Abdullah DevOps

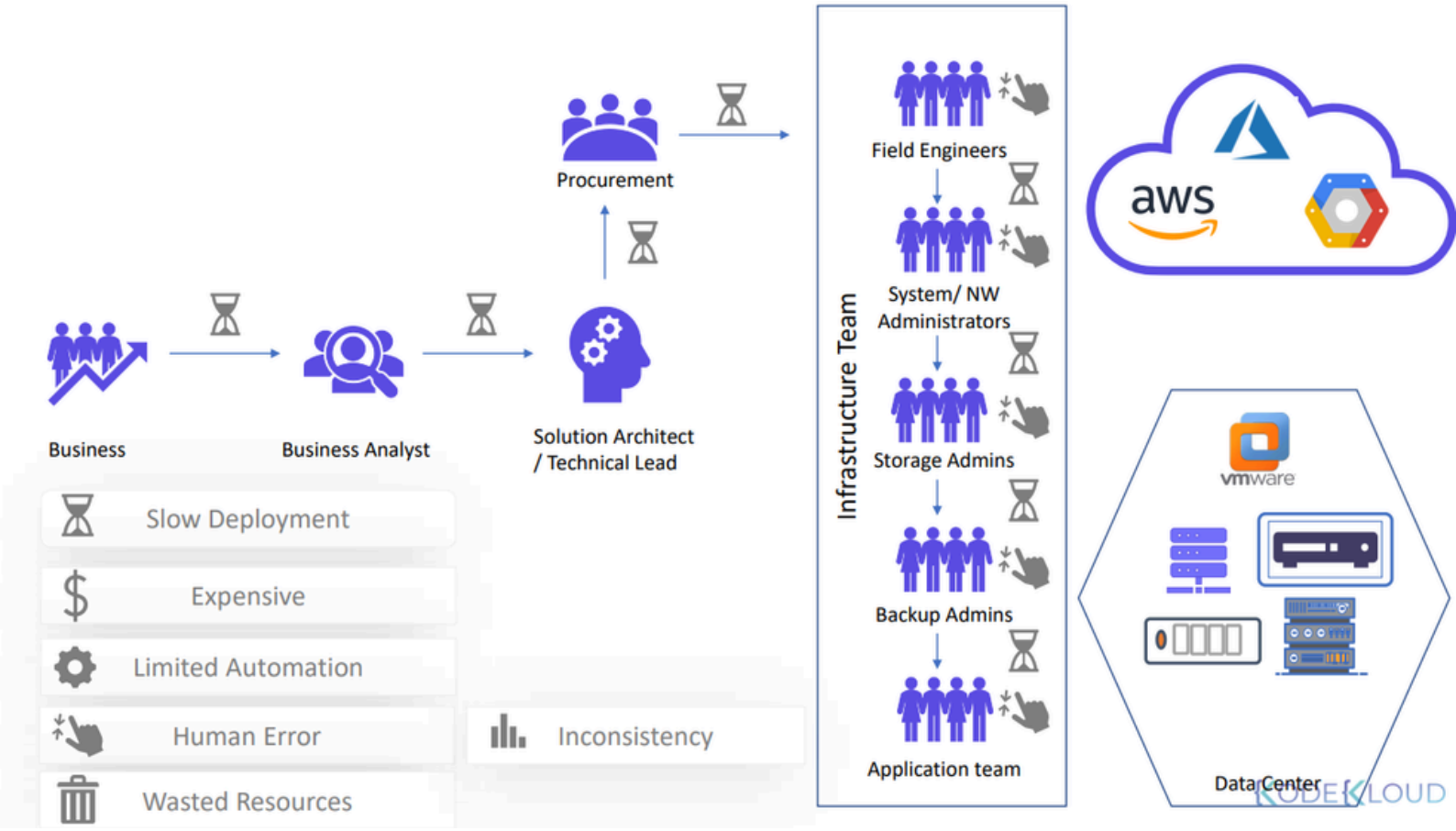
TERRAFORM COURSE

2025

Module# 1
(Introduction)

Course Objective

Challenges with Traditional IT infrastructure



Types of IAC Tools

Configuration Management

ANSIBLE

puppet

SALTSTACK

Designed to Install and Manage Software

Maintains Standard Structure

Version Control

Idempotent

Provisioning Tools

Terraform

CloudFormation

Deploy Immutable Infrastructure resources

Servers, Databases, Network Components etc.

