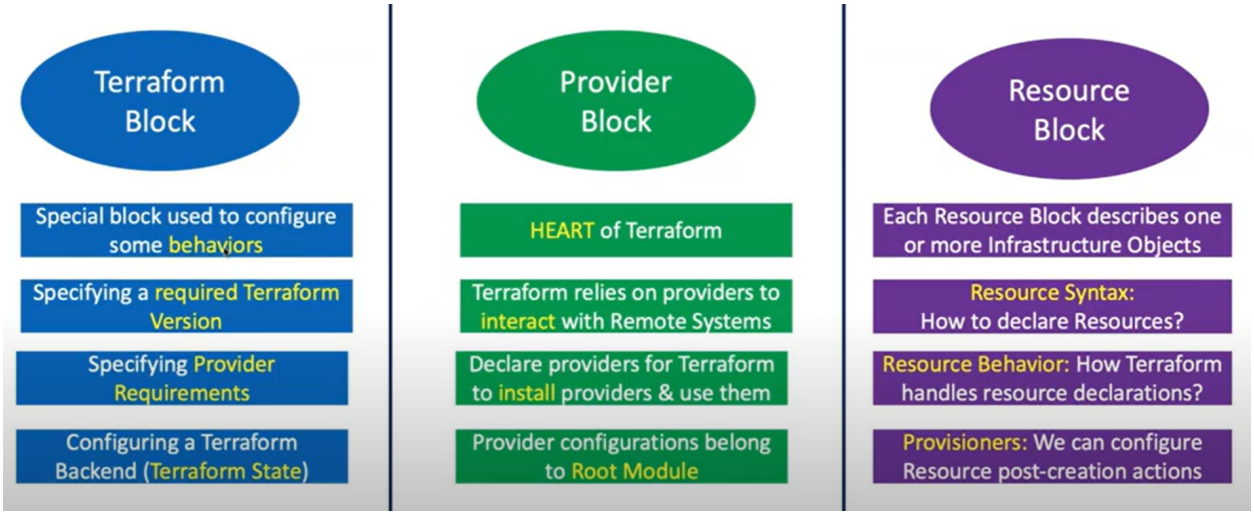
Terraform Video Resource

Video 1

Terraform Basic Block:

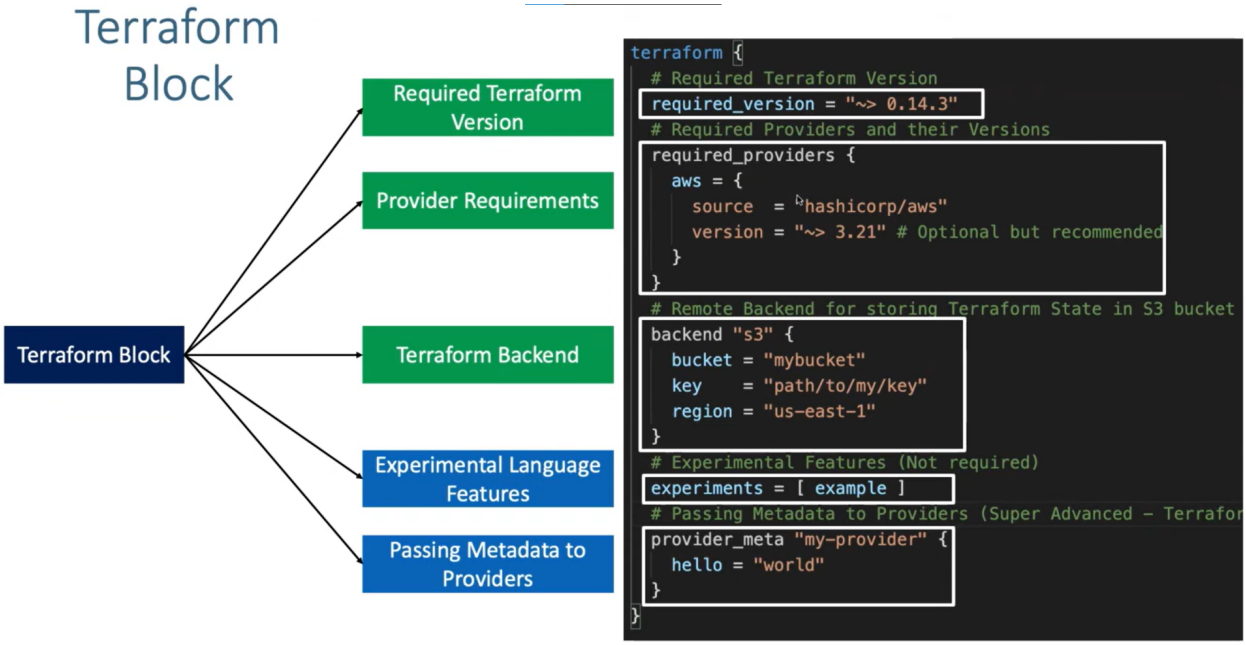


[1]

[1] Terraform Naming convention: [Terraform Coding Style and Naming Convention Best Practices & The Reasons | by Abrar Hasin | Medium]

Video 2

Terraform Block:



Required Terraform Version:

- Helps to stop breaking the code as Terraform is a highly evolving language.
- Helps to lock the version.

Provider Requirements:

- Helps to make sure the providers needed to download are locked.
- Specify the provider version needed as well.

Terraform Backend:

- Stores the state information of the configuration.
- Setup remote Backend.

#Rest 2 are Super Advanced and not required at basic level

Video 3

Required Version:

- Helps to lock the version of Terraform depending on the constraints.

- The most commonly used constraints are ~> which means it will allow all version increments compared to the rightmost version number and = which means the exact version provided is required.[2]
- Example for "~>": Provided version constraint "~>0.14.1" means it will allow the version specified and the rightmost increment. Example for "=": Provided version constraint "=0.14.1" means it will only allow the version specified.
- The documents related to version constraints: Link

Video 4

Required Providers

- Used to specify the required providers used in an Infrastructure.
- The source and the version should be mentioned.
- Similar version constraints for provider except for specifying the exact version no equals is needed.
- We can specify more than one provider as well.
- We can use the "terraform init -upgrade" to upgrade our provider version.

```
terraform init -upgrade
```

Video 5

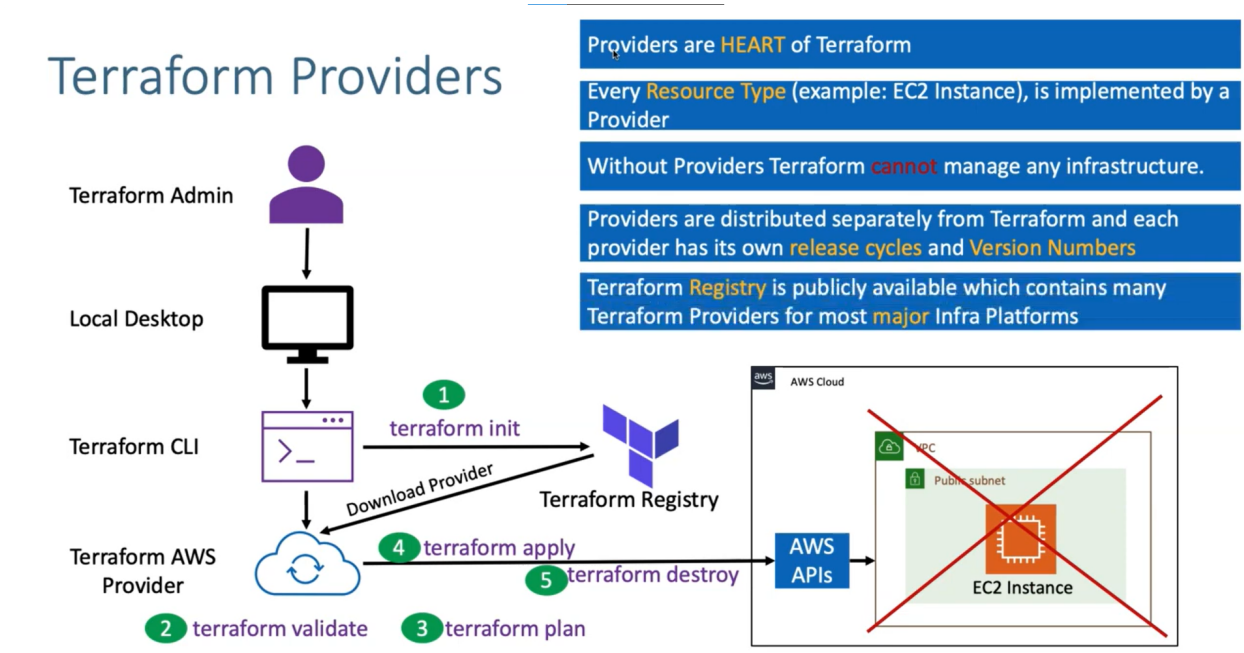
Note for Version control

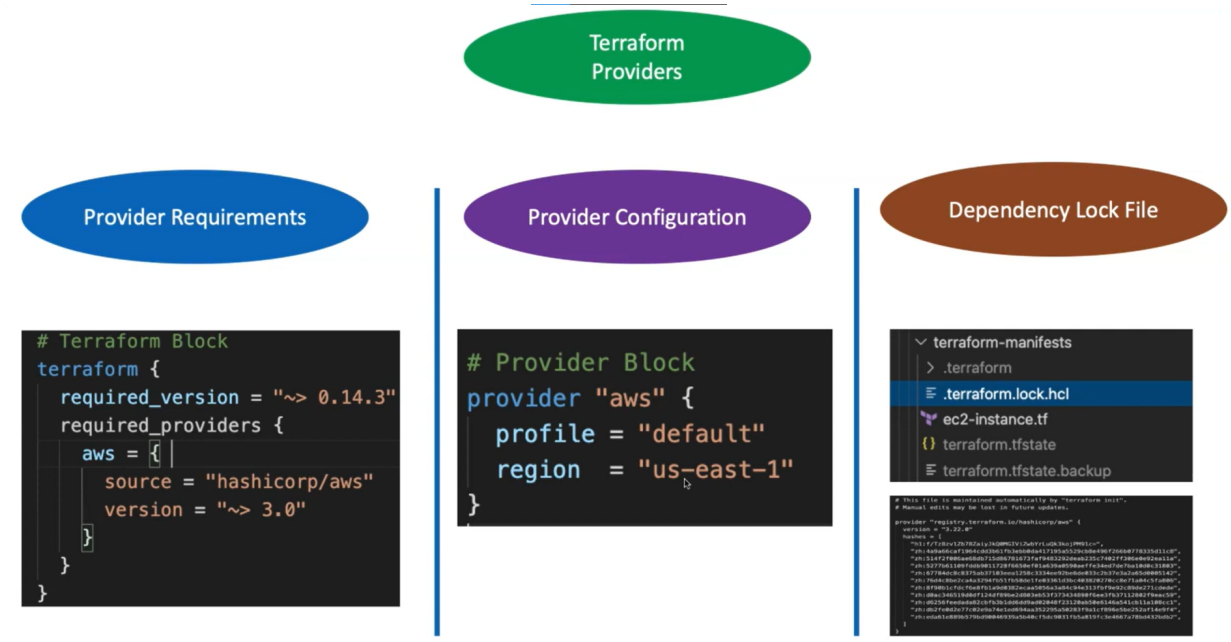
★ When using the version control it should be made sure to use the full form of the version for example "0.14.0" instead of "0.14". This makes sure that the version that is being used does not get too far mismatched. For "0.14.0" the next increment will be "0.14.1" but if the given version is "0.14" then the next version will be "0.15" which creates a huge gap in between.

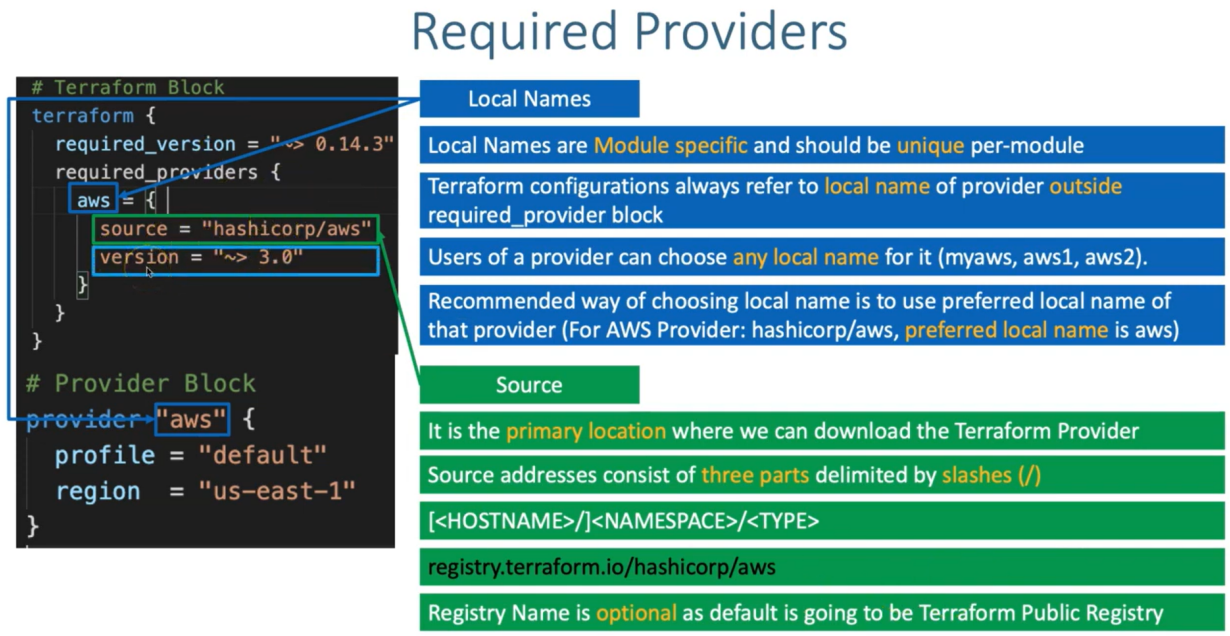
[2] Data types in HCL: How To Improve Flexibility Using Terraform Variables, Dependencies, and Conditionals | DigitalOcean

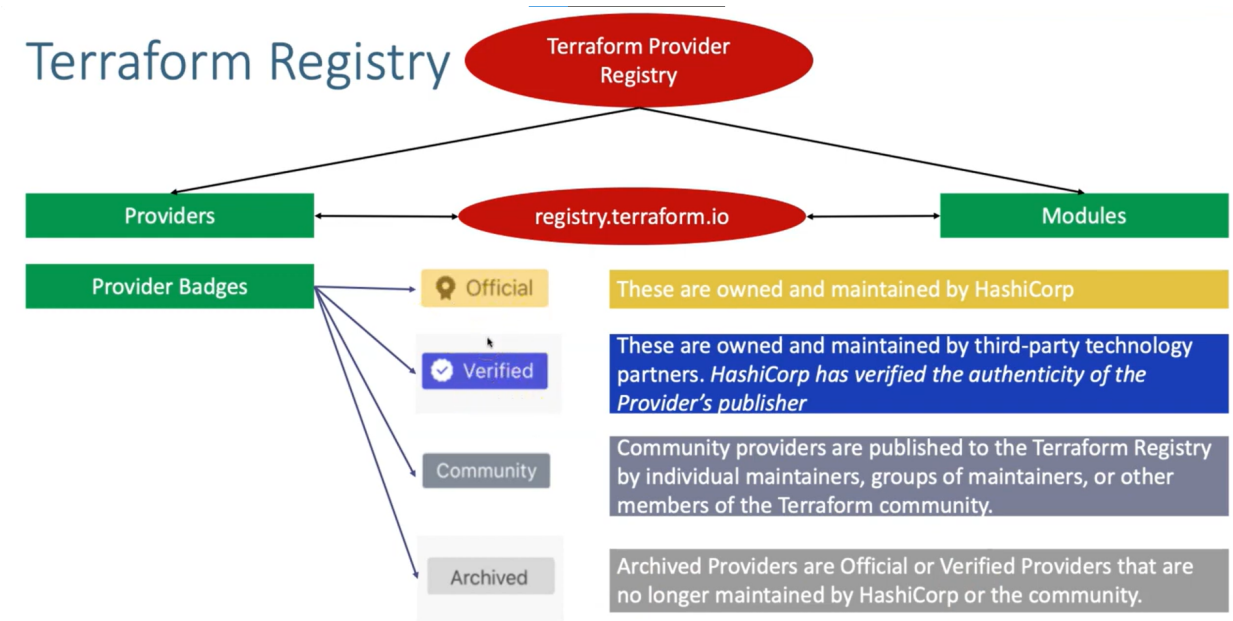
Video 6

Terraform Provider









Video 7

Provider Block

- We use it to configure our provider.
- We can use the provider we will be using for example AWS.
- We can use many arguments in the provider block.
- Region and profile can be taken as common arguments.

- We can give access and a secret key there but recommended no to do so.
- These sensitive data can be given in aws cli by using “aws configure”
- The keys can be found using “`cat $HOME/.aws/credentials`”
- Related resources given in the Link

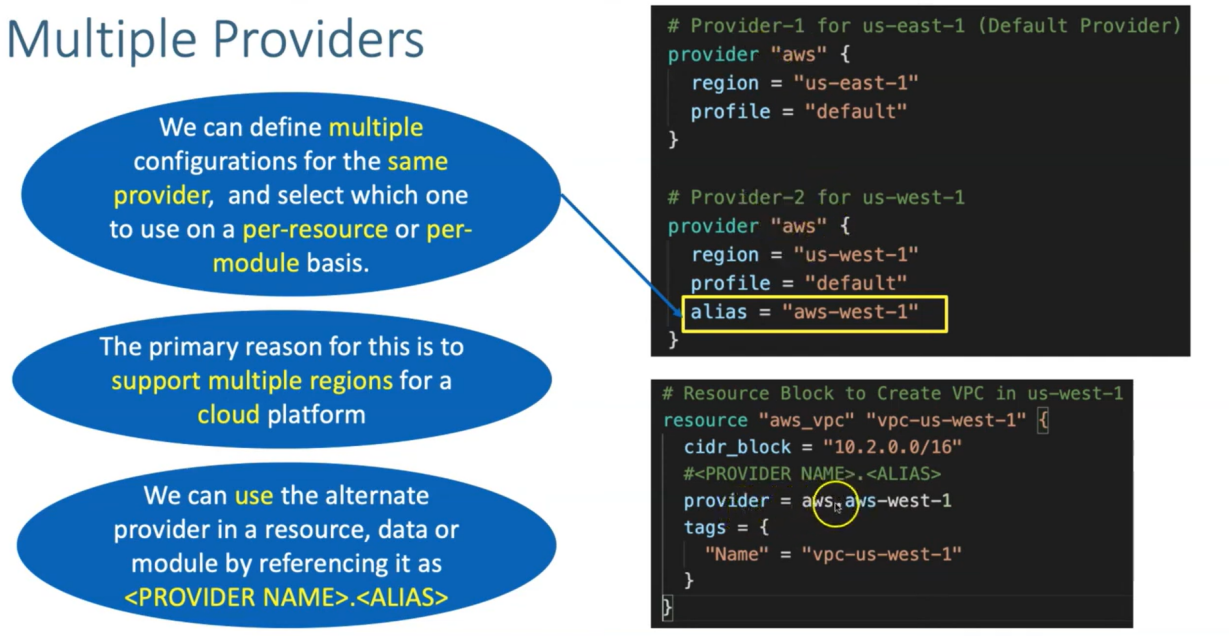
Video 8

VPC resource, Test and Release

- Terraform docs related to VPC.
- Required argument: CIDR
- We can add resources like a VPC giving it a unique identifier.
- We can give various arguments but the mandatory required argument must be provided.
- When we use terraform init, the terraform.lock.hcl and .terraform are created.
- When we use terraform apply, the terraform.tfstate is created. This file contains the representative information related to the resources used.
- Terraform uses the .terraform directory to store the project's providers and modules. Terraform will refer to these components when we run validate, plan, and apply.

Video 9

Multiple Providers



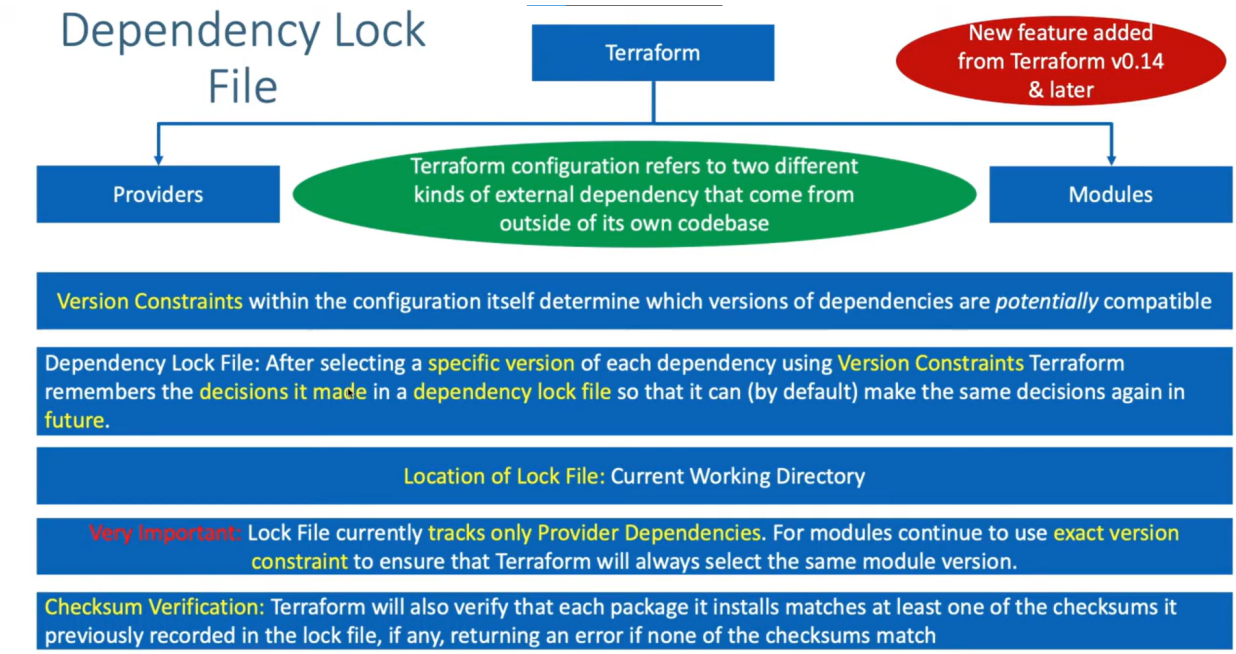
Video 10

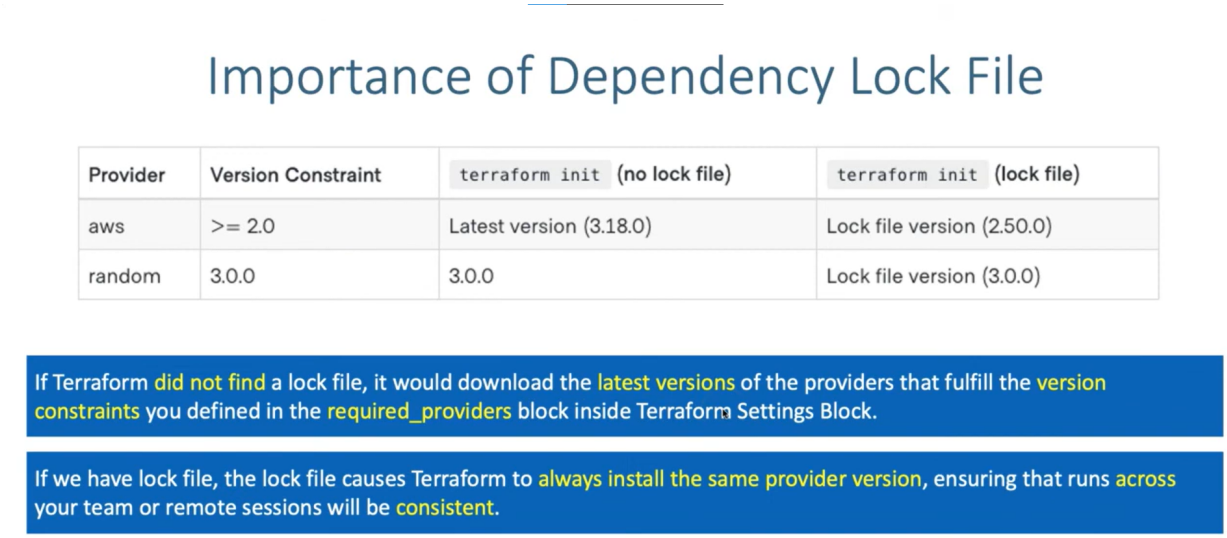
Multiple Provider Implementation

- We can implement multiple providers in Terraform.
- One will be default providers and the others will be alias providers.
- If a resource is not specified based on the meta argument of provider, the default provider will be selected
- The resource given the provider meta argument will use one of the alias providers.
- The way to apply the alias is: provider_name.alias_name

Video 11

Terraform Dependency Lock File





Video 12

Terraform Create S3 bucket with existing Lock File AWS provider version

- We can use a terraform lock file to pull the exact versions of the providers.
- This can help to lock the version of the providers used for a project to make sure that the project does not break.

