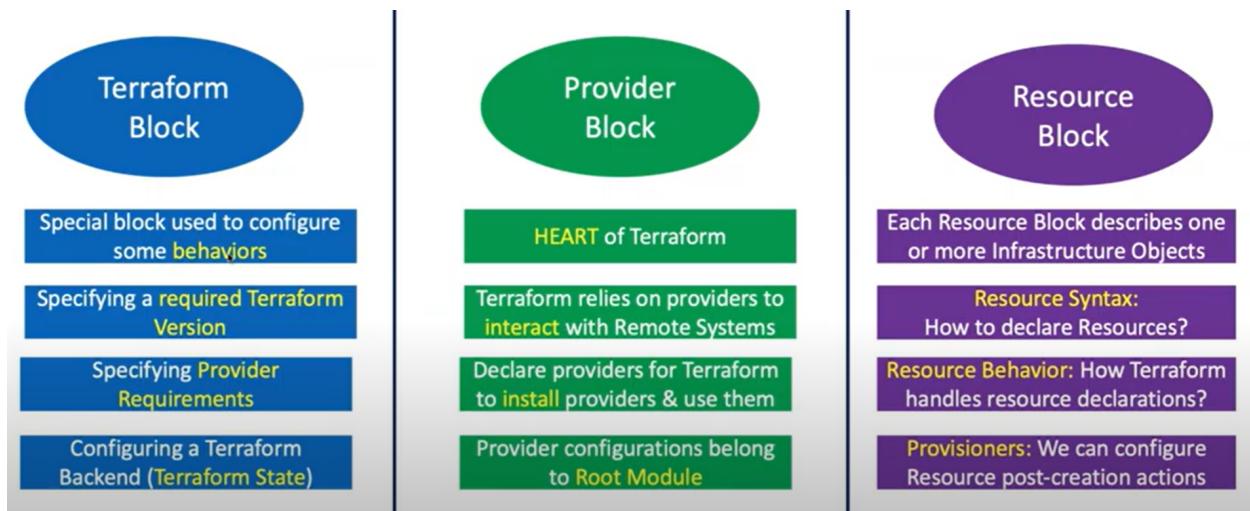


Terraform Video Resource

Video 1

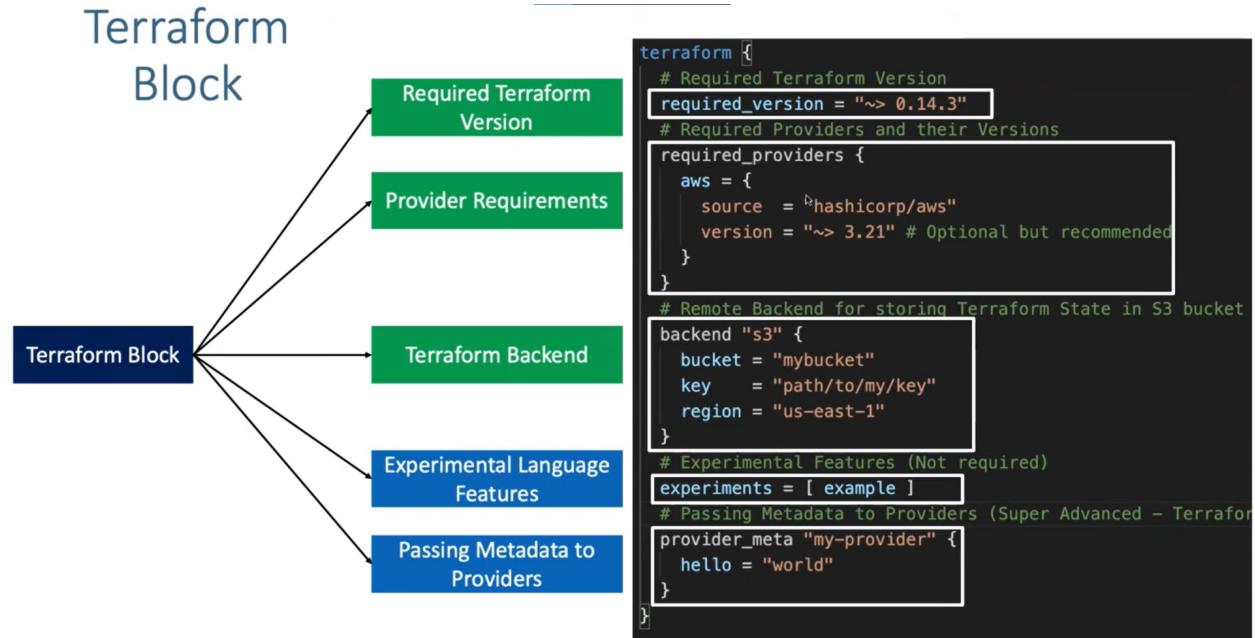
Terraform Basic Block:



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

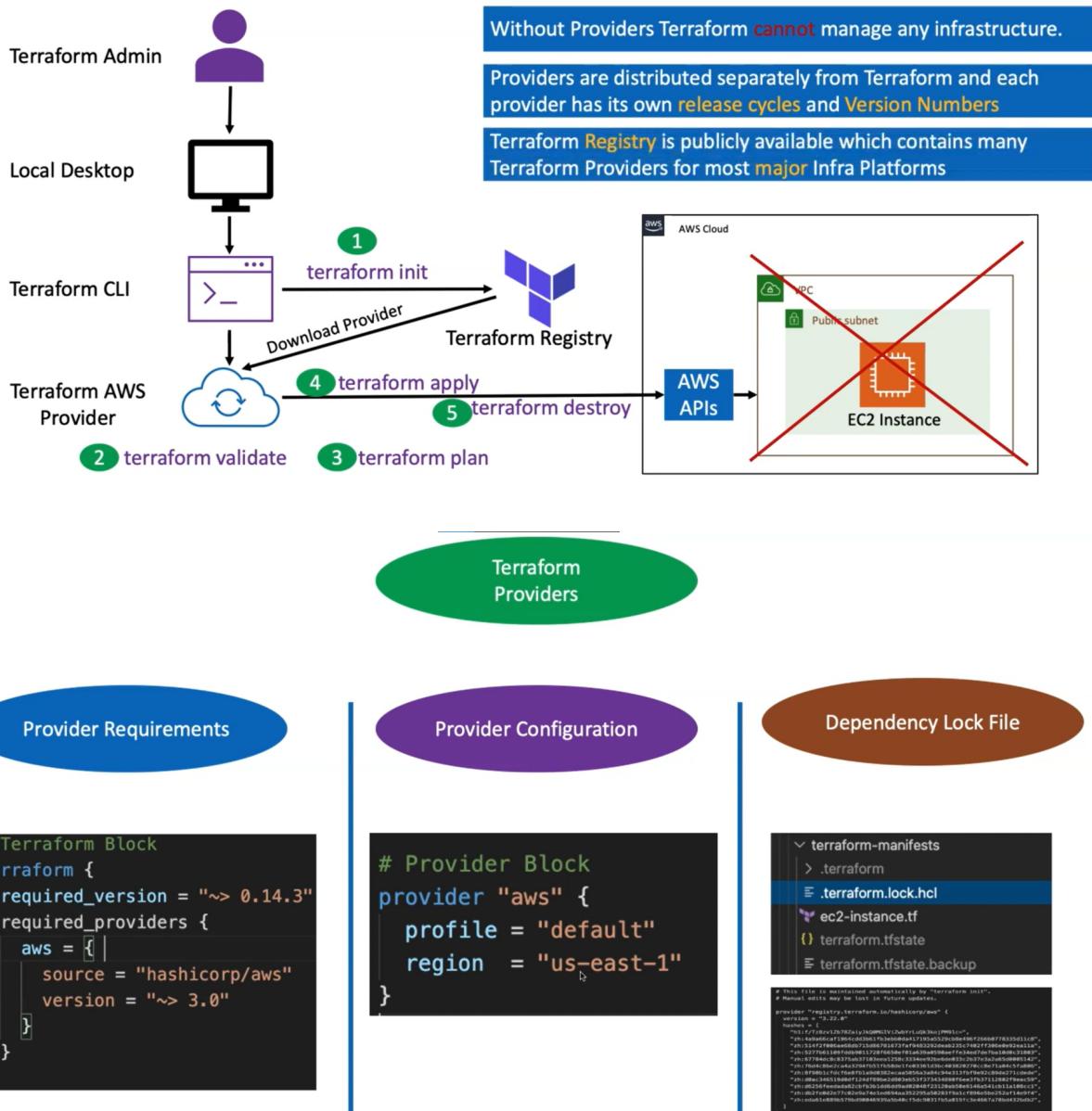
- ★ When using the version control it should be made sure to use the full form of the version for example “`0.14.0`” instead of “`0.14`”. This makes sure that the version that is being used does not get too far mismatched. For “`0.14.0`” the next increment will be “`0.14.1`” but if the given version 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](#)