Multiple Providers

Server Templating

docker

Packer

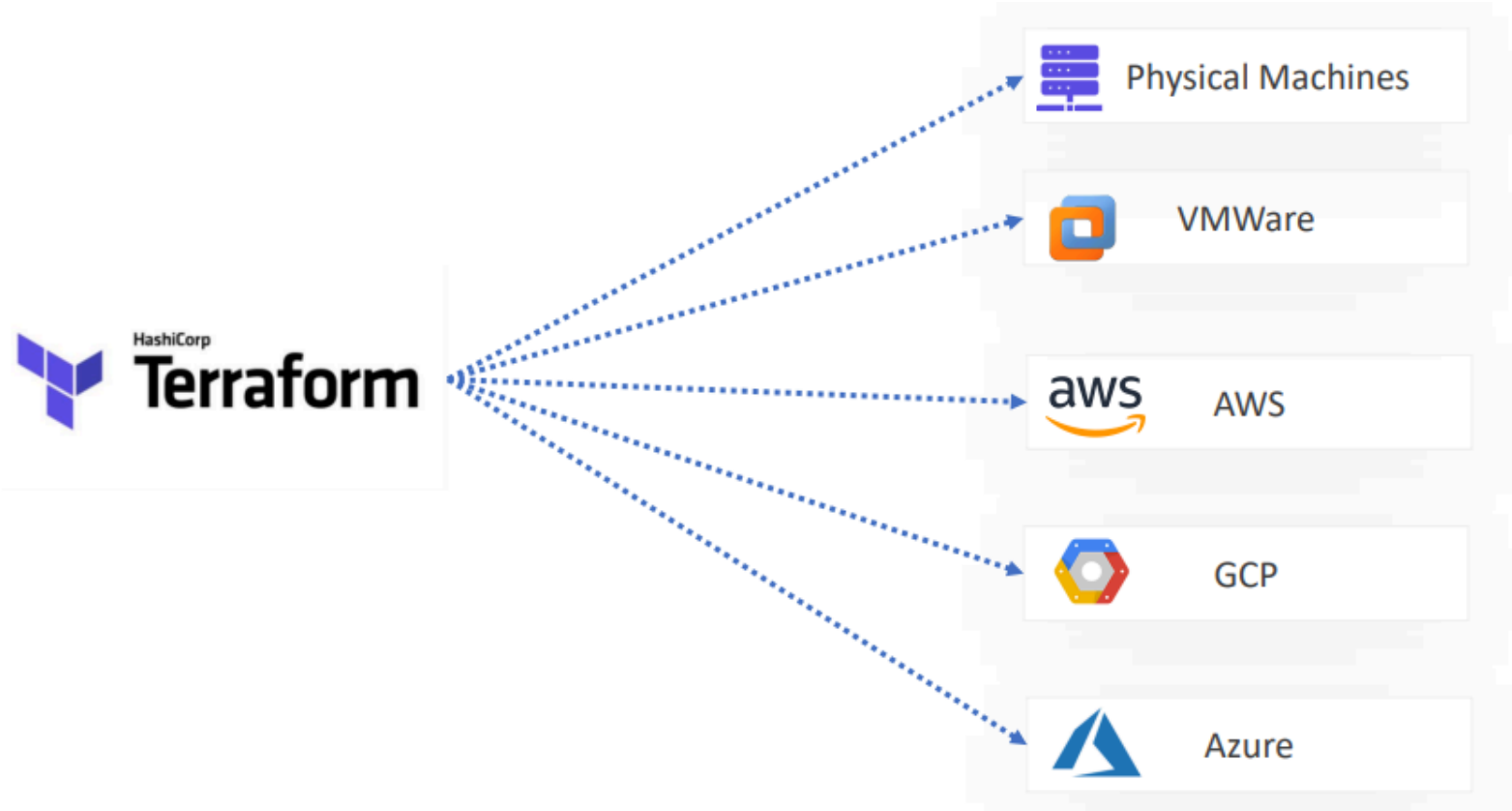
Vagrant

Pre Installed Software and Dependencies

Virtual Machine or Docker Images

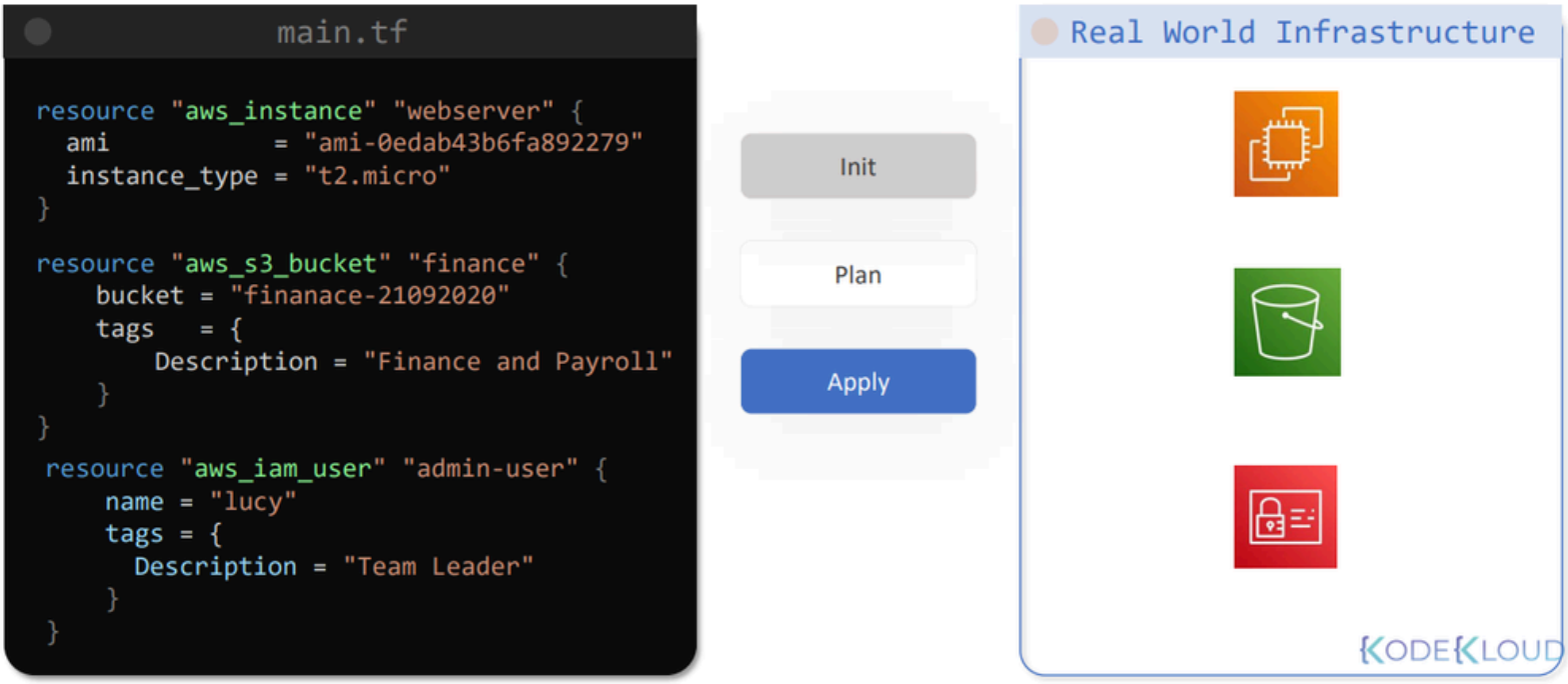
Immutable Infrastructure

Why Terraform?



Declarative

Terraform is declarative means **you define what the desired state of your instance should look** like instead of writing code in step by step detailed instruction.

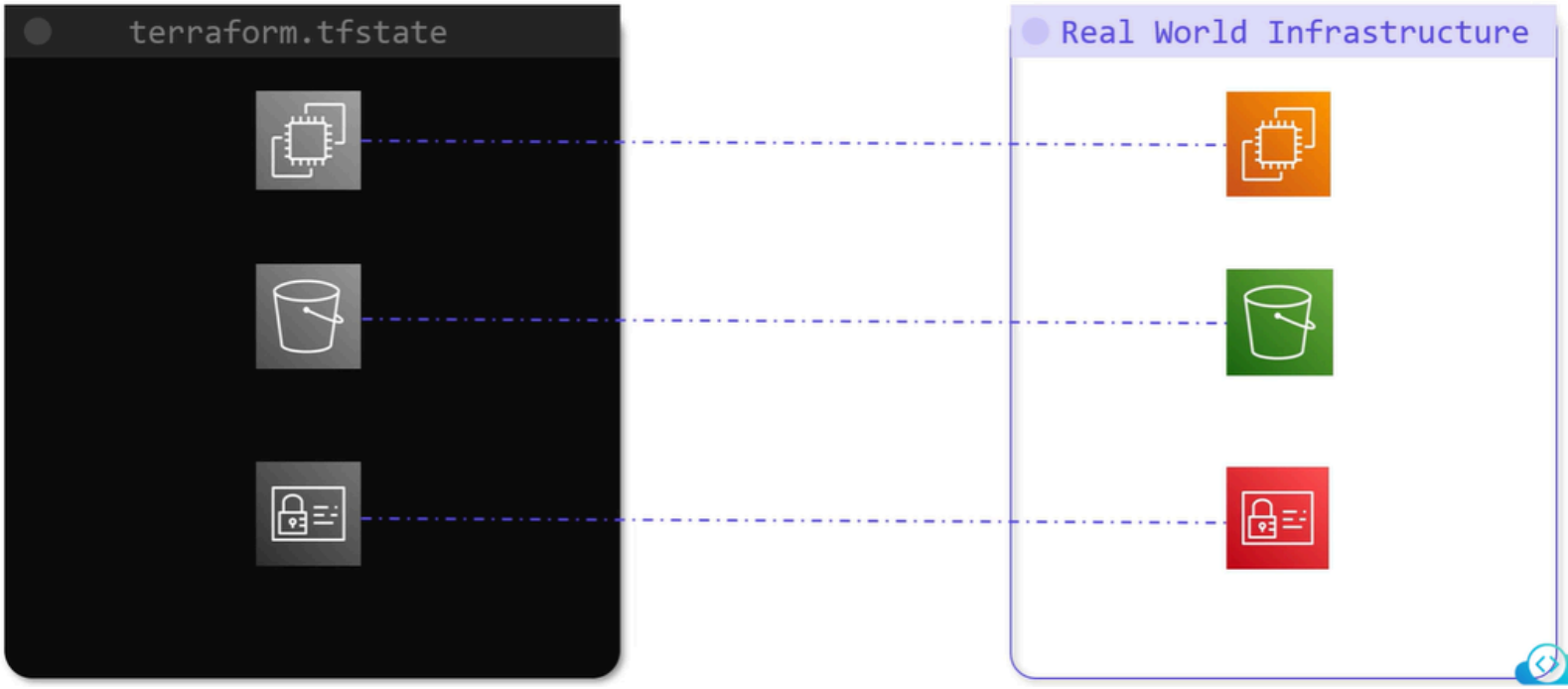


Resource

Everything that terraform manages is called **terraform resource**. It includes anything managed by terraform like S3, EC2 on-prem.

Terraform State

It is the blueprint of infrastructure deployed by Terraform. Terraform manages lifecycle of resources like provisioning or configuration. It records states of infrastructure running in real time and based on this it knows that which to change on updating the infra.



Terraform Import

Terraform can read attribute of other system components. It can import other resources that were not created by terraform & bring them under its management for future.

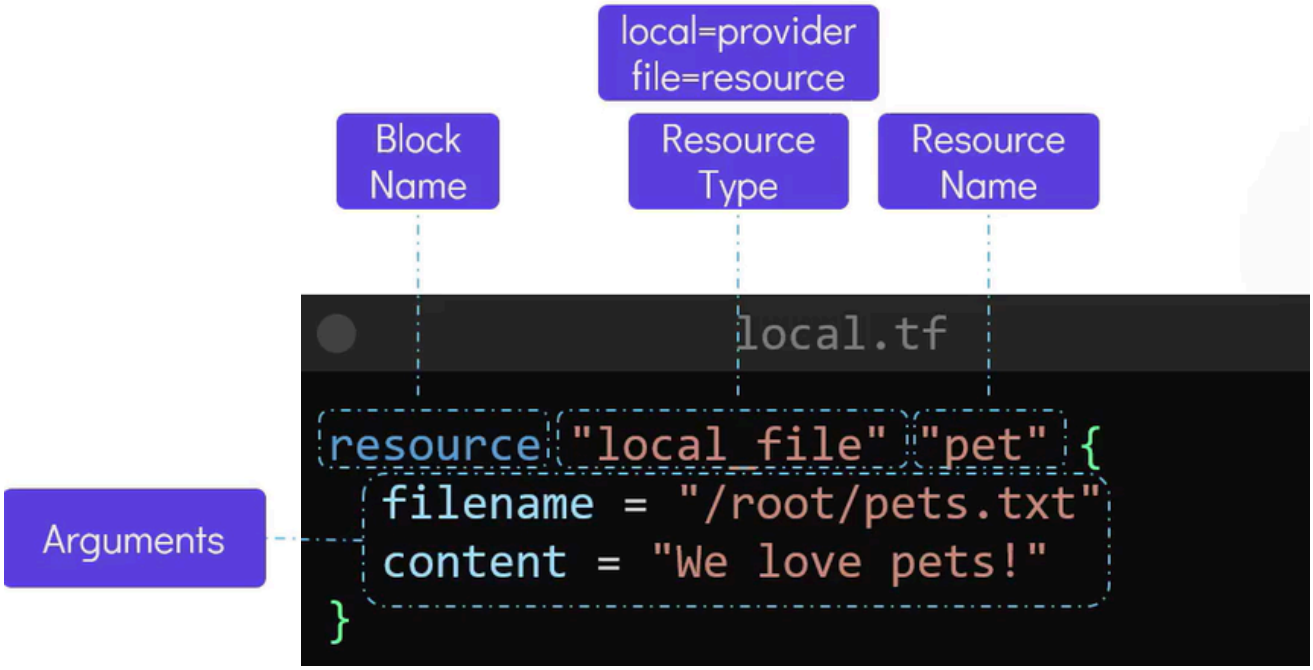
Installing Terraform

```
$ wget https://releases.hashicorp.com/terraform/0.13.0/terraform_0.13.0_linux_amd64.zip
$ unzip terraform_0.13.0_linux_amd64.zip
$ mv terraform /usr/local/bin
$ terraform version
  Terraform v0.13.0
```

HCL Basics

```
>_
$ mkdir /root/terraform-local-file
$ cd /root/terraform-local-file
```

```
<block> <parameters> {
    key1 = value1
    key2 = value2
}
```



Sample file for EC2

```
resource "aws_instance" "webserver" {
    ami = "ami-0c2f25c1f66a1ff4d"
    instance_type = "t2.micro"
}
```

Sample file for S3

```
resource "aws_s3_bucket" "data" {
    bucket = "webserver-bucket-org-2207"
    acl    = "private"
}
```

Terraform Workflow

Init: Checks the configuration file and initialize working directory.