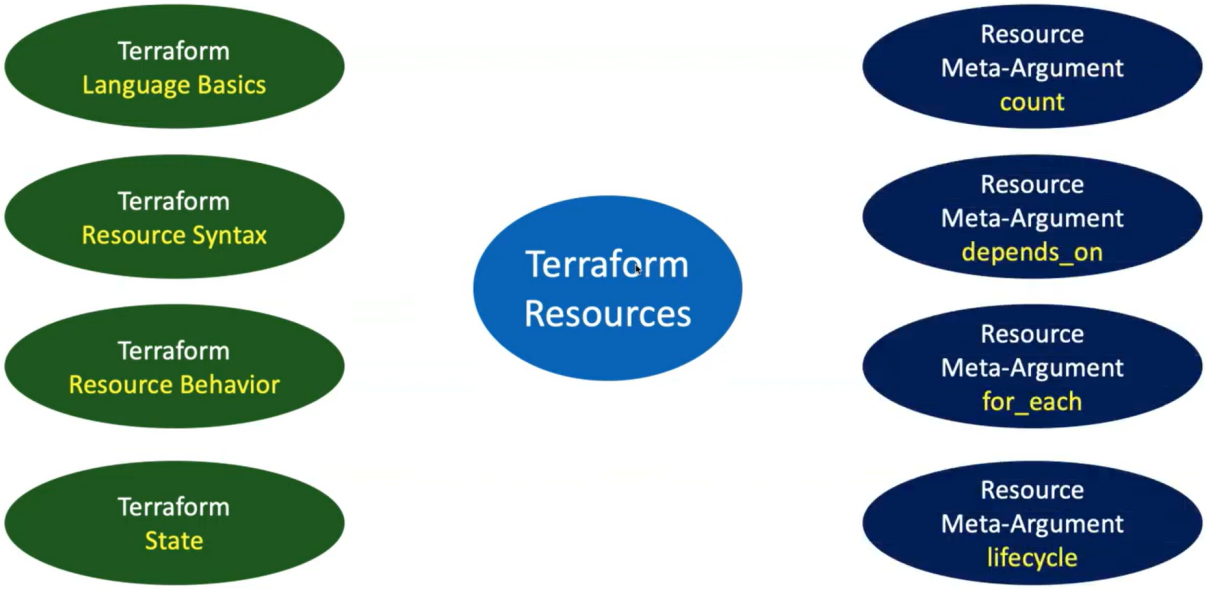
Video 13

Terraform Upgrade AWS Provider, Test, Fix S3 Bucket Issues and Clean Up

- By using “terraform init -upgrade” we can upgrade the lock file.
- There may be many arguments which will work for some specific versions and not for others.
- For example: for s3 bucket we no longer need to use “region” argument and the acl argument is also deprecated for latest versions.

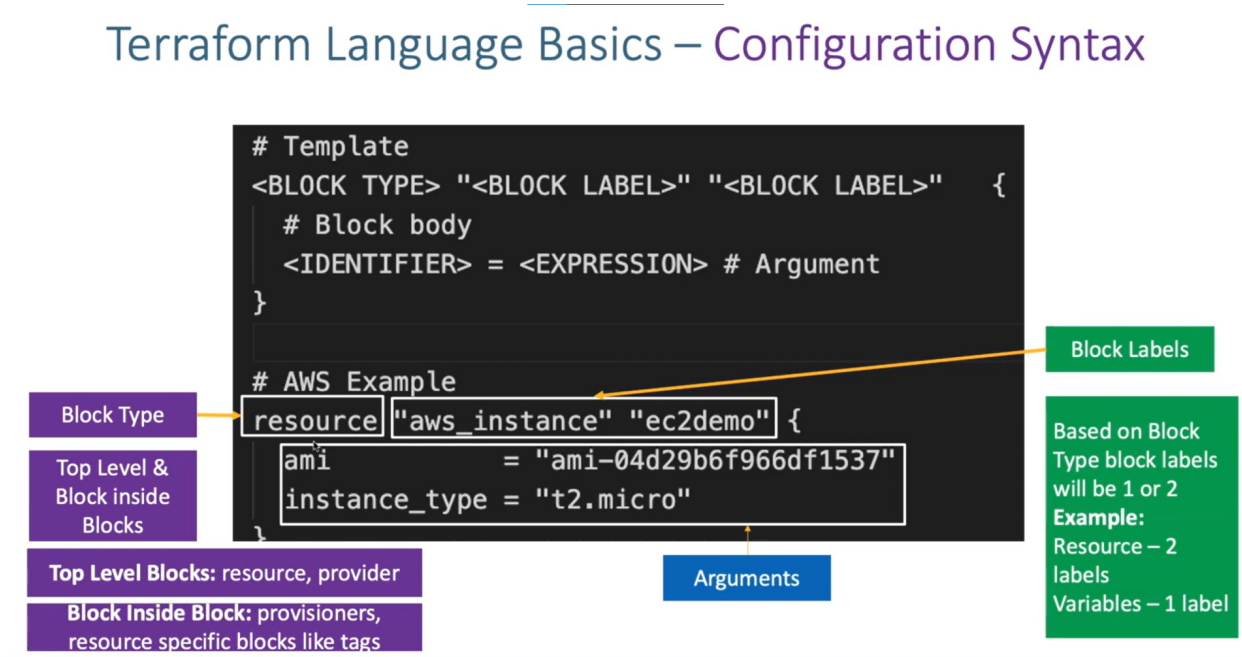
Video 14

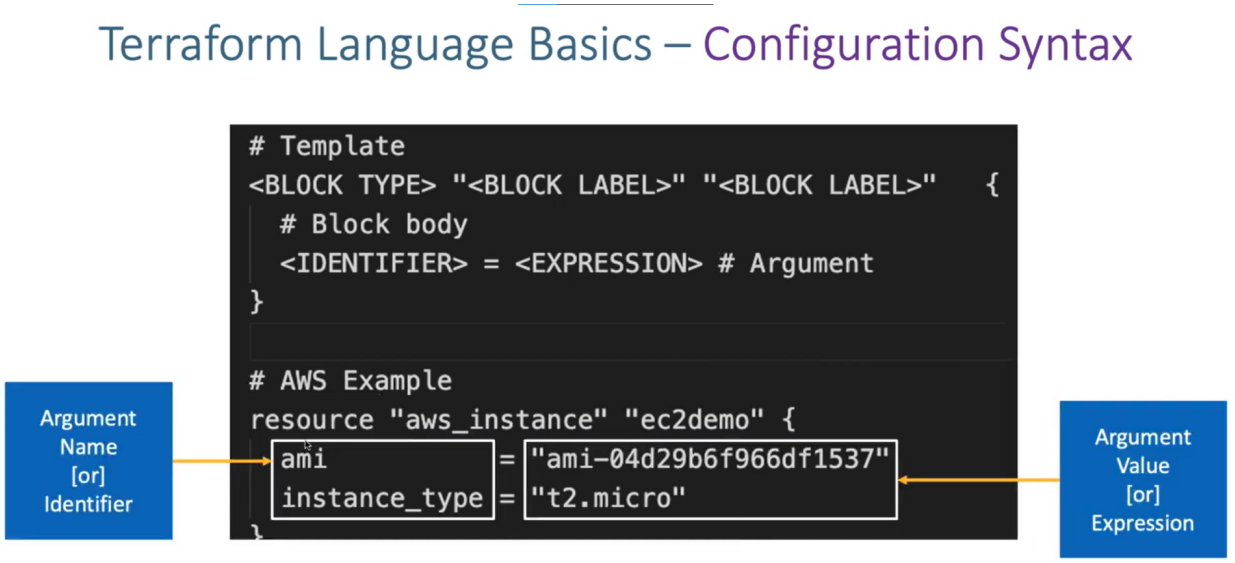
Terraform Resources Introduction

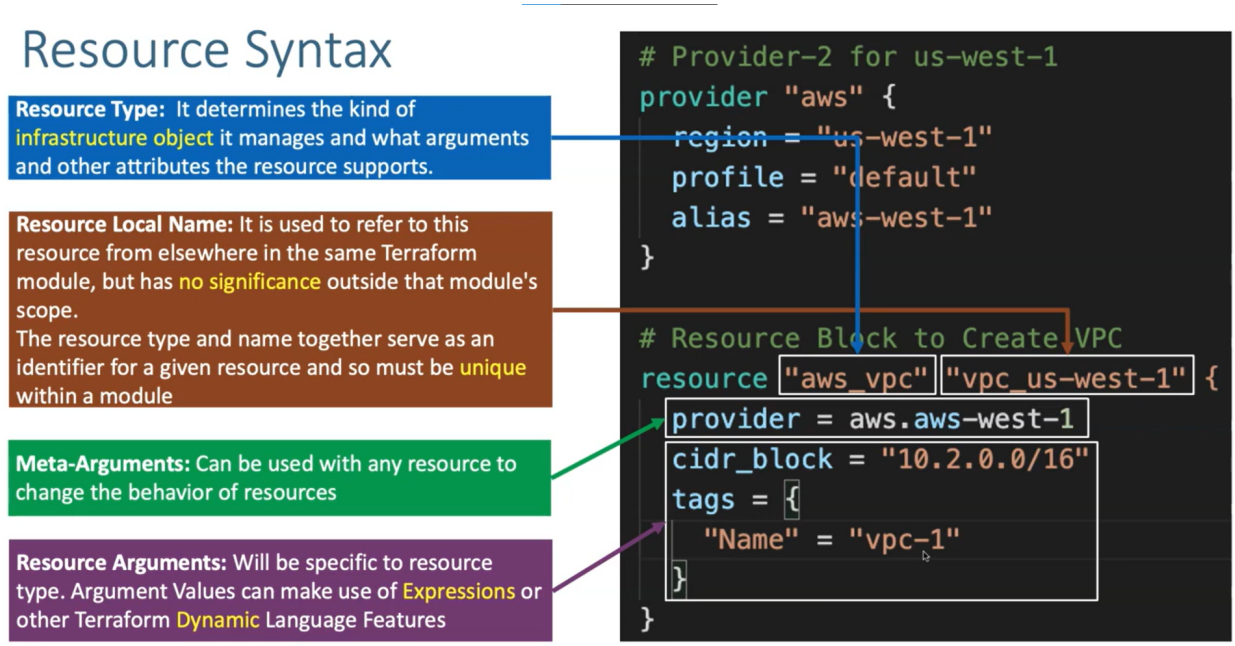


Video 15

Terraform Resources Syntax Introduction



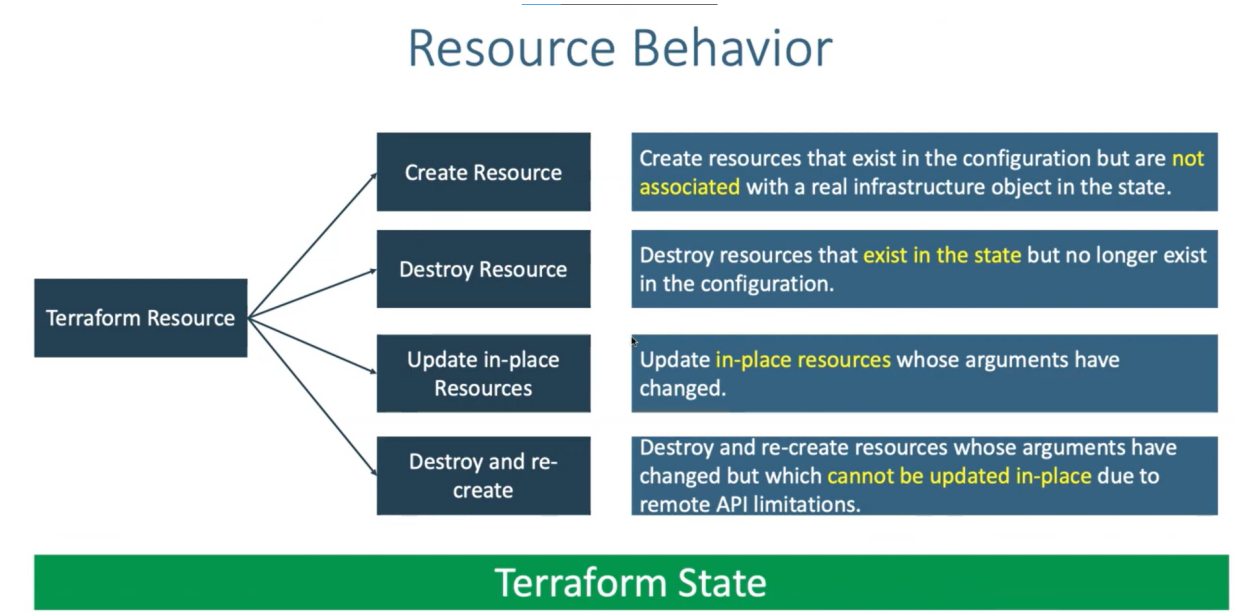




Video 16

Terraform Resources Behavior Part 1

- We can create, destroy, update and destroy and recreate resources.
- All of these resource behaviors will depend on the terraform state which stores the information related to the resources.



Video 17

Terraform Resources Behavior Part 2

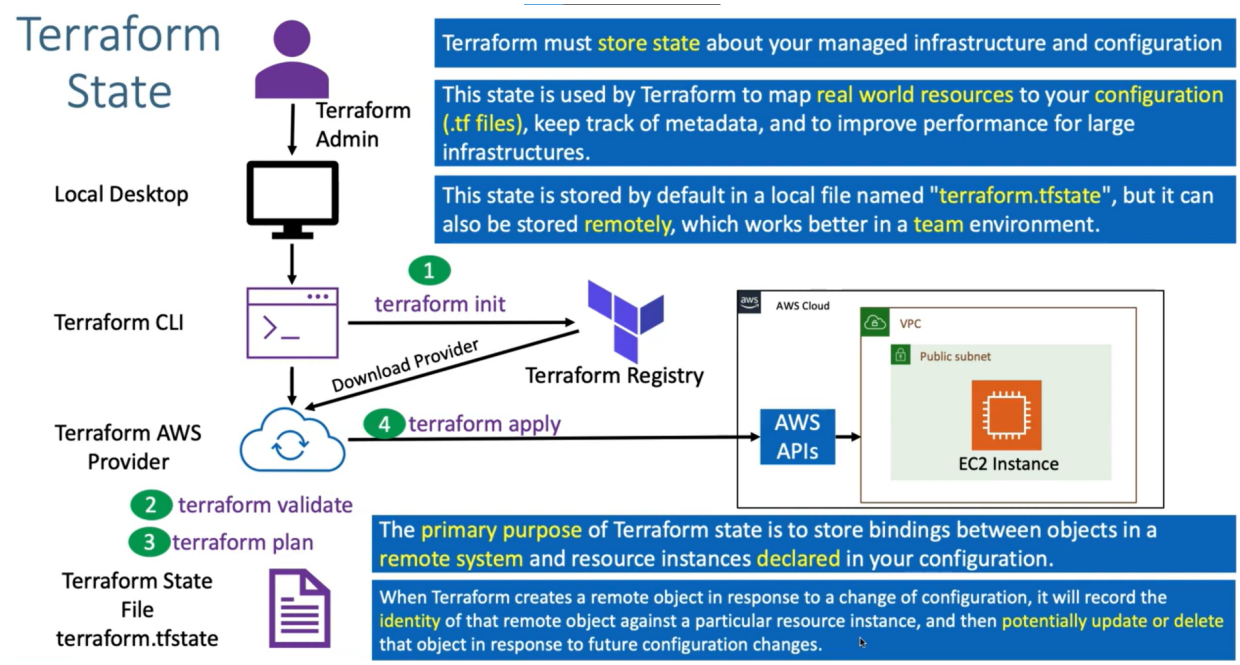
- Terraform stores all the metadata of the resource created in the terraform.state file.
- Terraform will notify what resource it will create, change, destroy or destroy and recreate while planning.

```
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:

  # aws_vpc.fazle-vpc1 will be created
  + resource "aws_vpc" "fazle-vpc1" {
      + arn                                  = (known after apply)
      + cidr_block                           = "192.78.0.0/27"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "Fazle-vpc1"
        }
      + tags_all                             = {
          + "Name" = "Fazle-vpc1"
        }
    }
```

In the above code we can see how create works.

Video 18

Understand Terraform State and Review terraform tfstate file



Video 19

Understand Terraform Resource Behavior Update In Place

When we update in place it shows something like this:

```
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc1 will be updated in-place
  ~ resource "aws_vpc" "fazle-vpc1" {
        id                                   = "vpc-0f9e694bc54cbf31b"
      ~ tags                                 = {
          ~ "Name" = "Fazle-vpc1" -> "Fazle-Vpc1"
        }
      ~ tags_all                             = {
          ~ "Name" = "Fazle-vpc1" -> "Fazle-Vpc1"
```

```
        }
        # (14 unchanged attributes hidden)
    }

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

- ★ Desired State: What we want our resources to be. It also can be seen when we make changes to our resources.
- ★ Current state: The state where the resources created are currently in.

Video 20

Understand Terraform Resource Behavior Destroy and Recreate

When we apply a change to our resource it may need to destroy and recreate our resources as shown below:

```
aws_vpc.fazle-vpc1: Refreshing state... [id=vpc-0612ec3504f805760]
Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc1 must be replaced
-/+ resource "aws_vpc" "fazle-vpc1" {
          ~ arn                                                          =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-0612ec3504f805760"  ->  (known
after apply)
      - assign_generated_ipv6_cidr_block     = false -> null
        ~ cidr_block                                 = "192.77.0.0/27" ->
"192.78.0.0/27" # forces replacement
       ~ default_network_acl_id                = "acl-0c0553d18965ab082" ->
(known after apply)
       ~ default_route_table_id                = "rtb-0d880178415b5cec2" ->
(known after apply)
       ~ default_security_group_id              = "sg-0b2bbc2415c9f8e7a" ->
(known after apply)
      ~ dhcp_options_id                       = "dopt-0b9ceb20cd0988fd9" ->
(known after apply)
       ~ enable_dns_hostnames                      = false -> (known after
apply)
```

```
        ~ enable_network_address_usage_metrics = false -> (known after
apply)
      ~ id                                     = "vpc-0612ec3504f805760" ->
(known after apply)
      + ipv6_association_id                    = (known after apply)
      + ipv6_cidr_block                        = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      - ipv6_netmask_length                    = 0 -> null
      ~ main_route_table_id                     = "rtb-0d880178415b5cec2" ->
(known after apply)
      ~ owner_id                                 = "430704357939" -> (known
after apply)
        tags                                   = {
            "Name" = "Fazle-vpc1"
        }
        # (3 unchanged attributes hidden)
    }

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

Video 21

Terraform Desired & Current State High Level and Clean Up

- The desired state is the state of the resources declared in the configuration files. The current state is the state of the resources in real world resources.

