Lecture 1

Terraform basic blocks:

1. Terraform block
2. Provider block
3. Resoruce block

Terraform block: Terraform block contains: required versions, provider requirements and configure: some behaviour and terraform backend(Terraform state). This is the first step for creating anything with terraform.

Provider block: It relies on providers to interact with remote system(cloud), declare them(provider) to install and use them and lastly provider configuration belong to Root Module.

Resource Block: Each block describes one or more infra objects.
Resource syntax: How resource is declared
Resource Behavior: How Terraform handles resource declarations
Provisioners: Can configure resource post-creation actions

Lecture2
Terraform block

- Within a terraform block, only constant values can be used.
- Can't use any built in functions of terraform here also need to write value of the respective arguments, can't refer any variables.
- **Terraform block is introduced from 0.13 onwards**

- Example:

Terraform 0.13 and later:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

- **Terraform block required:**
  - required version. (need to match with local cli version else error, need to lock the version)

    ```
    required_version = "~> 0.14.3"
    ```

  - Required_providers: Will install during the **terraform init** command
  - Backend:

    ```
    backend "s3" {
      bucket = "mybucket"
      key    = "path/to/my/key"
      region = "us-east-1"
    }
    ```

    Storing the state information of our configuration.

- Terraform block optional:
- Experimental language features: This is experimental.

```
experiments = [ example ]
```

- Passing Metadata to Providers: More advanced terraform feature.

```
provider_meta "my-provider" {
  hello = "world"
}
```

  etc.

- **Random Provider:** The "random" provider allows the use of randomness within Terraform configurations. This is a logical provider, which means that it works entirely within Terraform's logic, and doesn't interact with any other services.
    - Where random providers are used: Generate unique resource name, testing and simulation, load balancing and routing etc.

Lecture 3

Creation of terraform block.

Code:

1.

```
terraform {
  required_version = "1.5.4"
}
```

Won't work.

```
terraform {
  required_version =  "~> 1.5.4"
}
```

Will work because it is allowing the right most version as I have already 1.5.5 so this syntax will work. Because it ensure the latest minor version upgrade. It allows anything >1.5 but less than 1.6

Version Constraint Syntax

Terraform's syntax for version constraints is very similar to the syntax used by other dependency management systems.

version = ">= 1.2.0, < 2.0.0"

A version constraint is a string literal containing one or more conditions, which are separated by commas.

Each condition consists of an operator and a version number.

Version numbers should be a series of numbers separated by periods (like 1.2.0), optionally with a suffix to indicate a beta release.

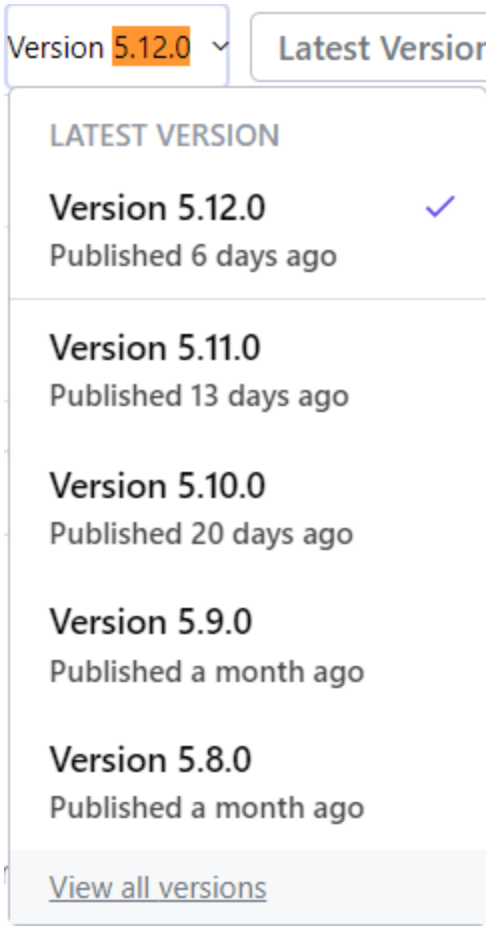
The following operators are valid:

- = (or no operator): Allows only one exact version number. Cannot be combined with other conditions.

- != : Excludes an exact version number.
- ≥, >=, <, <=: Comparisons against a specified version, allowing versions for which the comparison is true. "Greater-than" requests newer versions, and "less-than" requests older versions.
- ~>: Allows only the *rightmost* version component to increment. For example, to allow new patch releases within a specific minor release, use the full version number: ~> 1.0.4 will allow installation of 1.0.5 and 1.0.10 but not 1.1.0. This is usually called the pessimistic constraint operator.(Important)

Lecture 4

It specifies all of the required providers by the current module, mention the source form which they need to download the provider and also ensures that provider constraints are there.
Latest aws provider version:



and we can see that the latest one is 5.12.0 and **terraform init** is installing this version.

Caution regarding required_version: always use the complete minor version. Because if we write anything like:
"~>0.14" it wil search for upgraded version for 14(15, 16..) and there is some major changes. So Always use complete minor version. Like: 0.14.0 then it will search for 0.14.1, 0.14.2….

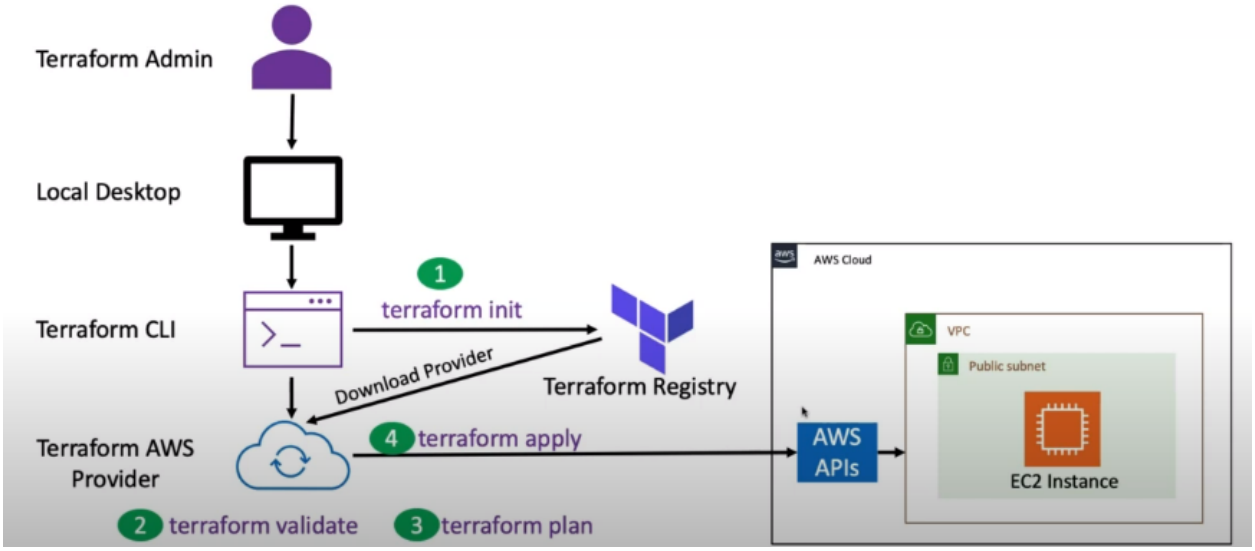
terraform init -upgrade will upgrade or downgrade based on the condition provided.

terraform version variation and check

```
/*
Play with Terraform Version
  required_version = "~> 0.14.3"
  required_version = "= 0.14.4"
  required_version = ">= 0.13"
  required_version = "= 0.13"
  required_version = "~> 0.13"


Play with Provider Version
      version = "~> 3.0"
      version = ">= 3.0.0, < 3.1.0"
      version = ">= 3.0.0, <= 3.1.0"
      version = "~> 2.0"
      version = "~> 3.0"
*/
```

Lecture 6

Terraform Providers



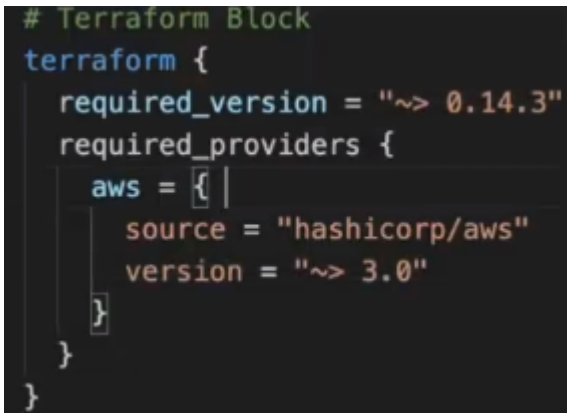
So the provider will communicate with the AWS APIs and those API's will provision the ec2 instance. The same as terraform apply is done during terraform destroy.

Without providers terrafrom can’t manage any infrastructure. And these are distributed separately from the terraform and each provider has it’s own release cycle and version number.
Terraform got 3443 providers.

Terraform provider block:
Profile = “default”
It will use the default configuration form local cli or we can use access key, secret access key like static information.

Dependency lock file: Have the provider related information including version.

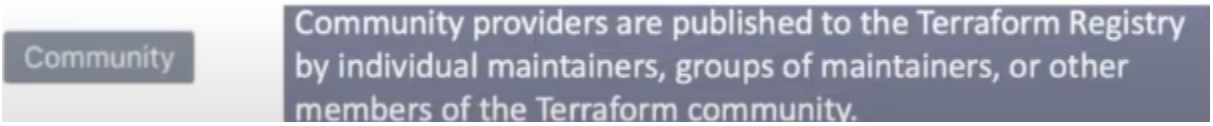
Local name for required providers:



Here aws is the local name and it is named after source=”hashicorp/**aws**”
It is recommended to use this name but any other local name can also be created.
Local names are module specific and should be unique per-module. And if I want to refer outside of the provider, then I can refer using the local

name.
Source: primary location from where we can download the provider.
It got 3 part:
[<HOSTNAME>/]<NAMESPACE>/<TYPE>
Example: registry.terraform.io/hashicorp/aws
But registry name is optional when we are using terraform public registry.

Registry contains 4 things: Providers, Modules, Policy Libraries and Run Tasks.
Provider got badges(provided by terraform=Official, Verified, Community and archived(No longer maintained by hashicorp))



Lecture 7

Terraform needs acces to respective cloud provider. There are several ways:

Configuration for the AWS Provider can be derived from several sources, which are applied in the following order:

1. Parameters in the provider configuration(Static credenrtial is not recommended)
2. Environment variables(aws configure)
3. Shared credentials files
4. Shared configuration files
5. Container credentials

6. Instance profile credentials and region

Here first 3 are the main. **And already saved files are in .aws/credentials folder.**

Lecture 8

```
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"(required)
}
```

Here aws_vpc is mandatory and main the convenient label to refer this vpc object inside terraform.