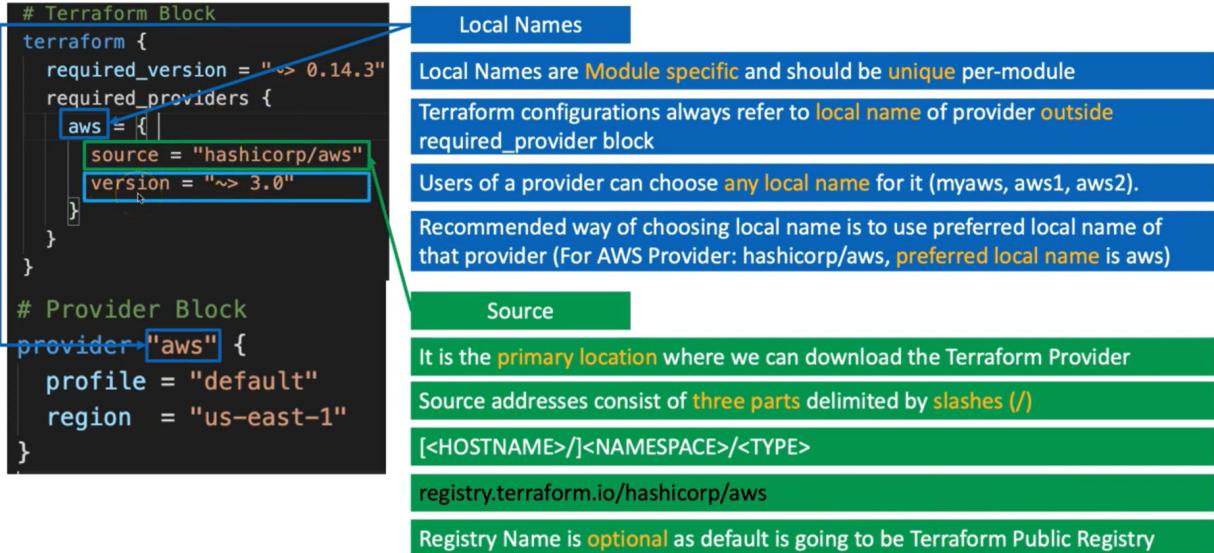
Video 6

Terraform Provider

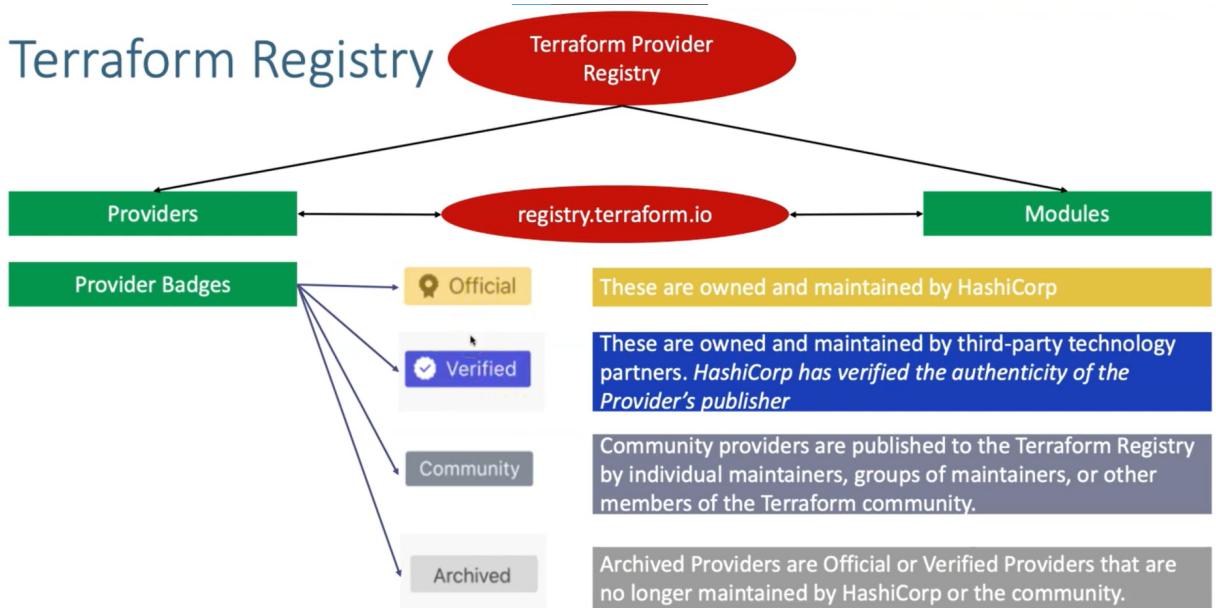
Terraform Providers



Required Providers



Terraform Registry



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

Multiple Providers

We can define multiple configurations for the same provider, and select which one to use on a per-resource or per-module basis.

The primary reason for this is to support multiple regions for a cloud platform

We can use the alternate provider in a resource, data or module by referencing it as <PROVIDER NAME>. <ALIAS>

```
# Provider-1 for us-east-1 (Default Provider)
provider "aws" {
  region = "us-east-1"
  profile = "default"
}

# Provider-2 for us-west-1
provider "aws" {
  region = "us-west-1"
  profile = "default"
  alias = "aws-west-1"
}
```

```
# Resource Block to Create VPC in us-west-1
resource "aws_vpc" "vpc-us-west-1" {
  cidr_block = "10.2.0.0/16"
  #<PROVIDER NAME>. <ALIAS>
  provider = aws. aws-west-1
  tags = {
    "Name" = "vpc-us-west-1"
  }
}
```

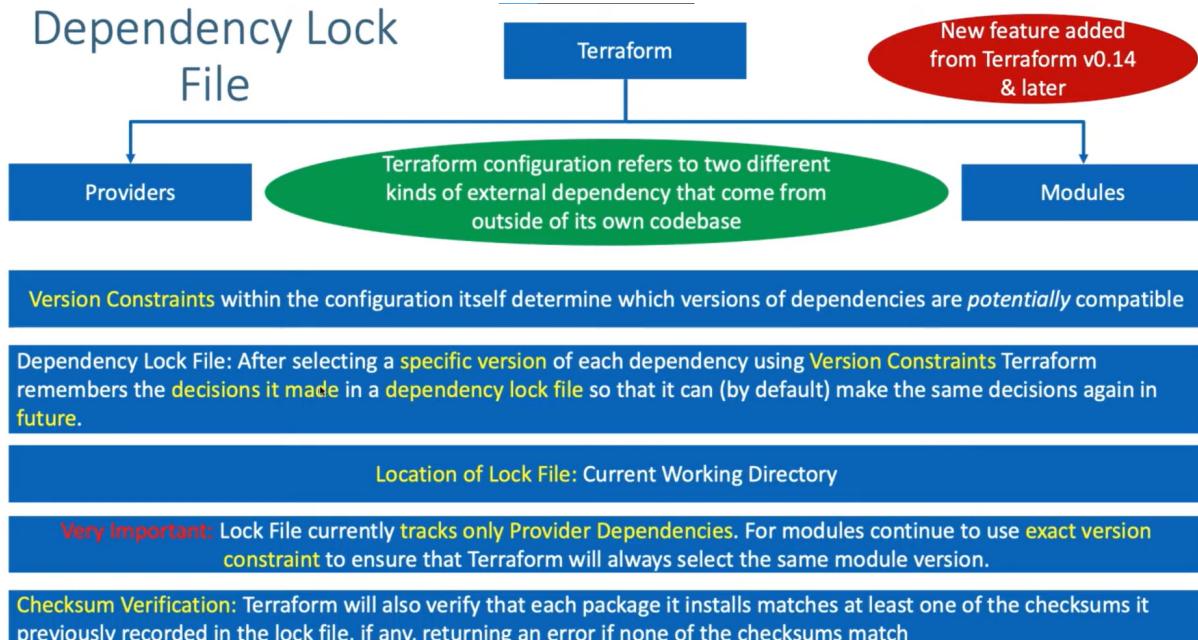
Video 10

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



Importance of Dependency Lock File

Provider	Version Constraint	terraform init (no lock file)	terraform init (lock file)
aws	<code>>= 2.0</code>	Latest version (3.18.0)	Lock file version (2.50.0)
random	<code>3.0.0</code>	<code>3.0.0</code>	Lock file version (3.0.0)

If Terraform did not find a lock file, it would download the latest versions of the providers that fulfill the version constraints you defined in the required_providers block inside Terraform Settings Block.