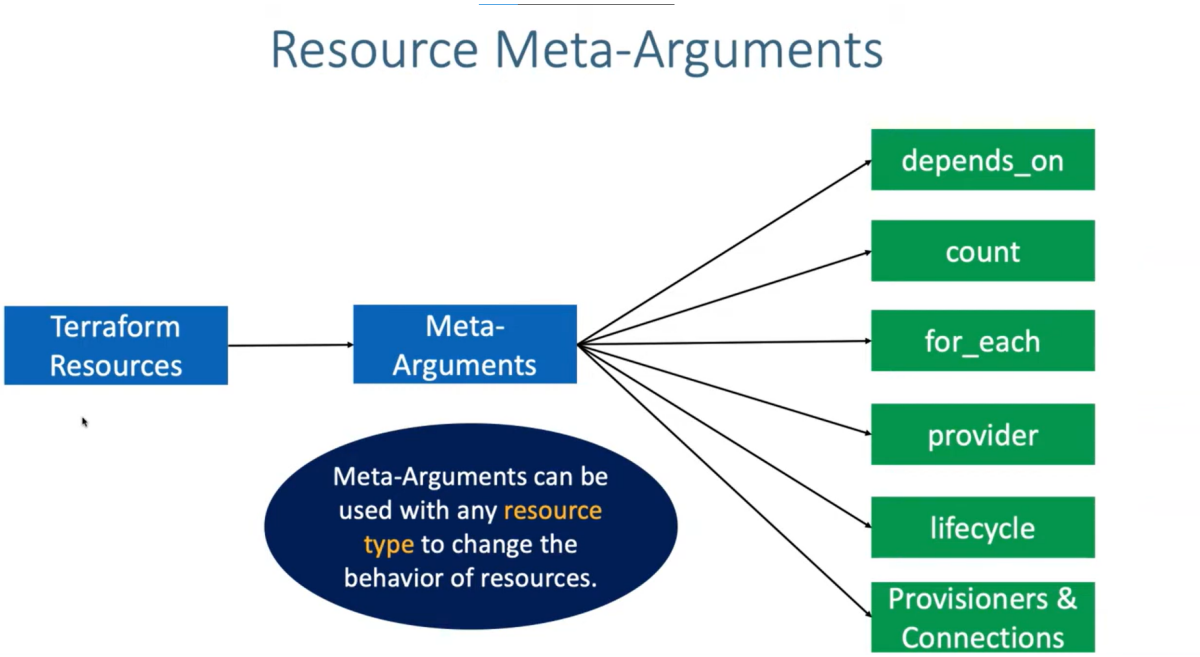
In case of destroying resource, it will give a similar output as given below:

```
aws_vpc.fazle-vpc1: Refreshing state... [id=vpc-0f9e694bc54cbf31b]

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

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc1 will be destroyed
  - resource "aws_vpc" "fazle-vpc1" {
```

```
      - arn                                  =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-0f9e694bc54cbf31b" -> null
      - assign_generated_ipv6_cidr_block     = false -> null
      - cidr_block                           = "192.78.0.0/27" -> null
      - default_network_acl_id               = "acl-06cb1fa1bbefa978b" ->
null
      - default_route_table_id               = "rtb-07f720eb4ff40b14a" ->
null
      - default_security_group_id            = "sg-0602a251b5d857130" ->
null
      - dhcp_options_id                      = "dopt-0b9ceb20cd0988fd9" ->
null
      - enable_dns_hostnames                 = false -> null
      - enable_dns_support                   = true -> null
      - enable_network_address_usage_metrics = false -> null
      - id                                   = "vpc-0f9e694bc54cbf31b" ->
null
      - instance_tenancy                     = "default" -> null
      - ipv6_netmask_length                  = 0 -> null
      - main_route_table_id                  = "rtb-07f720eb4ff40b14a" ->
null
      - owner_id                             = "430704357939" -> null
      - tags                                 = {
          - "Name" = "Fazle-Vpc1"
        } -> null
      - tags_all                             = {
          - "Name" = "Fazle-Vpc1"
        } -> null
    }

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

Video 22

Terraform Introduction to Resources Meta Arguments

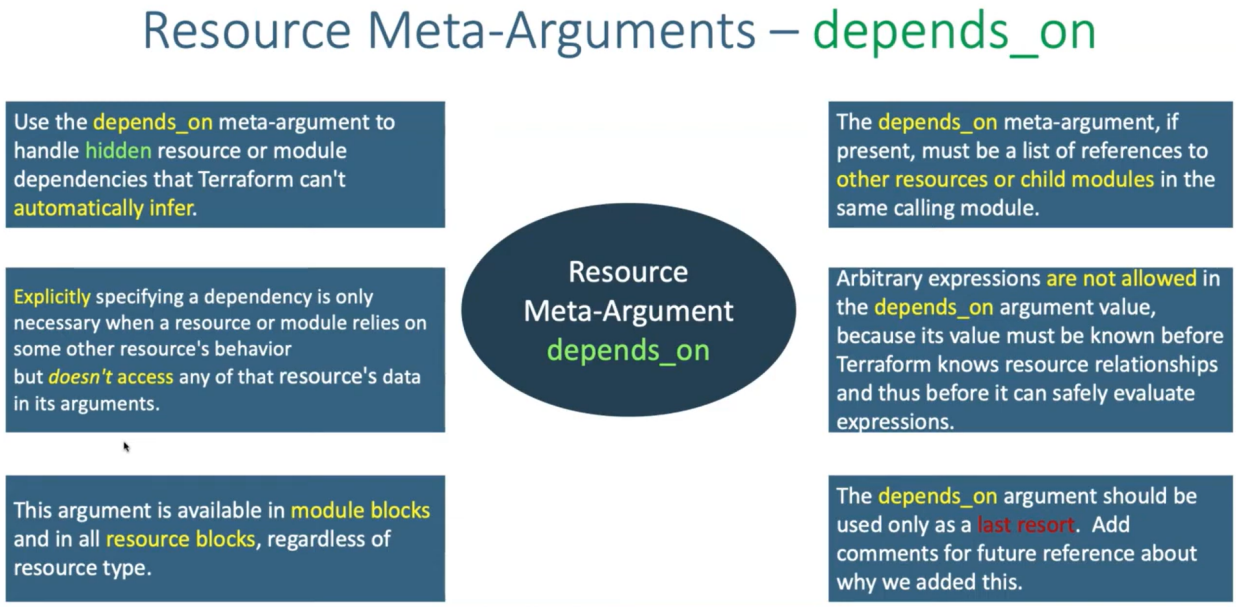


Resource Meta-Arguments

depends_on	To handle hidden resource or module dependencies that Terraform can't automatically infer.
count	For creating multiple resource instances according to a count
for_each	To create multiple instances according to a map, or set of strings
provider	For selecting a non-default provider configuration
lifecycle	Standard Resource behavior can be altered using special nested lifecycle block within a resource block body
Provisioners & Connections	For taking extra actions after resource creation (Example: install some app on server or do something on local desktop after resource is created at remote destination)

Video 23

Terraform Introduction to Resource Meta Argument depends on



Video 24

Create Key Pair terraform key and also review c1 versions tf

- Created a key-pair .pem file.
- Reviewed the versions of the providers we are using.

Video 25

Terraform Create VPC Manually to understand

- Manual creation of VPC, subnet, route table and IGW.

Video 26

Create VPC Resources using Terraform

- Created the following resources using Terraform: VPC, subnet, RT, IGW, route table allocation and security group.

Video 27

Create EC2 Instance Resource using Terraform

- Created an EC2 instance with user data for deploying an index .html on apache2.

Video 28

Create Elastic IP Resource with depends on Meta Argument

- Created an elastic IP resource which will depend on the IGW.

Video 29

Verify and Clean Up VPC, EC2 Instance and Elastic IP

- Verifying if all the resources were properly made and clean it up after we confirm.

Video 30

Terraform Introduction to Resources Meta Argument count

Resource Meta-Arguments – count

If a resource or module block includes a count argument whose value is a whole number, Terraform will create that many instances.

count.index: The distinct index number (starting with 0) corresponding to this instance.

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

Resource Meta-Argument count

When count is set, Terraform distinguishes between the block itself and the multiple resource or module instances associated with it. Instances are identified by an index number, starting with 0. aws_instance.myvm[0]

The count meta-argument accepts numeric expressions. The count value must be known *before* Terraform performs any remote resource actions.

Module support for count was added in Terraform 0.13, and previous versions can only use it with resources.

A given resource or module block cannot use both count and for_each

Use case: What are we going implement?

Resource-1: Use Meta-Argument Count to create multiple EC2 Instances using single Resource

```
# Create EC2 Instance
resource "aws_instance" "web" {
  ami = "ami-047a51fa27710816e" # Amazon Linux
  instance_type = "t2.micro"
  count = 5
  tags = {
    #"Name" = "web"
    "Name" = "web-${count.index}"
  }
}
```

count

count.index

aws_instance.web[0]

aws_instance.web[1]

aws_instance.web[2]

aws_instance.web[3]

aws_instance.web[4]

Video 31

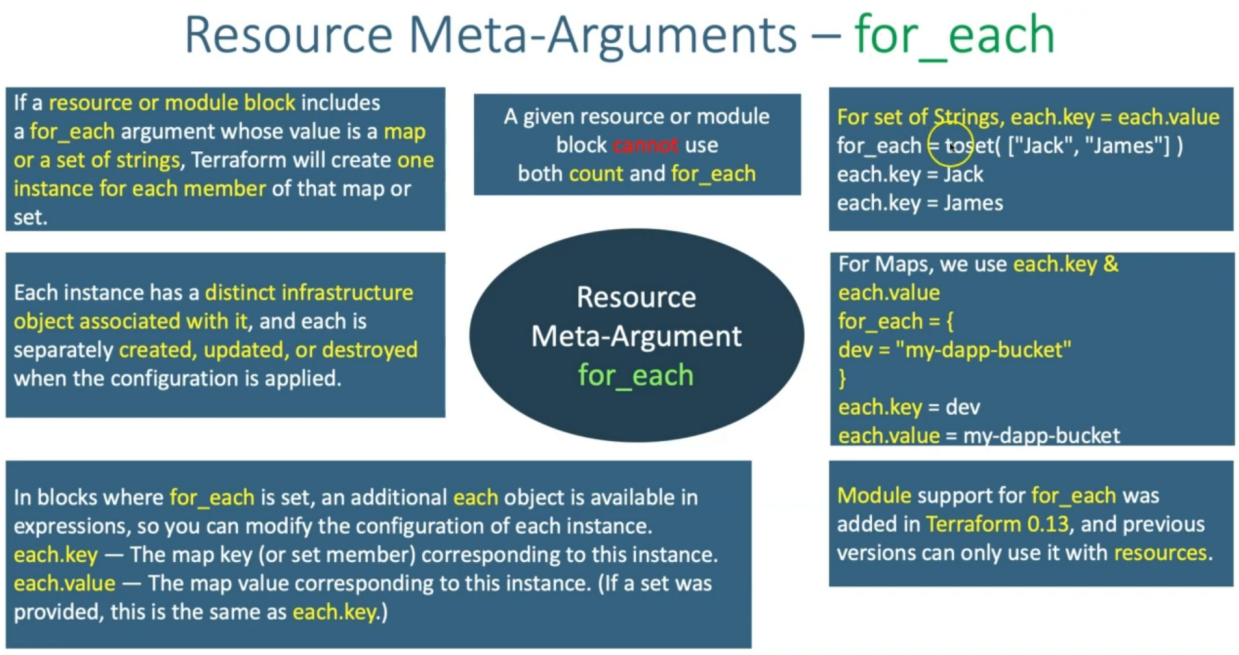
Create EC2 Instance with Meta Argument Count & Count Index and Clean Up

- Implemented count meta argument to tag Name an instance by the index of the count.

```
resource "aws_instance" "fazle-EC2" {
  ami           = "ami-08a52ddb321b32a8c"
  subnet_id     = aws_subnet.fazle-subnet.id
  instance_type = "t2.micro"
  count         = 5
  tags = {
    Name = "Fazle-EC2-${count.index}"
  }
}
```

Video 32

Terraform Introduction to Resources Meta Argument for each



Video 33

Terraform Implement Meta Argument for each with Maps

```
resource "aws_s3_bucket" "fazle-s3-bucket" {
  for_each = {
    lapp = "this-pot-bucket"
    dapp = "this-hot-bucket"
    tap  = "this-bot-bucket"
  }
  bucket = "${each.value}-${each.key}"
  #acl = "private"

  tags = {
    Name        = "fazle-bucket-${each.value}"
    Environment = "dev"
  }
}

resource "aws_s3_bucket_acl" "fazle-s3-acl" {
  for_each = aws_s3_bucket.fazle-s3-bucket
  bucket   = each.value.id
  depends_on = [ aws_s3_bucket_ownership_controls.fazle-s3-owner ]
  acl      = "private"
}

resource "aws_s3_bucket_ownership_controls" "fazle-s3-owner" {
  for_each = aws_s3_bucket.fazle-s3-bucket
  bucket   = each.value.id
  rule {
    object_ownership = "ObjectWriter"
  }
}
```

Video 34

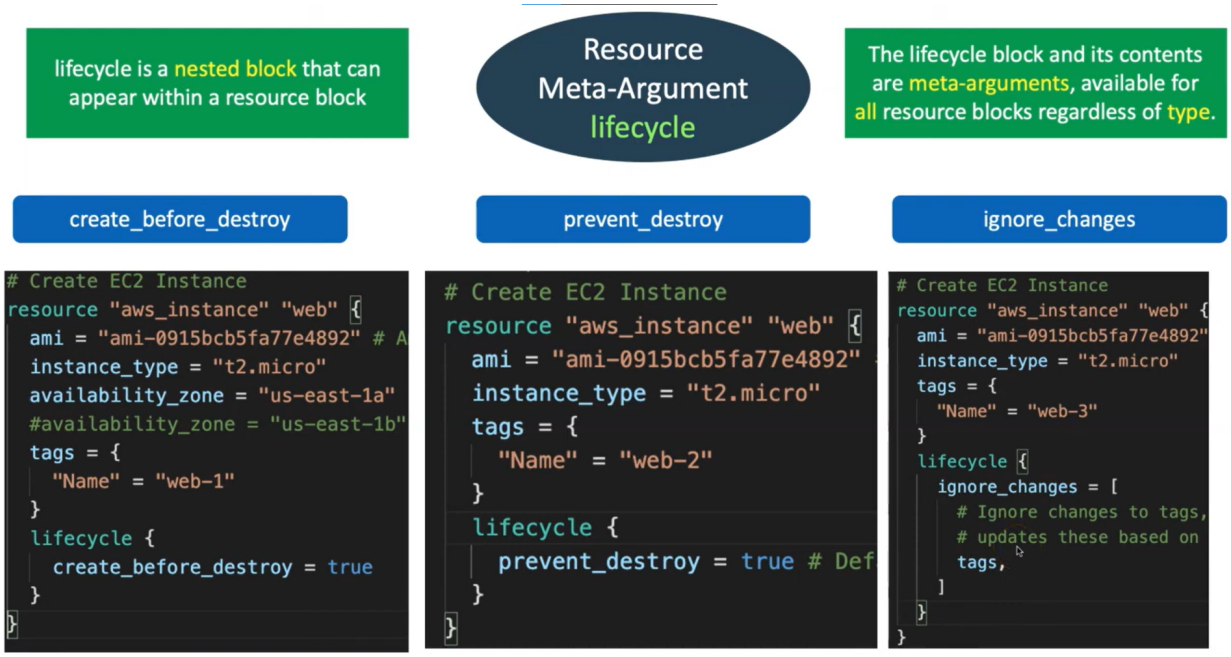
Terraform Implement Meta Argument for each with Set of Strings

```
resource "aws_s3_bucket" "fazle-aws-s3-bucket" {
```

```
  for_each = toset(["my-111-fazle-bucket", "my-112-fazle-bucket",
"my-113-fazle-bucket"])
  bucket   = each.key
  tags = {
    Name = "Fazle-Bucket-${each.key}"
  }
}
```

Video 35

Terraform Introduction to Resource Meta Argument lifecycle



Video 36

Terraform lifecycle Meta Argument create before destroy verify default behavior

- The default behavior for example: if we change the availability zone for us-east-1a to us-east-1b then the resource first gets destroyed then gets created.

Video 37

Terraform lifecycle Meta Argument create before destroy verify the altered behavior

```
resource "aws_vpc" "fazle-vpc" {
  cidr_block = "192.134.0.0/16"
}

resource "aws_subnet" "fazle-subnet" {
  cidr_block = "192.134.0.0/24"
  vpc_id = aws_vpc.fazle-vpc.id
  availability_zone = "us-east-1a"
  lifecycle {
    create_before_destroy = true
  }
}
```

```
resource "aws_instance" "fazle-INSTANCE" {
  ami               = "ami-08a52ddb321b32a8c"
  instance_type     = "t2.micro"
  subnet_id = aws_subnet.fazle-subnet.id
  availability_zone = "us-east-1a"
  #availability_zone = "us-east-1b"
  lifecycle {
    create_before_destroy = true
  }
}
```

If we change the availability zone and the CIDR blocks it will create first and destroy afterwards.

```
Plan: 3 to add, 0 to change, 3 to destroy.
aws_vpc.fazle-vpc: Creating...
aws_vpc.fazle-vpc: Creation complete after 4s [id=vpc-01cfd262a956c94e4]
aws_subnet.fazle-subnet: Creating...
aws_subnet.fazle-subnet: Creation complete after 2s [id=subnet-05d671847b9b32977]
aws_instance.fazle-INSTANCE: Creating...
aws_instance.fazle-INSTANCE: Still creating... [10s elapsed]
aws_instance.fazle-INSTANCE: Still creating... [21s elapsed]
aws_instance.fazle-INSTANCE: Still creating... [31s elapsed]
aws_instance.fazle-INSTANCE: Creation complete after 36s [id=i-0132c3aed9d080198]
aws_instance.fazle-INSTANCE (deposed object 7d4ca958): Destroying... [id=i-065641d5a0a79950c]
aws_instance.fazle-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 10s elapsed]
aws_instance.fazle-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 20s elapsed]
aws_instance.fazle-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 30s elapsed]
aws_instance.fazle-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 40s elapsed]
aws_instance.fazle-INSTANCE: Destruction complete after 42s
aws_subnet.fazle-subnet (deposed object 0d0cf51c): Destroying... [id=subnet-0bb14eadbe389bbc1]
aws_subnet.fazle-subnet: Destruction complete after 1s
aws_vpc.fazle-vpc (deposed object 9b56683a): Destroying... [id=vpc-07b40f56aae48e7f3]
aws_vpc.fazle-vpc: Destruction complete after 2s

Apply complete! Resources: 3 added, 0 changed, 3 destroyed.
```

Video 38

Terraform lifecycle Meta Argument prevent destroy implement and test

```hcl
resource "aws_instance" "fazle-INSTANCE" {
  ami               = "ami-08a52ddb321b32a8c"
  instance_type     = "t2.micro"
  subnet_id         = aws_subnet.fazle-subnet.id
  availability_zone = "us-east-1a"
  #availability_zone = "us-east-1b"
  lifecycle {
    #create_before_destroy = true
    prevent_destroy = true
  }
}
```

```
PS D:\Terraform\Files\Video Resource Files\video38> terraform destroy
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-0f4fe82ecbc3636d3]
aws_subnet.fazle-subnet: Refreshing state... [id=subnet-0fe35871d652d5e33]
aws_instance.fazle-INSTANCE: Refreshing state... [id=i-06eb6ef8b337334d4]

Error: Instance cannot be destroyed

  on resource.tf line 1:
   1: resource "aws_instance" "fazle-INSTANCE" {

Resource aws_instance.fazle-INSTANCE has lifecycle.prevent_destroy set, but the plan calls for this resource to be destroyed. To avoid this error and continue
with the plan, either disable lifecycle.prevent_destroy or reduce the scope of the plan using the -target flag.
```

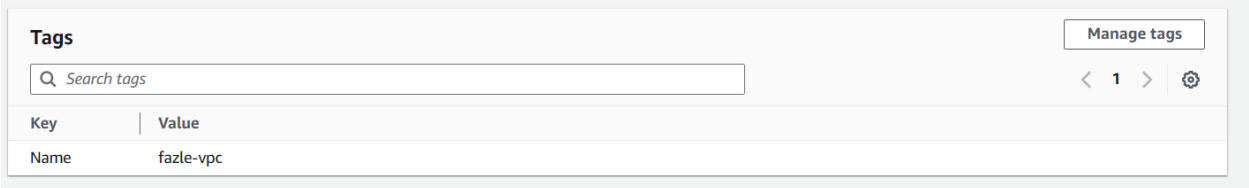
Video 39

Terraform lifecycle Meta Argument ignore changes implement and test

If we manually apply a new tag to the vpc down below the tag will be remove if we apply again

```
resource "aws_vpc" "fazle-vpc" {
  cidr_block = "192.155.0.0/16"
  tags = {
    Name = "fazle-vpc"
  }
#   lifecycle {
#     ignore_changes = [ tags ]
#   }
}
```

```
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-0c9e1d830857d1392]

Terraform used the selected providers to generate the following execution plan. Resource
actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc will be updated in-place
  ~ resource "aws_vpc" "fazle-vpc" {
        id                                   = "vpc-0c9e1d830857d1392"
      ~ tags                                 = {
            "Name" = "fazle-vpc"
          - "web"  = "my-vpc" -> null
        }
      ~ tags_all                             = {
          - "web"  = "my-vpc" -> null
            # (1 unchanged element hidden)
        }
        # (14 unchanged attributes hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
aws_vpc.fazle-vpc: Modifying... [id=vpc-0c9e1d830857d1392]
aws_vpc.fazle-vpc: Modifications complete after 4s [id=vpc-0c9e1d830857d1392]
```



After adding the ignore changes argument for the tags it shows nothing to change:

```
resource "aws_vpc" "fazle-vpc" {
```

```
  cidr_block = "192.155.0.0/16"
  tags = {
    Name = "fazle-vpc"
  }
  lifecycle {
    ignore_changes = [ tags ]
  }
}
```

```
PS D:\Terraform\Files\Video Resource Files\video39> terraform apply -auto-approve
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-0c9e1d830857d1392]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