In terraform init = .terraform.lock.hcl file is created with .terraform folder which got provider version inside it.
Terraform tf state file: Terraform configuration database. Resources created in the aws cloud, it's equivalent information will be stored here. This file is created at terraform apply.

Lecture 9

Multiple provider: Define multiple configuration for the same provider and we can use on a per-resource basis.

Difference is to use alias.
Way to refer them: <PROVIDER NAME>.<ALIAS>

```
resource "aws_vpc" "my_vpc" {
  cidr_block       = "192.168.0.0/26"

  tags = {
    Name = "main"
  }
}
```

```
resource "aws_vpc" "my_vpc-us-east-2" {
  cidr_block       = "192.168.0.0/26"
  provider = aws.aws-east-2

  tags = {
    Name = "vpc-us-east-2"
  }
}
```

Here provider is a meta argument.

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

provider "aws" {
  profile = "default"
  region  = "us-east-2"
  alias = "aws-east-2"
}
```

Lecture 11

Terraform dependency lock file.

Terraform configuration refers to 2 different kinds of external dependency that come from outside of it's own codebase.
They are: **Providers and Modules.**
Dependency lock file works towards only provider version locking. After selecting a specific version of each dependency using Version Constraints Terraform remembers the decision it made in a dependency lock file so that it can make the same decision again in future.
Inserted into github so that this version will be automatically downloaded in the users directory.

Lock file currently tracks only Provider dependencies.
Hashes in the lock file will be created during the **terraform init**.
If terraform didn't find any lock file it downloads the latest versions that fulfills the version constraints, but if not then terraform will use existing lock file and will install same provider version always.

Lecture 15

Terraform resource syntax introduction

Resource template:

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

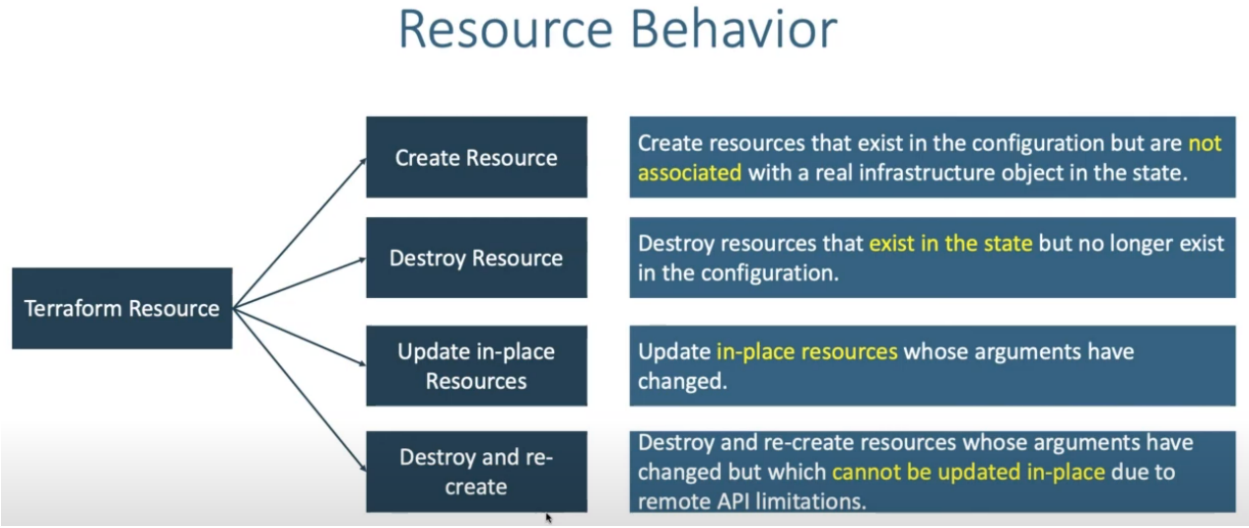
Top level blocks: resource, provider, variables etc.
Block inside block: tags etc.
Resources have 2 block labels. And variable have 1 block level.
Arguments: resource level details. They reside in the block body.

First block label determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports = **Resource Type**

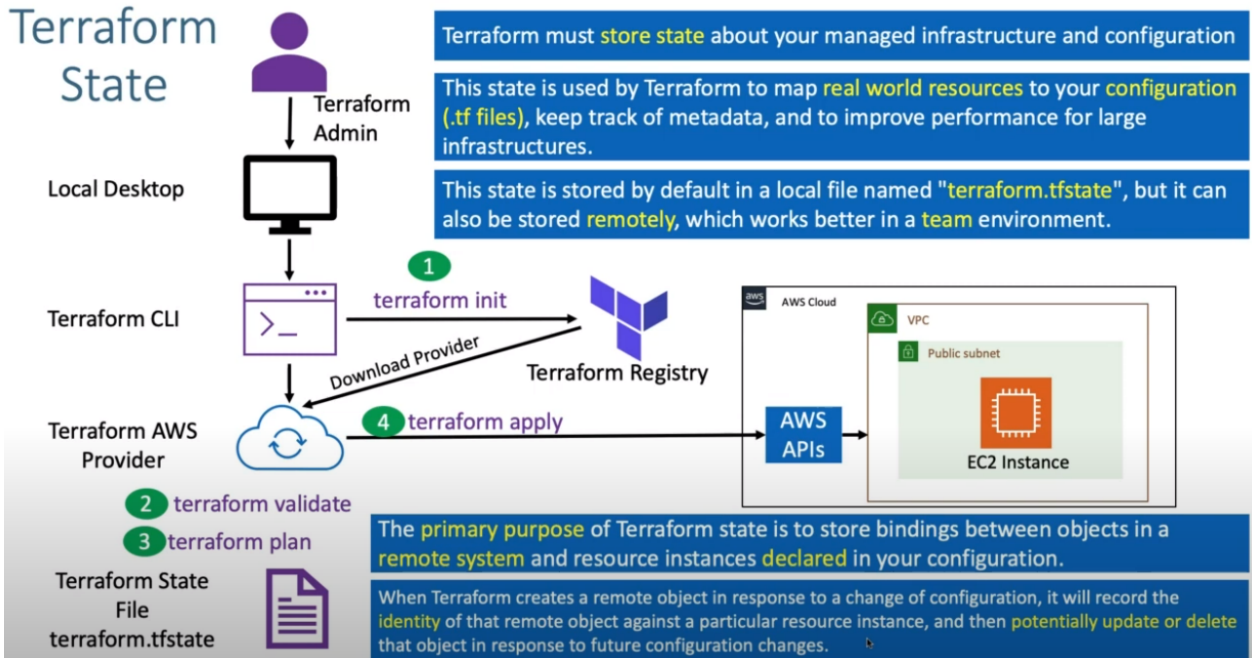
Second block label is used to refer this resource within module. But it got no scope outside this module = **Resource Local Name**
Local name needs to be unique also.

Can be used with any resource to chance the behaviour of resources = **Meta-Argument.** Othets are argument.

Lecture 16



Lecture 18




```
Terraform used the selected providers to gener
  ~ update in-place

Terraform will perform the following actions:
```

- **Desired State:** Local Terraform Manifest (All *.tf files)
- **Current State:** Real Resources present in your cloud

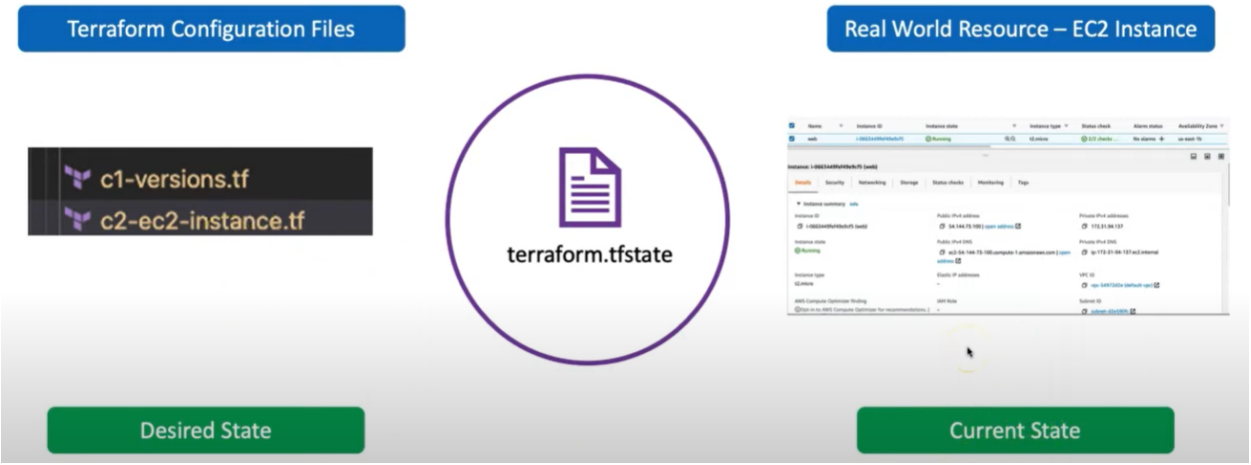
Lecture 20:

Destroy and recreate resources