Apply: Prints the execution plan and confirms the user exwcution

Plan: Show the action carried out by terraform to create resource

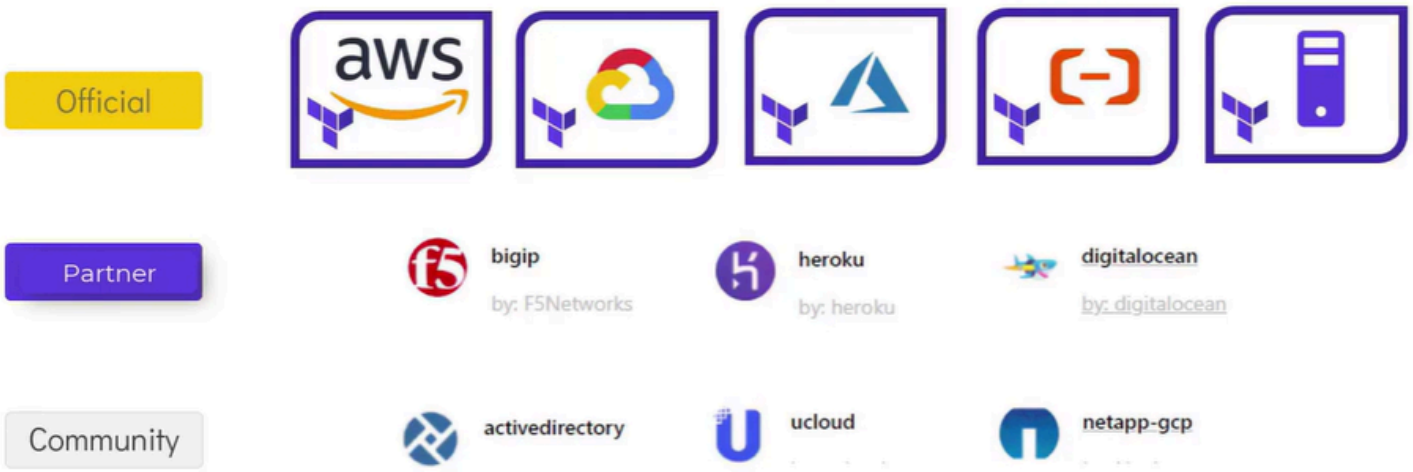
Update & Destroy Infrastructure

Terraform destroy command is use to delete resources.
When the command is executed it shows the deletion plan and shows “-” symbol before the resources that will be deleted.

Module# 2
(Terraform Basics)

Terraform Providers

Provider is a plugin for Terraform that offers a collection of resource types.



Plugins are installed when we run "**terraform init**".
Plugin are saved in **.terraform** file under the project directory.

Terraform Providers

Directory where all the terraform configuration files (.tf) stored.
i.e: you can store multiple files like cat.tf, dog.tf etc.
You can also store all the configuration blocks in a single file.

main.tf

```
resource "local_file" "pet" {
   filename = "/root/pets.txt"
   content = "We love pets!"
}

resource "local_file"  "cat" {
   filename = "/root/cat.txt"
   content = "My favorite pet is Mr. Whiskers"

}
```

File Name	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

Multiple Providers

Terraform can include multiple providers in a single configuration (.tf) file.

```
resource "local_file" "pet" {
    filename = "/root/pets.txt"
    content = "We love pets!"
}


resource "random_pet" "my-pet" {
    prefix = "Mrs"
    separator = "."
    length = "1"
}
```

Input Variables

Input variable are used to assign value to variable.
For this we have to assign variable in different configuration file (variable.tf).

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content = var.content

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```

variables.tf

```
variable "filename" {
        default = "/root/pets.txt"
}
variable "content" {
        default = "We love pets!"
}
variable "prefix" {
        default = "Mrs"
}
variable "separator" {
        default = "."
}
variable "length" {
        default = "1"
}
```

Understanding Variables

There are 3 parameters of variable.

- Default
- Type
- Description

```
variable "filename" {
   default = "/root/pets.txt"
   type = string
   description = "the path of local file"

}
```

Type	Example
string	"/root/pets.txt"
number	1
bool	true/false
any	Default Value
list	["cat", "dog"]
map	pet1 = cat pet2 = dog
object	Complex Data Structure
tuple	Complex Data Structure

Type variable are optional but when mentioned, terraform enforces to use it. If not used type parameter it uses **any** by default.

List Variable

variables.tf

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

maint.tf

```
resource "random_pet" "my-pet" {
  prefix    = var.prefix[0]
}
```

Map Variable

variables.tf

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

maint.tf

```
resource local_file my-pet {
  filename  = "/root/pets.txt"
  content = var.file-content["statement2"]
}
```

List of a Type

variables.tf

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

variables.tf

```
variable "prefix" {
  default = [  1 , 2 , 3  ]
  type = list(number)
}
```

Map of a Type

variables.tf

```
variable "cats" {
  default = {
    "color" = "brown"
    "name" = "bella"
  }
  type = map(string)
}
```

variables.tf

```
variable "pet_count" {
  default = {
    "dogs" =  3
    "cats" =  1
    "goldfish" =  2
  }
  type = map(number)
}
```

Set

Set is same as of list only key difference is that sets don't have duplicate values.

variables.tf

```
variable "age" {
  default = [    10 , 12 , 15    ]
  type = set(number)
}
```

variables.tf

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

variables.tf

```
variable "fruit" {
  default = ["apple", "banana"]
  type = set(string)
}
```

Objects

With objects we can create complex data structures using all the variable data types

Key	Example	Type
name	bella	string
color	brown	string
age	7	number
food	["fish", "chicken", "turkey"]	list
favorite_pet	true	bool

variables.tf

```
variable  "bella"  {
  type = object({
      name = string
      color = string
      age = number
      food = list(string)
      favorite_pet = bool
  })

  default = {
      name = "bella"
      color = "brown"
      age = 7
      food = ["fish", "chicken", "turkey"]
      favorite_pet = true
  }
}
```

Tuples

Tuples are similar to list but consists of sequence of elements. The only key difference is that it can use elements of different variable types.

variables.tf

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

}
```

Using Variables

Passing directly

```
$ terraform apply -var "filename=/root/pets.txt" -var "content=We love
Pets!"  -var "prefix=Mrs" -var "separator=." -var "length=2"
```

Use as Env 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
```

Variable Definition file

terraform.tfvars

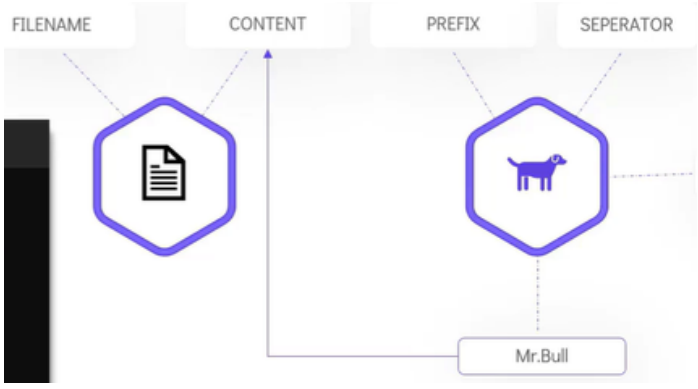
```
filename = "/root/pets.txt"
content = "We love pets!"
prefix = "Mrs"
separator = "."
length = "2"
```

Resource Attributes

Resource attribute is a way of linking two resources.
i.e below the content have the id of other resource.