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

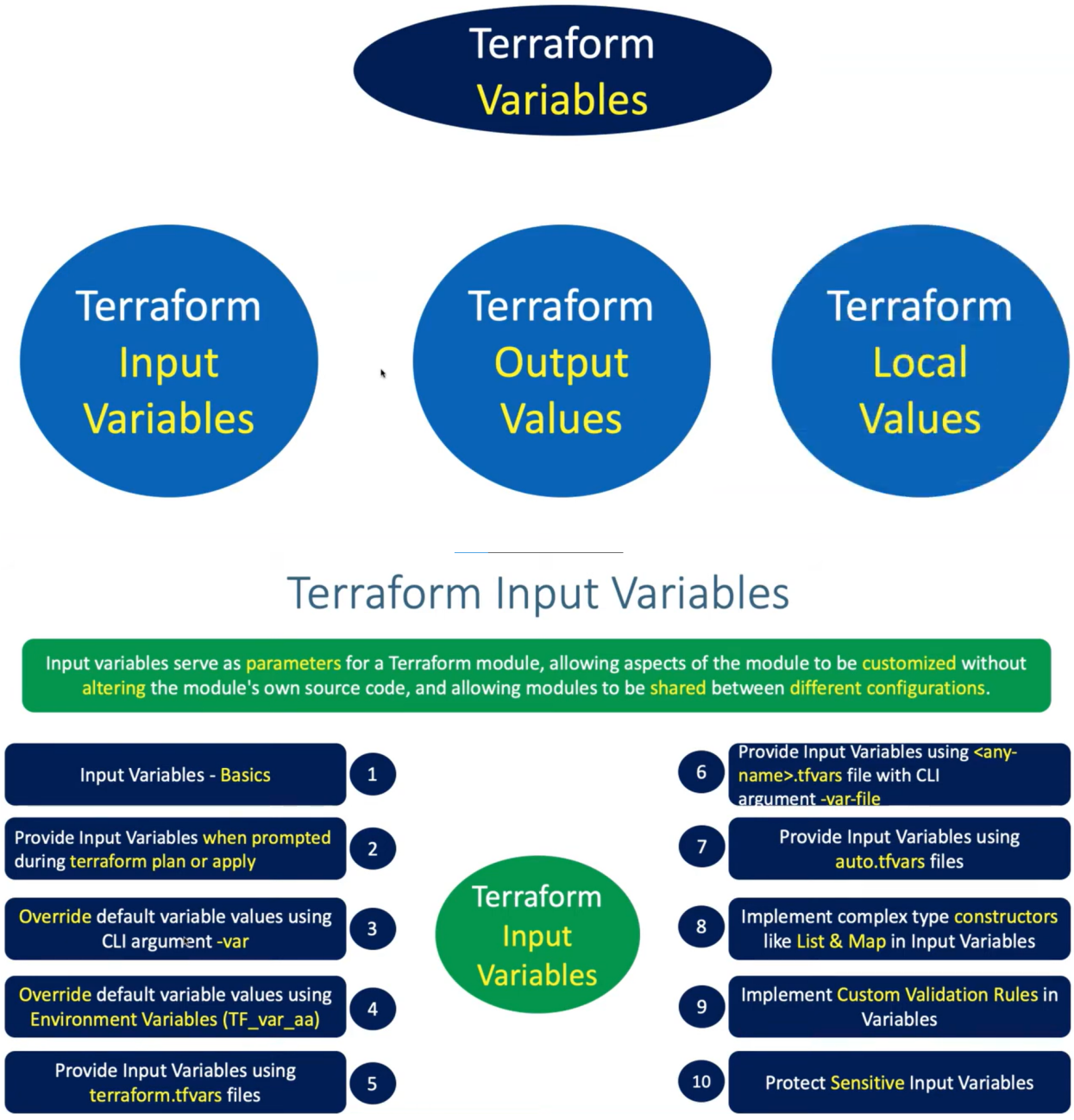
Video 40

Terraform Resource Provisioners

- Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

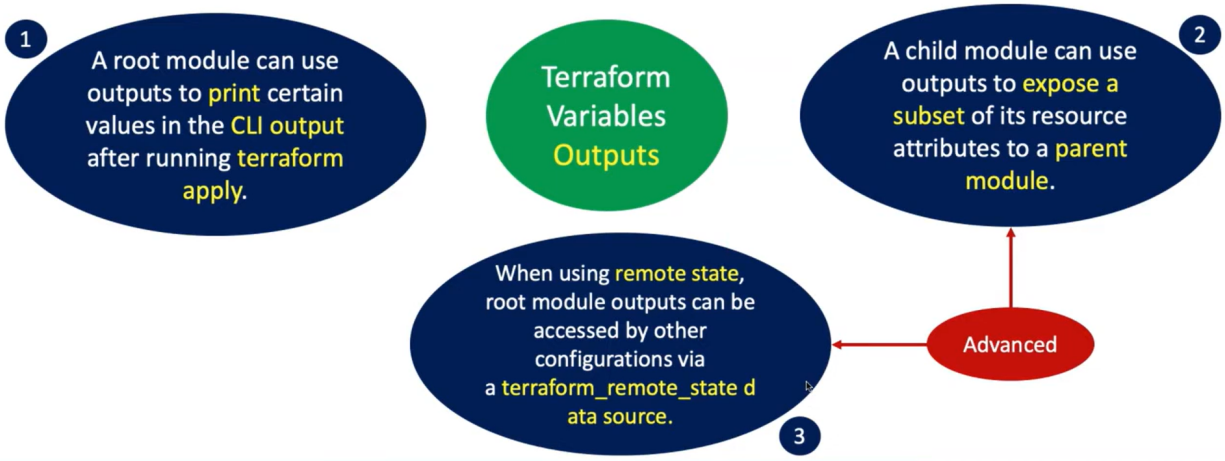
Video 41

Terraform Variables Introduction



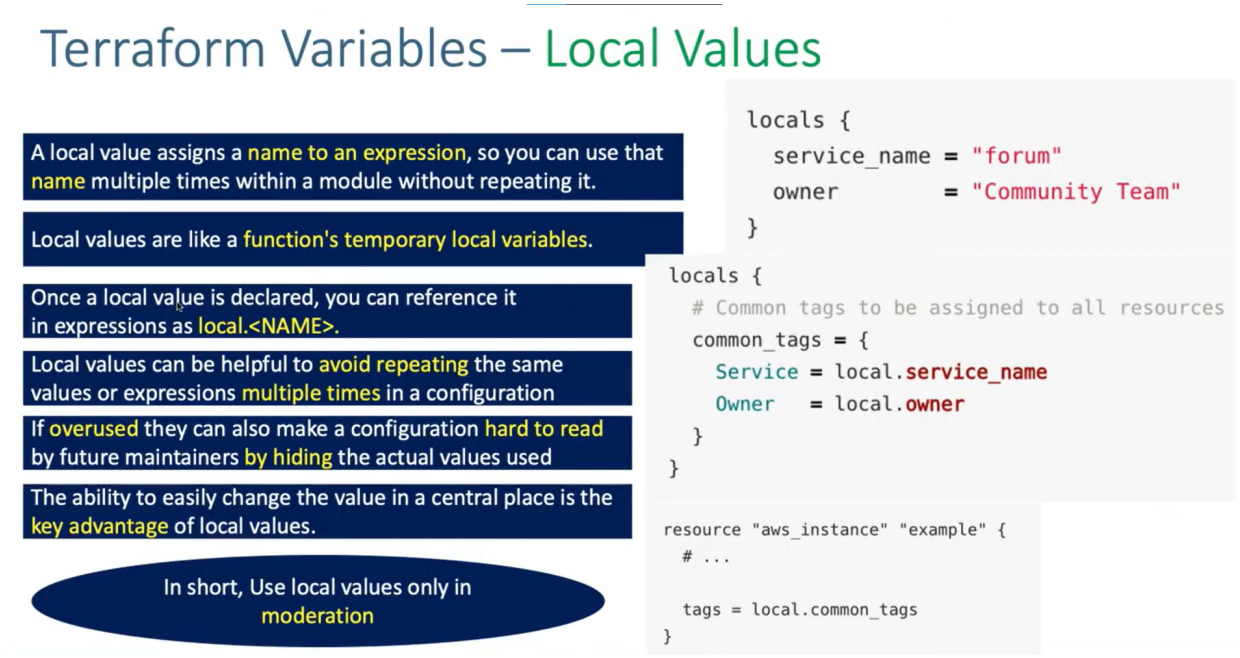
Terraform Variables – Output Values

Output values are like the return values of a Terraform module and have several uses



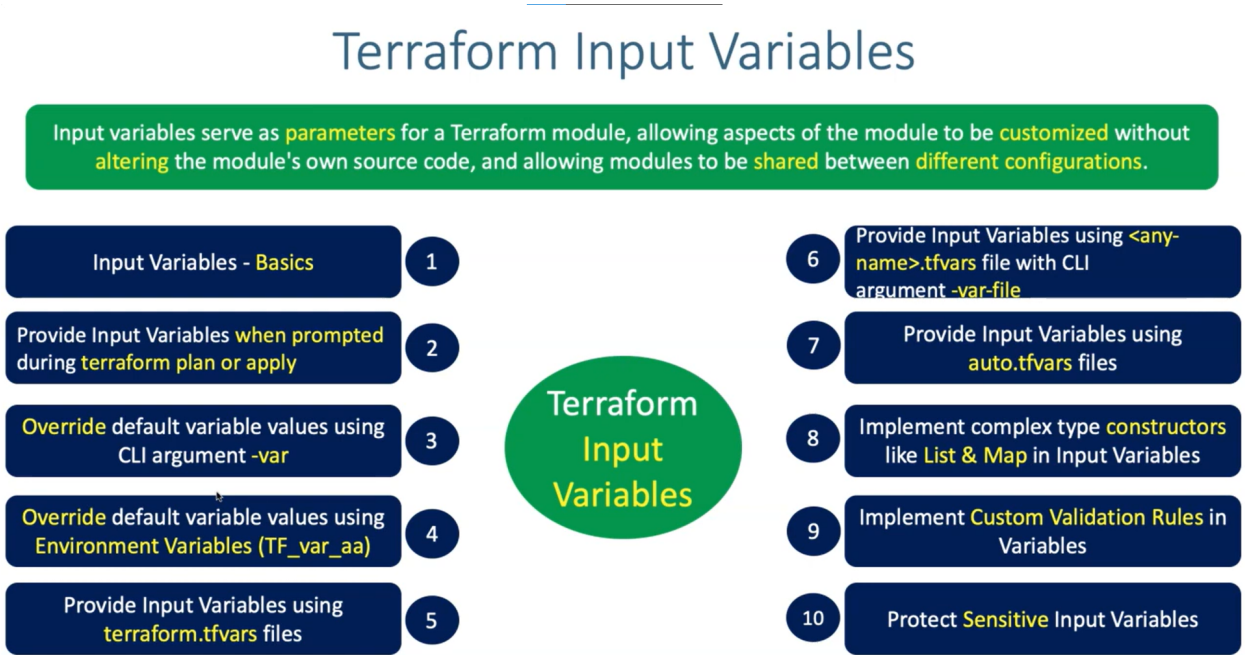
DRY Principle

Don't Repeat Yourself



Video 42

Terraform Input Variables Introduction



Video 43

Terraform Input Variable Basics Part 1

- Used a file to create some variables.
- Attached that variable to some arguments with the use of var.variableName

Video 44

Terraform Input Variable Basics Part 2

Used a variable.tf file to add variable to the CIDR blocks of my VPC and Subnet and the availability zone of my subnet as well:

```
variable "vpc_cidr_block" {
  description = "This is the cidr block for my vpc"
  type = string
  default = "192.123.0.0/16"
}

variable "subnet_cidr_block" {
  description = "This the cidr block for my subnet"
  type = string
  default = "192.123.1.0/24"
}

variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type = string
  default = "us-east-1a"
}
```

```
resource "aws_vpc" "fazle-vpc" {
  cidr_block = var.vpc_cidr_block
}

resource "aws_subnet" "fazle-subnet" {
  cidr_block        = var.subnet_cidr_block
  availability_zone = var.subnet_az
  vpc_id = aws_vpc.fazle-vpc.id
}
```

Video 45

Terraform Input Variables Assigning When Prompted using CLI

If we do not set a default value then the cli will prompt for an input of that variable:

```
variable "vpc_cidr_block" {
  description = "This is the cidr block for my vpc"
  type        = string
  default     = "192.123.0.0/16"
}

variable "subnet_cidr_block" {
  description = "This the cidr block for my subnet"
  type        = string
  default     = "192.123.1.0/24"
}

variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type        = string
  #default     = "us-east-1a"
}
```

- The description argument acts as a message for the prompt.

```
PS D:\Terraform\Files\Video Resource Files\video45> terraform apply -auto-approve
var.subnet_az
  This is the availability zone for my subnet

  Enter a value: us-east-1a
```

Video 46

Terraform Input Variables Override default value with var argument

```
variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type        = string
```

```
  default     = "us-east-1a"
}
```

```
PS D:\Terraform\Files\Video Resource Files\video46> terraform plan -var="subnet_az=us-east-1b"

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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1b"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "192.123.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags_all                                       = (known after apply)
      + vpc_id                                         = (known after apply)
    }
```

Video 47

Terraform Input Variables Override default value with environment variables

We can use TF_VAR environment variables to change the value of the variable through the cli in the following way:

```
variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type        = string
  default     = "us-east-1a"
}
```

```
$ export TF_VAR_subnet_az=us-east-1b
```

```
BJIT@11729-fazle-mubin MINGW64 /d/Terraform/Files/Video Resource Files/video47
$ echo $TF_VAR_subnet_az
us-east-1b

BJIT@11729-fazle-mubin MINGW64 /d/Terraform/Files/Video Resource Files/video47
$ unset TF_VAR_subnet_az

BJIT@11729-fazle-mubin MINGW64 /d/Terraform/Files/Video Resource Files/video47
$ echo $TF_VAR_subnet_az
```

Video 48

Terraform Input Variables Assign with terraform tfvars

variables.tf - here, you define the variables that must have values in order for your Terraform code to validate and run. You can also define default values for your variables in this file. Note that you don't need to define all of your variables in a file named variables.tf - they can be defined anywhere, but this practice is encouraged for organizational purposes.

terraform.tfvars - this file contains one or more variablename=variablevalue pairs. When Terraform loads this file, it looks for any variables in your Terraform with the name variablename and sets their value to be variablevalue. You can't define new variables here, and can only set the values of existing ones defined in variables.tf.

These are the values in my variables.tf file which we will override with terraform.tfvars:

```
variable "vpc_cidr_block" {
  description = "This is the cidr block for my vpc"
  type        = string
  default     = "192.123.0.0/16"
}

variable "subnet_cidr_block" {
  description = "This the cidr block for my subnet"
  type        = string
  default     = "192.123.1.0/24"
}

variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type        = string
  default     = "us-east-1a"
}
```

The terraform.tfvars file will look like this:

```
vpc_cidr_block = "192.13.0.0/16"
subnet_cidr_block = "192.13.5.0/24"
subnet_az = "us-east-1c"
```

The output will be the following:

```
# aws_subnet.fazle-subnet will be created
+ resource "aws_subnet" "fazle-subnet" {
    + arn                                            = (known after apply)
    + assign_ipv6_address_on_creation                = false
    + availability_zone                              = "us-east-1c"
    + availability_zone_id                           = (known after apply)
    + cidr_block                                     = "192.13.5.0/24"
    + enable_dns64                                   = false
    + enable_resource_name_dns_a_record_on_launch    = false
    + enable_resource_name_dns_aaaa_record_on_launch = false
    + id                                             = (known after apply)
    + ipv6_cidr_block_association_id                 = (known after apply)
    + ipv6_native                                    = false
    + map_public_ip_on_launch                        = false
    + owner_id                                       = (known after apply)
    + private_dns_hostname_type_on_launch            = (known after apply)
    + tags_all                                       = (known after apply)
    + vpc_id                                         = (known after apply)
  }

# aws_vpc.fazle-vpc will be created
+ resource "aws_vpc" "fazle-vpc" {
    + arn                                = (known after apply)
    + cidr_block                         = "192.13.0.0/16"
    + default_network_acl_id             = (known after apply)
    + default_route_table_id             = (known after apply)
    + default_security_group_id          = (known after apply)
    + dhcp_options_id                    = (known after apply)
    + enable_dns_hostnames               = (known after apply)
    + enable_dns_support                 = true
    + enable_network_address_usage_metrics = (known after apply)
```

Video 49

Terraform Input Variables Assign with var file argument

If we want to change the name of terraform.tfvars to a different name for example: example.tfvars then we need to use the var file argument.

The terraform.tfvars had the following code:

```
vpc_cidr_block = "192.13.0.0/16"
subnet_cidr_block = "192.13.5.0/24"
subnet_az = "us-east-1c"
```

We change the value of subnet_az in web.tfvars:

```
subnet_az = "us-east-1d"
```

This resulted in the subnet_az value to change only if we give the -var-file=”web.tfvars” input:

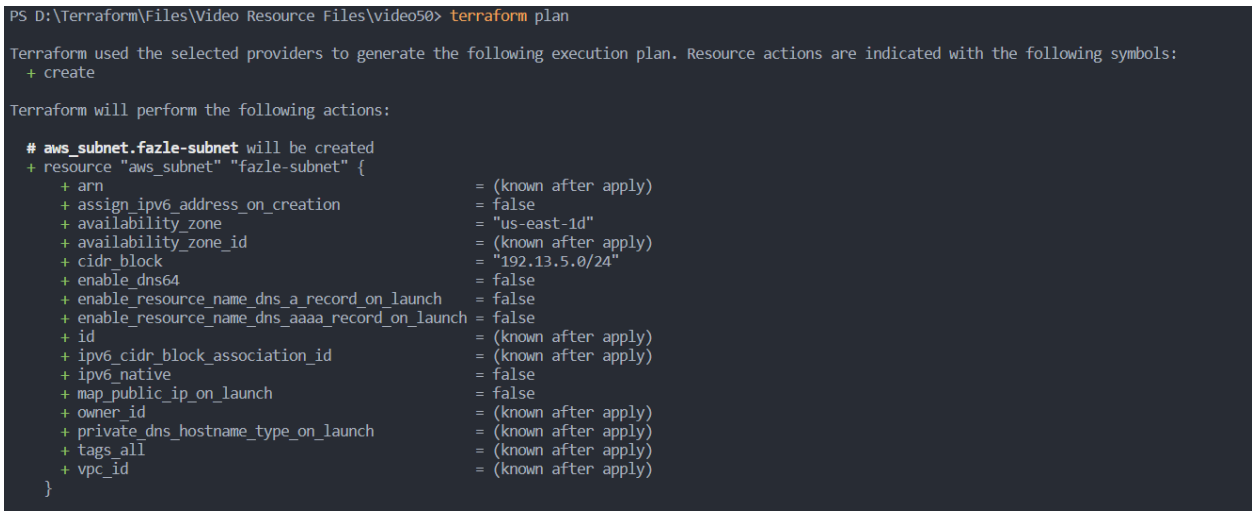
```
PS D:\Terraform\Files\Video Resource Files\video49> terraform plan -var-file="web.tfvars"

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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1d"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "192.13.5.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags_all                                       = (known after apply)
      + vpc_id                                         = (known after apply)
    }
```

Video 50

Terraform Input Variables Assign with auto tfvars files

We can change the name of terraform.tfvars to any name without using -var-file argument by changing the name to for example: example.auto.tfvars

So by just changing the name from terraform,tfvars to web.auto.tfvars the we get the same output as using -var-file argument which is shown below:



```
PS D:\Terraform\Files\Video Resource Files\video50> 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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1d"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "192.13.5.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags_all                                       = (known after apply)
      + vpc_id                                         = (known after apply)
    }
```

Video 51

Terraform Input Variables Complex Constructor of Type List

We can use List in Terraform in the following way:

```
variable "subnet_az" {
  description = "This is the availability zone for my subnet"
  type        = list(string)
  default     = ["us-east-1a","us-east-1b","us-east-1c","us-east-1d"]
}
```

```
resource "aws_subnet" "fazle-subnet" {
  cidr_block        = var.subnet_cidr_block
  availability_zone = var.subnet_az[1]
  vpc_id            = aws_vpc.fazle-vpc.id
}
```

```
PS D:\Terraform\Files\Video Resource Files\video51> 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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1b"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "192.123.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
      + id                                             = (known after apply)
      + ipv6_cidr_block_association_id                 = (known after apply)
      + ipv6_native                                    = false
      + map_public_ip_on_launch                        = false
      + owner_id                                       = (known after apply)
      + private_dns_hostname_type_on_launch            = (known after apply)
      + tags_all                                       = (known after apply)
      + vpc_id                                         = (known after apply)
    }
```

Video 52

Terraform Input Variables Complex Constructor of Type Map

We can use Map or object in Terraform as well. It can be applied in the following way:

```
variable "subnet_az" {
  description = "This is the availability zone for my subnet"
```

```
  type         = map(string)
  default = {
    aZone = "us-east-1a",
    bZone = "us-east-1b",
    cZone = "us-east-1c",
    dZone = "us-east-1d"
  }
}
```

```
resource "aws_subnet" "fazle-subnet" {
  cidr_block        = var.subnet_cidr_block
  availability_zone = var.subnet_az["cZone"]
  vpc_id            = aws_vpc.fazle-vpc.id
}
```

The following is the output:

```
PS D:\Terraform\Files\Video Resource Files\video52> 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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                                            = (known after apply)
      + assign_ipv6_address_on_creation                = false
      + availability_zone                              = "us-east-1c"
      + availability_zone_id                           = (known after apply)
      + cidr_block                                     = "192.123.1.0/24"
      + enable_dns64                                   = false
      + enable_resource_name_dns_a_record_on_launch    = false
      + enable_resource_name_dns_aaaa_record_on_launch = false
```

Video 53

Terraform Input Variables Length and SubString Functions

The length function can be found here. The use of length can be seen in the terraform console as given below:

```
> length("hello!")
6
> length([1,2,5,2,7,2,3])
7
> length({first = "value1", sec = "value2"})
2
```

The substring extracts a substring from a given string by offset and (maximum) length. substr(string, offset, length). It is to be noted that The offset in substr specifies the position from which the substring starts. The use case is shown below:

```
> substr("what up!", 3, 5)
"t up!"
> substr("hello fazle!", 2, 5)
"llo f"
>
```

Video 54

Terraform Input Variables Custom Validation Rules

We can add a validation block which can take condition and error_message arguments to verify the code:

```
variable "subnet_cidr_block" {
  description = "This the cidr block for my subnet"
  type        = string
  default     = "192.123.1.0/24"
  validation {
                  condition     =     length(var.subnet_cidr_block)>7     &&
substr(var.subnet_cidr_block,0,2) == "182"
    error_message = "The given subnet id has errors"
  }
}
```

```
Error: Invalid value for variable

  on variable.tf line 7:
   7: variable "subnet_cidr_block" {
     ├────────────────
     │ var.subnet_cidr_block is "192.123.1.0/24"

The given subnet id has errors

This was checked by the validation rule at variable.tf:11,3-13.
```

Video 55

Terraform Input Variables Sensitive Variables

When you use sensitive variables in your Terraform configuration, you can use them as you would any other variable.

Terraform will redact these values in command output and log files, and raise an error when it detects that they will be exposed in other ways.

```
variable "vpc_cidr_block" {
  description = "This is the cidr block for my vpc"
  type        = string
  #default     = "192.123.0.0/16"
  sensitive = true
}
```

```
# aws_vpc.fazle-vpc will be created
+ resource "aws_vpc" "fazle-vpc" {
    + arn                              = (known after apply)
    + cidr_block                       = (sensitive value)
    + default_network_acl_id           = (known after apply)
    + default_route_table_id           = (known after apply)
    + default_security_group_id        = (known after apply)
    + dhcp_options_id                  = (known after apply)
    + enable_dns_hostnames             = (known after apply)
    + enable_dns_support               = true
```

- ❖ It is to be noted that the Terraform state file contains values for these sensitive variables terraform.tfstate. You must keep your state file secure to avoid exposing this data.

Video 56

Terraform Input Variables Variable Definition Precedence

The above mechanisms for setting variables can be used together in any combination. If the same variable is assigned multiple values, Terraform uses the last value it finds, overriding any previous values. Note that the same variable cannot be assigned multiple values within a single source.

Terraform loads variables in the following order, with later sources taking precedence over earlier ones:

- Environment variables
- The terraform.tfvars file, if present.
- The terraform.tfvars.json file, if present.
- Any *.auto.tfvars or *.auto.tfvars.json files, processed in lexical order of their filenames.
- Any -var and -var-file options on the command line, in the order they are provided. (This includes variables set by a Terraform Cloud workspace.)

We can find more here.

Video 57

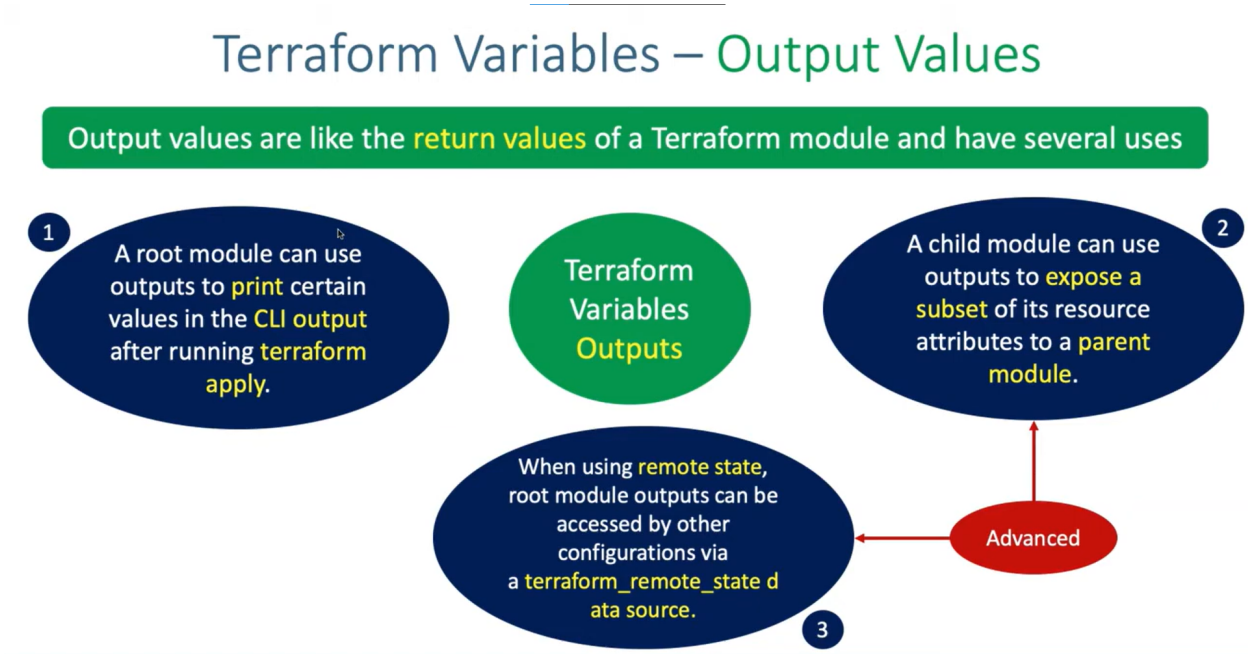
Terraform File Function

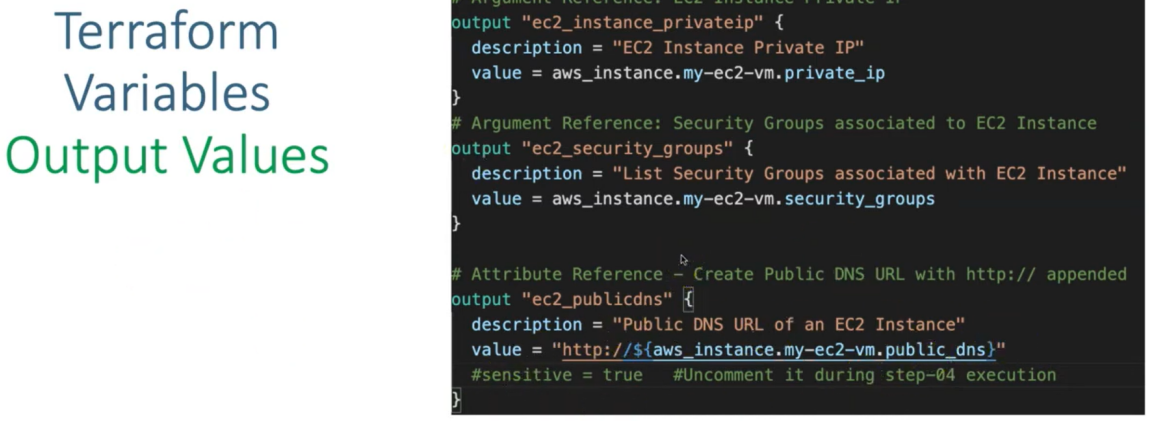
We can read about the file function here. In short, file reads the contents of a file at the given path and returns them as a string. The file function has been used in the following way:

```
resource "aws_instance" "fazle-EC2" {
  ami                         = "ami-08a52ddb321b32a8c"
  subnet_id                   = aws_subnet.fazle-subnet.id
  instance_type               = "t2.micro"
  associate_public_ip_address = "true"
  vpc_security_group_ids = [aws_security_group.fazle_SG_EC2.id]
  user_data                   = file("user_data.sh")
  user_data_replace_on_change = true

  tags = {
    Name = "Fazle-EC2"
  }
}
```