```
  # aws_instance.my-ec2-vm must be replaced
-/+ resource "aws_instance" "my-ec2-vm" {
```

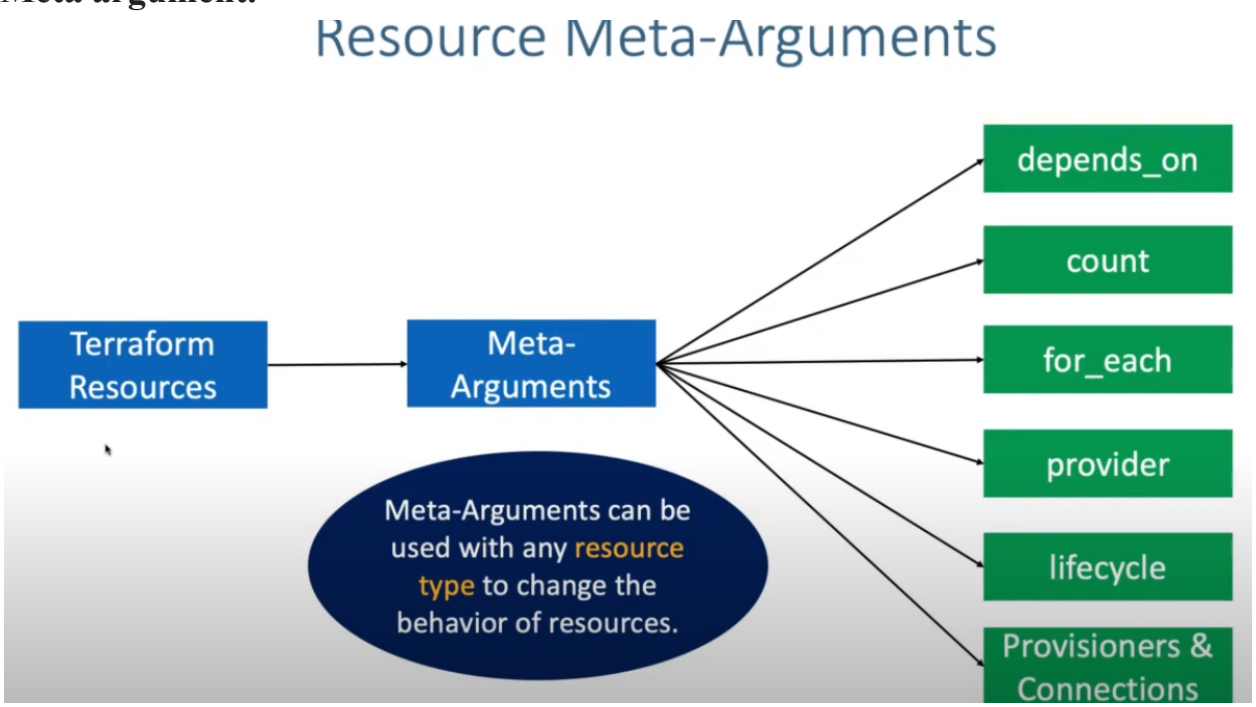
Because in case of available zone change, it will first destroy and then create into another zone. So, destroy and create.

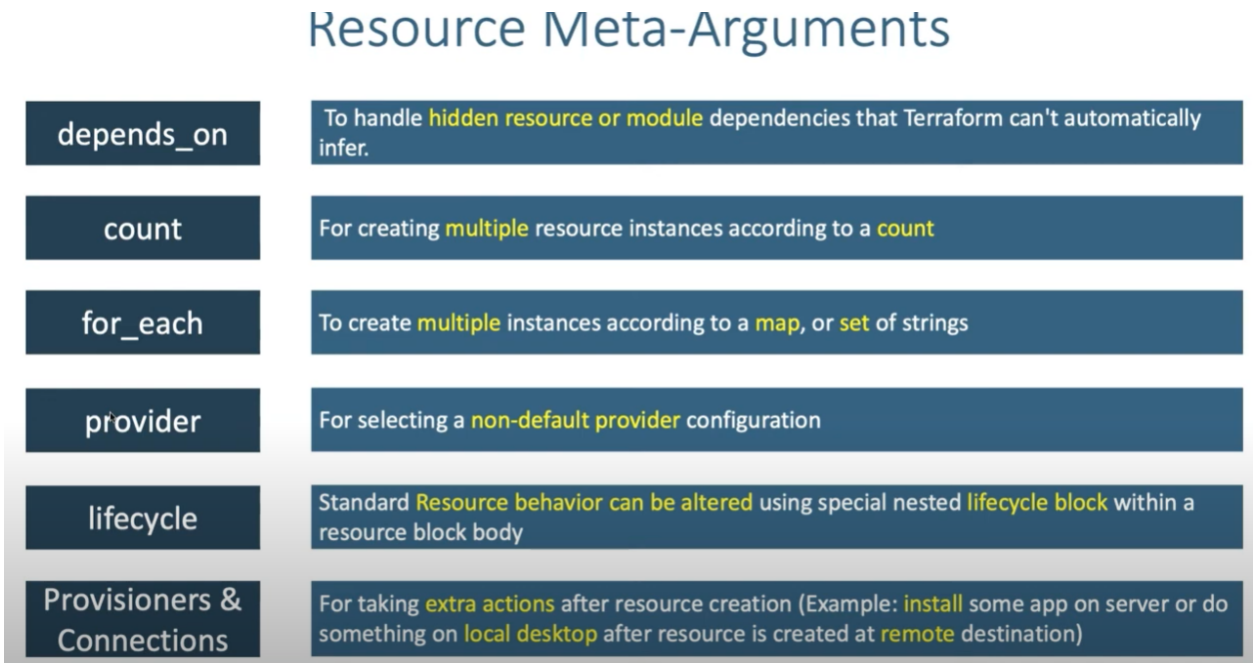
Lecture 21

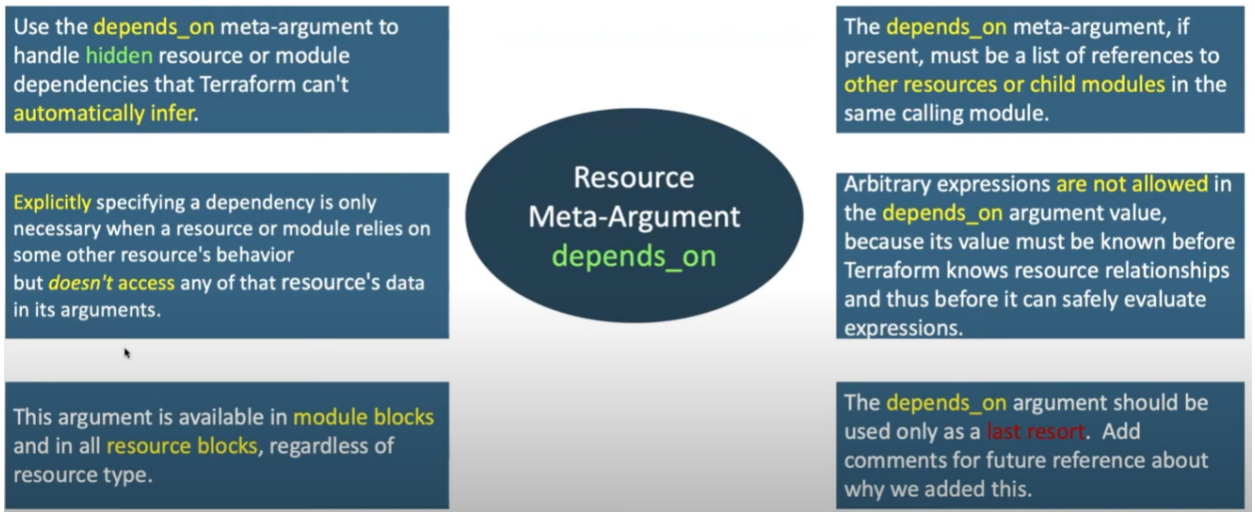

Terraform Configuration Files
Real World Resource – EC2 Instance
c1-versions.tf
c2-ec2-instance.tf
terraform.tfstate
Desired State
Current State

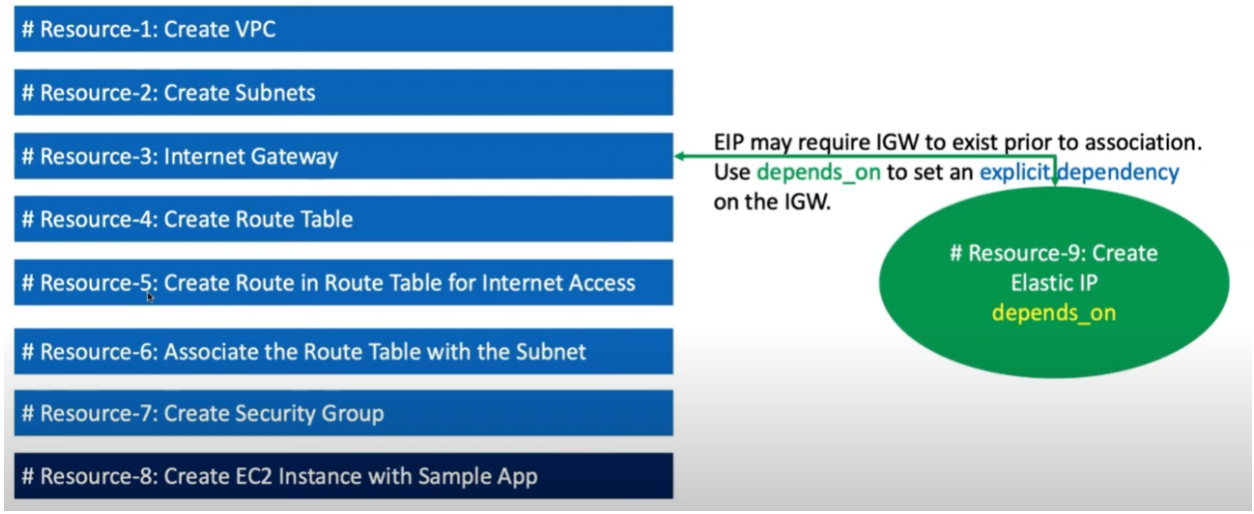
Lecture 22

Meta argument.









Lecture 29

