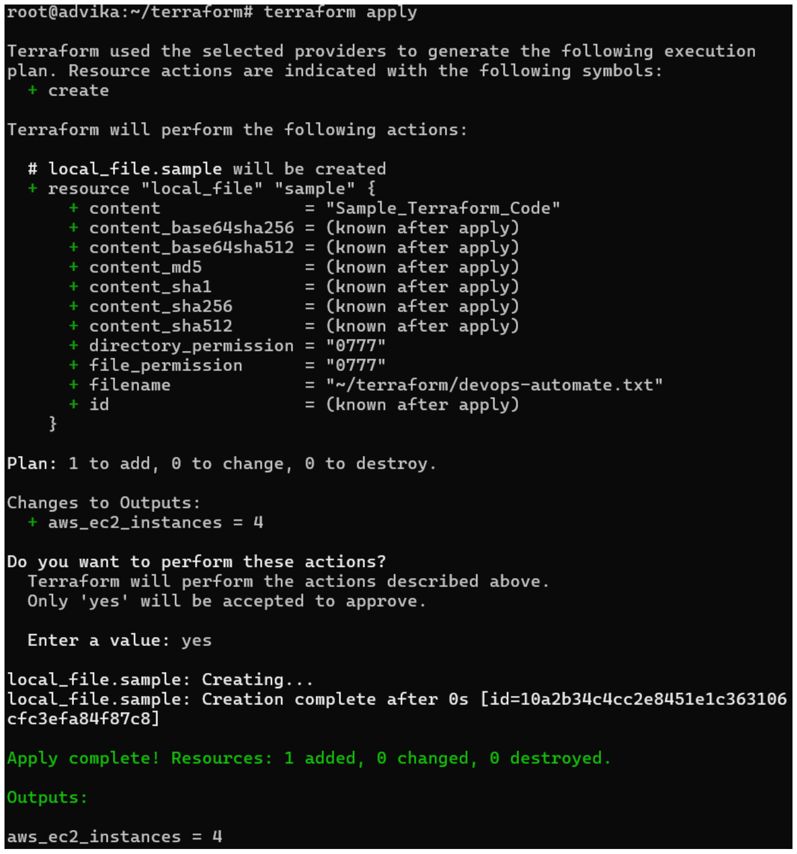
```
root@advika:~/terraform# terraform validate
Success! The configuration is valid.
```

- terraform plan: creates an execution plan, shows actions terraform will take to achieve the desired state defined in the config files



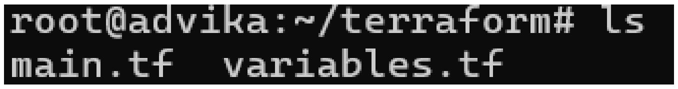
```
root@advika:~/terraform# terraform plan

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # local_file.sample will be created
  + resource "local_file" "sample" {
      + content              = "Sample_Terraform_Code"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "~/terraform/devops-automate.txt"
      + id                   = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + aws_ec2_instances = 4

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
```

- terraform apply: applies the changes reqd to reach the desired state



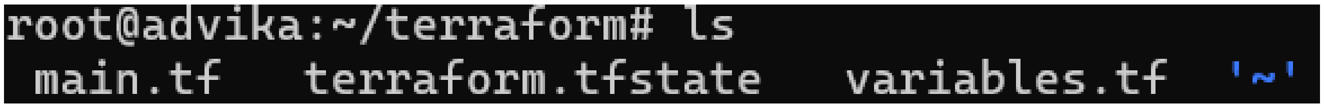
```
root@advika:~/terraform# terraform apply

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # local_file.sample will be created
  + resource "local_file" "sample" {
      + content              = "Sample_Terraform_Code"
      + content_base64sha256 = (known after apply)
      + content_base64sha512 = (known after apply)
      + content_md5          = (known after apply)
      + content_sha1         = (known after apply)
      + content_sha256       = (known after apply)
      + content_sha512       = (known after apply)
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "~/terraform/devops-automate.txt"
      + id                   = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + aws_ec2_instances = 4

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

local_file.sample: Creating...
local_file.sample: Creation complete after 0s [id=10a2b34c4cc2e8451e1c363106
cfc3efa84f87c8]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

aws_ec2_instances = 4
```

Before applying the changes:



```
root@advika:~/terraform# ls
main.tf  variables.tf
```

After applying:



```
root@advika:~/terraform# ls
 main.tf   terraform.tfstate   variables.tf  '~'
```