Video 58

Terraform Output Values Introduction






```
# Define Output Values
# Attribute Reference: EC2 Instance Public IP
output "ec2_instance_publicip" {
  description = "EC2 Instance Public IP"
  value = aws_instance.my-ec2-vm.public_ip
}

# Argument Reference: EC2 Instance Private IP
output "ec2_instance_privateip" {
  description = "EC2 Instance Private IP"
  value = aws_instance.my-ec2-vm.private_ip
}
# Argument Reference: Security Groups associated to EC2 Instance
output "ec2_security_groups" {
  description = "List Security Groups associated with EC2 Instance"
  value = aws_instance.my-ec2-vm.security_groups
}

# Attribute Reference - Create Public DNS URL with http:// appended
output "ec2_publicdns" {
  description = "Public DNS URL of an EC2 Instance"
  value = "http://${aws_instance.my-ec2-vm.public_dns}"
  #sensitive = true   #Uncomment it during step-04 execution
}
```

Terraform Variables - Output Values

```
aws_instance.my-ec2-vm: Creation complete after 24s [id=i-0406c6733

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

ec2_instance_privateip = "172.31.78.184"
ec2_instance_publicip = "3.235.244.111"
ec2_publicdns = "http://ec2-3-235-244-111.compute-1.amazonaws.com"
ec2_security_groups = toset([
  "vpc-ssh",
  "vpc-web",
])
```

Video 59

Terraform Outputs Implementation Part 1

```
output "ec2_public_ip" {
  description = "This is the value for the public IP of the EC2 instance"
  value = aws_instance.fazle-EC2.public_ip
}

output "ec2_private_ip" {
  description = "This is the value for the private IP of the EC2 instance"
  value = aws_instance.fazle-EC2.private_ip
}

output "ec2_SG" {
   description = "This is the value for the security group of the EC2 instance"
  value = aws_instance.fazle-EC2.vpc_security_group_ids
}
```

```
Outputs:

ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = "35.171.203.129"
PS D:\Terraform\Files\Video Resource Files\video59>
```

Video 60

Terraform Outputs Implementation Part 2

The terraform.tfstate file stores all the output may it be sensitive or not it will store in a redacted manner. We can always get the output anytime we want in the following manner.

```
Outputs:

ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = "35.171.203.129"
PS D:\Terraform\Files\Video Resource Files\video59> terraform output
ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = "35.171.203.129"
PS D:\Terraform\Files\Video Resource Files\video59> terraform output private_ip

  Error: Output "private_ip" not found

  The output variable requested could not be found in the state file. If you recently added this to your configuration, be sure to run `terraform apply`, since the
  state won't be updated with new output variables until that command is run.

PS D:\Terraform\Files\Video Resource Files\video59> terraform output ec2_private_ip
"192.168.56.13"
PS D:\Terraform\Files\Video Resource Files\video59>
```

If the output is labeled sensitive then it will give that in the following way:

```
PS D:\Terraform\Files\Video Resource Files\video59> terraform output
ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = "35.171.203.129"
PS D:\Terraform\Files\Video Resource Files\video59> terraform apply -auto-approve
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-01c396d3a38601d0e]
aws_internet_gateway.fazle_IGW: Refreshing state... [id=igw-0edda6a4cf0069617]
aws_subnet.fazle-subnet: Refreshing state... [id=subnet-02211075993643188]
aws_security_group.fazle_SG_EC2: Refreshing state... [id=sg-0012e15b6cb780814]
aws_route_table.Public_RT: Refreshing state... [id=rtb-0e58aed0ec771f692]
aws_instance.fazle-EC2: Refreshing state... [id=i-0a048946462136370]
aws_route_table_association.PRT_Associate: Refreshing state... [id=rtbassoc-0c3548d4b11378a9a]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

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

Outputs:

ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = <sensitive>
PS D:\Terraform\Files\Video Resource Files\video59>
```

But if we take the output from the tfstate file it will give us the non-redacted value:

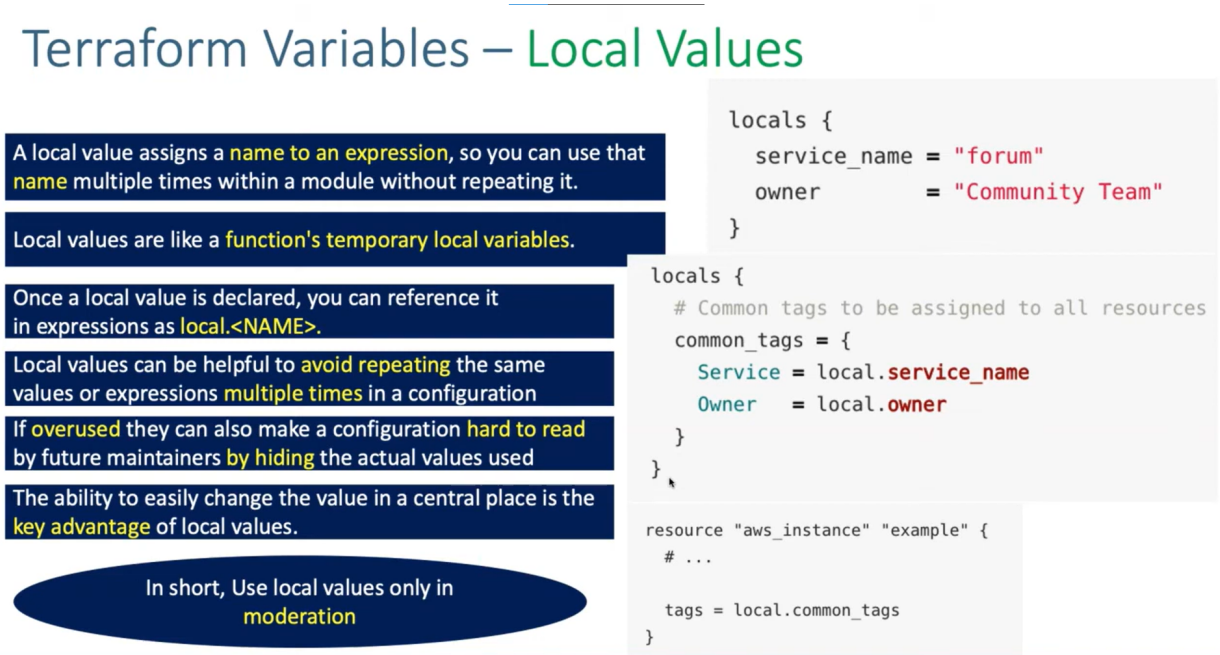
```
PS D:\Terraform\Files\Video Resource Files\video59> terraform output
ec2_SG = toset([
  "sg-0012e15b6cb780814",
])
ec2_private_ip = "192.168.56.13"
ec2_public_ip = <sensitive>
PS D:\Terraform\Files\Video Resource Files\video59> terraform output ec2_public_ip
"35.171.203.129"
PS D:\Terraform\Files\Video Resource Files\video59>
```

The output can also be extracted in a JSON format:

```
PS D:\Terraform\Files\Video Resource Files\video59> terraform output -json
{
  "ec2_SG": {
    "sensitive": false,
    "type": [
      "set",
      "string"
    ],
    "value": [
      "sg-0012e15b6cb780814"
    ]
  },
  "ec2_private_ip": {
    "sensitive": false,
    "type": "string",
    "value": "192.168.56.13"
  },
  "ec2_public_ip": {
    "sensitive": true,
    "type": "string",
    "value": "35.171.203.129"
  }
}
PS D:\Terraform\Files\Video Resource Files\video59>
```

Video 61

Terraform Local Values Introduction



Terraform Variables – Local Values

```
# Create S3 Bucket - with Input Variables & Local Values
locals {
  bucket-name = "${var.app_name}-${var.environment_name}-bucket" # Complex expression
}

resource "aws_s3_bucket" "mys3bucket" {
  bucket = local.bucket-name # Simplifed to use in many places
  acl = "private"
  tags = {
    Name = local.bucket-name # Simplifed to use in many places
    Environment = var.environment_name
  }
}
```

We can apply locals in the following way:

```
locals {
  bucket-name = "this-is-fazle-bucket-1"
}

resource "aws_s3_bucket" "fazle-bucket" {
  bucket = local.bucket-name
}
```

Video 63

Terraform Datasources Introduction

Terraform Datasources

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.

Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a data block

Each data resource is associated with a single data source, which determines the kind of object (or objects) it reads and what query constraint arguments are available

Data resources have the same dependency resolution behavior as defined for managed resources. Setting the depends_on meta-argument within data blocks defers reading of the data source until after all changes to the dependencies have been applied.

```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux" {
  most_recent      = true
  owners           = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*"]
  }
  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }
  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
  filter {
    name   = "architecture"
    values = ["x86_64"]
  }
}
```

Terraform Datasources

We can refer the data resource in a resource as depicted

Meta-Arguments for Datasources

```
# Create EC2 Instance - Amazon Linux
resource "aws_instance" "my-ec2-vm" {
  ami           = data.aws_ami.amzlinux.id
  instance_type = var.ec2_instance_type
  key_name      = "terraform-key"
  user_data = file("apache-install.sh")
  vpc_security_group_ids = [aws_security_group
  tags = {
    "Name" = "amz-linux-vm"
  }
}
```

Data resources support the provider meta-argument as defined for managed resources, with the same syntax and behavior.

Data resources do not currently have any customization settings available for their lifecycle, but the lifecycle nested block is reserved in case any are added in future versions.

Data resources support count and for_each meta-arguments as defined for managed resources, with the same syntax and behavior.
Each instance will separately read from its data source with its own variant of the constraint arguments, producing an indexed result.

Video 64

Create a data resource to get latest AMI ID

We can follow this document to get the latest amazon ami machine. It is implemented below:

```
data "aws_ami" "latest_amazon_linux2" {
  most_recent = true
  owners      = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-ebs"]
  }
  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}

resource "aws_instance" "fazle-EC2" {
  ami = data.aws_ami.latest_amazon_linux2.id
  subnet_id = aws_subnet.fazle-subnet.id
  instance_type = "t2.micro"
}
```

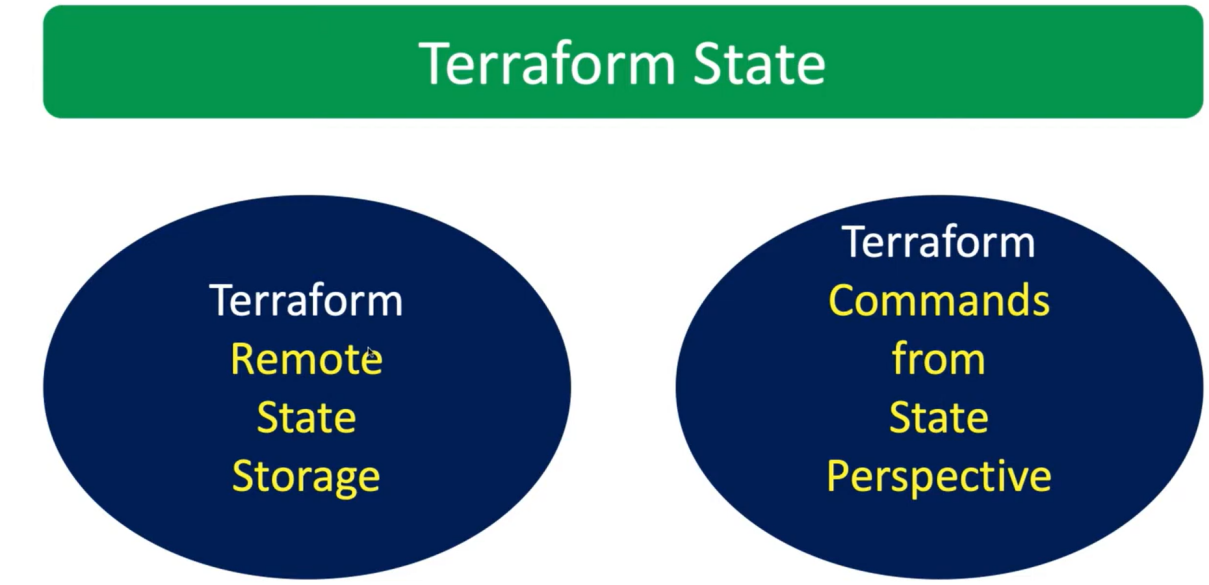
Video 65

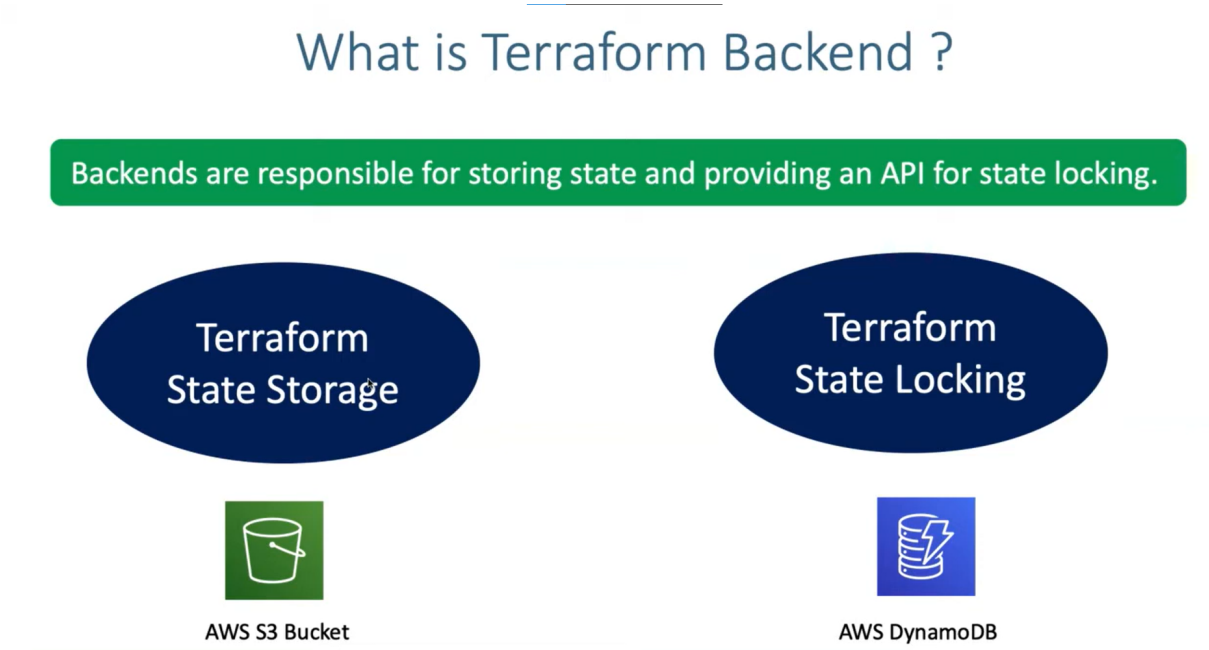
Verify the AMI ID by executing terraform plan

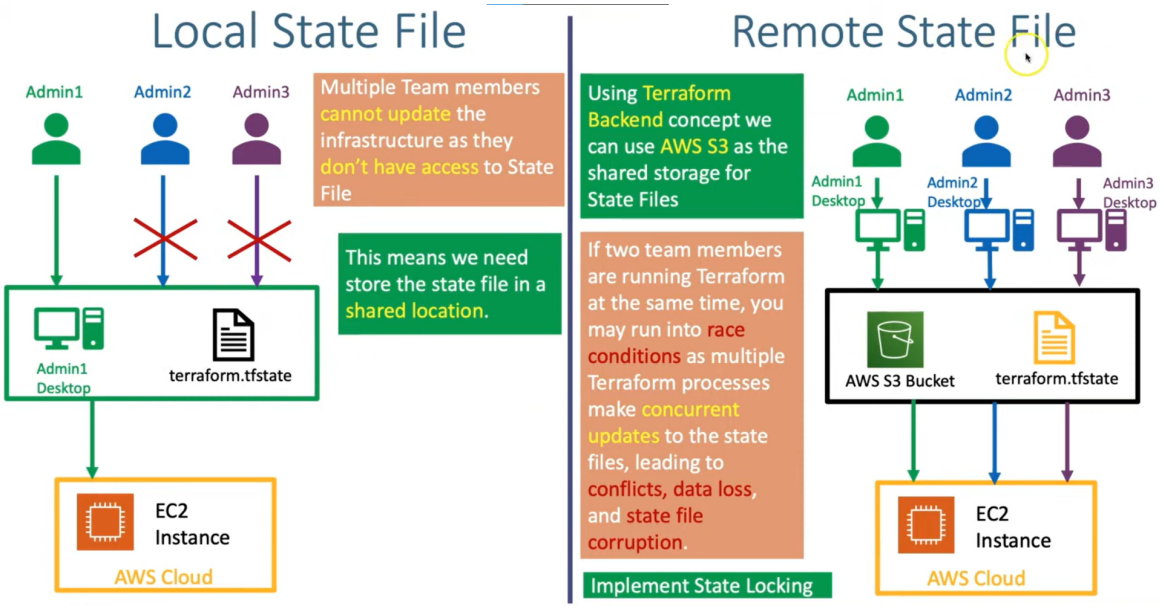
In order to get the latest used AMI we need to make sure that we have properly filtered our AMI enough that it generates the desired AMI instance. If it doesn't we need to keep on filtering the instance till we get the desired instance. It is best to use most_recent = true.

Video 66

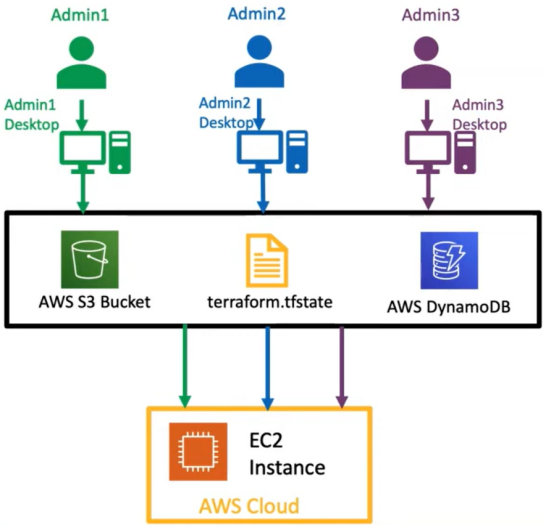
Terraform STATE Introduction






Local State File
Admin1
Admin2
Admin3
Admin1 Desktop
terraform.tfstate
EC2 Instance
AWS Cloud
Multiple Team members cannot update the infrastructure as they don't have access to State File
This means we need store the state file in a shared location.
Remote State File
Using Terraform Backend concept we can use AWS S3 as the shared storage for State Files
If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.
Implement State Locking
Admin1
Admin2
Admin3
Admin1 Desktop
Admin2 Desktop
Admin3 Desktop
AWS S3 Bucket
terraform.tfstate
EC2 Instance
AWS Cloud

Terraform Remote State File with State Locking



Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

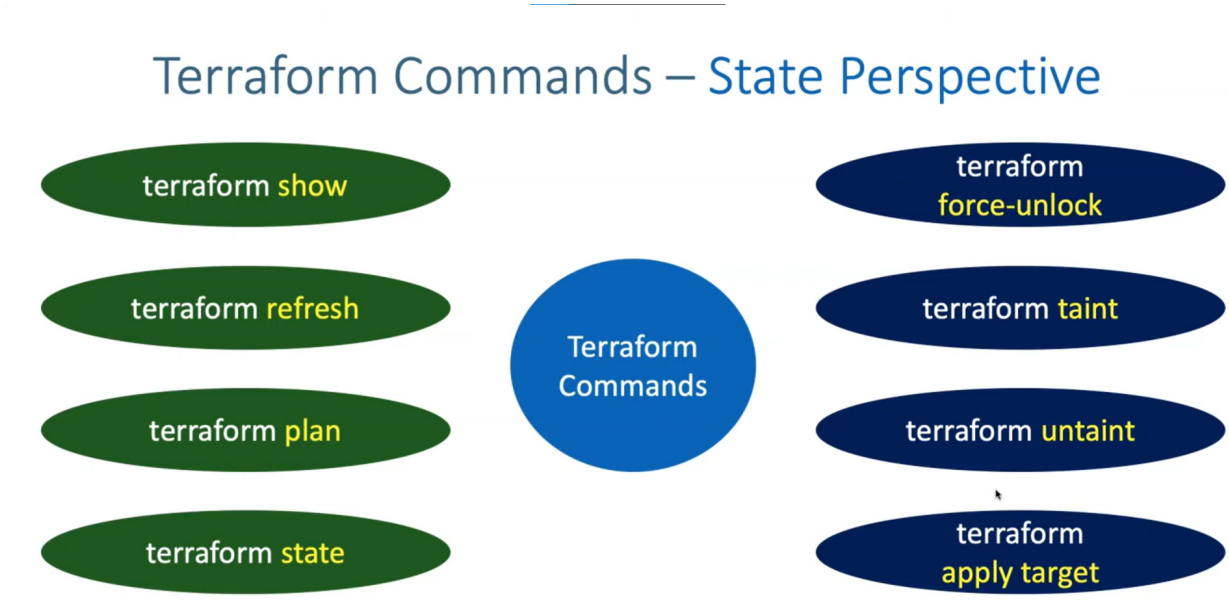
Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Terraform Remote State File with State Locking

Terraform State Storage to Remote Backend

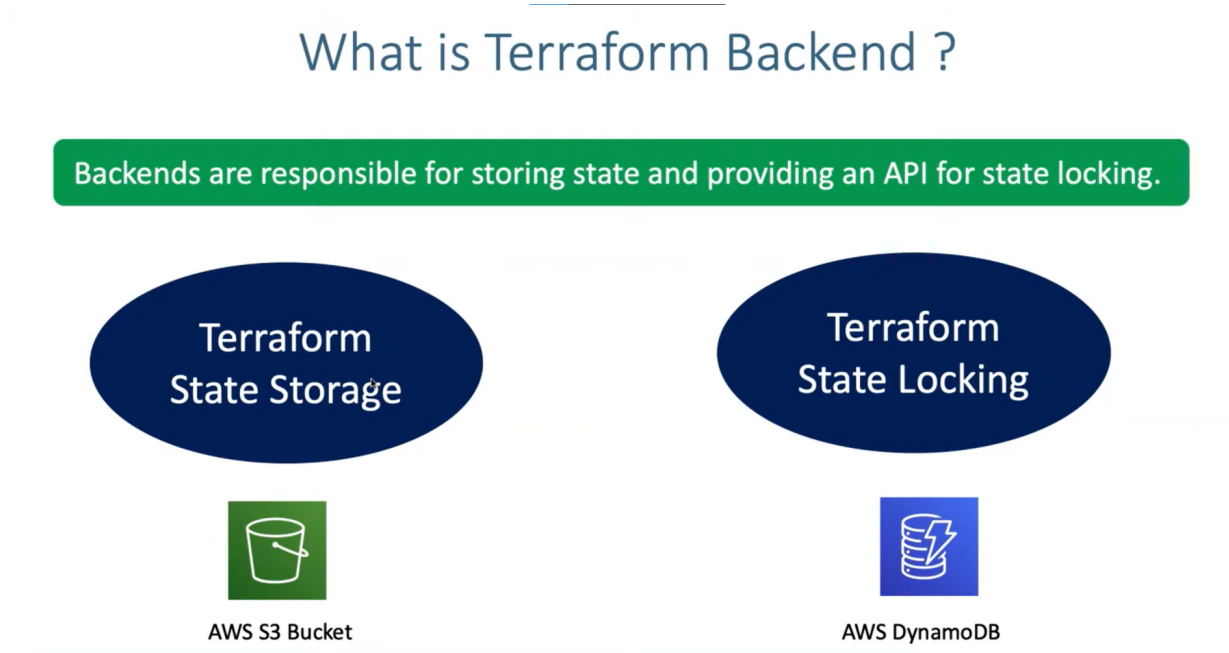
Terraform State Locking

```
# Terraform Block
terraform {
  required_version = "~> 0.14" # which means any ve
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  # Adding Backend as S3 for Remote State Storage
  backend "s3" {
    bucket = "terraform-stacksimplify"
    key    = "dev/terraform.tfstate"
    region = "us-east-1"

    # Enable during Step-09
    # For State Locking
    dynamodb_table = "terraform-dev-state-table"
  }
}
```

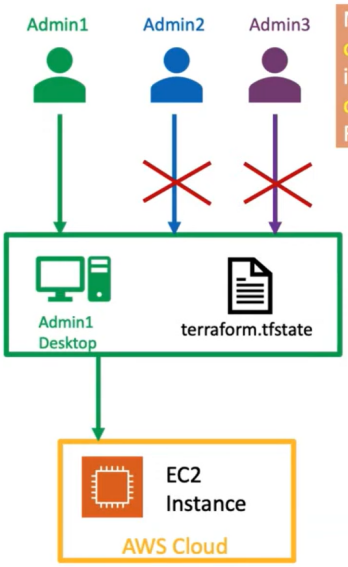


Video 67

Terraform Remote State Storage Introduction



Local State File

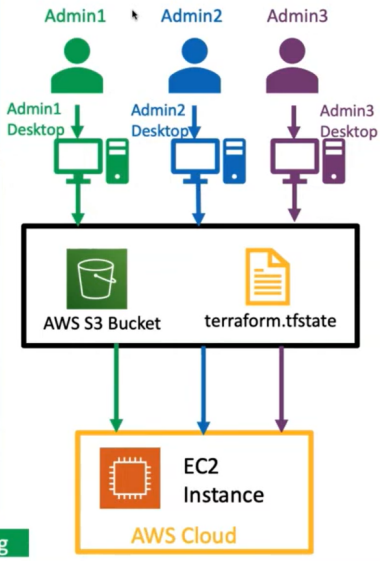


Multiple Team members cannot update the infrastructure as they don't have access to State File

This means we need store the state file in a shared location.