```
#user_data = file("apache-install.sh")
```

Lecture 30

Count Meta Argument.

The `count` Object

In blocks where `count` is set, an additional `count` object is available in expressions, so you can modify the configuration of each instance. This object has one attribute:

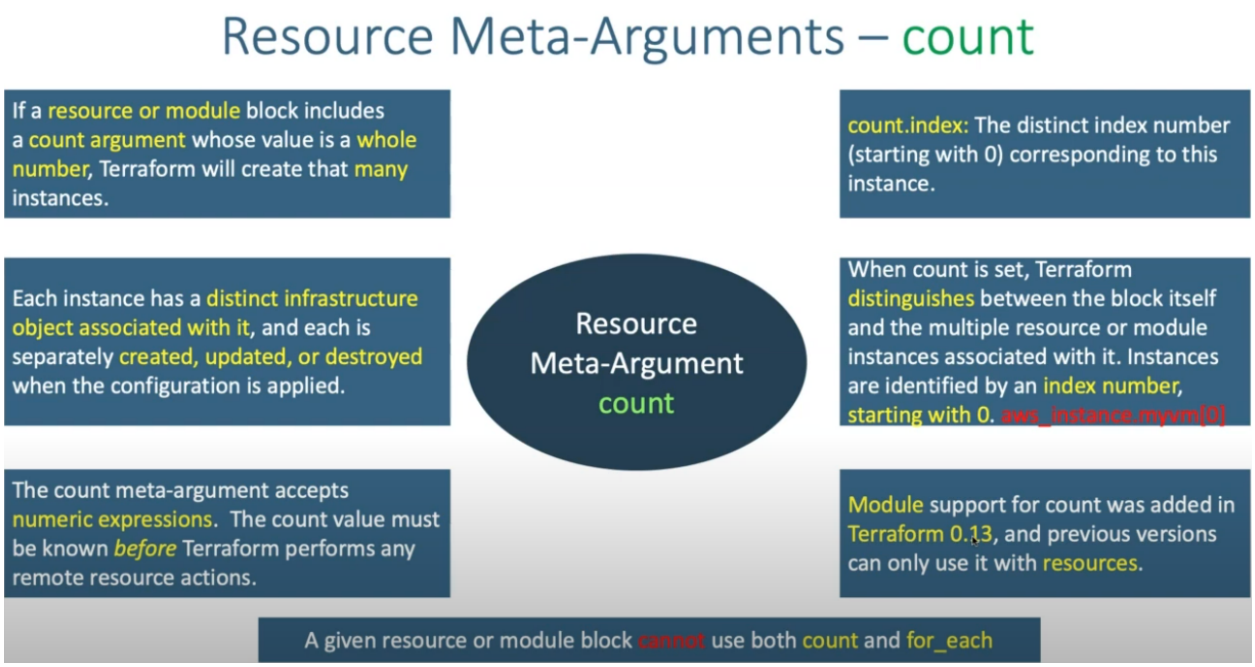
- **`count.index` — The distinct index number (starting with `0`) corresponding to this instance.**

Using Expressions in `count`

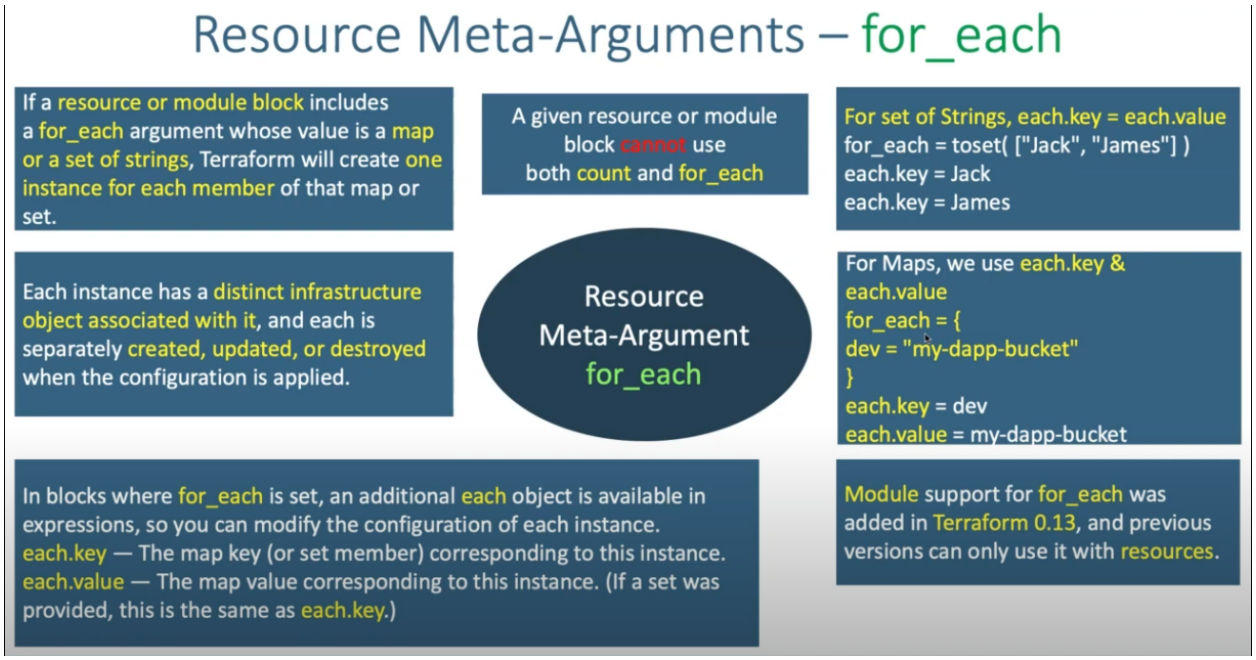
The `count` meta-argument accepts numeric expressions. However, unlike most arguments, the `count` value must be known *before* Terraform performs any remote resource actions. This means `count` can't refer to any resource attributes that aren't known until after a configuration is applied (such as a unique ID generated by the remote API when an object is created).

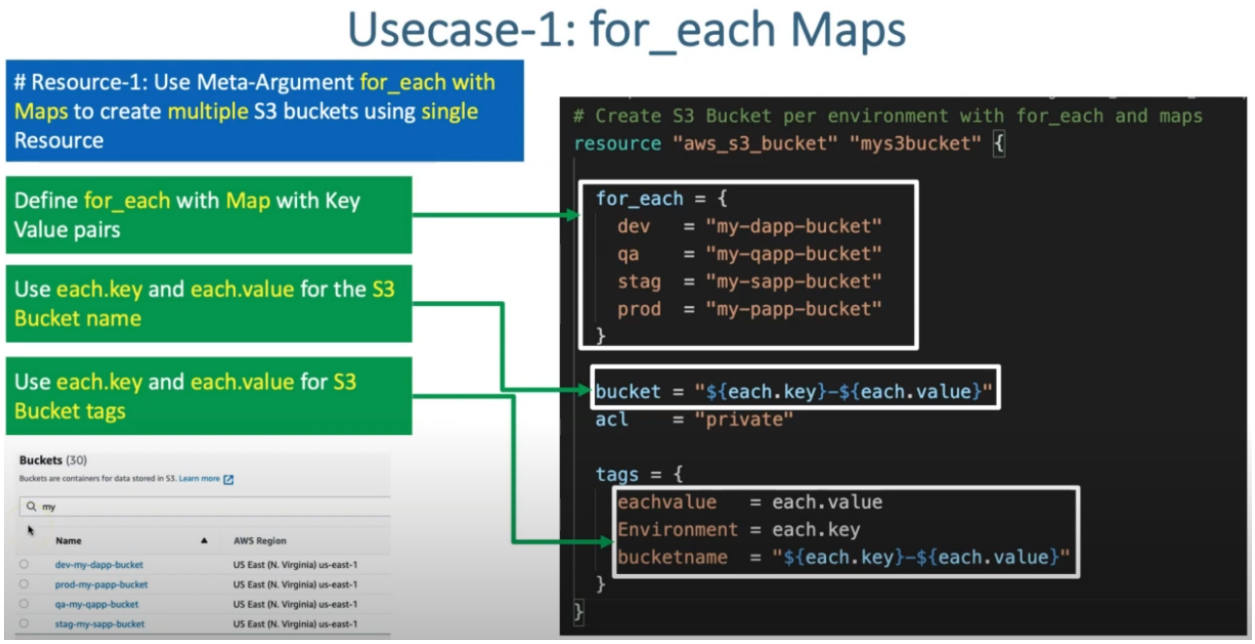
When to Use `for_each` Instead of `count`

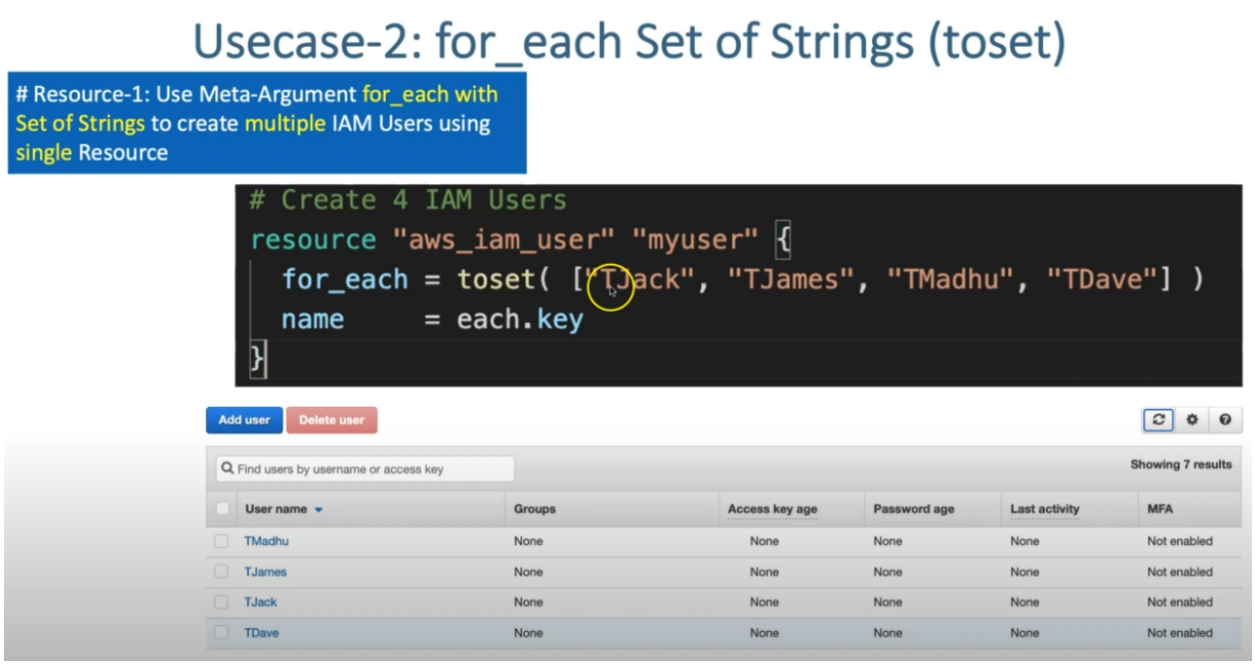
If your instances are almost identical, `count` is appropriate. If some of their arguments need distinct values that can't be directly derived from an integer, it's safer to use `for_each`.


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

Lecture 32


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.


Usecase-1: for_each Maps
Resource-1: Use Meta-Argument for_each with Maps to create multiple S3 buckets using single Resource
Define for_each with Map with Key Value pairs
Use each.key and each.value for the S3 Bucket name
Use each.key and each.value for S3 Bucket tags
Buckets (30)
Buckets are containers for data stored in S3. Learn more
my
Name
AWS Region
dev-my-dapp-bucket
US East (N. Virginia) us-east-1
prod-my-papp-bucket
US East (N. Virginia) us-east-1
qa-my-qapp-bucket
US East (N. Virginia) us-east-1
stag-my-sapp-bucket
US East (N. Virginia) us-east-1
# Create S3 Bucket per environment with for_each and maps
resource "aws_s3_bucket" "mys3bucket" {
for_each = {
dev = "my-dapp-bucket"
qa = "my-qapp-bucket"
stag = "my-sapp-bucket"
prod = "my-papp-bucket"
}
bucket = "${each.key}-${each.value}"
acl = "private"
tags = {
eachvalue = each.value
Environment = each.key
bucketname = "${each.key}-${each.value}"
}
}


Usecase-2: for_each Set of Strings (toset)
# Resource-1: Use Meta-Argument for_each with Set of Strings to create multiple IAM Users using single Resource
# Create 4 IAM Users
resource "aws_iam_user" "myuser" {
for_each = toset(["TJack", "TJames", "TMadhu", "TDave"])
name = each.key
}
Add user
Delete user
Find users by username or access key
Showing 7 results
User name
Groups
Access key age
Password age
Last activity
MFA
TMadhu
None
None
None
None
Not enabled
TJames
None
None
None
None
Not enabled
TJack
None
None
None
None
Not enabled
TDave
None
None
None
None
Not enabled

Lecture 33

```
resource "aws_s3_bucket" "tf_s3" {


  for_each = {

    dev  = "my-dapp-bucket"

    qa   = "my-qapp-bucket"

    stag = "my-sapp-bucket"

    prod = "my-papp-bucket"

