Objectives

- Introduction to IaC
- Types of IaC Tools
- Why Terraform?
- HCL Basics
- Provision, Update & Destroy

Infrastructure as Code

Configuration Management

- Ansible
- SaltStack
- Puppet

Server Templting

- docker
- Packer
- Vagrant

Provisioning Tools

- Terraform
- CloudFormation

-

Why Terraform?

https://developer.hashicorp.com/terraform/intro

Automate creation, management and tracking of Infrastructure in a collaborative environment.

Terraform comes with rich set of plugins called **providers,** which enables it to connect to remote system/infrastructure.

https://registry.terraform.io

HCL

HCL is the syntactical specification of Terraform language, which is a rich language designed to be relatively easy for humans to read and write. The constructs in the Terraform language can also be expressed in JSON syntax.

HCL is also used by configuration languages in other applications, and in particular other HashiCorp products.

https://developer.hashicorp.com/terraform/language/syntax/configuration

https://github.com/hashicorp/hcl/blob/main/hclsyntax/spec.md

Block

```
resource "aws_instance" "example" {
  ami = "abc123"

  network_interface {
    # ...
  }
}
```

Terraform Commads

init	Prepare your working directory for other commands
plan	Show changes required by the current configuration
apply	Create or update infrastructure

To list all terraform commands, fetch the help manual

terraform -h

Filename	Purpose
main.tf	Main configuration file containing resource definition
variables.tf	Contains variable declarations
outputs.tf	Contains outputs from resources
provider.tf	Contains Provider definition

Variable Type

- **string**
- **number**
- **bool**
- **any**
- **list**
- **map**
- **set**
- **object**
- **tuple**

List

```
variable "prefix" {
default = ["Mr", "Mrs", "Sir"]
type = list   0       1       2
}
```

Map

```
variable file-content {
  type = map
 default = {
    "statement1" = "We love pets!"
     "statement2" = “We love animals!"
 }
}
```

Set

```
variable "prefix" {
 default = ["Mr", "Mrs", "Sir"]
 type = set(string)
}
```

Objects

```
variable "cindrella" {
 type = object({
        name = string
        color = string
        age = number
        food = list(string)
        favorite_pet = bool
  })
 default = {
        name = "cindrella"
        color = "brown"
        age = 7
```

```
    food = ["fish", "chicken", "turkey"]
    favorite_pet = true
  }
}
```

Tuples

```
variable kitty {
    type = tuple([string, number, bool])
    default = ["cat", 7, true]
}
```

Set Variable values at runtime

1. Don’t set default values in variables.tf file
2. Specify the values with `terraform apply` command as below:

   terraform apply -var "filename=/root/pets.txt" -var "content=We love Pets!" -var "prefix=Mrs" -var "separator=." -var "length=2"
3. *Set the vlues using environment variables:*

   $ export TF_VAR_filename="/root/pets.txt"

   $ export TF_VAR_content="We love pets!"

   $ export TF_VAR_prefix="Mrs"

   $ export TF_VAR_separator="."

   $ export TF_VAR_length="2"

   $ terraform apply
4. *Set the values in terraform.tfvars files:*

   filename = "/root/pets.txt"

   content = "We love pets!"

   prefix = "Mrs"

   separator = "."

   length = "2"

Now apply it as below:

$ terraform apply -var-file variables.tfvars

Variable files with below name are automatically loaded:

terraform.tfvars | terraform.tfvars.json

**.auto.tfvars | *.auto.tfvars.json*

Variable Precedence

*0 *.tf variable block*

1 Environment Variables

2 terraform.tfvars

*3 *.auto.tfvars (alphabetical order)*

4 -var or –var-file (command-line flags)

Lifecycle Rules

- create_before_destroy
- prevent_destroy
- ignore_changes

Data Source

```
data <prvider-resource> <resource-name> {
  # arguements
}
```

State

$ terraform state list

$ terraform state show <resorce>

Provisioner

https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax

Taint

https://developer.hashicorp.com/terraform/cli/commands/taint

Debugging

Log Level

- Info
- Warning
- Error
- Debug
- Trace

Variables

- TF_LOG
- TF_LOG_PATH

Imports

$ terraform import <resource_type>.<resource-name> id

Modules

module "dev-webserver" {

source = "../aws-instance"

}

Advantages

- Simpler Configuration Files
- Lower Risk
- Re-Usability
- Standardized Configuration

https://registry.terraform.io/browse/modules

Functions

- Numeric Functions
- String Functions
- Collection Functions
- Type Conversion Functions

Numeric Functions

- Max
- min
- ceil
- floor

String Functions

- Split
- lower
- upper
- title
- substr
- join

Collection Functions

- Length

- index
- element
- contains

Map Functions

- Keys
- values
- lookup

Operators & Conditional Expressions

- Numeric Operators → **+, -, *, /**
- Equality Operator → **==, !=**
- Comparison Operator → **>, >=, <, <=**
- Logical Operator → **&&, ||, !**

condition ? true_val : false_val

Workspace

$ terraform workspace new <workspace-name>

$ terraform workspace list

Provision VM on KVM

https://computingforgeeks.com/how-to-provision-vms-on-kvm-with-terraform/