Remote State File

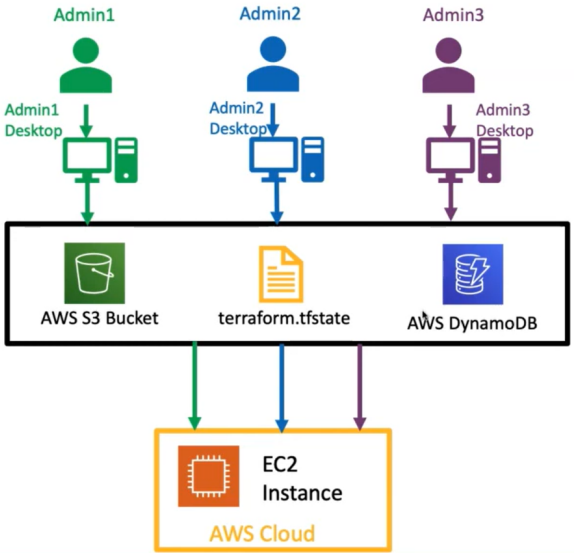
Using Terraform Backend concept we can use AWS S3 as the shared storage for State Files



If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

Implement State Locking

Terraform Remote State File with State Locking



Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could write state.

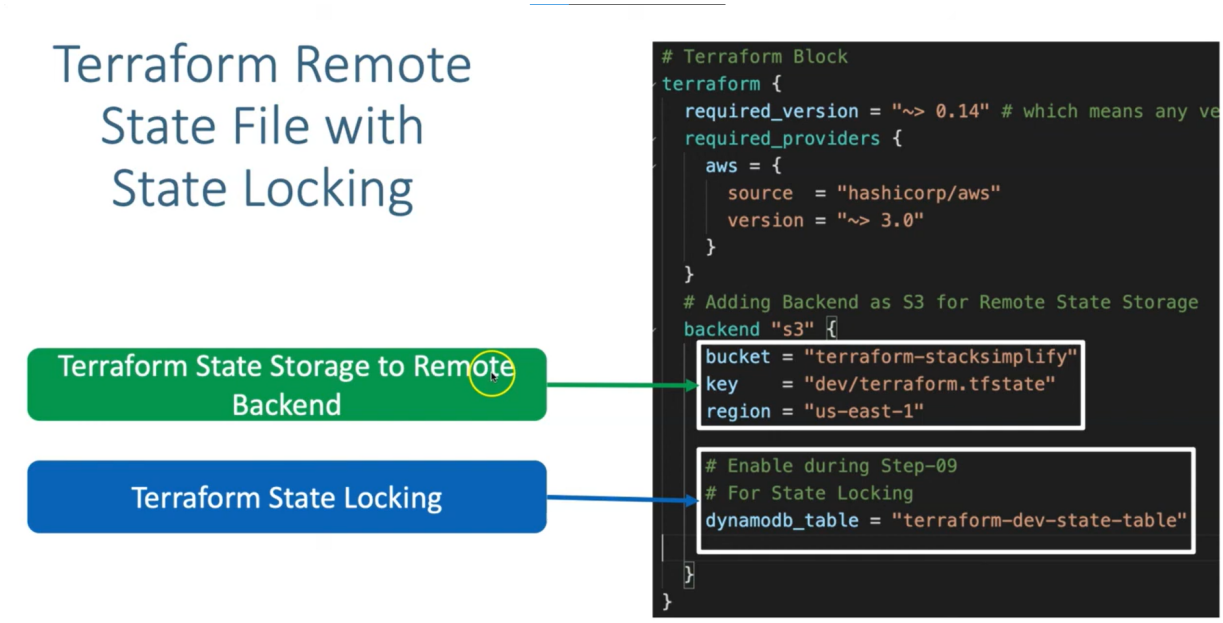
If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.



Video 68

Configure AWS S3 as Terraform Backend for Remote State Storage

First we need to create an S3 bucket. The bucket may have a folder for storing the tfstate file if needed. The backend for the tfstate will refer to the s3 and the specific path which is denoted as key. If any change is made to the code which brings change to the tfstate file then the tfstate file will be modified into a new version. The code is shown below:

```
terraform {
  required_version = "~>1.5.4"
  required_providers {
    aws = {
      version = "~>3.11.0"
      source  = "hashicorp/aws"
    }
  }
  backend "s3" {
    bucket = "fazle-test-bucket"
    key    = "tfstate/fazle.tfstate"
    region = "us-east-1"
  }
}
```

Video 69

Implement State Locking using AWS DynamoDB

We need to create a Dynamodb table after the work from the previous video. We need to name the encryption as “LockID”. DynamoDB will help to lock the file during Terraform plan, apply and destroy. The code is shown below:

```
terraform {
  required_version = "~>1.5.4"
  required_providers {
    aws = {
      version = "~>3.11.0"
      source  = "hashicorp/aws"
    }
  }
  backend "s3" {
    bucket = "fazle-test-bucket"
    key    = "tfstate/terraform.tfstate"
    region = "us-east-1"
    //Lock file using DynamoDB
    dynamodb_table = "fazle-stateLock-test"
  }
}
```

aws_vpc.fazle-vpc: Creating...
aws_vpc.fazle-vpc: Still creating... [10s elapsed]
aws_vpc.fazle-vpc: Creation complete after 15s [id=vpc-07fd7688b6dfb9638]
Releasing state lock. This may take a few moments...

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

BJIT@11729-fazle-mubin MINGW64 /d/Terraform/Files/Video Resource Files/video69
$ Terraform destroy -auto-approve
Acquiring state lock. This may take a few moments...
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-07fd7688b6dfb9638]

Video 71

Learn about terraform show command

- The terraform show command is used to provide human-readable output from a state or plan file.
- This can be used to inspect a plan to ensure that the planned operations are expected, or to inspect the current state as
- Terraform plan output files are binary files. We can read them using terraform show command

```
$ terraform show tfplan.out

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:

  # aws_vpc.fazle-vpc will be created
  + resource "aws_vpc" "fazle-vpc" {
      + arn                              = (known after apply)
      + assign_generated_ipv6_cidr_block = false
      + cidr_block                       = "192.169.0.0/16"
      + default_network_acl_id           = (known after apply)
      + default_route_table_id           = (known after apply)
      + default_security_group_id        = (known after apply)
      + dhcp_options_id                  = (known after apply)
      + enable_classiclink               = (known after apply)
      + enable_classiclink_dns_support   = (known after apply)
      + enable_dns_hostnames             = (known after apply)
      + enable_dns_support               = true
      + id                               = (known after apply)
      + instance_tenancy                 = "default"
      + ipv6_association_id              = (known after apply)
      + ipv6_cidr_block                  = (known after apply)
      + main_route_table_id              = (known after apply)
      + owner_id                         = (known after apply)
      + tags                             = {
          + "Name" = "Fazle-vpc"
```

```
        }
    }

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

```
$ terraform show
The state file is empty. No resources are represented.
```

After Terraform apply the state file will be made. Then we can use terraform show.

```
$ terraform show
# aws_vpc.fazle-vpc:
resource "aws_vpc" "fazle-vpc" {
        arn                                                                =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-07906d7cf3d4a6d53"
    assign_generated_ipv6_cidr_block = false
    cidr_block                       = "192.169.0.0/16"
    default_network_acl_id           = "acl-0c0caefe63260e38a"
    default_route_table_id           = "rtb-0e3000a13fba49c4e"
    default_security_group_id        = "sg-0fd5b3bae2d0cefa1"
    dhcp_options_id                  = "dopt-0b9ceb20cd0988fd9"
    enable_classiclink               = false
    enable_classiclink_dns_support   = false
    enable_dns_hostnames             = false
    enable_dns_support               = true
    id                               = "vpc-07906d7cf3d4a6d53"
    instance_tenancy                 = "default"
    main_route_table_id              = "rtb-0e3000a13fba49c4e"
    owner_id                         = "430704357939"
    tags                             = {
        "Name" = "Fazle-vpc"
    }
}
```

Video 72

Learn about terraform refresh command

- This commands comes under Terraform Inspecting State
- Understanding terraform refresh clears a lot of doubts in our mind and terraform state file and state feature
- The terraform refresh command is used to reconcile the state Terraform knows about (via its state file) with the real-world infrastructure.
- This can be used to detect any drift from the last-known state, and to update the state file.
- This does not modify infrastructure, but does modify the state file. If the state is changed, this may cause changes to occur during the next plan or apply.
- **terraform refresh:** Update local state file against real resources in cloud
- **Desired State:** Local Terraform Manifest (All *.tf files)
- **Current State:** Real Resources present in your cloud
- **Command Order of Execution:** refresh, plan, make a decision, apply
- Why? Lets understand that in detail about this order of execution

```
      + tags                                 = {
          + "Name" = "Fazle-vpc"
        }
    }

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

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

aws_vpc.fazle-vpc: Creating...
aws_vpc.fazle-vpc: Still creating... [10s elapsed]
aws_vpc.fazle-vpc: Creation complete after 15s [id=vpc-027260947a6d89d05]

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

```
BJIT@11729-fazle-mubin    MINGW64    /d/Terraform/Files/Video    Resource
Files/video72
$ Terraform refresh
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-027260947a6d89d05]

BJIT@11729-fazle-mubin    MINGW64    /d/Terraform/Files/Video    Resource
Files/video72
$ terraform plan
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-027260947a6d89d05]

Terraform used the selected providers to generate the following execution
plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc will be updated in-place
  ~ resource "aws_vpc" "fazle-vpc" {
        id                               = "vpc-027260947a6d89d05"
      ~ tags                             = {
            "Name"  = "Fazle-vpc"
          - "Owner" = "Fazle" -> null
        }
        # (14 unchanged attributes hidden)
    }

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

Video 73

Learn about terraform state command List, Show and mv

Terraform state list and Terraform state show

- These two commands comes under Terraform Inspecting State
- terraform state list: This command is used to list resources within a Terraform state.
- terraform state show: This command is used to show the attributes of a single resource in the Terraform state.

```
terraform state list
```

```
aws_vpc.fazle-vpc
PS D:\Terraform\Files\Video Resource Files\video73> terraform state show
aws_vpc.fazle-vpc
# aws_vpc.fazle-vpc:
resource "aws_vpc" "fazle-vpc" {
    arn                                  =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-0644b7bb063d3855c"
    assign_generated_ipv6_cidr_block = false
    cidr_block                           = "192.169.0.0/16"
    default_network_acl_id               = "acl-0fbd948b3acbff468"
    default_security_group_id            = "sg-007aaa80720423d65"
    enable_classiclink                   = false
    enable_classiclink_dns_support       = false
    enable_dns_hostnames                 = false
    enable_dns_support                   = true
    id                                   = "vpc-0644b7bb063d3855c"
    instance_tenancy                     = "default"
    main_route_table_id                  = "rtb-0183bf0e533284c0a"
    owner_id                             = "430704357939"
    tags                                 = {
        "Name" = "Fazle-vpc"
    }
}
```

Terarform mv

- This commands comes under Terraform Moving Resources
- This command will move an item matched by the address given to the destination address.
- This command can also move to a destination address in a completely different state file
- Very dangerous command
- Very advanced usage command
- Results will be unpredictable if concept is not clear about terraform state files mainly desired state and current state.
- Try this in production environments, only when everything worked well in lower environments.

```
terraform state mv aws_vpc.fazle-vpc aws_vpc.fazle-vpc-new
Move "aws_vpc.fazle-vpc" to "aws_vpc.fazle-vpc-new"
Successfully moved 1 object(s).
PS D:\Terraform\Files\Video Resource Files\video73> terraform state list
```

```
aws_vpc.fazle-vpc-new
PS D:\Terraform\Files\Video Resource Files\video73> terraform plan
aws_vpc.fazle-vpc-new: Refreshing state... [id=vpc-0176e3fb6c022d9cf]

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

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc will be created
  + resource "aws_vpc" "fazle-vpc" {
      + arn                              = (known after apply)
      + assign_generated_ipv6_cidr_block = false
      + cidr_block                       = "192.169.0.0/16"
      + default_network_acl_id           = (known after apply)
      + default_route_table_id           = (known after apply)
      + default_security_group_id        = (known after apply)
      + dhcp_options_id                  = (known after apply)
      + enable_classiclink               = (known after apply)
      + enable_classiclink_dns_support   = (known after apply)
      + enable_dns_hostnames             = (known after apply)
      + enable_dns_support               = true
      + id                               = (known after apply)
      + instance_tenancy                 = "default"
      + ipv6_association_id              = (known after apply)
      + ipv6_cidr_block                  = (known after apply)
      + owner_id                         = (known after apply)
      + tags                             = {
          + "Name" = "Fazle-vpc"
        }
    }

  # aws_vpc.fazle-vpc-new will be destroyed
  # (because aws_vpc.fazle-vpc-new is not in configuration)
  - resource "aws_vpc" "fazle-vpc-new" {
      - arn                              =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-0176e3fb6c022d9cf" -> null
      - assign_generated_ipv6_cidr_block = false -> null
```

```
      - cidr_block                          = "192.169.0.0/16" -> null
      - default_network_acl_id              = "acl-0e9e9da13d87c9866" -> null
      - default_route_table_id              = "rtb-0e9d1e6ed10397e2f" -> null
      - default_security_group_id           = "sg-08f0a1fccec653696" -> null
      - dhcp_options_id                     = "dopt-0b9ceb20cd0988fd9" ->
null
      - enable_classiclink                  = false -> null
      - enable_classiclink_dns_support      = false -> null
      - enable_dns_hostnames                = false -> null
      - enable_dns_support                  = true -> null
      - id                                  = "vpc-0176e3fb6c022d9cf" -> null
      - instance_tenancy                    = "default" -> null
      - main_route_table_id                 = "rtb-0e9d1e6ed10397e2f" -> null
      - owner_id                            = "430704357939" -> null
      - tags                                = {
          - "Name" = "Fazle-vpc"
        } -> null
    }

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

Video 74

Learn about terraform state command rm and replace provider

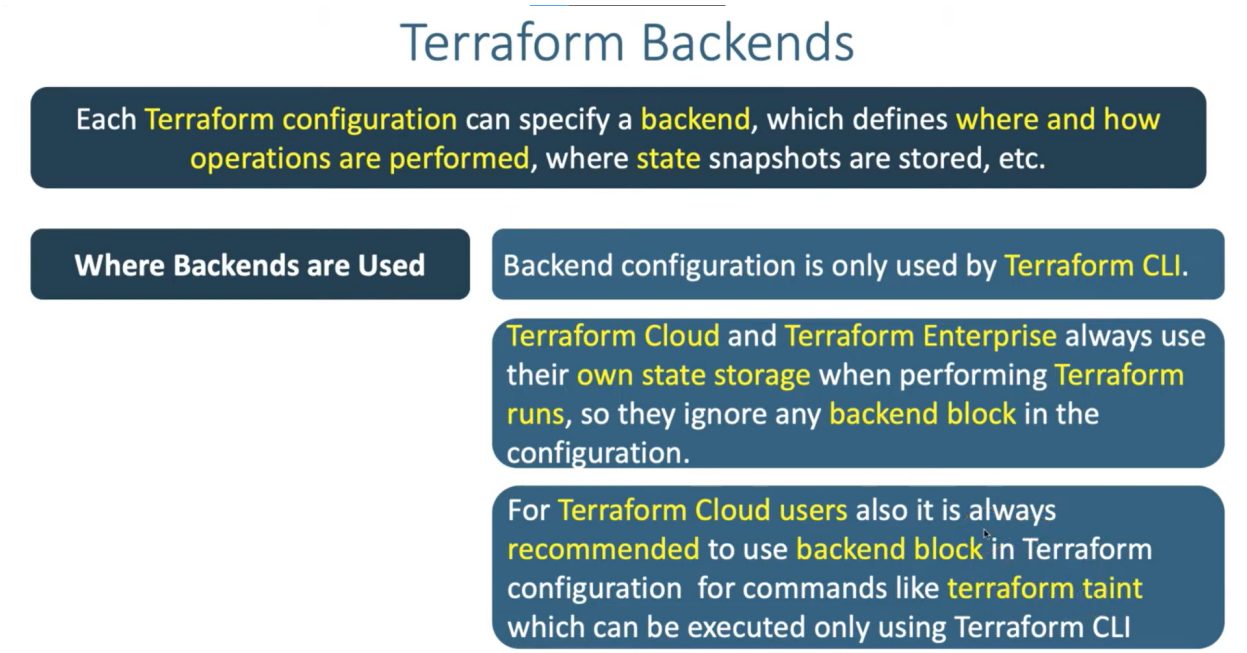
- This commands comes under Terraform Moving Resources
- The terraform state rm command is used to remove items from the Terraform state.
- This command can remove single resources, single instances of a resource, entire modules, and more.
- The removed resource will not be destroyed but will not be controlled by Terraform.

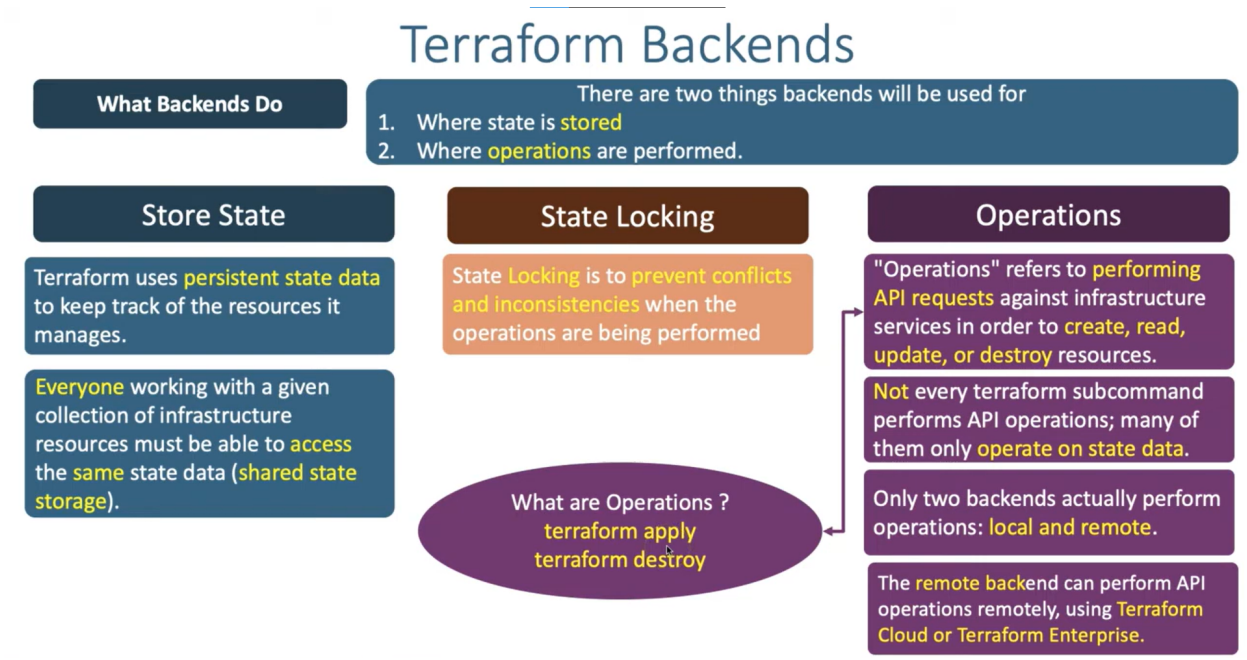
```
PS D:\Terraform\Files\Video Resource Files\video74> terraform state rm
-dry-run aws_vpc.fazle-vpc-
Would have removed nothing.
PS D:\Terraform\Files\Video Resource Files\video74> terraform state rm
-dry-run aws_vpc.fazle-vpc
Would remove aws_vpc.fazle-vpc
PS D:\Terraform\Files\Video Resource Files\video74> terraform state rm
aws_vpc.fazle-vpc
```

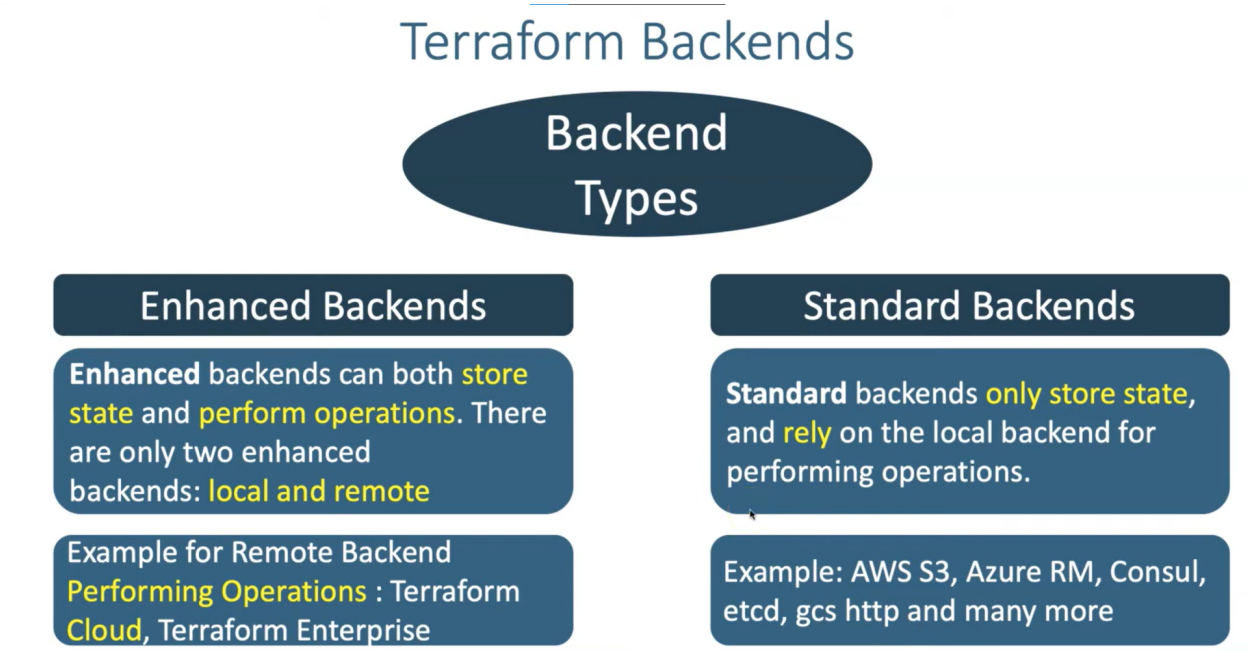
```
Removed aws_vpc.fazle-vpc
Successfully removed 1 resource instance(s).
```

Video 70

Step 04 Understand more about Terraform Backends







Video 75

Learn about terraform state command pull, push and force unlock

State Pull

- This command comes under Terraform Disaster Recovery Concept
- terraform state pull:
- The terraform state pull command is used to manually download and output the state from remote state.
- This command also works with local state.
- This command will download the state from its current location and output the raw format to stdout.
- terraform state push: The terraform state push command is used to manually upload a local state file to remote state.

```
PS D:\Terraform\Files\Video Resource Files\video74> terraform state pull
{
  "version": 4,
  "terraform_version": "1.5.4",
  "serial": 6,
  "lineage": "1d474f52-5fed-43ae-ed6c-4c5e7711c914",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```

Terraform force-unlock

- This command comes under Terraform Disaster Recovery Concept
- Manually unlock the state for the defined configuration.
- This will not modify your infrastructure.
- This command removes the lock on the state for the current configuration.
- The behavior of this lock is dependent on the backend (aws s3 with dynamodb for state locking etc) being used.
- Important Note: Local state files cannot be unlocked by another process.