If we have lock file, the lock file causes Terraform to always install the same provider version, ensuring that runs across your team or remote sessions will be consistent.

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.

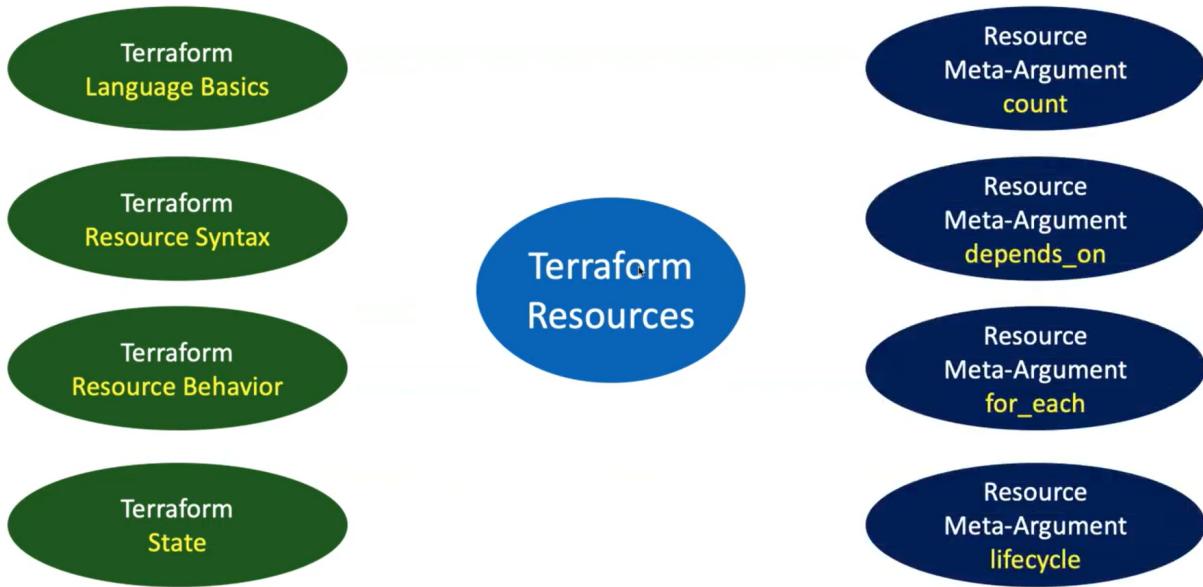
Video 13

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

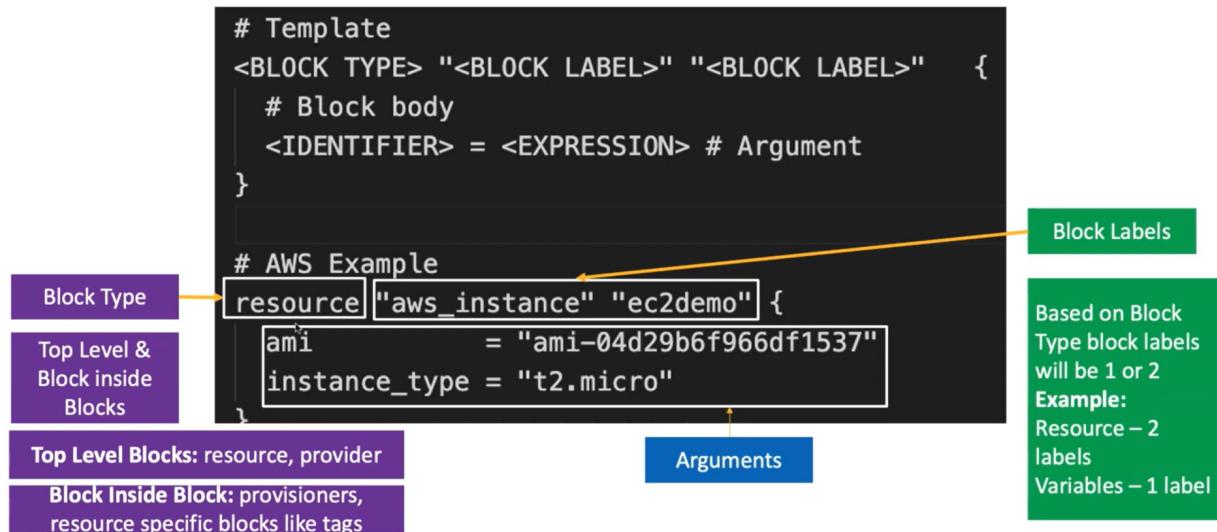
Terraform Resources Introduction



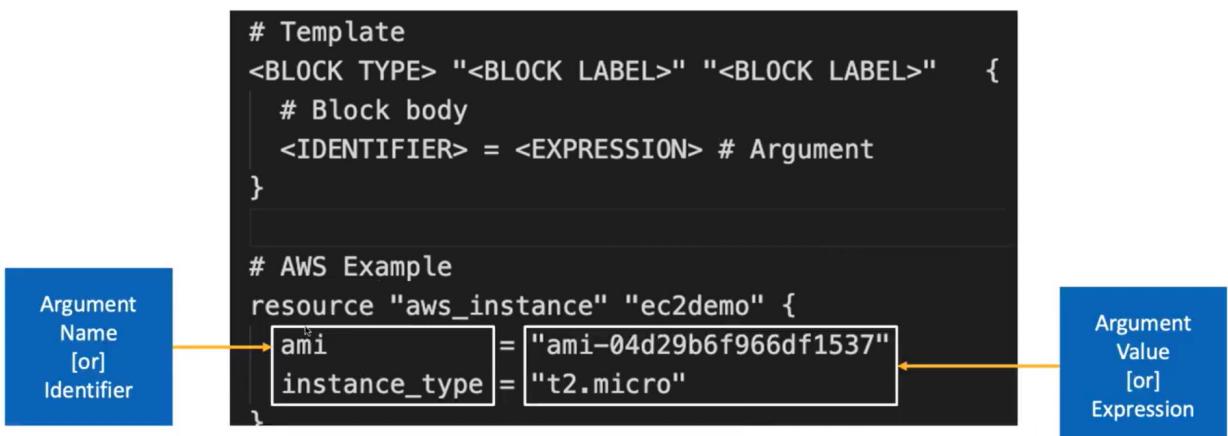
Video 15

Terraform Resources Syntax Introduction

Terraform Language Basics – Configuration Syntax



Terraform Language Basics – Configuration Syntax



Resource Syntax

Resource Type: It determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.