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content =  "My favorite pet is ${random_pet.my-pet.id}"
}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```



The following attributes are supported:

- `id` - (string) The random pet name

Resource Dependencies

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content =  "My favorite pet is Mr.Cat"
    depends_on = [
      random_pet.my-pet
    ]

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```

Output Variables

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content =  "My favorite pet is ${random_pet.my-pet.id}"
}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}

output  "pet-name" {
  value       = random_pet.my-pet.id
  description = "Record the value of pet ID generated by the
 random_pet resource"
}
```

Sample commands

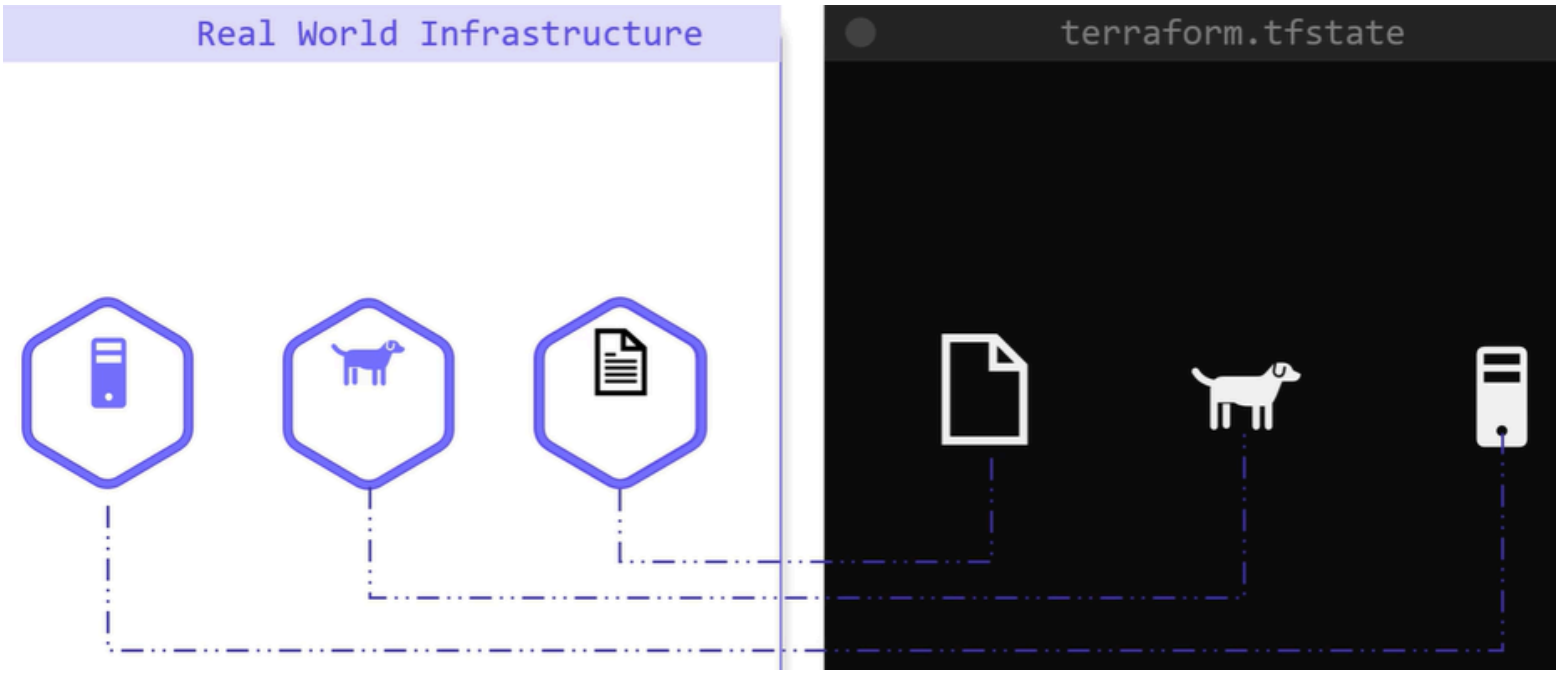
```
$ terraform output
pet-name = Mrs.gibbon
```

```
$ terraform output pet-name
Mrs.gibbon
```

Module# 3
(Terraform State)

Terraform state are the state of process that has been done when we initiate terraform commands.

- **Terraform init:** Use to download necessary plugins.
- **Terraform plan:** Generate output of execution plan.
- **Terraform apply:** Create resources as expected in plan command.



Purpose of State

Terraform state file is the blueprint of all the resources that terraform manages in the real world infrastructure.

- Terraform records the id of resource to identify it.

Terraform State consideration

- Terraform state consists of all the sensitive data of our infra.
- So it is better to store it in a secured storage.

Module# 4
(Working with Terraform)

Terraform Commands

1. **Terraform Validate:** Checks the file has correct syntax or not!
2. **Terraform Format:** Formats the file into proper format for readability.
3. **Terraform Show:** Prints the current state of infrastructure.
4. **Terraform Providers:** List all the providers in configuration directory.
5. **Terraform Output:** Print all the output variable of configuration directory.
6. **Terraform Refresh:** Sync terraform with real world architecture.
7. **Terraform Graph:** Visual representation of dependencies in configuration dir.

```
$ apt install graphviz -y
$ terraform graph | dot -Tsvg > graph.svg
```

Mutable v/s immutable Infrastructure

"**Mutable infrastructure"** refers to a system where existing infrastructure components can be directly modified and updated after deployment, while **"Immutable infrastructure"** means that once deployed, components are never changed and instead are replaced with entirely new versions whenever a modification is needed

Life Cycle Rule Infrastructure

Create before destroy

```
resource "local_file" "pet" {
   filename = "/root/pets.txt"
   content = "We love pets!"
   file_permission = "0700"

   lifecycle  {
      create_before_destroy  = true
   }
}
```

Prevent destroy

```
resource "local_file" "pet" {
   filename = "/root/pets.txt"
   content = "We love pets!"
   file_permission = "0700"

   lifecycle  {
      prevent_destroy  = true
   }
```

Ignore Changes

This will ignore the changes that are being chosen.

For Specific Tags

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
      Name = "ProjectA-Webserver"
  }
  lifecycle {
        ignore_changes = [
          tags,ami

        ]
    }
}
```

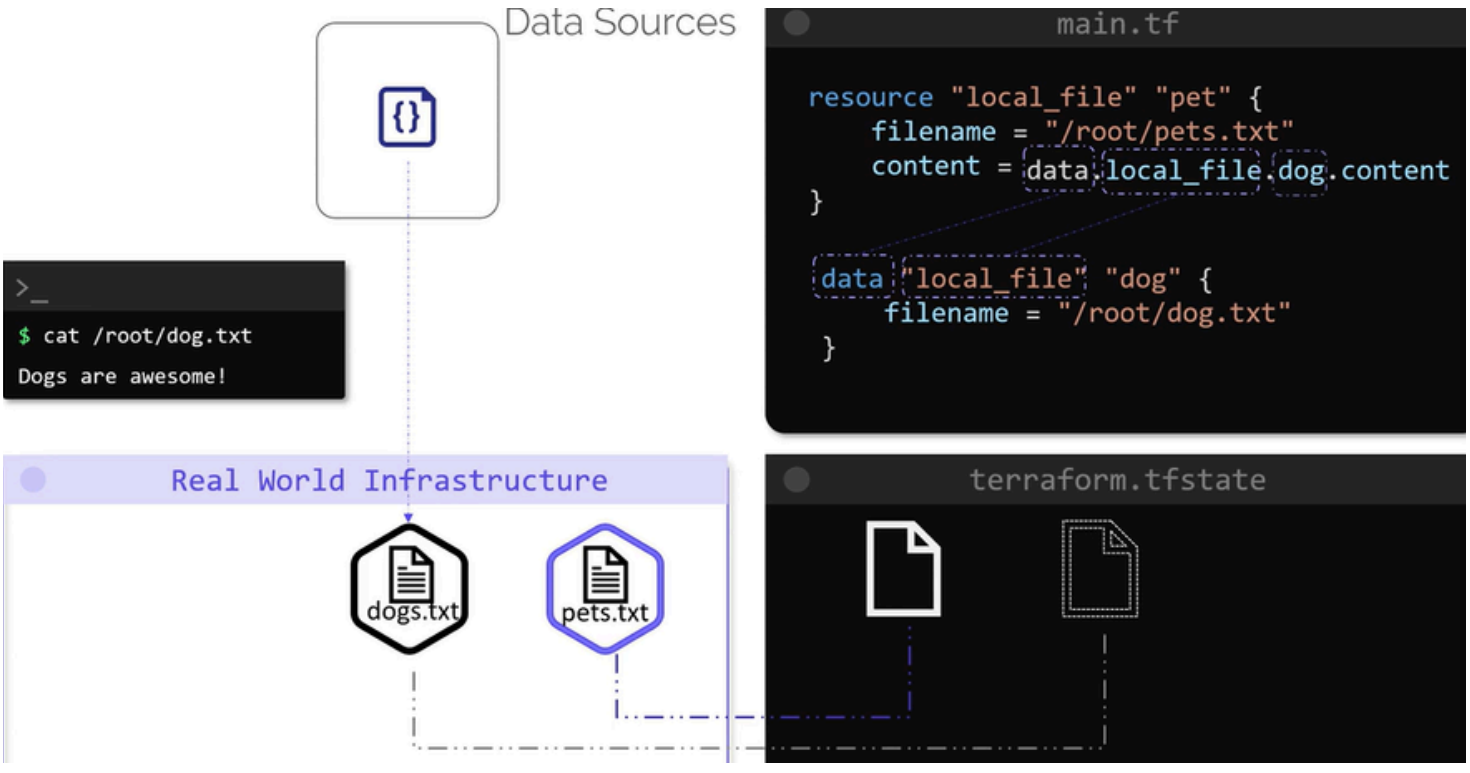
For All Tags

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
      Name = "ProjectA-Webserver"
  }
  lifecycle {
        ignore_changes = all
    }
}
```

Order	Option	
1	create_before_destroy	Create the resource first and then destroy older
2	prevent_destroy	Prevents destroy of a resource
3	ignore_changes	Ignore Changes to Resource Attributes (specific/all)

Data Sources

Data sources allows terraform to read attribute from the resources that are out of terraform control.



Meta Argument

Use in any resource block to change behaviour of resources.
i.e lifecycle rules, depend_on.
Types: Count & For_Each

Count

Use in any resource block to change behaviour of resources.
i.e lifecycle rules, depend_on

main.tf

```
resource "local_file" "pet" {
    filename = var.filename[count.index]
    count    = length(var.filename)
}
```

variables.tf

```
variable "filename" {
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

For Each

Method #1

main.tf

```
resource "local_file" "pet" {
    filename = each.value
    for_each = var.filename
}
```

variables.tf

```
variable "filename" {
  type=set(string)
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

Method #2

main.tf

```
resource "local_file" "pet" {
    filename = each.value
    for_each = toset(var.filename)
}
```

variables.tf

```
variable "filename" {
  type=list(string)
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

The key difference between for_each and count is that whenever count is applied it destroys all the resources but for_each only destroyed the changed resources.