```
# Manually Unlock the State
terraform force-unlock LOCK_ID
```

Video 76

Terraform taint and untaint

- These commands comes under Terraform Forcing Re-creation of Resources
- When a resource declaration is modified, Terraform usually attempts to update the existing resource in place (although some changes can require destruction and re-creation, usually due to upstream API limitations).
- Example: A virtual machine that configures itself with cloud-init on startup might no longer meet your needs if the cloud-init configuration changes.
- terraform taint: The terraform taint command manually marks a Terraform-managed resource as tainted, forcing it to be destroyed and recreated on the next apply.
- terraform untaint:
- The terraform untaint command manually unmarks a Terraform-managed resource as tainted, restoring it as the primary instance in the state.
- This reverses either a manual terraform taint or the result of provisioners failing on a resource.
- This command will not modify infrastructure, but does modify the state file in order to unmark a resource as tainted.
- Notice that terraform taint is **deprecated** from v0.15.2. Use terraform replace instead.

```
terraform taint aws_vpc.fazle-vpc-new
Resource instance aws_vpc.fazle-vpc-new has been marked as tainted.
```

```
PS D:\Terraform\Files\Video Resource Files\video76> terraform apply
-auto-approve
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-06efc6279fcdc7689]
aws_vpc.fazle-vpc-new: Refreshing state... [id=vpc-046c1284ca2ada47f]

Terraform used the selected providers to generate the following execution
plan. Resource
actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc-new is tainted, so must be replaced
-/+ resource "aws_vpc" "fazle-vpc-new" {
          ~ arn                                                =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-046c1284ca2ada47f" -> (known
after apply)
      ~ default_network_acl_id             = "acl-048584df53b664cc3" ->
(known after apply)
      ~ default_security_group_id           = "sg-0d562ddb3834b6df7" ->
(known after apply)
      ~ dhcp_options_id                    = "dopt-0b9ceb20cd0988fd9" ->
(known after apply)
      ~ enable_classiclink                 = false -> (known after apply)
      ~ enable_classiclink_dns_support     = false -> (known after apply)
      ~ enable_dns_hostnames               = false -> (known after apply)
      ~ id                                 = "vpc-046c1284ca2ada47f" ->
(known after apply)
      + ipv6_association_id                = (known after apply)
      ~ main_route_table_id                = "rtb-0fcbcbae70827556e" ->
(known after apply)
        tags                               = {
            "Name" = "Fazle-vpc-#2"
        }
        # (4 unchanged attributes hidden)
    }

Plan: 1 to add, 0 to change, 1 to destroy.
aws_vpc.fazle-vpc-new: Destroying... [id=vpc-046c1284ca2ada47f]
aws_vpc.fazle-vpc-new: Destruction complete after 2s
```

```
aws_vpc.fazle-vpc-new: Creating...
aws_vpc.fazle-vpc-new: Still creating... [10s elapsed]
aws_vpc.fazle-vpc-new:       Creation       complete       after       15s
[id=vpc-0a084128eacaa766f]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
PS  D:\Terraform\Files\Video  Resource  Files\video76>  terraform  untaint
aws_vpc.fazle-vpc-new
╷
│ Error: Resource instance is not tainted
│
│ Resource instance aws_vpc.fazle-vpc-new is not currently tainted, and so
it cannot be
│ untainted.
╵


PS  D:\Terraform\Files\Video  Resource  Files\video76>  terraform  taint
aws_vpc.fazle-vpc-new
Resource instance aws_vpc.fazle-vpc-new has been marked as tainted.
PS  D:\Terraform\Files\Video  Resource  Files\video76>  terraform  untaint
aws_vpc.fazle-vpc-new
Resource instance aws_vpc.fazle-vpc-new has been successfully untainted.
```

Instead of terraform taint we can use terraform replace argument. It does the same function but it can work with plan and apply. Moreover, when using replace we can actually see the change of the resource without having others code over it. It can be done in the following way:

```
PS  D:\Terraform\Files\Video  Resource  Files\video76>  terraform  plan
-replace="aws_vpc.fazle-vpc-new"
aws_vpc.fazle-vpc-new: Refreshing state... [id=vpc-0a084128eacaa766f]
aws_vpc.fazle-vpc: Refreshing state... [id=vpc-06efc6279fcdc7689]

Terraform used the selected providers to generate the following execution
plan. Resource
actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_vpc.fazle-vpc-new will be replaced, as requested
```

```
-/+ resource "aws_vpc" "fazle-vpc-new" {
            ~  arn                                                          =
"arn:aws:ec2:us-east-1:430704357939:vpc/vpc-0a084128eacaa766f"  ->  (known
after apply)
        ~ default_network_acl_id              = "acl-074b04b42354afe4c" ->
(known after apply)
        ~ default_route_table_id              = "rtb-084620915d0d126b2" ->
(known after apply)
        ~ default_security_group_id           = "sg-0d842afa8d3413d2c" ->
(known after apply)
        ~ dhcp_options_id                     = "dopt-0b9ceb20cd0988fd9" ->
(known after apply)
      ~ enable_classiclink                  = false -> (known after apply)
      ~ enable_classiclink_dns_support      = false -> (known after apply)
      ~ enable_dns_hostnames                = false -> (known after apply)
        ~ id                                      = "vpc-0a084128eacaa766f" ->
(known after apply)
      + ipv6_association_id                 = (known after apply)
      + ipv6_cidr_block                     = (known after apply)
        ~ main_route_table_id                     = "rtb-084620915d0d126b2" ->
(known after apply)
       ~ owner_id                           = "430704357939" -> (known after
apply)
        tags                                = {
            "Name" = "Fazle-vpc-#2"
        }
        # (4 unchanged attributes hidden)
    }

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

Video 77

Learn about terraform plan or apply target command

- The -target option can be used to focus Terraform's attention on only a subset of resources.
- Terraform Resource Targeting

- This targeting capability is provided for exceptional circumstances, such as recovering from mistakes or working around Terraform limitations.
- It is not recommended to use -target for routine operations, since this can lead to undetected configuration drift and confusion about how the true state of resources relates to configuration.
- Instead of using -target as a means to operate on isolated portions of very large configurations, prefer instead to break large configurations into several smaller configurations that can each be independently applied.
- Any dependency of the resource will also be applied as shown below:

```
PS D:\Terraform\Files\Video Resource Files\video77> terraform plan
-target="aws_subnet.fazle-subnet"

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:

  # aws_subnet.fazle-subnet will be created
  + resource "aws_subnet" "fazle-subnet" {
      + arn                             = (known after apply)
      + assign_ipv6_address_on_creation = false
      + availability_zone               = (known after apply)
      + availability_zone_id            = (known after apply)
      + cidr_block                      = "192.166.0.0/24"
      + id                              = (known after apply)
      + ipv6_cidr_block_association_id  = (known after apply)
      + map_public_ip_on_launch         = false
      + owner_id                        = (known after apply)
      + vpc_id                          = (known after apply)
    }

  # aws_vpc.fazle-vpc-new will be created
  + resource "aws_vpc" "fazle-vpc-new" {
      + arn                              = (known after apply)
      + assign_generated_ipv6_cidr_block = false
      + cidr_block                       = "192.166.0.0/16"
      + default_network_acl_id           = (known after apply)
      + default_route_table_id           = (known after apply)
      + default_security_group_id        = (known after apply)
```

```
      + dhcp_options_id                     = (known after apply)
      + enable_classiclink                  = (known after apply)
      + enable_classiclink_dns_support      = (known after apply)
      + enable_dns_hostnames                = (known after apply)
      + enable_dns_support                  = true
      + id                                  = (known after apply)
      + instance_tenancy                    = "default"
      + ipv6_association_id                 = (known after apply)
      + ipv6_cidr_block                     = (known after apply)
      + main_route_table_id                 = (known after apply)
      + owner_id                            = (known after apply)
      + tags                                = {
          + "Name" = "Fazle-vpc-#2"
        }
    }

Plan: 2 to add, 0 to change, 0 to destroy.
```

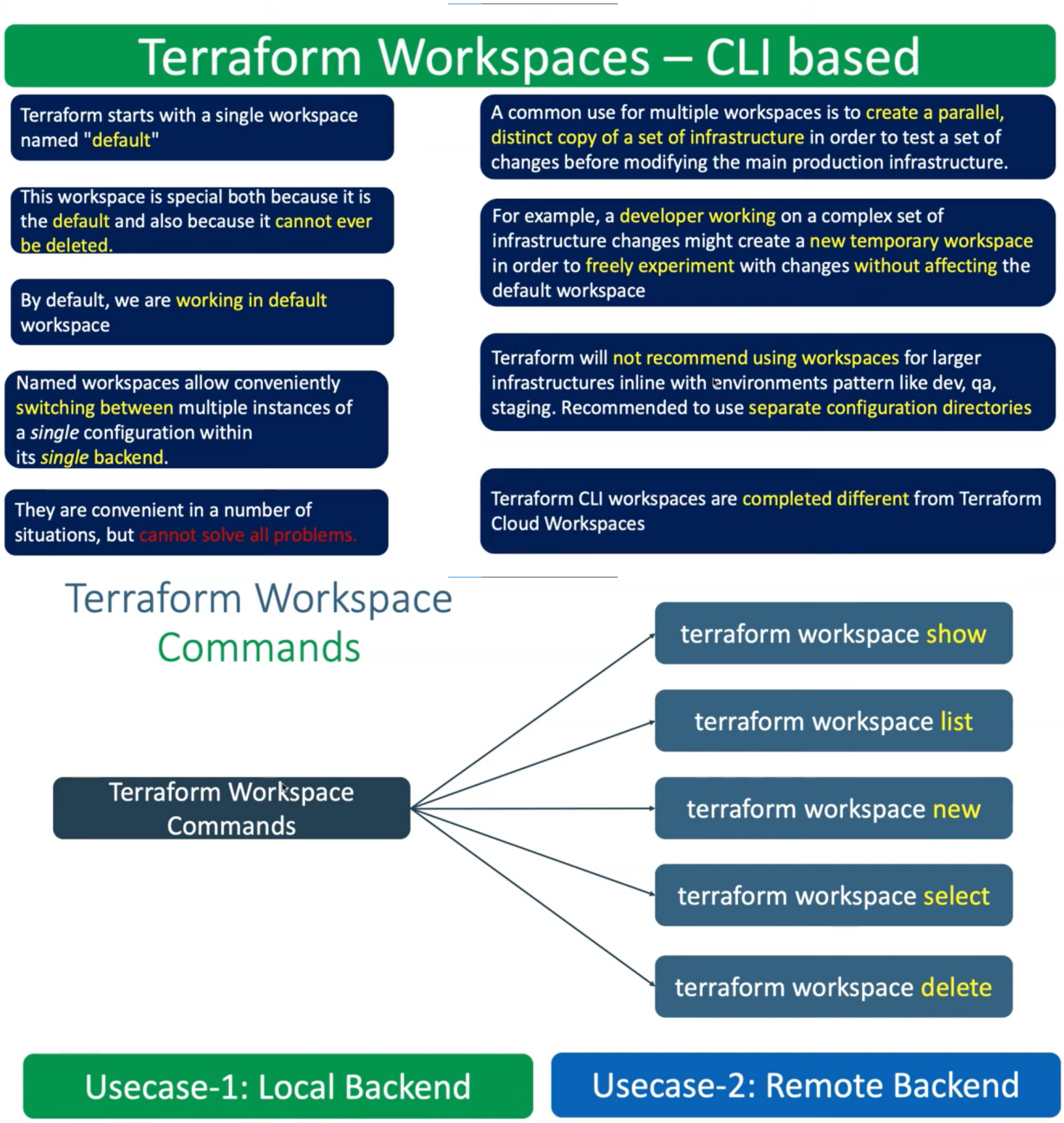
Video 78

Terraform State Commands Clean Up

- We should destroy all the resources which we have created.
- We should make sure that we have deleted all the files we don’t need.

Video 79

Introduction to Terraform Workspaces



Video 80

Review Create Terraform Manifests to support multiple workspaces

- Sub-folder we are working on: v1-local-backend
- Ideally, AWS don't allow to create a security group with same name twice.
- With that said, we need to change our security group names in our c2-security-groups.tf
- At the same time, just for reading convenience we can also have our EC2 Instance Name tag also updated inline with workspace name.
- What is ${terraform.workspace}? - It will get the workspace name
- Popular Usage-1: Using the workspace name as part of naming or tagging behavior
- Popular Usage-2: Referencing the current workspace is useful for changing behavior based on the workspace. For example, for non-default workspaces, it may be useful to spin up smaller cluster sizes.

```
# Change-1: Security Group Names
  name        = "vpc-ssh-${terraform.workspace}"
  name        = "vpc-web-${terraform.workspace}"

# Change-2: For non-default workspaces, it may be useful to spin up smaller cluster sizes.
  count = terraform.workspace == "default" ? 2 : 1
This will create 2 instances if we are in default workspace and in any other workspaces it will create 1 instance

# Change-3: EC2 Instance Name tag
    "Name" = "vm-${terraform.workspace}-${count.index}"

# Change-4: Outputs
  value = aws_instance.my-ec2-vm.*.public_ip
  value = aws_instance.my-ec2-vm.*.public_dns
You can create a list of all of the values of a given attribute for the items in the collection with a star. For instance, aws_instance.my-ec2-vm.*.id will be a list of all of the Public IP of the instances.
```

Video 81

Local Backend Create Resources in default workspace

```
PS D:\Terraform\Files\Video Resource Files\video81> terraform workspace
list
* default

PS D:\Terraform\Files\Video Resource Files\video81> terraform workspace
show
default
```

```
  # aws_instance.fazle-EC2[0] will be created
  + resource "aws_instance" "fazle-EC2" {
      + ami                          = "ami-08a52ddb321b32a8c"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
      + private_dns                  = (known after apply)
      + private_ip                   = (known after apply)
      + public_dns                   = (known after apply)
      + public_ip                    = (known after apply)
      + secondary_private_ips        = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
```

```
    + subnet_id                     = (known after apply)
    + tags                          = {
        + "Name"  = "Fazle-ec2"
        + "Owner" = "Fazle"
      }
    + tenancy                       = (known after apply)
    + volume_tags                   = (known after apply)
    + vpc_security_group_ids        = (known after apply)
  }

# aws_instance.fazle-EC2[1] will be created
+ resource "aws_instance" "fazle-EC2" {
    + ami                           = "ami-08a52ddb321b32a8c"
    + arn                           = (known after apply)
    + associate_public_ip_address   = (known after apply)
    + availability_zone             = (known after apply)
    + cpu_core_count                = (known after apply)
    + cpu_threads_per_core          = (known after apply)
    + get_password_data             = false
    + host_id                       = (known after apply)
    + id                            = (known after apply)
    + instance_state                = (known after apply)
    + instance_type                 = "t2.micro"
    + ipv6_address_count            = (known after apply)
    + ipv6_addresses                = (known after apply)
    + key_name                      = (known after apply)
    + outpost_arn                   = (known after apply)
    + password_data                 = (known after apply)
    + placement_group               = (known after apply)
    + primary_network_interface_id  = (known after apply)
    + private_dns                   = (known after apply)
    + private_ip                    = (known after apply)
    + public_dns                    = (known after apply)
    + public_ip                     = (known after apply)
    + secondary_private_ips         = (known after apply)
    + security_groups               = (known after apply)
    + source_dest_check             = true
    + subnet_id                     = (known after apply)
    + tags                          = {
        + "Name"  = "Fazle-ec2"
```

```
          + "Owner" = "Fazle"
        }
      + tenancy                      = (known after apply)
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = (known after apply)
    }
```

Video 82

Local Backend Create Resources in new workspace

```
PS D:\Terraform\Files\Video Resource Files\video82> terraform workspace
new test
Created and switched to workspace "test"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS D:\Terraform\Files\Video Resource Files\video82> terraform show
No state.
PS D:\Terraform\Files\Video Resource Files\video82> terraform workspace
show
test
PS D:\Terraform\Files\Video Resource Files\video82> terraform workspace
list
  default
* test
```

```
 # aws_instance.fazle-EC2[0] will be created
  + resource "aws_instance" "fazle-EC2" {
      + ami                          = "ami-08a52ddb321b32a8c"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)
      + cpu_core_count               = (known after apply)
      + cpu_threads_per_core         = (known after apply)
      + get_password_data            = false
      + host_id                      = (known after apply)
      + id                           = (known after apply)
```

```
      + instance_state               = (known after apply)
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + ipv6_addresses               = (known after apply)
      + key_name                     = (known after apply)
      + outpost_arn                  = (known after apply)
      + password_data                = (known after apply)
      + placement_group              = (known after apply)
      + primary_network_interface_id = (known after apply)
      + private_dns                  = (known after apply)
      + private_ip                   = (known after apply)
      + public_dns                   = (known after apply)
      + public_ip                    = (known after apply)
      + secondary_private_ips        = (known after apply)
      + security_groups              = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tags                         = {
          + "Name"  = "Fazle-ec2"
          + "Owner" = "Fazle"
        }
      + tenancy                      = (known after apply)
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = (known after apply)
    }

  # aws_security_group.fazle_SG_EC2 will be created
```

Video 83

Local Backend Switch and Delete non default workspaces

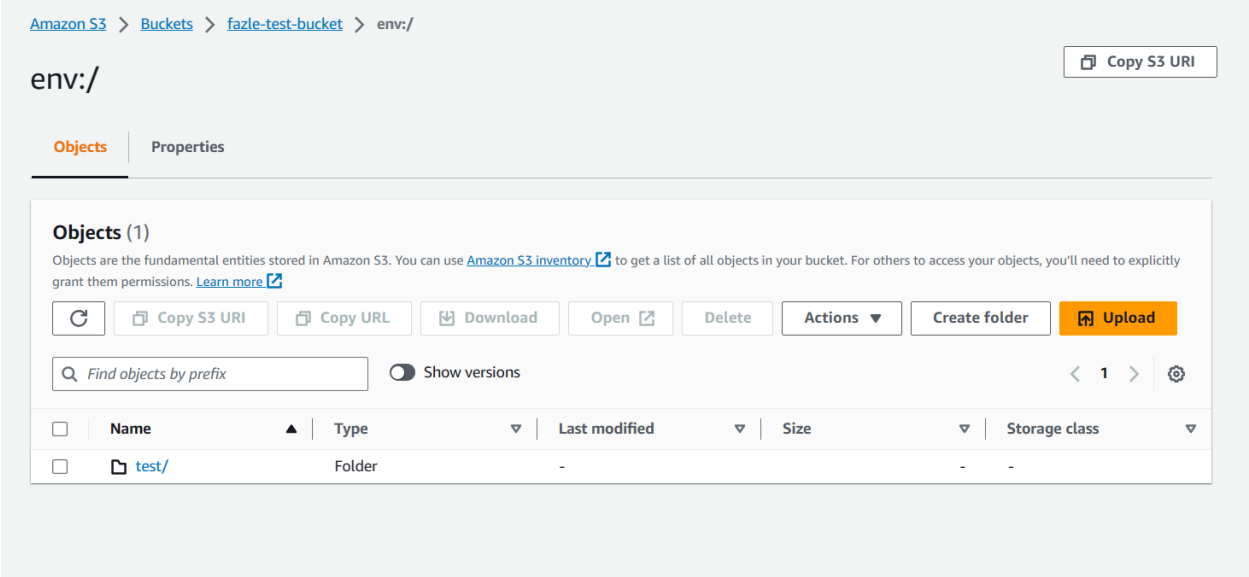
- We cannot delete "default" workspace
- We can delete workspaces which we created (dev, qa etc)
- We cannot delete a workspace while being in that workspace.
- A workspace won't be deleted if there are resources in it but if it is forcefully deleted the resource will be dangling(Not managed by Terraform).

```
PS D:\Terraform\Files\Video Resource Files\video83> terraform workspace
new dev
Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS D:\Terraform\Files\Video Resource Files\video83> terraform workspace
list
  default
* dev

PS D:\Terraform\Files\Video Resource Files\video83> terraform workspace
delete dev
Workspace "dev" is your active workspace.

You cannot delete the currently active workspace. Please switch
to another workspace and try again.
```

```
PS D:\Terraform\Files\Video Resource Files\video83> terraform workspace
select default
Switched to workspace "default".
PS D:\Terraform\Files\Video Resource Files\video83> terraform workspace
delete dev
Deleted workspace "dev"!
```

Video 84

Remote Backends with Workspaces




```
PS D:\Terraform\Files\Video Resource Files\video84> terraform workspace
list
* default

PS D:\Terraform\Files\Video Resource Files\video84> terraform workspace
new test
Created and switched to workspace "test"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
PS D:\Terraform\Files\Video Resource Files\video84> terraform worspace
select default
Terraform has no command named "worspace". Did you mean "workspace"?

To see all of Terraform's top-level commands, run:
  terraform -help

PS D:\Terraform\Files\Video Resource Files\video84> terraform workspace
select default
Switched to workspace "default".
PS D:\Terraform\Files\Video Resource Files\video84> terraform workspace
delete test
Deleted workspace "test"!
```

Video 85

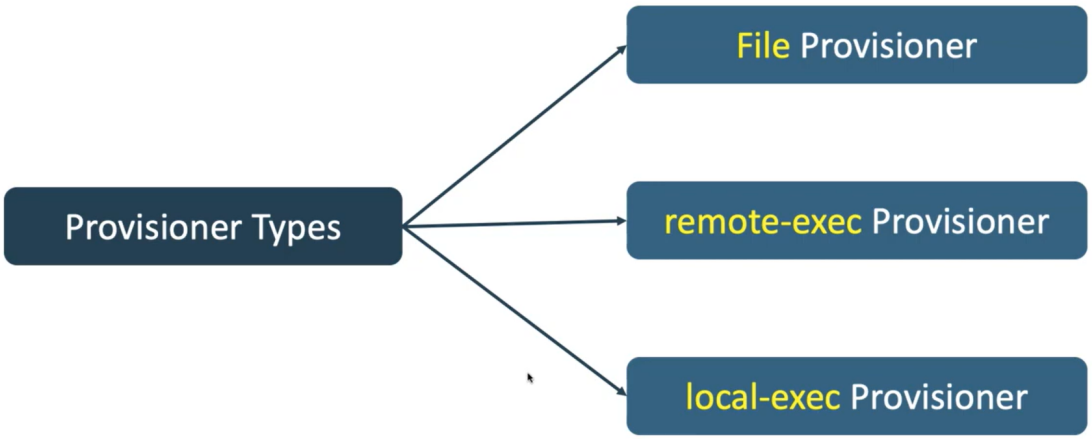
Introduction to Terraform Provisioner's

Terraform Provisioners

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers

- Passing data into virtual machines and other compute resources
- Running configuration management software (packer, chef, ansible)
- Creation-Time Provisioners
- Failure Behaviour: Continue: Ignore the error and continue with creation or destruction.
- Provisioners are a Last Resort
- First-class Terraform provider functionality may be available
- Destroy-Time Provisioners
- Failure Behaviour: Fail: Raise an error and stop applying (the default behavior). If creation provisioner, taint resource

Types of Provisioners



Connection Block

Most provisioners require access to the remote resource via SSH or WinRM, and expect a nested connection block with details about how to connect.