Resource Local Name: It is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside that module's scope.

The resource type and name together serve as an identifier for a given resource and so must be unique within a module

Meta-Arguments: Can be used with any resource to change the behavior of resources

Resource Arguments: Will be specific to resource type. Argument Values can make use of Expressions or other Terraform Dynamic Language Features

```
# Provider-2 for us-west-1
provider "aws" {
  region = "us-west-1"
  profile = "default"
  alias = "aws-west-1"
}

# Resource Block to Create VPC
resource "aws_vpc" "vpc_us-west-1" {
  provider = aws.aws-west-1
  cidr_block = "10.2.0.0/16"
  tags = {
    "Name" = "vpc-1"
  }
}
```

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.

Resource Behavior



Terraform State

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

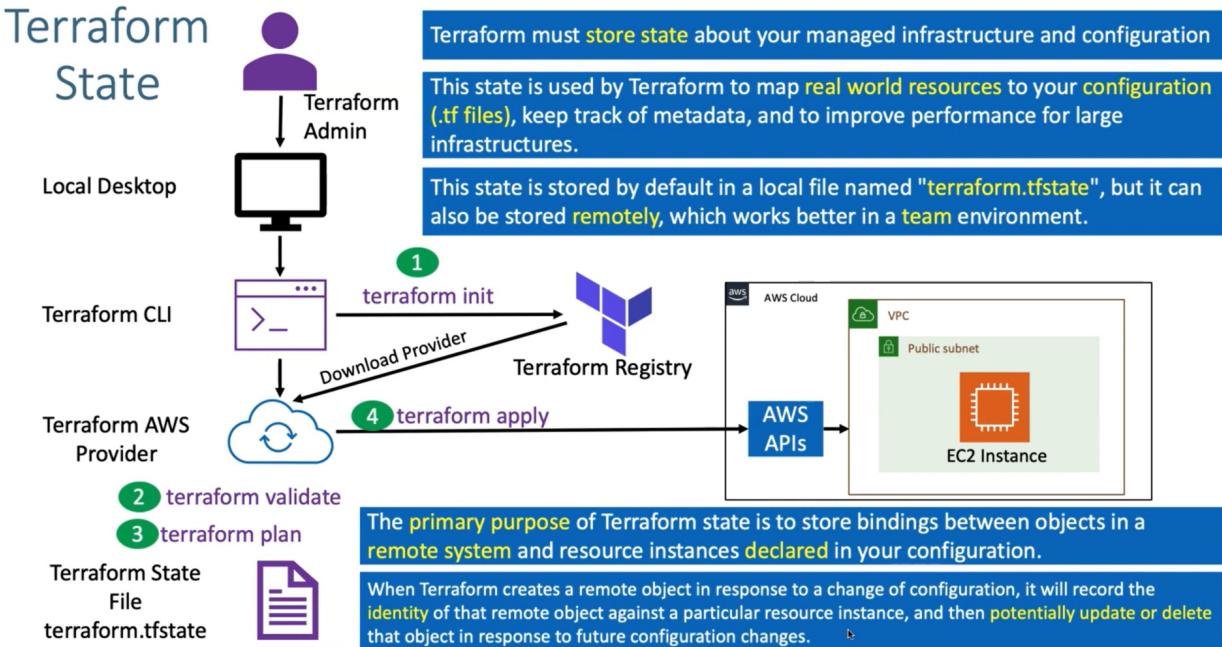
Terraform will perform the following actions:

```
# aws_vpc.fazole-vpc1 will be created
+ resource "aws_vpc" "fazole-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" = "Fazole-vpc1"
    }
    + tags_all                            = {
        + "Name" = "Fazole-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.fazole-vpc1 will be updated in-place
~ resource "aws_vpc" "fazole-vpc1" {
    id                               = "vpc-0f9e694bc54cbf31b"
    tags                            = {
        ~ "Name" = "Fazole-vpc1" -> "Fazole-Vpc1"
    }
    tags_all                         = {
        ~ "Name" = "Fazole-vpc1" -> "Fazole-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" = "Fazole-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.fazole-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.fazole-vpc1 will be destroyed
- resource "aws_vpc" "fazole-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
}

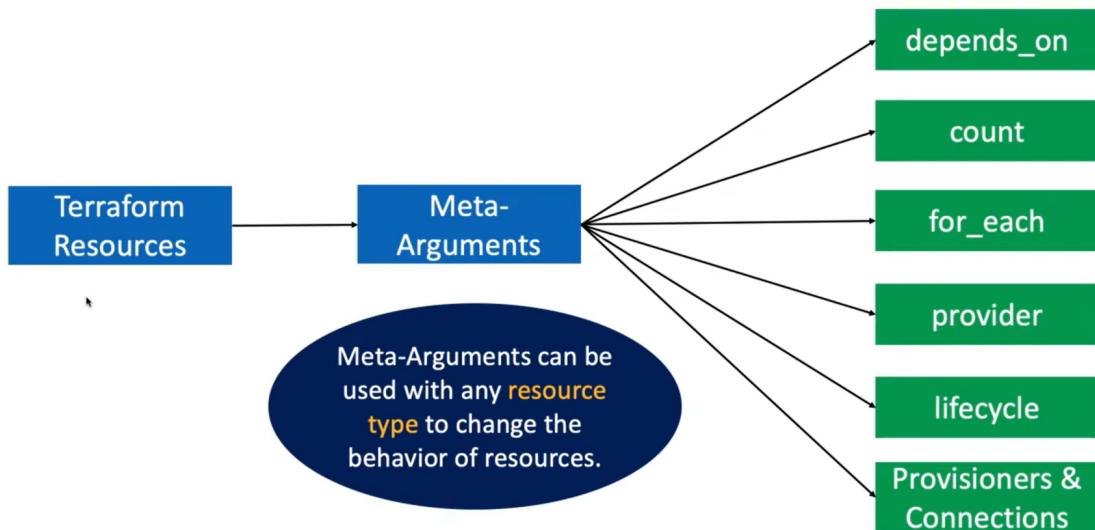
Plan: 0 to add, 0 to change, 1 to destroy.