Version Constraints

It is used when we make sure that our terraform file will use a specific version of provider when we run terraform init command

Users of Version Constraints:

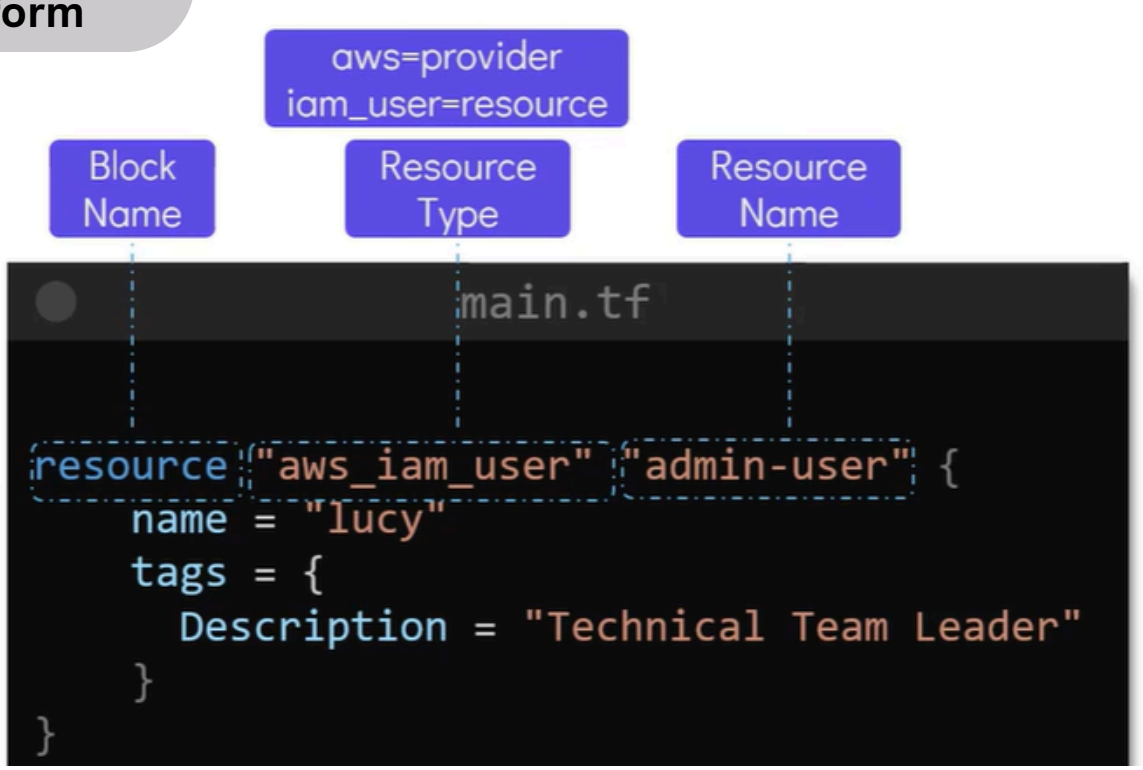
1. Don't use specific version **"!= 2.0"**
2. Use smaller than version **"< 2.0"**
3. Use greater than version **"> 2.0"**
4. **Same or greater version "~> 1.2"**
5.
6. Use multiple operators
   " >1.2, < 2.0, != 1.4"

main.tf

```
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "1.4.0"
    }
  }
}

resource "local_file" "pet" {
  filename     = "/root/pet.txt"
  content  = "We love pets!"
}
```

Module# 5
(Terraform with AWS)

AWS IAM with Terraform



Methods of using IAM

Method #1

```
main.tf
provider "aws" {
    region = "us-west-2"
    access_key = "AKIAI44QH8DHBEXAMPLE"
    secret_key = "je7MtGbClwBF/2tk/h3yCo8n…"
}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

Method #2

```
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

```
.aws/credentials
[default]
aws_access_key_id =
aws_secret_access_key =
```

```
$ export AWS_ACCESS_KEY_ID=AKIAI44QH8DHBEXAMPLE
$ export AWS_SECRET_ACCESS_KEY_ID=je7MtGbClwBF/2tk/h3yCo
$ export AWS_REGION=us-west-2
```

Method #1 of creating IAM with Terraform

Using Policy with EOF parameters.

```
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy =  <<EOF
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "*",
                "Resource": "*"
            }
        ]
    }
    EOF
}

  resource "aws_iam_user_policy_attachment" "lucy-admin-access" {

     user = aws_iam_user.admin-user.name

     policy_arn = aws_iam_policy.adminUser.arn

  }
```

Method #1 of creating IAM with Terraform

Using policy from a different file.

main.tf

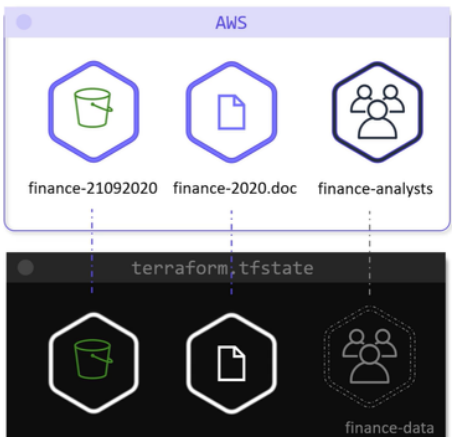
```
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy = file("admin-policy.json")
}

resource "aws_iam_user_policy_attachment" "lucy-admin-access" {

    user = aws_iam_user.admin-user.name

    policy_arn = aws_iam_policy.adminUser.arn

}
```

admin-policy.json

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": "*",
        "Resource": "*"
    }
]
```

AWS S3 with Terraform

```
resource "aws_s3_bucket" "finance" {
    bucket = "finanace-21092020"
    tags   = {
        Description = "Finance and Payroll"
    }
}

resource "aws_s3_bucket_object" "finance-2020" {
    content = "/root/finance/finance-2020.doc"
    key     = "finance-2020.doc"
    bucket = aws_s3_bucket.finance.id
}

data "aws_iam_group" "finance-data" {
  group_name = "finance-analysts"

}
resource "aws_s3_bucket_policy" "finance-policy" {
  bucket = aws_s3_bucket.finance.id
  policy = <<EOF

  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Action": "*",
              "Effect": "Allow",
              "Resource": "arn:aws:s3:::${aws_s3_bucket.finance.id}/*",
              "Principal": {
                  "AWS": [
                      "${data.aws_iam_group.finance-data.arn}"
                  ]
              }
          }
      ]
  }

EOF
}
```



AWS Dynamo DB with Terraform

DynamoDB

- Highly Scalable
- Fully Managed by AWS
- NoSQL Database
- Single-Digit MilliSecond Latency
- Data Replicated across Regions

Manufacturer	Model	Year	VIN
Toyota	Corolla	2004	4Y1SL65848Z411439
Honda	Civic	2017	DY1SL65848Z411432
Dodge	Journey	2014	SD1SL65848Z411443
Ford	F150	2020	DH1SL65848Z41100

main.tf

```
resource "aws_dynamodb_table" "cars" {
  name     = "cars"
  hash_key = "VIN"
  billing_mode = "PAY_PER_REQUEST"
  attribute {
    name = "VIN"
    type = "S"
  }
}
resource "aws_dynamodb_table_item" "car-items" {
  table_name = aws_dynamodb_table.cars.name
  hash_key   = aws_dynamodb_table.cars.hash_key
  item       = <<EOF

  {
      "Manufacturer": {"S": "Toyota"},
      "Make": {"S": "Corolla"},
      "Year": {"N": "2004"},
      "VIN" : {"S": "4Y1SL65848Z411439"},
  }
EOF
}
```

cars

```
{
    "Manufacturer": "Toyota",
    "Make": "Corolla",
    "Year": 2004,
    "VIN" : "4Y1SL65848Z411439"
}
```

```
{
    "Manufacturer": "Honda",
    "Make": "Civic",
    "Year": 2017,
    "VIN" : "DY1SL65848Z411432"
}
```

```
{
    "Manufacturer": "Dodge",
    "Make": "Journey",
    "Year": 2014,
    "VIN" : "SD1SL65848Z411443"
}
```