Expressions in connection blocks cannot refer to their parent resource by name. Instead, they can use the special self object.

```
# Connection Block for Provisioners to connect to EC2
connection {
  type = "ssh"
  host = self.public_ip # Understand what is "self"
  user = "ec2-user"
  password = ""
  private_key = file("private-key/terraform-key.pem")
}
```

File Provisioner

File Provisioner

- File Provisioner is used to copy files or directories from the machine executing Terraform to the newly created resource.
- The file provisioner supports both ssh and winrm type of connections

```
# Create EC2 Instance - Amazon2 Linux
resource "aws_instance" "my-ec2-vm" {
  ami           = data.aws_ami.amzlinux.id
  instance_type = var.instance_type
  key_name      = "terraform-key"
  #count = terraform.workspace == "default" ? 1 : 1
  user_data = file("apache-install.sh")
  vpc_security_group_ids = [aws_security_group.vpc-ssh.id,
  tags = {
    "Name" = "vm-${terraform.workspace}-0"
  }
# PLAY WITH /tmp folder in EC2 Instance with File Provisio
  # Connection Block for Provisioners to connect to EC2 In
  connection {
    type = "ssh"
    host = self.public_ip # Understand what is "self"
    user = "ec2-user"
    password = ""
    private_key = file("private-key/terraform-key.pem")
  }
```

```
# Copies the file-copy.html file to /tmp/
provisioner "file" {
  source      = "apps/file-copy.html"
  destination = "/tmp/file-copy.html"
}

# Copies the string in content into /tmp
provisioner "file" {
  content     = "ami used: ${self.ami}"
  destination = "/tmp/file.log"
}

# Copies the app1 folder to /tmp - FOLDE
provisioner "file" {
  source      = "apps/app1"
  destination = "/tmp"
}
```

local-exec Provisioner

local-exec Provisioner

- The local-exec provisioner invokes a local executable after a resource is created.
- This invokes a process on the machine running Terraform, not on the resource.

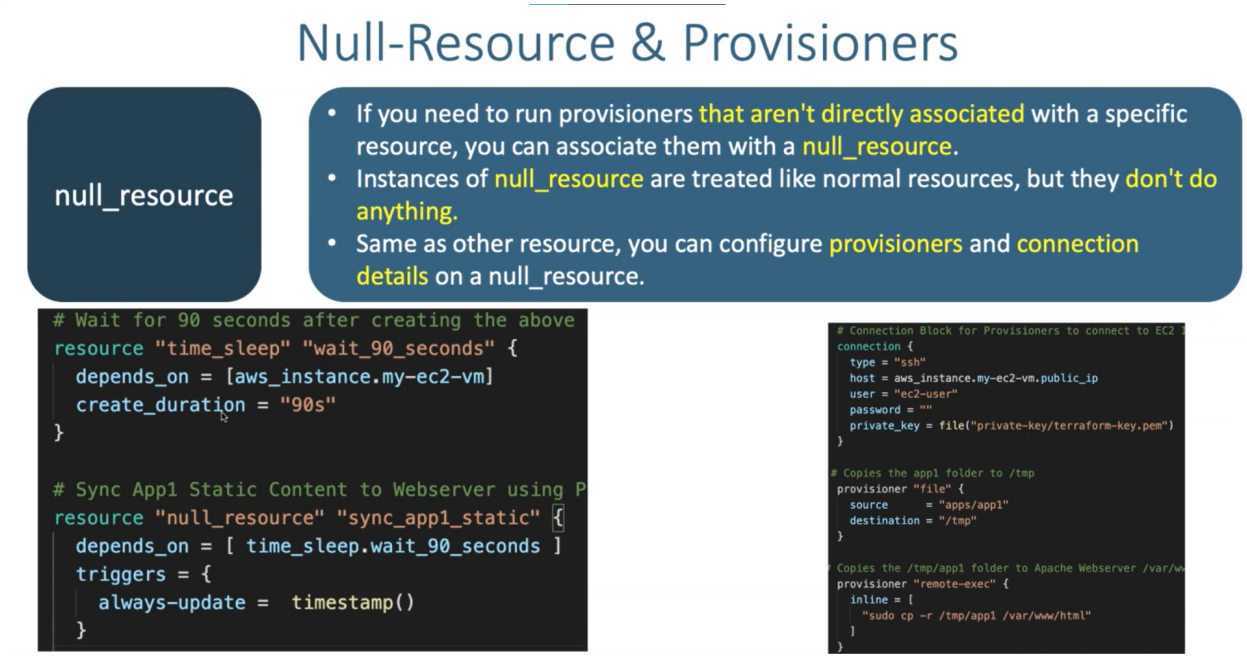
```
# local-exec provisioner (Creation-Time Provisioner - Triggered during Create Resource)
provisioner "local-exec" {
  command = "echo ${aws_instance.my-ec2-vm.private_ip} >> creation-time-private-ip.txt"
  working_dir = "local-exec-output-files/"
  #on_failure = continue
}

# local-exec provisioner - (Destroy-Time Provisioner - Triggered during Destroy Resource)
provisioner "local-exec" {
  when    = destroy
  command = "echo Destroy-time provisioner Instanace Destroyed at `date` >> destroy-time.txt"
  working_dir = "local-exec-output-files/"
}
```

remote-exec Provisioner

remote-exec Provisioner

- The remote-exec provisioner invokes a script on a remote resource after it is created.
- This can be used to run a configuration management tool, bootstrap into a cluster, etc.

```
# Copies the file-copy.html file to /tmp/file-copy.html
 provisioner "file" {
   source      = "apps/file-copy.html"
   destination = "/tmp/file-copy.html"
 }

# Copies the file to Apache Webserver /var/www/html directory
 provisioner "remote-exec" {
   inline = [
     "sleep 120",  # Will sleep for 120 seconds to ensure Apache
     "sudo cp /tmp/file-copy.html /var/www/html"
   ]
 }
```



Video 86

Implement Connection Block for File Provisioner's & Review Manifests

Connection Block

- We can have connection block inside resource block for all provisioners -[or] We can have connection block inside a provisioner block for that respective provisioner

Self Object

- Important Technical Note: Resource references are restricted here because references create dependencies. Referring to a resource by name within its own block would create a dependency cycle.
- Expressions in provisioner blocks cannot refer to their parent resource by name. Instead, they can use the special self object.
- The self object represents the provisioner's parent resource, and has all of that resource's attributes.

```
resource "aws_instance" "fazle-EC2-Public" {
  ami           = data.aws_ami.latest_amazon_linux2.id
  subnet_id = aws_subnet.fazle-subnet.id
  //vpc_security_group_ids = [aws_security_group.fazle_SG_EC2.id]
  instance_type                = "t2.micro"
  associate_public_ip_address = true
  //key_name = "ec2-key"
```

```
  connection {
    type = "ssh"
    host = self.public_ip
    user = "ec2-user"
    password =  ""
    private_key = file("key.pem")
  }

  tags = {
    Name  = "Fazle-ec2-Public"
    Owner = "Fazle"
  }

}
```

Video 87

File Provisioner's Execute Terraform Commands to Verify

Creation-Time Provisioners:

- By default, provisioners run when the resource they are defined within is created.
- Creation-time provisioners are only run during creation, not during updating or any other lifecycle.
- They are meant as a means to perform bootstrapping of a system.
- If a creation-time provisioner fails, the resource is marked as tainted.
- A tainted resource will be planned for destruction and recreation upon the next terraform apply.
- Terraform does this because a failed provisioner can leave a resource in a semi-configured state.
- Because Terraform cannot reason about what the provisioner does, the only way to ensure proper creation of a resource is to recreate it. This is tainting.

```
resource "aws_instance" "fazle-EC2-Public" {
  ami                         = data.aws_ami.latest_amazon_linux2.id
  subnet_id                   = aws_subnet.fazle-subnet.id
  vpc_security_group_ids      = [aws_security_group.sg1.id]
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = "fazle1"

  connection {
```

```
    type        = "ssh"
    host        = self.public_ip
    user        = "ec2-user"
    password    = ""
    private_key = file("fazle1.pem")
  }

  provisioner "file" {
    source      = "file/comment.html"
    destination = "/tmp/comment.html"
  }
```

Video 88

Learn Provisioner's failure behavior using onfailure=continue

- By default, provisioners that fail will also cause the Terraform apply itself to fail. The on_failure setting can be used to change this. The allowed values are:
- continue: Ignore the error and continue with creation or destruction.
- fail: (Default Behavior) Raise an error and stop applying (the default behavior). If this is a creation provisioner, taint the resource.
- If we try to run the provisioner block it will not work as there is no such file in the given directory to copy the file in "/tmp" using file provisioner
- Try two scenarios
- No on_failure attribute (Same as on_failure = fail) - default what happens It will Raise an error and stop applying. If this is a creation provisioner, it will taint the resource.
- When on_failure = continue, will continue creating resources
- Verify: Verify terraform.tfstate for "status": "tainted"

```
resource "aws_instance" "fazle-EC2-Public" {
  ami                         = data.aws_ami.latest_amazon_linux2.id
  subnet_id                   = aws_subnet.fazle-subnet.id
  vpc_security_group_ids      = [aws_security_group.sg1.id]
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = "fazle1"

  connection {
    type        = "ssh"
    host        = self.public_ip
```

```
    user        = "ec2-user"
    password    = ""
    private_key = file("fazle1.pem")
  }

  provisioner "file" {
    source      = "file/comment.html"
    destination = "/tmp/comment.html"
  }

  provisioner "file" {
    source      = "file/not_there.html"
    destination = "/tmp/not_there.html"
    on_failure = continue
  }

  tags = {
    Name  = "Fazle-ec2-Public"
    Owner = "Fazle"
  }

}
```

Video 89

Remote exec Provisioner's Demo

Understand about remote-exec Provisioner

- The remote-exec provisioner invokes a script on a remote resource after it is created.
- This can be used to run a configuration management tool, bootstrap into a cluster, etc.
- We will tell the instance to sleep for 120 sec and then copy the “comment.html” from /tmp/ to /var/

```
resource "aws_instance" "fazle-EC2-Public" {
  ami                         = data.aws_ami.latest_amazon_linux2.id
  subnet_id                   = aws_subnet.fazle-subnet.id
  vpc_security_group_ids      = [aws_security_group.sg1.id]
  instance_type               = "t2.micro"
  associate_public_ip_address = true
```

```
  key_name                    = "fazle1"

  connection {
    type        = "ssh"
    host        = self.public_ip
    user        = "ec2-user"
    password    = ""
    private_key = file("fazle1.pem")
  }

  provisioner "file" {
    source      = "file/comment.html"
    destination = "/tmp/comment.html"
  }

  provisioner "remote-exec" {
    inline = [
      "sleep 120",
      "sudo cp /tmp/comment.html /var"
    ]
  }

  tags = {
    Name  = "Fazle-ec2-Public"
    Owner = "Fazle"
  }

}
```

Video 90

Local exec Provisioner's Demo

Understand about local-exec Provisioner

- The local-exec provisioner invokes a local executable after a resource is created.
- This invokes a process on the machine running Terraform, not on the resource.

- We will create one provisioner during creation-time. It will output private ip of the instance in to a file named created.txt
- We will create one more provisioner during destroy time. It will output destroy time with date in to a file named destroy.txt

```
resource "aws_instance" "fazle-EC2-Public" {
  ami                         = data.aws_ami.latest_amazon_linux2.id
  subnet_id                   = aws_subnet.fazle-subnet.id
  vpc_security_group_ids      = [aws_security_group.sg1.id]
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = "fazle1"

  connection {
    type        = "ssh"
    host        = self.public_ip
    user        = "ec2-user"
    password    = ""
    private_key = file("fazle1.pem")
  }

  provisioner "local-exec" {
     command       = "echo ${aws_instance.fazle-EC2-Public.private_ip} >>
created.txt"
    working_dir = "creation-time/"
  }

  provisioner "local-exec" {
    when        = destroy
    command     = "echo ${self.public_ip} >> destroy.txt"
    working_dir = "destruction-time/"
  }

  tags = {
    Name  = "Fazle-ec2-Public"
    Owner = "Fazle"
  }

}
```

Video 91

Terraform Null Resource Introduction & Review Manifests

The null provider is a rather-unusual provider that has constructs that intentionally do nothing. This may sound strange, and indeed these constructs do not need to be used in most cases, but they can be useful in various situations to help orchestrate tricky behavior or work around limitations.

The documentation of each feature of this provider, accessible via the navigation, gives examples of situations where these constructs may prove useful.

Usage of the null provider can make a Terraform configuration harder to understand. While it can be useful in certain cases, it should be applied with care and other solutions preferred when available.

Video 92

Null Resource Execute Terraform Commands and Test

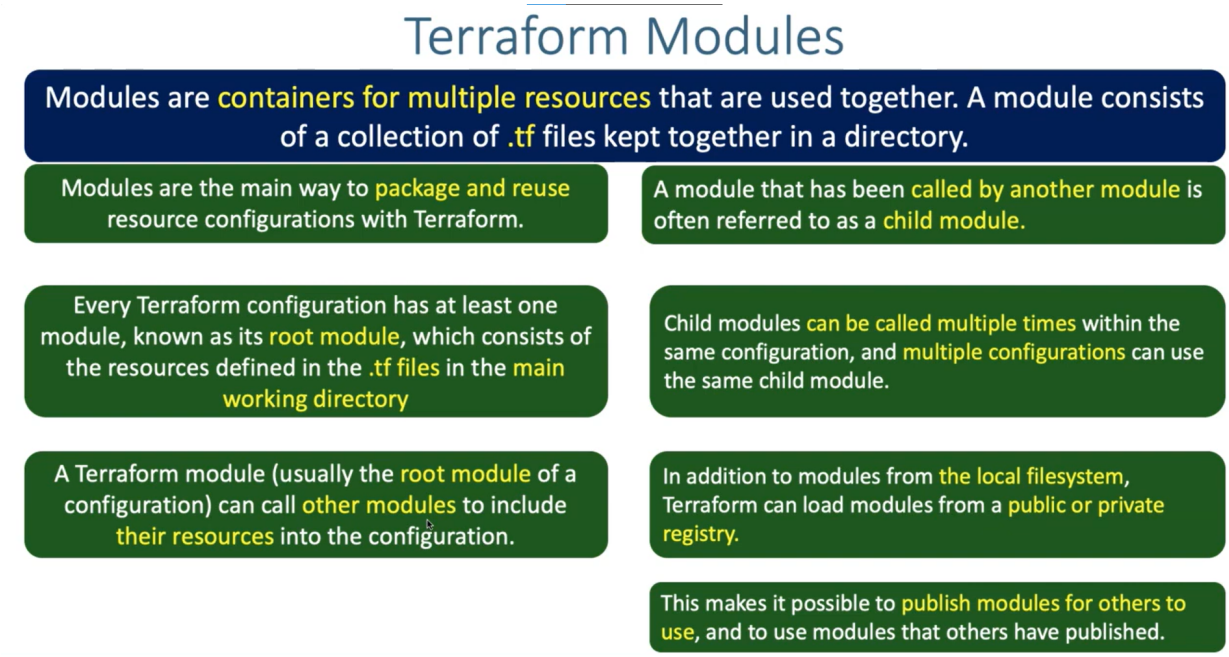
- We will create a null resource so that we can connect to the ec2 instance.
- In the null resource we also make sure that the actions performed in the null resource will get triggered based on the timestamp.
- Basically the null resource will get updated every time the terraform code is executed.
- The provisioners copies a file from the local machine to the ec2 through the remote-exec and then copies form one to another folder via file provisioner

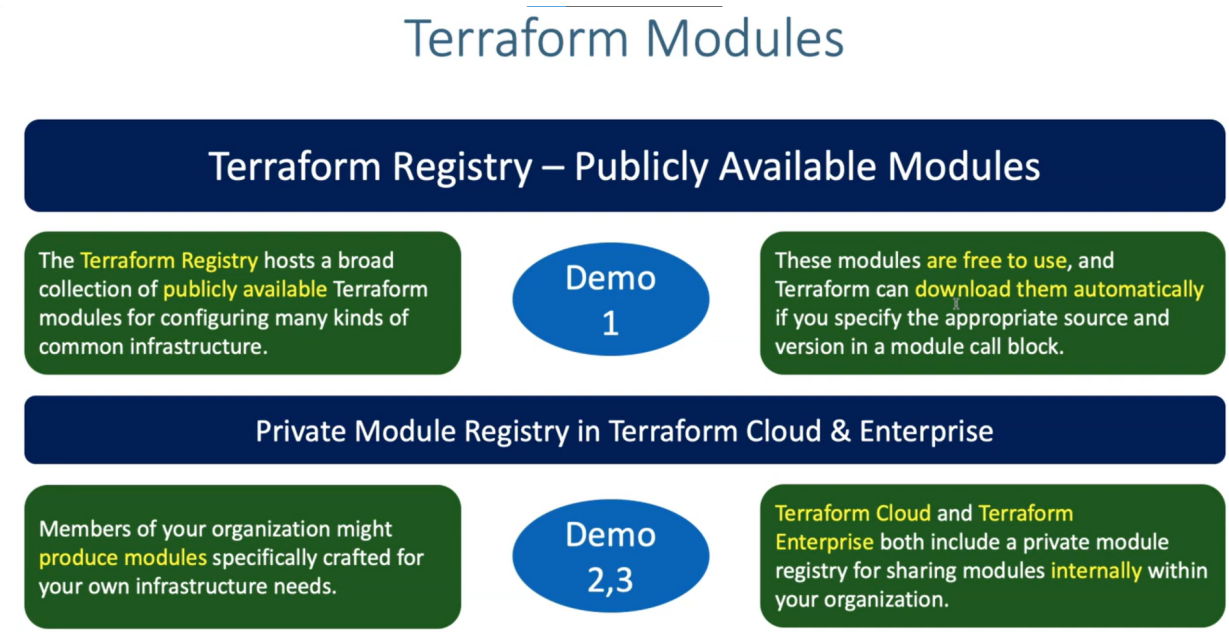
```
resource "aws_instance" "fazle-EC2-Public" {
  ami                         = data.aws_ami.latest_amazon_linux2.id
  subnet_id                   = aws_subnet.fazle-subnet.id
  vpc_security_group_ids      = [aws_security_group.sg1.id]
  instance_type               = "t2.micro"
  associate_public_ip_address = true
  key_name                    = "fazle1"

  user_data = file("install.sh")

  tags = {
```

```
    Name  = "Fazle-ec2-Public"
    Owner = "Fazle"
  }

}

resource "time_sleep" "sleep-90-sec" {
  depends_on      = [aws_instance.fazle-EC2-Public]
  create_duration = "90s"
}

resource "null_resource" "trigger-condition" {
  depends_on = [time_sleep.sleep-90-sec]
  triggers = {
    always-update = timestamp()
  }

  connection {
    type        = "ssh"
    host        = aws_instance.fazle-EC2-Public.public_ip
    user        = "ec2-user"
    password    = ""
    private_key = file("fazle1.pem")
  }

  provisioner "file" {
    source      = "file/comment.html"
    destination = "/tmp/comment.html"
  }

  provisioner "remote-exec" {
    inline = [
      "sudo cp /tmp/comment.html /var/www/html"
    ]
  }
}
```

Video 93

Introduction to Terraform Modules





The Root Module

Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory.

Child Modules

A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration. A module that has been called by another module is often referred to as a child module.

Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.

Published Modules
In addition to modules from the local filesystem, Terraform can load modules from a public or private registry. This makes it possible to publish modules for others to use, and to use modules that others have published.

Video 94

Defining Child Modules Create EC2 Instances using Terraform Modules

```
module "ec2_instance" {
  source = "terraform-aws-modules/ec2-instance/aws"

  name = "Fazle-single-instance"

  ami                    = data.aws_ami.latest_amazon_linux2.id
  instance_type          = "t2.micro"
  key_name               = "fazle1"
  monitoring             = false
  vpc_security_group_ids = [aws_security_group.sg1.id]
  subnet_id              = aws_subnet.fazle-subnet.id

  tags = {
    Name        = "Fazle-EC2"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

Video 95

Create Outputs for EC2 Instance Module, Execute Commands and Test

```
module "ec2_instance" {
  source = "terraform-aws-modules/ec2-instance/aws"

  count = 2

  name = "Fazle-single-instance"

  ami                    = data.aws_ami.latest_amazon_linux2.id
  instance_type          = "t2.micro"
  key_name               = "fazle1"
  monitoring             = false
  vpc_security_group_ids = [aws_security_group.sg1.id]
  subnet_id              = aws_subnet.fazle-subnet.id

  tags = {
    Name        = "Fazle-EC2"
    Terraform   = "true"
    Environment = "dev"
  }
}
```

```
output "private-address" {
  description = "This wil give the pivate ip as an update"
  value       = module.ec2_instance.*.private_ip
}
```