```

Video 22

Terraform Introduction to Resources Meta Arguments

Resource Meta-Arguments



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

Resource Meta-Arguments – depends_on

Use the `depends_on` meta-argument to handle `hidden` resource or module dependencies that Terraform can't automatically infer.

Explicitly specifying a dependency is only necessary when a resource or module relies on some other resource's behavior but `doesn't access` any of that resource's data in its arguments.

Resource
Meta-Argument
`depends_on`

The `depends_on` meta-argument, if present, must be a list of references to other resources or child modules in the same calling module.

Arbitrary expressions are not allowed in the `depends_on` argument value, because its value must be known before Terraform knows resource relationships and thus before it can safely evaluate expressions.

This argument is available in `module blocks` and in all `resource blocks`, regardless of resource type.

The `depends_on` argument should be used only as a `last resort`. Add comments for future reference about why we added this.

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.

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

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

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

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]`

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

Resource
Meta-Argument
count

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

Video 32

Terraform Introduction to Resources Meta Argument for each

Resource Meta-Arguments – `for_each`

If a resource or module block includes a `for_each` argument whose value is a map or a set of strings, Terraform will create one instance for each member of that map or set.

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

For set of Strings, `each.key = each.value`
`for_each = toset(["Jack", "James"])`
`each.key = Jack`
`each.key = James`

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
`for_each`

For Maps, we use `each.key` & `each.value`
`for_each = {`
 `dev = "my-dapp-bucket"`
`}`
`each.key = dev`
`each.value = my-dapp-bucket`

In blocks where `for_each` is set, an additional `each` object is available in expressions, so you can modify the configuration of each instance.
`each.key` — The map key (or set member) corresponding to this instance.
`each.value` — The map value corresponding to this instance. (If a set was provided, this is the same as `each.key`.)

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

Video 33

Terraform Implement Meta Argument for each with Maps

```
resource "aws_s3_bucket" "fazole-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      = "fazole-bucket-${each.value}"
    Environment = "dev"
  }
}

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

resource "aws_s3_bucket_ownership_controls" "fazole-s3-owner" {
  for_each = aws_s3_bucket.fazole-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" "fazole-aws-s3-bucket" {
```

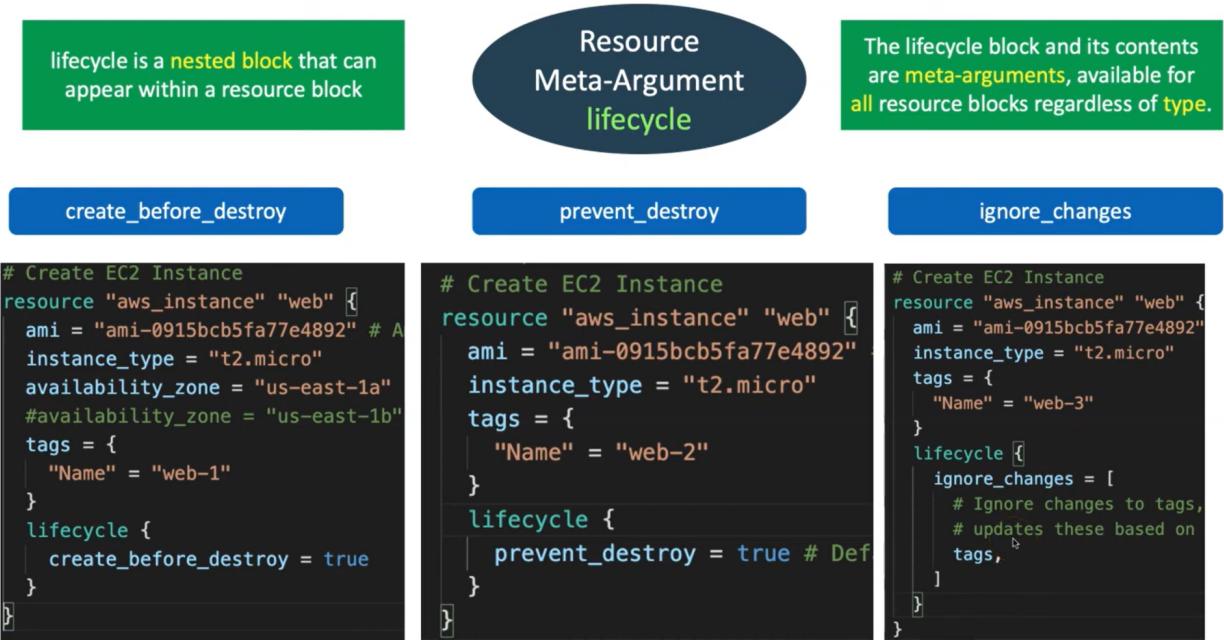
```

for_each = toset(["my-111-fazle-bucket", "my-112-fazle-bucket",
"my-113-fazle-bucket"])
bucket    = each.key
tags = {
  Name = "Fazole-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.fazole-vpc: Creating...
aws_vpc.fazole-vpc: Creation complete after 4s [id=vpc-01cf262a956c94e4]
aws_subnet.fazole-subnet: Creating...
aws_subnet.fazole-subnet: Creation complete after 2s [id=subnet-05d671847b9b32977]
aws_instance.fazole-INSTANCE: Creating...
aws_instance.fazole-INSTANCE: Still creating... [10s elapsed]
aws_instance.fazole-INSTANCE: Still creating... [21s elapsed]
aws_instance.fazole-INSTANCE: Still creating... [31s elapsed]
aws_instance.fazole-INSTANCE: Creation complete after 36s [id=i-0132c3aed9d080198]
aws_instance.fazole-INSTANCE (deposed object 7d4ca958): Destroying... [id=i-065641d5a0a79950c]
aws_instance.fazole-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 10s elapsed]
aws_instance.fazole-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 20s elapsed]
aws_instance.fazole-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 30s elapsed]
aws_instance.fazole-INSTANCE: Still destroying... [id=i-065641d5a0a79950c, 40s elapsed]
aws_instance.fazole-INSTANCE: Destruction complete after 42s
aws_subnet.fazole-subnet (deposed object 0decf51c): Destroying... [id=subnet-0bb14eadbe389bbc1]
aws_subnet.fazole-subnet: Destruction complete after 1s
aws_vpc.fazole-vpc (deposed object 9b56683a): Destroying... [id=vpc-07b40f56aae48e7f3]
aws_vpc.fazole-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