```
{
    "Manufacturer": "Ford",
    "Make": "F150",
    "Year": 2020,
    "VIN" : "DH1SL65848Z41100"
}
```

Module# 6
(Remote State)

Remote State

Terraform state does

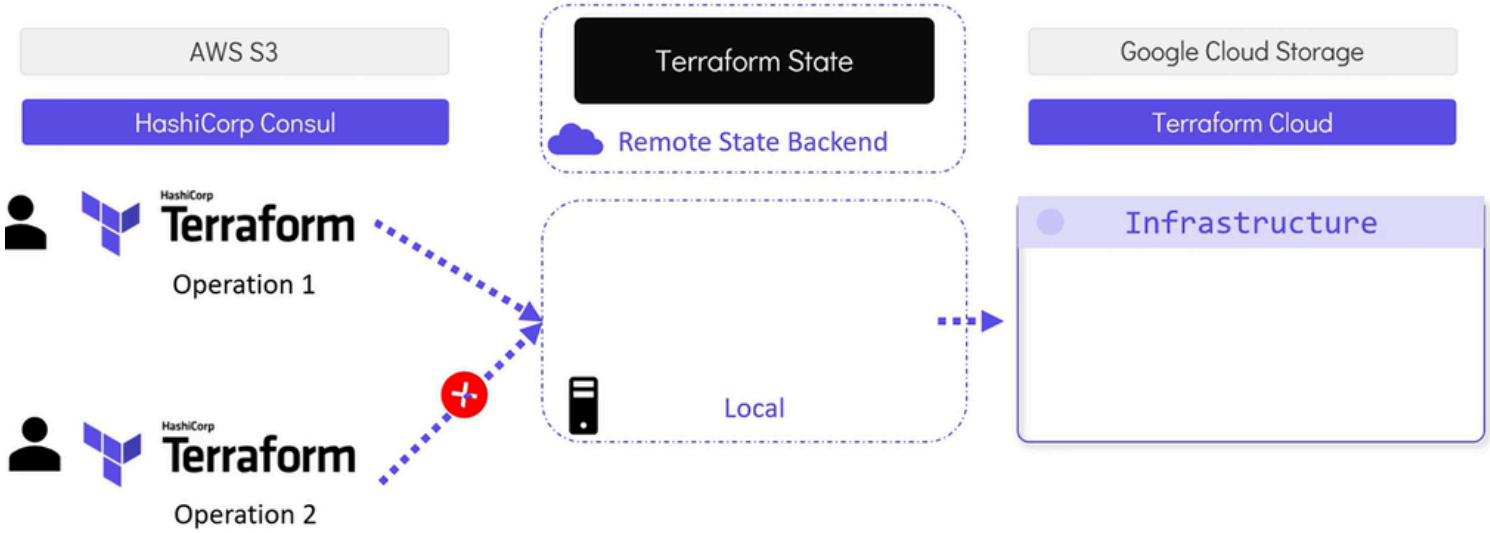
- Mapping configuration to real world.
- Tracking metadata.
- Performance.
- collaborate members

Terraform state file is saved at local machine when terraform apply command is run, but it is not recommended to save this file on SCM.

State Locking

State locking happened when two persons tries to run configuration file at once, or it runs without completing the previous execution.

So in such scenario we save the terraform state at remote backend.
This results in auto load and upload, prevent state locking and security.



Remote Backends with S3

main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}
```

terraform.tf

```
terraform {
  backend "s3" {
    bucket         = "kodekloud-terraform-state-bucket01"
    key            = "finance/terraform.tfstate"
    region         = "us-west-1"
    dynamodb_table = "state-locking"
  }
}
```

Terraform State Commands

Terraform state commands are use to manipulate terraform state.

```
# terraform state <subcommand> [options] [args]
```

Sub-command
list
mv
pull
rm
show

List: Lists all records of terraform state

```
# terraform state list [options] [address]

$ terraform state list
aws_dynamodb_table.cars
aws_s3_bucket.finance-2020922
```

Show: Lists detail of single resource

```
# terraform state show [options] [address]

$ terraform state show aws_s3_bucket.finance-2020922

resource "aws_s3_bucket" "terraform-state" {
    acl                         = "private"
    arn                         = "arn:aws:s3::: finance-2020922 "
    bucket                      = "finance-2020922 "
    bucket_domain_name          = "finance-2020922.s3.amazonaws.com"
    bucket_regional_domain_name = " finance-2020922.s3.us-west-1.amazonaws.com"
    force_destroy               = false
    hosted_zone_id              = "Z2F5ABCDE1ACD"
    id                          = "finance-2020922 "
    region                      = "us-west-1"
    request_payer               = "BucketOwner"
    tags                        = {
        "Descritpion" = "Bucket to store Finance and Payroll Information"
    }
```

Pull: To view the contents of remote state

```
#  terraform state pull [options] SOURCE  DESTINATION

$ terraform state pull
```

Pull with specific resource attribute

```
$ terraform state pull | jq '.resources[] | select(.name  == "state-locking-
db")|.instances[].attributes.hash_key'

"LockID"
```

Move: Use to move items in terraform state file

Note: *You have to manually change resource name in main configuration file.*

```
# terraform state mv [options] SOURCE  DESTINATION

$ terraform state mv aws_dynamodb_table.state-locking aws_dynamodb_table.state-locking-db
Move "aws_dynamodb_table.state-locking" to "aws_dynamodb_table.state-locking-db"
Successfully moved 1 object(s).
```

RM: Use to remove items form terraform state command

```
#  terraform state rm ADDRESS

$ terraform state rm aws_s3_bucket.finance-2020922
```

Module# 7
(Terraform Provisioners)

AWS EC2 with Terraform

provider.tf

```
provider "aws" " {
  region = “us-west-1"
}
```

- ***User Data:*** *Commands to run when EC2 starts.*
- ***KeyPair:*** *Sets keypair for ssh.*
- ***SecurityGroup:*** *Manage firewall/ ports.*
- ***PublicIP:*** *Prints public IP*

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name        = "webserver"
    Description = "An Nginx WebServer on Ubuntu"
  }
  user_data = <<-EOF
              #!/bin/bash
              sudo apt update
              sudo apt install nginx -y
              systemctl enable nginx
              systemctl start nginx
              EOF
}

 key_name  = aws_key_pair.web.id
 vpc_security_group_ids = [ aws_security_group.ssh-access.id ]

}
resource "aws_key_pair" "web" {
    public_key = file("/root/.ssh/web.pub")
}
resource "aws_security_group" "ssh-access" {
  name        = "ssh-access"
  description = "AllowSSH access from the Internet"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
 output publicip {
   value       = aws_instance.webserver.public_ip
 }
```

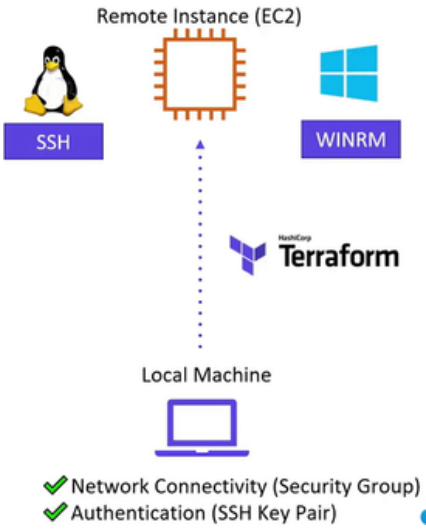
Terraform Provisioners

Terraform provisioners way us to carry out tasks or scripts on remote resources or locally on the machine where terraform is installed.

Remote Exec: *Used to run bash script after the resource is created.*

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "remote-exec" {
      inline = [ "sudo apt update",
                 "sudo apt install nginx -y",
                 "sudo systemctl enable nginx",
                 "sudo systemctl start nginx",
               ]
    }
  connection {
     type        = "ssh"
     host        = self.public_ip
     user        = "ubuntu"
     private_key = file("/root/.ssh/web")
   }
  key_name   = aws_key_pair.web.id
  vpc_security_group_ids = [ aws_security_group.ssh-access.id ]
}
 resource "aws_security_group" "ssh-access" {
      << code hidden >>
 }
 resource "aws_key_pair" "web" {
      << code hidden >>
 }
```

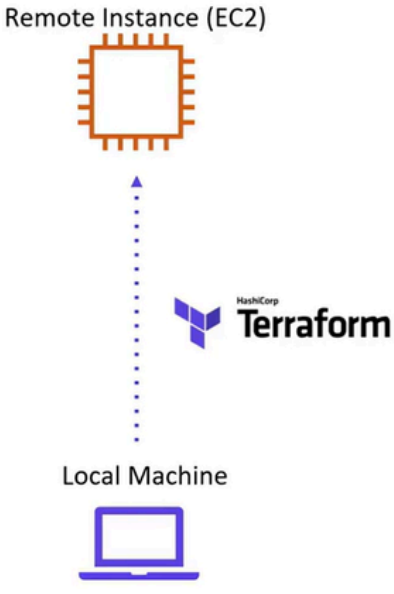


Local Exec: *Used to run bash script from the local machine.*

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"

  provisioner "local-exec" {
    command = "echo ${aws_instance.webserver .public_ip} >> /tmp/ips.txt"
   }