  }

  #   each.key

  #   each.value

  bucket = each.value

  #   acl = "private"




  tags = {

    # Name         = "My bucket"

    Environment = each.key

    bucketname  = "${each.key}-${each.value}"

    eachvalue   = each.value

  }
```

```
}

# resource "aws_s3_bucket_acl" "name" {



# }
```

—

```
# resource "aws_instance" "tf_ec2" {

#   ami                    = "ami-08a52ddb321b32a8c"

#   instance_type          = "t2.micro"

#   key_name               = "tf-key"

#   subnet_id              = aws_subnet.pubsub_1.id

#   vpc_security_group_ids = [aws_security_group.pubsub-1-sg.id]

#   availability_zone      = "us-east-1a"

#   tags = {

#     Name = "E2-pubsub-1"

#     tag1 = "New tag"

#   }



#   # here EOF is used but we can also use file function.

#   user_data = <<-EOF

#               #!/bin/bash

#               sudo yum update -y

#               sudo yum install -y httpd

#               sudo systemctl start httpd
```

```
#               sudo systemctl enable httpd

#               echo "<h1>Welcome : Output form EOF</h1>" | sudo tee
/var/www/html/index.html

#               EOF

# }



resource "aws_instance" "tf_ec2" {

  ami           = "ami-08a52ddb321b32a8c" #Amazon Linux of us-east-1
region

  instance_type = "t2.micro"

  # key_name               = "tf-key"

  subnet_id = aws_subnet.pubsub_1.id

  # count     = 5

  # vpc_security_group_ids = [aws_security_group.pubsub-1-sg.id]

  # availability_zone      = "us-east-1a"

  tags = {

    Name = "E2-pubsub-1"

    # Name = "E2-pubsub-1-${count.index}"

    # tag1 = "New tag"

  }



}
```

```
terraform {

  # required_version = "~> 1.5"

  required_version = "~> 0"

  required_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "= 5.12.0"

    }



    random = {

      source  = "hashicorp/random"

      version = "3.4.3"

    }



  }

}



# provider "aws" {

#   profile = "default"

#   region = "us-ease-1"

# }
```

```
provider "aws" {

  profile = "default"

  region  = "us-east-1"

}



# provider "aws" {

#   profile = "default"

#   region  = "us-east-2"

#   alias = "aws-east-2"

# }
```

```
resource "aws_eip" "pubsub-1-eip" {

  instance = aws_instance.tf_ec2.id

  domain   = "vpc"



  #Meta argument

#   depends_on = [aws_internet_gateway.gw]

}
```

—

```
resource "aws_vpc" "my_vpc" {
  cidr_block = "192.168.0.0/26"

  tags = {
    Name = "main"
  }
}

# resource "aws_vpc" "my_vpc-us-east-2" {
#   cidr_block       = "192.168.0.0/26"
#   provider = aws.aws-east-2

#   tags = {
#     Name = "vpc-us-east-2"
#   }
# }

#Creating subnet
resource "aws_subnet" "pubsub_1" {
  vpc_id                  = aws_vpc.my_vpc.id
  cidr_block              = "192.168.0.0/27" #first public subnet
  availability_zone       = "us-east-1a"
```

```
  map_public_ip_on_launch = true #ensures the public ip of an instance


  tags = {
    Name = "Public Subnet 1"
  }
}


# Creating internet gateway
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.my_vpc.id


  tags = {
    Name = "pubsub-1-igw"
  }
}


# Creating route table
resource "aws_route_table" "pubsub-1-rt" {
  vpc_id = aws_vpc.my_vpc.id #required


  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.gw.id
```

```
  }


  tags = {
    Name = "pubsub-1-rt"
  }
}


# Subnet association to route table
resource "aws_route_table_association" "pubsub_1_association" {
  subnet_id      = aws_subnet.pubsub_1.id
  route_table_id = aws_route_table.pubsub-1-rt.id
}


# Creating security group
resource "aws_security_group" "pubsub-1-sg" {
  name = "pubsub-1-sg"
  # description = "Allow TLS inbound traffic"
  vpc_id = aws_vpc.my_vpc.id


  ingress {
    description = "Allow port 22"
    from_port   = 22
    to_port     = 22
```

```
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

ingress {
  description = "Allow port 80"
  from_port   = 80
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

egress {
  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
  # ipv6_cidr_blocks = ["::/0"]
}

tags = {
  Name = "Allow port 22 and 80"
}
```

```
}
```

```
resource "aws_instance" "tf_ec2" {
  ami           = "ami-0ccabb5f82d4c9af5"
  instance_type = "t2.micro"
  availability_zone = "us-east-2a"
  tags = {
    Name = "terraform_ec2"
    tag1 = "New tag"
  }
}
```

```
resource "aws_vpc" "my_vpc" {
  cidr_block       = "192.168.0.0/26"

  tags = {
    Name = "main"
  }
}

# resource "aws_vpc" "my_vpc-us-east-2" {
#   cidr_block       = "192.168.0.0/26"
#   provider = aws.aws-east-2

#   tags = {
```

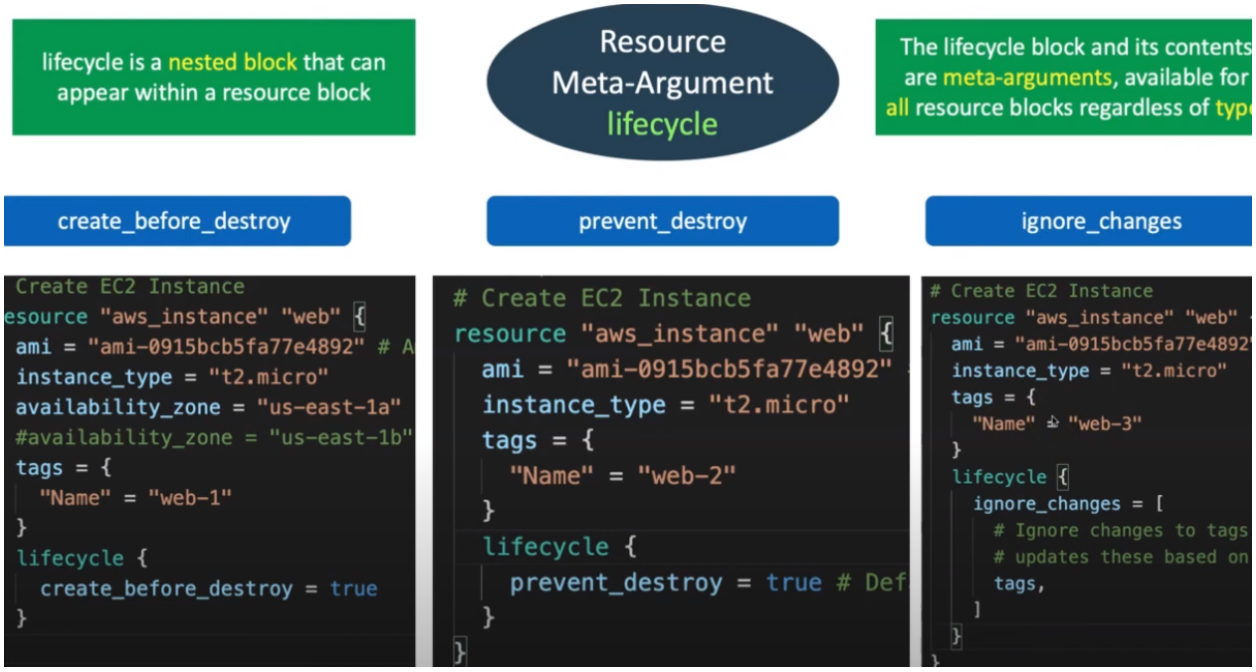
```
#    Name = "vpc-us-east-2"
#  }
# }
```

Lecture 34

Toset example:

```
resource "aws_iam_role" "toset_foreach" {
    for_each = toset(["TJack", "Tjames, "Tmahhdu"])
    name                = each.key
}
```

Lecture 33



Lecture 38

lifecycle is a nested block that can appear within a resource block

Resource Meta-Argument lifecycle

The lifecycle block and its contents are meta-arguments, available for all resource blocks regardless of type.

create_before_destroy

```
# Create EC2 Instance
resource "aws_instance" "web" {
  ami = "ami-0915bcb5fa77e4892" # A
  instance_type = "t2.micro"
  availability_zone = "us-east-1a"
  #availability_zone = "us-east-1b"
  tags = {
    "Name" = "web-1"
  }
  lifecycle {
    create_before_destroy = true
  }
}
```

prevent_destroy

```
# Create EC2 Instance
resource "aws_instance" "web" {
  ami = "ami-0915bcb5fa77e4892"
  instance_type = "t2.micro"
  tags = {
    "Name" = "web-2"
  }
  lifecycle {
    prevent_destroy = true # Def
  }
}
```

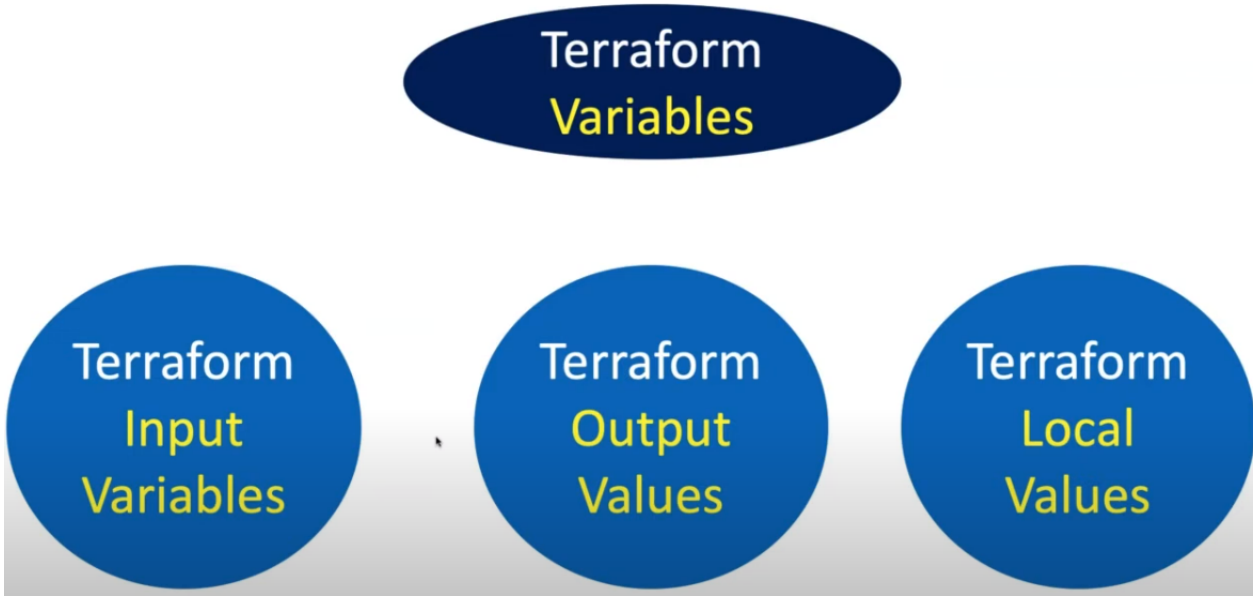
ignore_changes