```

resource "aws_instance" "fazole-INSTANCE" {
    ami                  = "ami-08a52ddb321b32a8c"
    instance_type        = "t2.micro"
    subnet_id            = aws_subnet.fazole-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\video08> terraform destroy
aws_vpc.fazole-vpc: Refreshing state... [id=vpc-0f4fe82ecbc3636d3]
aws_subnet.fazole-subnet: Refreshing state... [id=subnet-0fe35871d652d5e33]
aws_instance.fazole-INSTANCE: Refreshing state... [id=i-06eb6ef8b337334d4]

Error: Instance cannot be destroyed

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

Resource aws_instance.fazole-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 removed 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]

Tags		Manage tags
Search tags		< 1 > ⚙
Key	Value	
Name	fazle-vpc	

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.
```

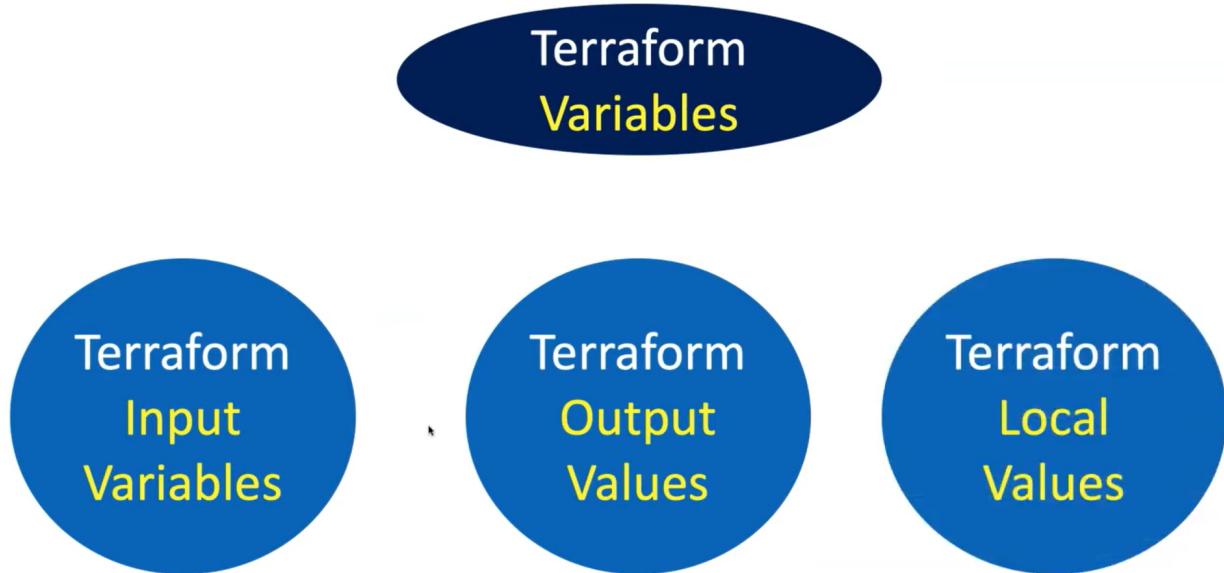
Video 40

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 Input Variables

Input variables serve as **parameters** for a Terraform module, allowing aspects of the module to be **customized** without **altering** the module's own source code, and allowing modules to be **shared** between **different configurations**.

-
- A central green circle contains the text "Terraform Input Variables". Surrounding it are ten numbered boxes, each describing a method for providing input variables, arranged in two columns:
- | | |
|---|---|
| 1
Input Variables - Basics | 6
Provide Input Variables using <any-name>.tfvars file with CLI argument -var-file |
| 2
Provide Input Variables when prompted during <code>terraform plan</code> or <code>apply</code> | 7
Provide Input Variables using <code>auto.tfvars</code> files |
| 3
Override default variable values using CLI argument -var | 8
Implement complex type constructors like <code>List</code> & <code>Map</code> in Input Variables |
| 4
Override default variable values using Environment Variables (<code>TF_var_aa</code>) | 9
Implement Custom Validation Rules in Variables |
| 5
Provide Input Variables using <code>terraform.tfvars</code> files | 10
Protect Sensitive Input Variables |

Terraform Variables – Output Values

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

1

A root module can use outputs to **print** certain values in the **CLI output** after running `terraform apply`.

Terraform
Variables
Outputs

2

A child module can use outputs to **expose a subset** of its resource attributes to a parent module.

When using **remote state**, root module outputs can be accessed by other configurations via a `terraform_remote_state` data source.

3

Advanced

DRY Principle

Don't Repeat Yourself

Terraform Variables – Local Values

A local value assigns a name to an expression, so you can use that name multiple times within a module without repeating it.

Local values are like a function's temporary local variables.

Once a local value is declared, you can reference it in expressions as `local.<NAME>`.

Local values can be helpful to avoid repeating the same values or expressions multiple times in a configuration

If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used

The ability to easily change the value in a central place is the key advantage of local values.

In short, Use local values only in moderation