}
```

```
>_
$ cat /tmp/ips.txt
54.214.68.27
```



Failure Behavior: *Destroy resource if failed*

main.tf

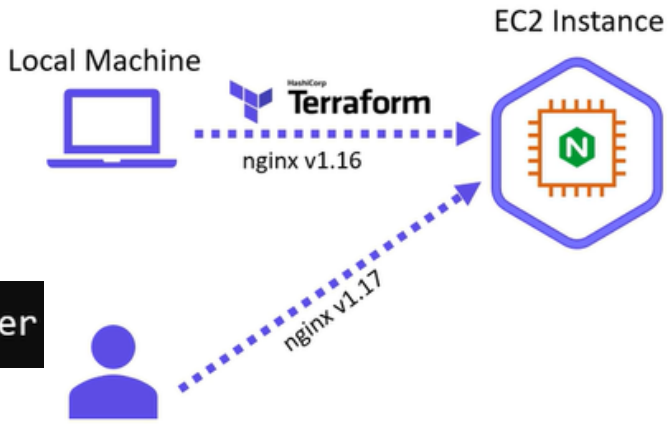
```
resource "aws_instance" "webserver" {
  ami                    = "ami-0edab43b6fa892279"
  instance_type          = "t2.micro"
  provisioner "local-exec" {
     on_failure = fail
     command = "echo Instance ${aws_instance.webserver.public_ip} Created! > /temp/instance_state.txt"
  }

  provisioner "local-exec" {
     when    = destroy
     command = "echo Instance ${aws_instance.webserver.public_ip} Destroyed! > /tmp/instance_state.txt"
   }
}
```

Module# 8
(Terraform Import, Taint & Debug)

Taint

Whenever the resource creation is failed due to any reason it will be marked as tainted by terraform.
You can see this when you run terraform plan command.

*You can **taint** a resource using taint command, it is useful if you have applied some manual changes to your resource and then you want to revert it.*




```
$ terraform taint aws_instance.webserver
```

*You can **untaint** a resource using untaint command, so terraform dont try to recreate it on terraform apply.*

```
$ terraform untaint aws_instance.webserver
```

Debugging

For debugging purpose we have to check logs.

Logs Level

Enable Logs

```
>_
# export TF_LOG=<log_level>
$ export TF_LOG=TRACE
```

Save logs in file

```
$ export TF_LOG_PATH=/tmp/terraform.log


$ head -10 /tmp/terraform.logs
----
2020/10/18 22:08:30 [INFO] Terraform version: 0.13.0
2020/10/18 22:08:30 [INFO] Go runtime version: go1.14.2
2020/10/18 22:08:30 [INFO] CLI args: []string{"C:\\Windows\\system32\\terraform.exe",
"plan"}
```

Disable Logs

```
$ unset TF_LOG_PATH
```

Terraform Import

Terraform import is use to import resources completely into the managment and operation of terraform

```
# terraform import  <resource_type>.<resource_name> <attribute>
$ terraform import aws_instance.webserver-2 i-026e13be10d5326f7

Error: resource address "aws_instance.webserver-2" does not exist
in the configuration.

Before importing this resource, please create its configuration in
the root module. For example:
```

At first it shows error, becases configuration file is not updated by import command. You have to write an empty block in config file for resource

main.tf

```
resource "aws_instance" "webserver-2" {
  # (resource arguments)
}
```

Now check the terraform state file and check for the configuration and add that accordingly in resource block.

terraform.tfstate

```
{
    "mode": "managed",
    "type": "aws_instance",
    "name": "webserver-2",
    "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
    "instances": [
      {
        "schema_version": 1,
        "attributes": {
          "ami": "ami-0edab43b6fa892279",
          "instance_state": "running",
          "instance_type": "t2.micro",
          "key_name": "ws",
          .
          "tags": {
            "Name": "old-ec2"
          },
          .
          .
          "vpc_security_group_ids": [
```

main.tf

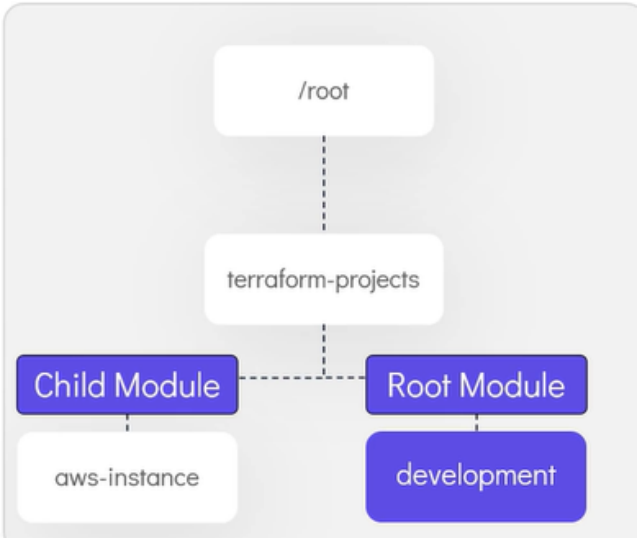
```
resource "aws_instance" "webserver-2" {
    ami                    = "ami-0edab43b6fa892279"
    instance_type          = "t2.micro"
    key_name               = "ws"
    vpc_security_group_ids = ["sg-8064fdee"]

}
```

Module# 9
(Terraform Modules)

Terraform Import

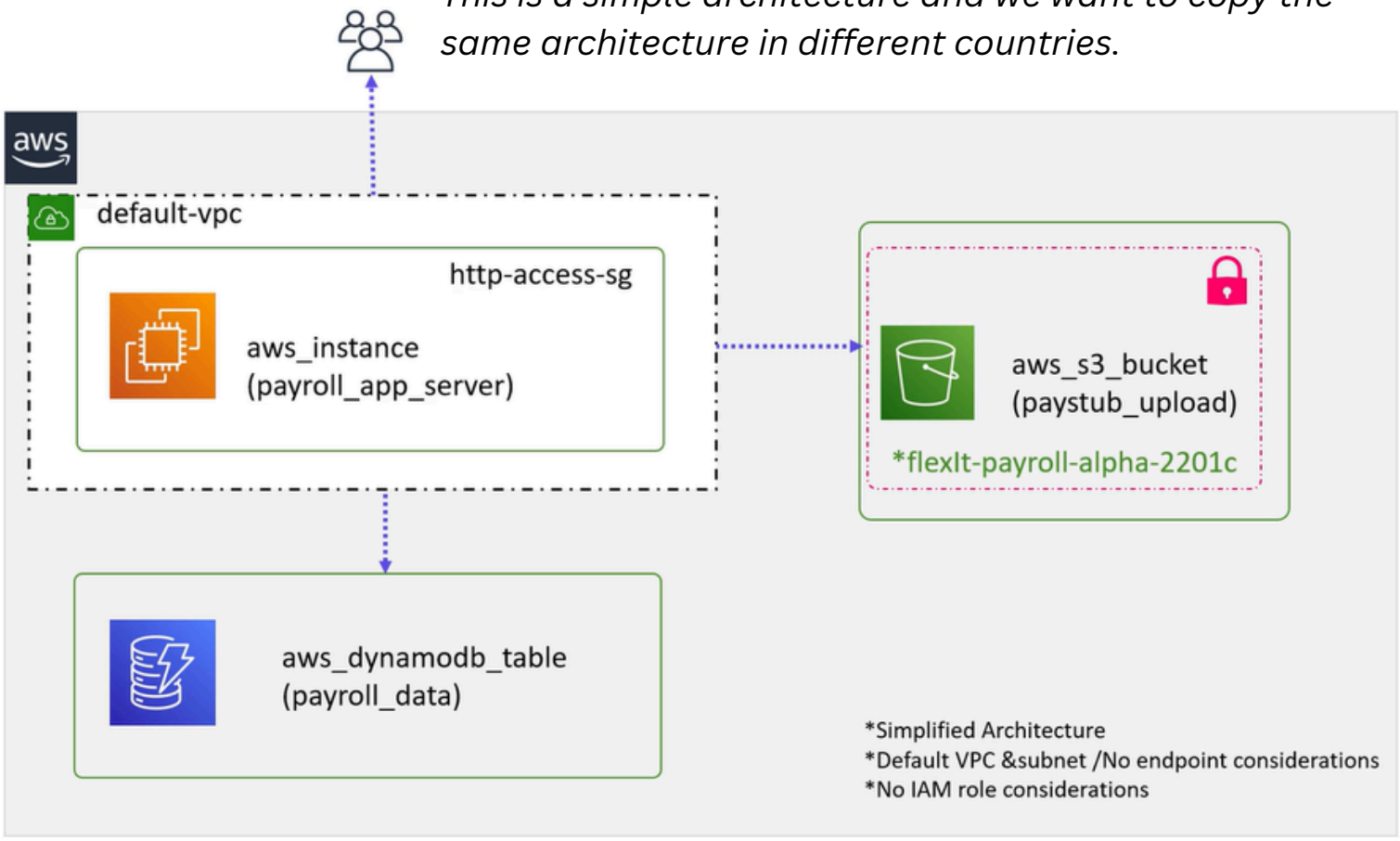
Terraform module is a set of terraform configuration files in a single directory.