```
# Create EC2 Instance
resource "aws_instance" "web" {
  ami = "ami-0915bcb5fa77e4892"
  instance_type = "t2.micro"
  tags = {
    "Name" = "web-3"
  }
  lifecycle {
    ignore_changes = [
      # Ignore changes to tags,
      # updates these based on
      tags,
    ]
  }
}
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
Kalyans-MacBook-Pro:v2-prevent_destroy kdaida$ terraform destroy -auto-approve

Error: Instance cannot be destroyed

  on c2-ec2-instance.tf line 2:
   2: resource "aws_instance" "web" {
```

```
1  # Create EC2 Instance
2  resource "aws_instance" "web" {
3    ami = "ami-0915bcb5fa77e4892" # Amazon Linux
4    instance_type = "t2.micro"
5    tags = {
6      "Name" = "web-2"
7    }
8    /*
9    lifecycle {
10     prevent_destroy = true # Default is false
11   }
12 }
13
14
```

Lecture 41



Why S3: S3 buckets aim to help enterprises and individuals achieve their data backup and delivery needs. S3 buckets also enable a large amount of data to be stored and accessed later through cloud storage. Most data stored by enterprises in S3 buckets is for big data analytics, disaster recovery, dynamic websites, and user-generated content, among other uses. Some enterprises also use S3 buckets to host static HTML websites and dynamically complex web applications.

`terraform destroy` does not delete the S3 Bucket ACL but does remove the resource from Terraform state.

What is an S3 bucket ACL?

Amazon S3 access control lists (ACLs) enable you to manage access to S3 buckets and objects. Every S3 bucket and object has an ACL attached to it as a subresource. The ACLs define which AWS accounts or groups are granted access along with the type of access.

- `bucket` - (Required) Name of the bucket that you want to associate this access point with.
- `rule` - (Required) Configuration block(s) with Ownership Controls rules. Detailed below.

rule Configuration Block

The following arguments are required:

- `object_ownership` - (Required) Object ownership. Valid values: `BucketOwnerPreferred`, `ObjectWriter` or `BucketOwnerEnforced`
    - `BucketOwnerPreferred` - Objects uploaded to the bucket change ownership to the bucket owner if the objects are uploaded with the `bucket-owner-full-control` canned ACL.
    - `ObjectWriter` - Uploading account will own the object if the object is uploaded with the `bucket-owner-full-control` canned ACL.
    - `BucketOwnerEnforced` - Bucket owner automatically owns and has full control over every object in the bucket. ACLs no longer affect permissions to data in the S3 bucket.

Why bucket ownership control block is used:

The aws_s3_bucket_ownership_controls resource is used to manage the ownership controls of an S3 bucket. Ownership controls are a feature of S3 that allow you to control who owns objects that are uploaded to the bucket.
By default, an object that is uploaded to an S3 bucket is owned by the AWS account that uploaded the object, not the owner of the bucket. This can lead to situations where the bucket

owner does not have access to objects in their own bucket, or where they are not able to manage access permissions for those objects.
The aws_s3_bucket_ownership_controls resource allows you to change this behavior. By setting the object_ownership attribute to BucketOwnerPreferred or BucketOwnerEnforced, you can ensure that the bucket owner will own all new objects that are uploaded to the bucket, regardless of who uploaded them.
This is particularly useful in situations where you want to ensure that the bucket owner has full control over all objects in the bucket, or where you want to centralize the management of access permissions.

Why ownership and acl is used:(Overall)

ACLs (Access Control Lists) and ownership controls are two mechanisms provided by AWS S3 to manage access permissions and ownership of objects within S3 buckets.

1. ACLs: ACLs allow you to control access to your S3 bucket and objects at the individual object level. With ACLs, you can grant or deny read and write permissions to specific AWS accounts or groups. ACLs provide a more granular level of access control compared to bucket policies or IAM policies, as they can be applied to individual objects within a bucket.

Key points about ACLs:

- ACLs define who can access the bucket or object and the level of access they have, such as read or write.
- ACLs are a legacy access control mechanism in S3.
- By default, ACLs are disabled, and the bucket owner automatically has full control over every object in the bucket.
- However, ACLs can be useful in scenarios where you need specific access control for individual objects within the bucket.

2. Ownership Controls: Ownership controls help ensure that the bucket owner has full control over the objects in their S3 bucket. By default, the AWS account that uploads an object owns that object. This can lead to situations where the bucket owner does not have access to their own objects or cannot manage access permissions for those objects.

Ownership controls address this issue by allowing the bucket owner to become the owner of every object uploaded to the bucket, regardless of who performed the upload. The most common ownership control rule is the "BucketOwnerPreferred" rule. When this rule is enabled, the bucket owner becomes the object owner, giving them complete control over the objects' access and permissions.
Key points about ownership controls:

- Ownership controls ensure that the bucket owner has full control over every object in the bucket.
- The "BucketOwnerPreferred" rule is commonly used to specify that the bucket owner should be the owner of every object uploaded to the bucket.
- Ownership controls provide a way to centralize ownership and access control within the bucket owner's AWS account.

```
Error: Insufficient rule blocks

  on s3_bucket.tf line 26, in resource "aws_s3_bucket_ownership_controls" "example":
  26: resource "aws_s3_bucket_ownership_controls" "example" {

At least 1 "rule" blocks are required.
```

```
$ terraform plan

Error: Missing required argument

  on s3_bucket.tf line 40, in resource "aws_s3_bucket_acl" "tf_s3_acl":
  40: resource "aws_s3_bucket_acl" "tf_s3_acl" {

The argument "bucket" is required, but no definition was found.
```

This problem occurs if

```
depends_on = [aws_s3_bucket_ownership_controls.example]
is not provided.
```

```
Error: creating S3 bucket ACL for prod-my-papp-bucket-keya: AccessControlListNotSupported: The bucket does not allow ACLs
      status code: 400, request id: HE4CYQA3JWG3989B, host id: UA855t+LA+3Sr8TRRRj3VDEKQwP4Dd4Hdm7xxQgk7e3jUfeSj25lchp2ui4T31HpsSl2cRoorws=

  with aws_s3_bucket_acl.tf_s3_acl["prod"],
  on s3_bucket.tf line 40, in resource "aws_s3_bucket_acl" "tf_s3_acl":
  40: resource "aws_s3_bucket_acl" "tf_s3_acl" {
```

Lecture 42

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.

Lecture 43

Setting the alias: terraform

Follow below steps:

1. **Open the file `.bashrc` which is found in location `C:\Users\USERNAME\.bashrc`**
   If file `.bashrc` not exist then create it using below steps:

   1. **Open Command Prompt and goto `C:\Users\USERNAME\`.**

2. Type command `notepad ~/.bashrc`
   It generates the `.bashrc` file.
2. Add below sample commands of WP CLI, Git, Grunt & PHPCS etc.

```
# ----------------------
# Git Command Aliases
# ----------------------
alias ga='git add'
alias gaa='git add .'
alias gaaa='git add --all'


# ----------------------
# WP CLI
# ----------------------
alias wpthl='wp theme list'
alias wppll='wp plugin list'
```

Var region:

```
provider "aws" {
  profile = "default"
  # region  = "us-east-1"
  region = var.aws_region
}
```

```
variable "aws_region" {
    type = string
    default = "us-east-1"
}
```

```
egress {
  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]
  # ipv6_cidr_blocks = ["::/0"]
}
```

In the code snippet you provided, the egress rule allows all outbound traffic (from_port = 0, to_port = 0) with any protocol (protocol = -1) to any destination (cidr_blocks = ["0.0.0.0/0"]). This effectively allows all traffic to be sent from the associated instances to any destination IP address.

Lecture 45:
Input variables assigned when prompted

```
instance_type = var.ec2_instance_type #promt example
```

```
$ tf plan
var.ec2_instance_type
  EC2 instance using variable

  Enter a value: t2.micro
```

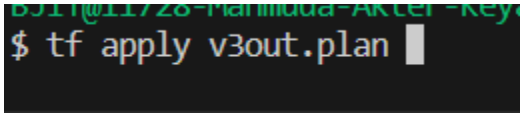
```
instance_type                                = "t2.micro" -> null
```

Lecture 46
-var in CLI

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ tf plan -var="ec2_instance_count=2" -var="ec2_instance_type=t2.micro"
```

tf plan -var="ec2_instance_count=2" -var="ec2_instance_type=t2.micro"

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ tf plan -var="ec2_instance_count=2" -var="ec2_instance_type=t2.micro" -out
v3out.plan
```

- **-out v3out.plan:** This flag is used to specify the output file name for the execution plan. In this case, the plan will be saved to a file named v3out.plan. -out=FILE option to save the generated plan to a file on disk.
- 

to run the code

Lecture 47

Input variable with environment variable.

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ export TF_VAR_ec2_instance_count=1

BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ export TF_VAR_ec2_instance_type=t2.micro
```

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ echo $TF_VAR_ec2_instance_count, $TF_VAR_ec2_instance_type
1, t2.micro
```

Unsetting the environment variable:

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$  unset TF_VAR_ec2_instance_count

BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$  unset TF_VAR_ec2_instance_type

BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ echo $TF_VAR_ec2_instance_count, $TF_VAR_ec2_instance_type
,
```

Lecture 48

Input from terraform.tfvars file.

- If the file name is `terraform.tfvars`, terraform will auto-load the variables present in this file by overriding the `default` values in `variables.tf`

```
terraform.tfvars > ec2_instance_type
1   ec2_instance_count = 2
2   ec2_instance_type = "t3.micro"
```

Lecture 49

Input Variables Assign with var file argument. We use this for using separate .tfvar files.