```
locals {  
    service_name = "forum"  
    owner        = "Community Team"  
}
```

```
locals {  
    # Common tags to be assigned to all resources  
    common_tags = {  
        Service = local.service_name  
        Owner   = local.owner  
    }  
}
```

```
resource "aws_instance" "example" {  
    # ...  
  
    tags = local.common_tags  
}
```

Video 42

Terraform Input Variables Introduction

Terraform Input Variables

Input variables serve as **parameters** for a Terraform module, allowing aspects of the module to be **customized** without altering the module's own source code, and allowing modules to be **shared** between different configurations.

1 Input Variables - Basics

2 Provide Input Variables when prompted during `terraform plan` or `apply`

3 Override default variable values using CLI argument `-var`

4 Override default variable values using Environment Variables (`TF_var_aa`)

5 Provide Input Variables using `terraform.tfvars` files

Terraform
Input
Variables

6 Provide Input Variables using `<any-name>.tfvars` file with CLI argument `-var-file`

7 Provide Input Variables using `auto.tfvars` files

8 Implement complex type **constructors** like `List & Map` in Input Variables

9 Implement **Custom Validation Rules** in Variables

10 Protect **Sensitive** Input Variables

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 is 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.fazole-subnet will be created
+ resource "aws_subnet" "fazole-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.fazole-vpc will be created
+ resource "aws_vpc" "fazole-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.fazole-subnet will be created
+ resource "aws_subnet" "fazole-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.fazole-subnet will be created
+ resource "aws_subnet" "fazole-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" "fazole-subnet" {
    cidr_block      = var.subnet_cidr_block
    availability_zone = var.subnet_az["cZone"]
    vpc_id          = aws_vpc.fazole-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.fazole-subnet will be created
+ resource "aws_subnet" "fazole-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.fazole-vpc will be created
+ resource "aws_vpc" "fazole-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:

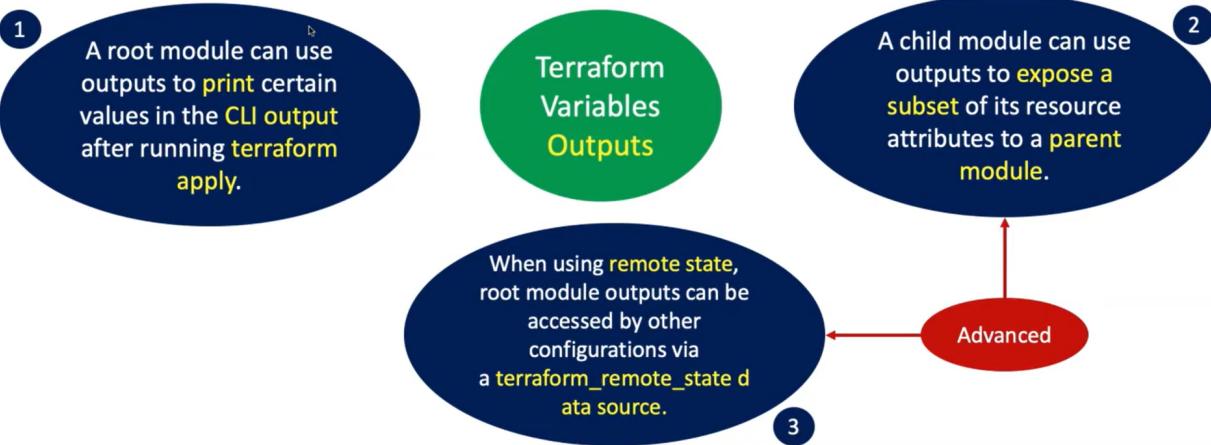
```
resource "aws_instance" "fazle-EC2" {  
    ami                      = "ami-08a52ddb321b32a8c"  
    subnet_id                = aws_subnet.fazle-subnet.id  
    instance_type            = "t2.micro"  
    associate_public_ip_address = "true"  
    vpc_security_group_ids  = [aws_security_group.fazle_SG_EC2.id]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

Video 58

Terraform Output Values Introduction

Terraform Variables – Output Values

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



Terraform Variables Output Values

```
# 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-0406c673]

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

Press Esc to exit full screen

DRY Principle



Don't Repeat Yourself

Terraform Variables – Local Values

A local value assigns a **name** to an expression, so you can use that **name** multiple times within a module without repeating it.

Local values are like a **function's temporary local variables**.

Once a local value is declared, you can reference it in expressions as `local.<NAME>`.

Local values can be helpful to **avoid repeating the same values or expressions multiple times** in a configuration.

If overused they can also make a configuration **hard to read** by future maintainers **by hiding the actual values used**.

The ability to easily change the value in a central place is the **key advantage** of local values.

In short, Use local values only in moderation

```
locals {  
    service_name = "forum"  
    owner        = "Community Team"  
}
```

```
locals {  
    # Common tags to be assigned to all resources  
    common_tags = {  
        Service = local.service_name  
        Owner   = local.owner  
    }  
}
```

```
resource "aws_instance" "example" {  
    # ...  
  
    tags = local.common_tags  
}
```

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

We can apply locals in the following way:

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

resource "aws_s3_bucket" "fazole-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"]]
  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"
}
```