```
module "dev-webserver" {
  source = "../aws-instance"
}
```

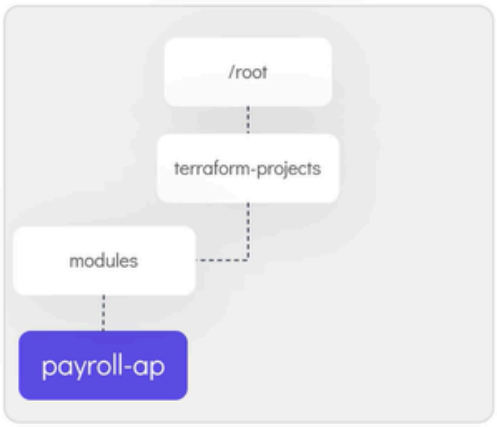


Creating & Using module

This is a simple architecture and we want to copy the same architecture in different countries.




```
$ mkdir /root/terraform-projects/modules/payroll-app

app_server.tf dynamodb_table.tf s3_bucket.tf variables.tf
```



app_server.tf

```
resource "aws_instance" "app_server" {
  ami           = var.ami
  instance_type =  "t2.medium"
  tags = {
    Name = "${var.app_region}-app-server"
  }
  depends_on = [ aws_dynamodb_table.payroll_db,
                 aws_s3_bucket.payroll_data
               ]
}
```

dynamodb_table.tf

```
resource "aws_dynamodb_table" "payroll_db" {
  name           = "user_data"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "EmployeeID"

  attribute {
    name = "EmployeeID"
    type = "N"
  }
}
```

s3_bucket.tf

```
resource  "aws_s3_bucket"  "payroll_data" {
  bucket = "${var.app_region}-${var.bucket}"
}
```

variables.tf

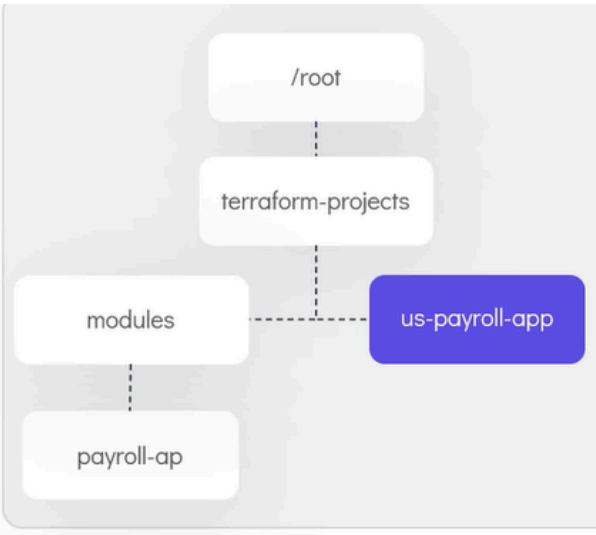
```
variable  "app_region" {
    type = string
}
variable "bucket" {
    default = "flexit-payroll-alpha-22001c"
}
variable "ami" {
    type = string
}
```

*Create infrastructure for **US-Region***

```
$ mkdir /root/terraform-projects/us-payroll-app

main.tf provider.tf
```

main.tf

```
module  "us_payroll" {
  source = "../modules/payroll-app"
  app_region = "us-east-1"
  ami        = "ami-24e140119877avm"
}
```



*Create infrastructure for **UK-Region***

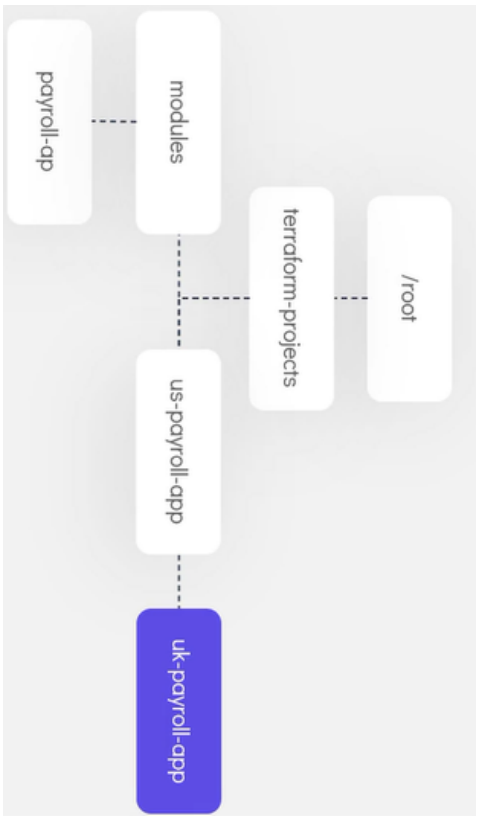
```
$ mkdir /root/terraform-projects/uk-payroll-app

main.tf provider.tf
```

main.tf

```
module  "uk_payroll" {
  source = "../modules/payroll-app"
  app_region = "eu-west-2"
  ami        = "ami-35e140119877avm"
}
```

provider.tf

```
provider "aws" {
  region    = "eu-west-2"
}
```



Benefits:

- *Reusability.*
- *Low risk (less human error).*
- *Simpler configuration files.*
- *Standardize configuration.*

The above is terraform local module, but incase we want to get remote modules here are some example for this.
You have to mention the module version or it will pull latest.

main.tf

```
module "security-group_ssh" {
  source  = "terraform-aws-modules/security-group/aws/modules/ssh"
  version = "3.16.0"
  # insert the 2 required variables here
  vpc_id = "vpc-7d8d215"
  ingress_cidr_blocks = [ "10.10.0.0/16"]
  name = "ssh-access"

}
```

Module# 10
(Terraform Functions)

Make Terraform Function

We have use some functions so far like length, file, list to set

main.tf

```
resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy = file("admin-policy.json")
}

resource "local_file" "pet" {
    filename = var.filename
    count = length(var.filename)
}
```

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    for_each = toset(var.region)
}

variable region {
  type        = list
  default     = ["us-east-1",
                 "us-east-1",
                 "ca-central-1"]
  description = "A list of AWS Regions"
}
```

```
$ terraform console
>file("/root/terraform-projects/main.tf)
 resource "aws_instance" "development" {
   ami           = "ami-0edab43b6fa892279"
   instance_type = "t2.micro"
 }
> length(var.region)
3
> toset(var.region)
[
  "ca-central-1",
  "us-east-1",

]
>
```

Numeric Functions:

Use to transform and manipulate numerical type data.

***Ceil** returns greater than or closer number.*

***Floor** return less than or closer number.*

variables.tf

```
variable "num" {
    type = set(number)
    default = [ 250, 10, 11, 5]
    description = "A set of numbers"
}
```

```
$ terraform console

> max (-1, 2, -10, 200, -250)
200

> min (-1, 2, -10, 200, -250)
-250

> max(var.num...)
250

> ceil(10.1)
11

> ceil(10.9)
11

> floor(10.1)
10
```

String Functions:

Use to transform and manipulate String type data values.

variables.tf

```
variable "ami" {
  type = string
  default = "ami-xyz,AMI-ABC,ami-efg"
  description = "A string containing ami ids"
}
```

```
$ terraform console
> split(",", "ami-xyz,AMI-ABC,ami-efg")
[ "ami-xyz","AMI-ABC","ami-efg" ]

> split(",", var.ami)
[ "ami-xyz","AMI-ABC","ami-efg“ ]

> lower(var.ami)
ami-xyz,ami-abc,ami-efg

> upper(var.ami)
AMI-XYZ,AMI-ABC,AMI-EFG

> title(var.ami)
Ami-Xyz,AMI-ABC,Ami-Efg

> substr(var.ami, 0, 7)
ami-xyz
> join(",", var.ami)
ami-xyz,AMI-ABC,ami-efg
```

Collection Functions:

Use for collection data types such as list, set and maps.

variables.tf

```
variable "ami" {
  type = list
  default = ["ami-xyz", "AMI-ABC", "ami-efg"]
    description = "A list of numbers"
}
```

```
$ terraform console
> length(var.ami)
3

> index(var.ami, "AMI-ABC")
1

> element(var.ami,2)
ami-efg

> contains(var.ami, "AMI-ABC")
true
```

Map Functions:

Use for Map data types.

variables.tf

```
variable "ami" {
  type = map
  default = { "us-east-1" = "ami-xyz",
              "ca-central-1" = "ami-efg",
              "ap-south-1" = "ami-ABC"
  }
  description = "A map of AMI ID's for specific regions"
}
```

```
$ terraform console
> keys(var.ami)
[
  "ap-south-1",
  "ca-central-1",
  "us-east-1",
]

> values(var.ami)
[
  "ami-ABC",
  "ami-efg",
  "ami-xyz",
]
> lookup (var.ami, "us-west-2", "ami-pqr")
ami-pqr
```

Conditional Expression

Comparison Operators

```
$ terraform console

> 5 > 7
false

> 5 > 4
true

> 5 > 5
False

> 5 >= 5
true

> 4 < 5
true

> 3 <= 4
true
```

Logical Operators

variables.tf

```
variable special {
  type        = bool
  default     = true
  description = "Set to true to
      use special characters"
}

variable b {
    type = number
    default = 25
}
```

```
$ terraform console

> 8 > 7 && 8 < 10
true

> 8 > 10 && 8 < 10
false

> 8 > 9 || 8 < 10
True

> var.special
true

> ! var.special
false

> ! (var.b > 30)
true
```

Equality Operators

```
terraform console

8 == 8
rue

== 7
alse

8 != "8"
rue
```

IF-ELSE

variables.tf

```
variable length {
  type        = number
  description = "The length of the password"
}
```