Example

terraform plan -var-file=”web.tfvars”

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ tf plan -var-file="app.tfvars"
```

```
~ resource "aws_instance" "tf_ec2" {
      id                                   = "i-0f1f5332ee4f30c8e"
    ~ instance_type                        = "t2.micro" -> "t3.micro"
```

```
BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ tf plan -var-file="web.tfvars"
```

```
~ instance_type                        = "t2.micro" -> "t3.small"
```

Lecture 50

Input variable with .auto.tfvars

```
web.auto.tfvars > ec2_instance_type
1   ec2_instance_type = "t3.micro"
```

Whatever stays in this file will auto load during terraform plan and apply. Apply terraform plan

Output:

```
# aws_instance.tf_ec2[1] will be updated in-place
~ resource "aws_instance" "tf_ec2" {
      id                                   = "i-0f1f5332ee4f30c8e"
    ~ instance_type                        = "t2.micro" -> "t3.micro"
      tags                                 = {
          "Name" = "E2-pubsub-1"
```

Lecture 51

- **list (or tuple): a sequence of values, like ["us-west-1a", "us-west-1c"]. Elements in a list or tuple are identified by consecutive whole numbers, starting with zero.**

```
#promt example
variable "ec2_instance_type" {
  description = "EC2 instance using variable"
  #   type         = string
  type = list(string)
  default = [ "t3.micro", "t3.small", "t3.large" ]
  #   default = "t3.micro" # -var cli
}
```

and output

```
# aws_instance.tf_ec2[0] will be updated in-place
~ resource "aws_instance" "tf_ec2" {
      id                                   = "i-0b718b8a564c62354"
    ~ instance_type                        = "t2.micro" -> "t3.micro"
      tags                                 = {
          "Name" = "E2-pubsub-1"
```

Lecture 52

Input Variables Complex Constructor of Type Map

```
instance_type = var.ec2_instance_type_map["small-apps"]
```

```
variable "ec2_instance_type_map" {
  description = "EC2 instance type"
  type = map(string)
  default = {
    "small-apps" = "t3.micro"
    "medium-apps" = "t3.medium"
    "big-apps" = "t3.large"
  }
}
```

Lecture 53

Custom validation rules in Variables

Terraform console:

The Terraform console is an interpreter that you can use to evaluate Terraform expressions and explore your Terraform project's state.

Scripting

```
$ echo "1+5" | terraform console
6
```

Terraform console command prompt

```
$ terraform console
> 1+5
6
```

Length function

- Length of a string

```
> length("😒😎😁🤣😊")
5
```

- Length of a list

```
> length(["a", "b", "c"])
3
```

- Length of Map

```
> length({"key" = "value"})
1
> length({"key1" = "value", "key2" = "value2"})
2
```

substr Function

substr(string, offset, length)
`substr` extracts a substring from a given string by offset and (maximum) length.

The offset index may be negative, in which case it is relative to the end of the given string. The length may be -1, in which case the remainder of the string after the given offset will be returned.

> substr("hello world", -5, -1)

world

If the length is greater than the length of the string, the substring will be the length of all remaining characters.

> substr("hello world", 6, 10)

world

Some example:

```
> substr("stack simplify", 1, 4)
"tack"
> substr("stack simplify", 0, 6)
"stack "
> substr("stack simplify", 0, 1)
"s"
> substr("stack simplify", 0,0)
""
> substr("stack simplify", 0, 10)
"stack simp"
```

Lecture 54

Custom validation rule

```
variable "ec2_ami_id" {
  type    = string
  default = "ami-08a52ddb321b32a8c"
  validation {
    condition     = length(var.ec2_ami_id) > 4 && substr(var.ec2_ami_id, 0, 4) == "ami-"
    error_message = "The image_id value must be a valid AMI id, starting with \"ami-\"."
  }
}
```

Lecture 55

Protect sensitive input variable

Often we need to configure our infrastructure using sensitive or secret information such as usernames, passwords, API tokens, or Personally Identifiable Information (PII). When we do so, we need to ensure that you do not accidentally expose this data in CLI output, log output, or source control. Terraform provides several features to help avoid accidentally exposing sensitive data. In this example, we are working with rds database username and password as sensitive information.

Example:

```
variable "db_useranme" {
  description = "AWS RDS Database Administrator Username"
  type = string
  sensitive = true
}

variable "db_password" {
  description = "AWS RDS Database Administrator Password"
  type = string
  sensitive = true
}
```

rds-db.tf > resource "aws_db_instance" "db1"

```
resource "aws_db_instance" "db1" {
  allocated_storage    = 5
  engine               = "mysql"
  instance_class       = "db.t2.micro"
#   name                 = "mydb1"
  engine_version       = "14.1"
  username             = var.db_useranme
  password             = var.db_password
  skip_final_snapshot = true
}
```

Lecture 56

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.

- 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.)

Lecture 57

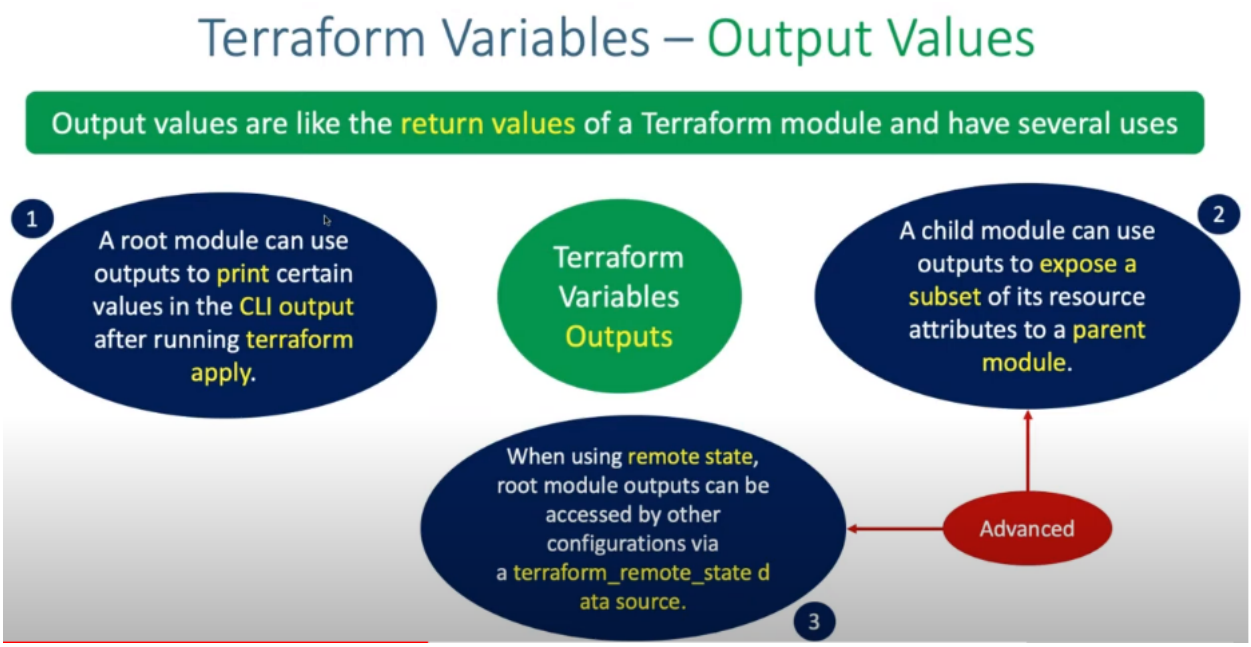
Terraform file function

`file` reads the contents of a file at the given path and returns them as a string.

```
user_data = file("apache-install.sh")
```

Lecture 58

Terraform output values



Lecture 59

Attribute reference

Argument reference

Output

```
Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

Outputs:

ec2_instacne_private_ip = "192.168.0.5"
ec2_instacne_public_ip = "54.86.192.214"
```

Lecture 60

We should get tf output ec2_security_groups in a non redacted value from terraform.tfstate file.

```
ec2_instacne_private_ip = "192.168.0.11"
ec2_instacne_public_ip = "44.201.209.55"
ec2_security_groups = <sensitive>

BJIT@11728-Mahmuda-Akter-Keya MINGW64 /d/terraform tutorial
$ tf output ec2_security_groups
toset([])
```

Terraform output in json format:

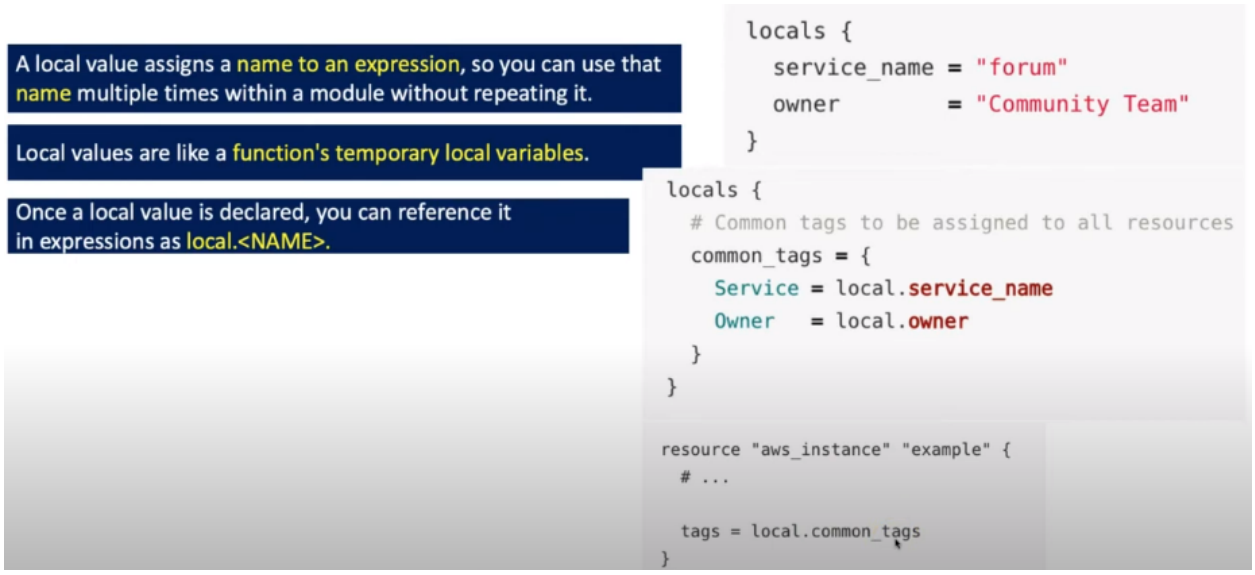
```
$ tf output -json
{
  "ec2_instacne_private_ip": {
    "sensitive": false,
    "type": "string",
    "value": "192.168.0.11"
  },
  "ec2_instacne_public_ip": {
    "sensitive": false,
    "type": "string",
    "value": "44.201.209.55"
  },
  "ec2_security_groups": {
    "sensitive": true,
    "type": [
      "set",
      "string"
    ],
    "value": []
  }
}
```

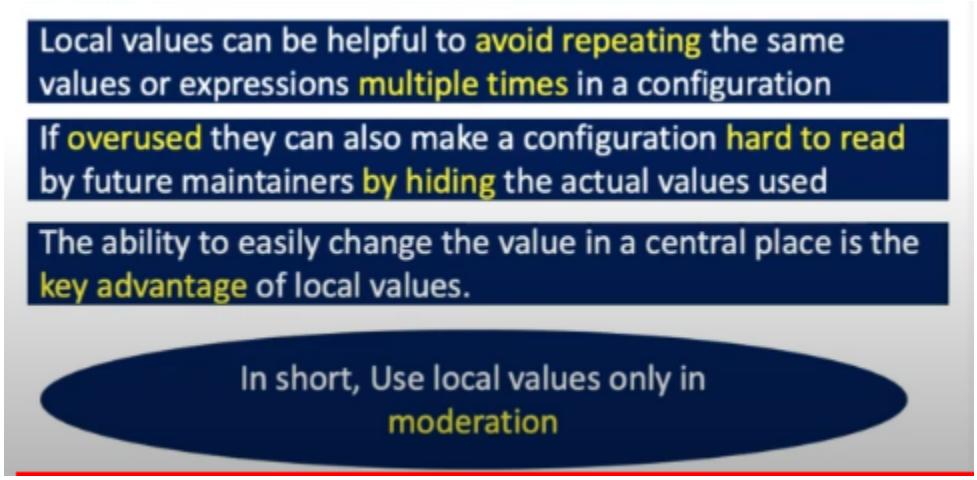
Lecture 61

Terraform Local Values

Dry principal

Using dry principal, we can reduce the repetition of the code or we can reuse the code.





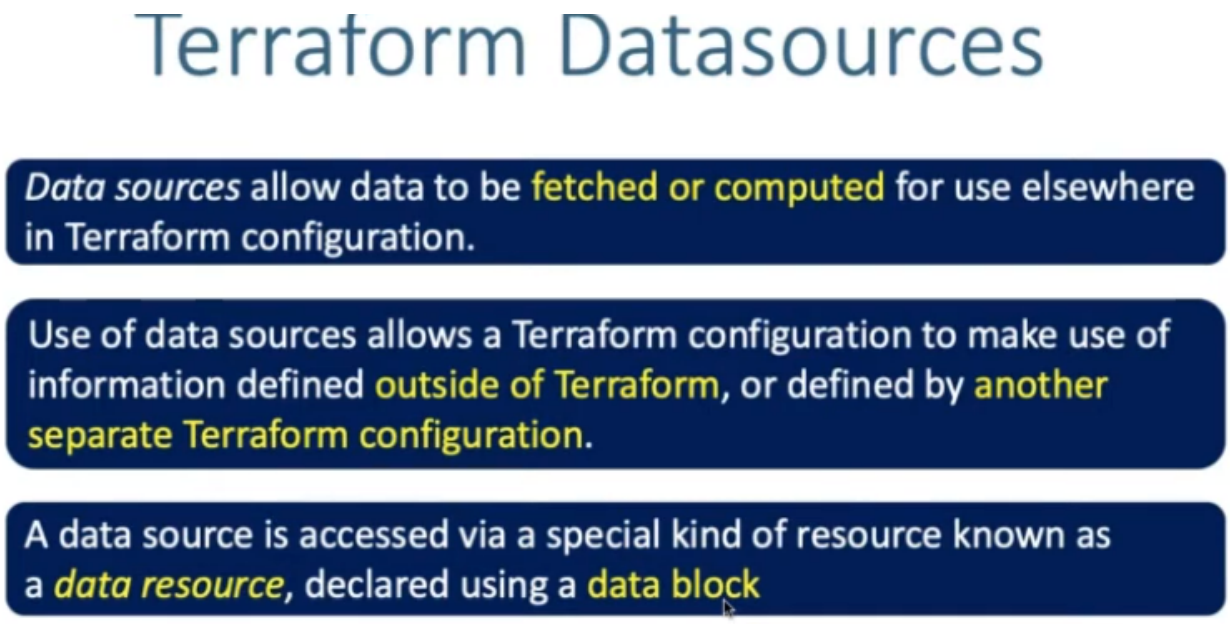
Lecture 62

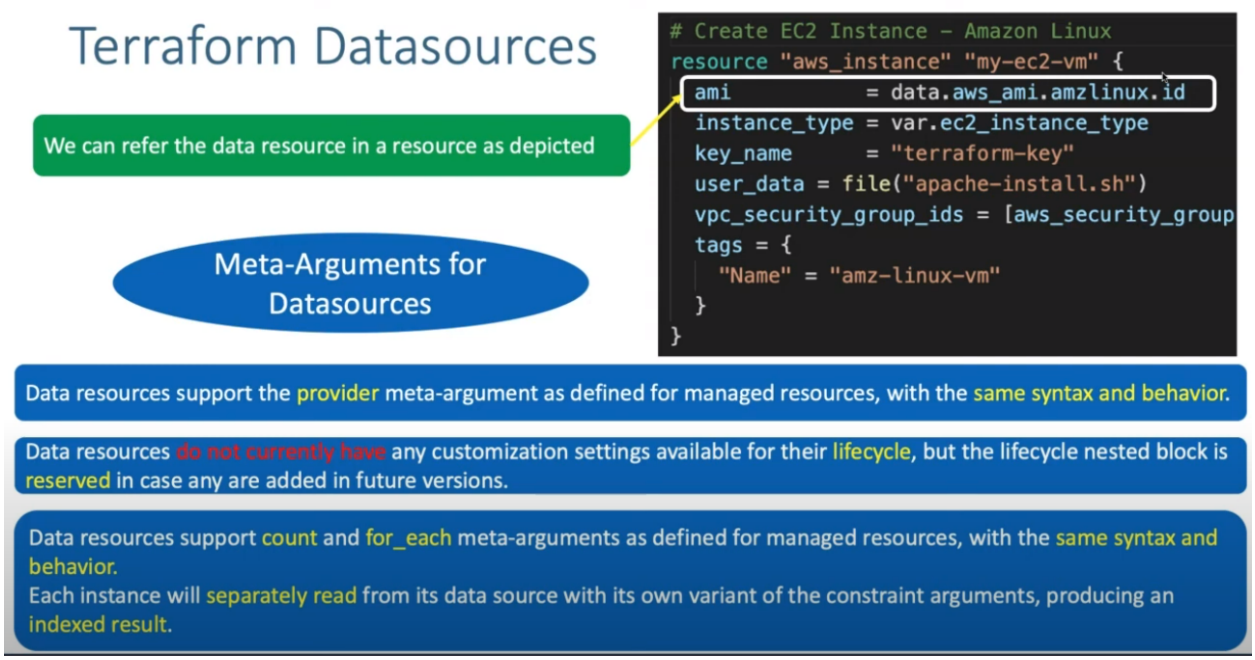
```
# local variable example

resource "aws_s3_bucket" "tf_s3" {

  bucket = local.bucket-name

  tags = {
    Name        = local.bucket-name
    Environment = var.environment_name
  }
}

#adding controller_access
```

```
resource "aws_s3_bucket_ownership_controls" "example" {

  bucket = local.bucket-name        #Required
  rule {                                  #Required
    object_ownership = "BucketOwnerPreferred" #Required
  }
  depends_on = [aws_s3_bucket.tf_s3]
}

resource "aws_s3_bucket_acl" "tf_s3_acl" {
  depends_on = [aws_s3_bucket_ownership_controls.example]


  bucket = local.bucket-name
  acl    = "private"
}

locals {
  bucket-name = "${var.app_name}-${var.environment_name}-bucket"
}
```

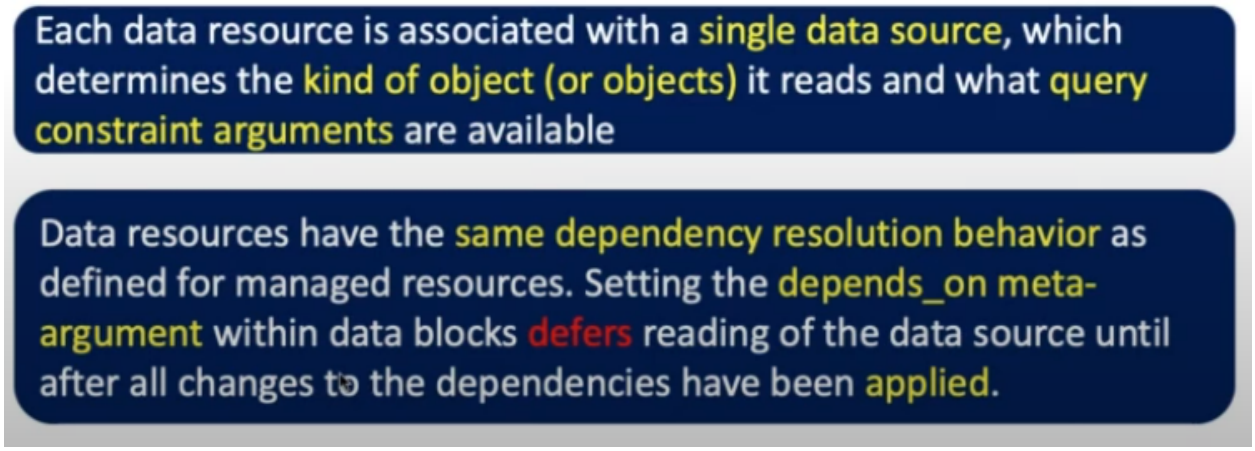
Lecture 63

Terarform Data Sources





Datasource is Data block.



Reference:

- https://www.youtube.com/playlist?list=PLxBANRJzAw7i__xMJH-w2vzV8nP_AclP6
- https://developer.hashicorp.com/terraform/language/functions/file
- https://registry.terraform.io/providers/figma/aws-4-49-0/latest/docs/data-sources/ami
- https://registry.terraform.io/providers/figma/aws-4-49-0/latest/docs
- https://developer.hashicorp.com/terraform/language/functions/file
- https://developer.hashicorp.com/terraform/cli/commands/console
- https://spacelift.io/blog/terraform-console
- https://registry.terraform.io/providers/terraform-redhat/rhcs/latest/docs/guides/terraform-vars
- https://github.com/stacksimplify/hashicorp-certified-terraform-associate/blob/main/05-Terraform-Variables/05-01-Terraform-Input-Variables/README.md
- https://spacelift.io/blog/terraform-tfvars
- https://developer.hashicorp.com/terraform/language/values/variables
- https://developer.hashicorp.com/terraform/language/providers/configuration
- https://developer.hashicorp.com/terraform/language/providers/configuration
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket_acl
- https://registry.terraform.io/providers/BigEyeLabs/aws-test/latest/docs/resources/s3_bucket_ownership_controls
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/eip
- https://developer.hashicorp.com/terraform/language/meta-arguments/for_each
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/route_table
- https://developer.hashicorp.com/terraform/language/expressions
- https://developer.hashicorp.com/terraform/language/meta-arguments/count
- https://developer.hashicorp.com/terraform/language/attr-as-blocks#arbitrary-expressions-with-argument-syntax
- https://developer.hashicorp.com/terraform/language/attr-as-blocks
- https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
- https://developer.hashicorp.com/terraform/tutorials/configuration-language/resource
- https://developer.hashicorp.com/terraform/tutorials/aws-get-started/aws-build
- https://developer.hashicorp.com/terraform/cli/commands/init#upgrade
- https://registry.terraform.io/providers/hashicorp/aws/3.74.3/docs/resources/vpc

- https://www.digitalocean.com/community/tutorials/how-to-create-reusable-infrastructure-with-terraform-modules-and-templates
- https://developer.hashicorp.com/terraform/language/resources/provisioners/syntax