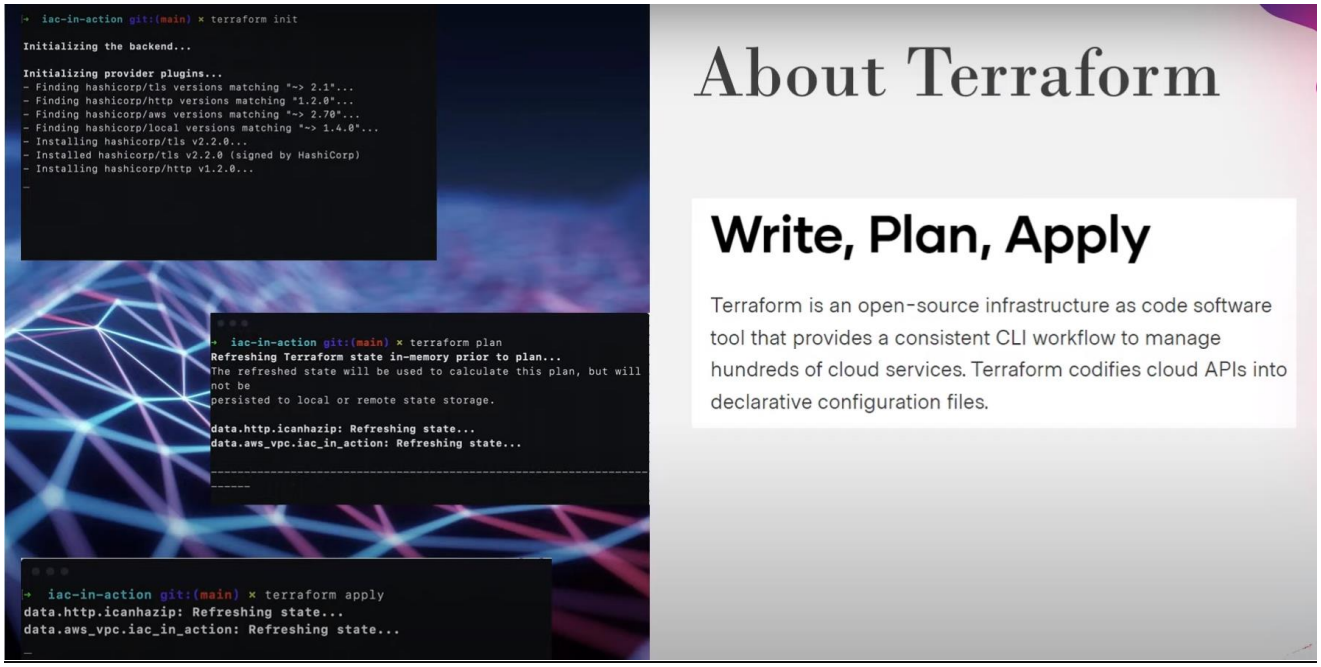
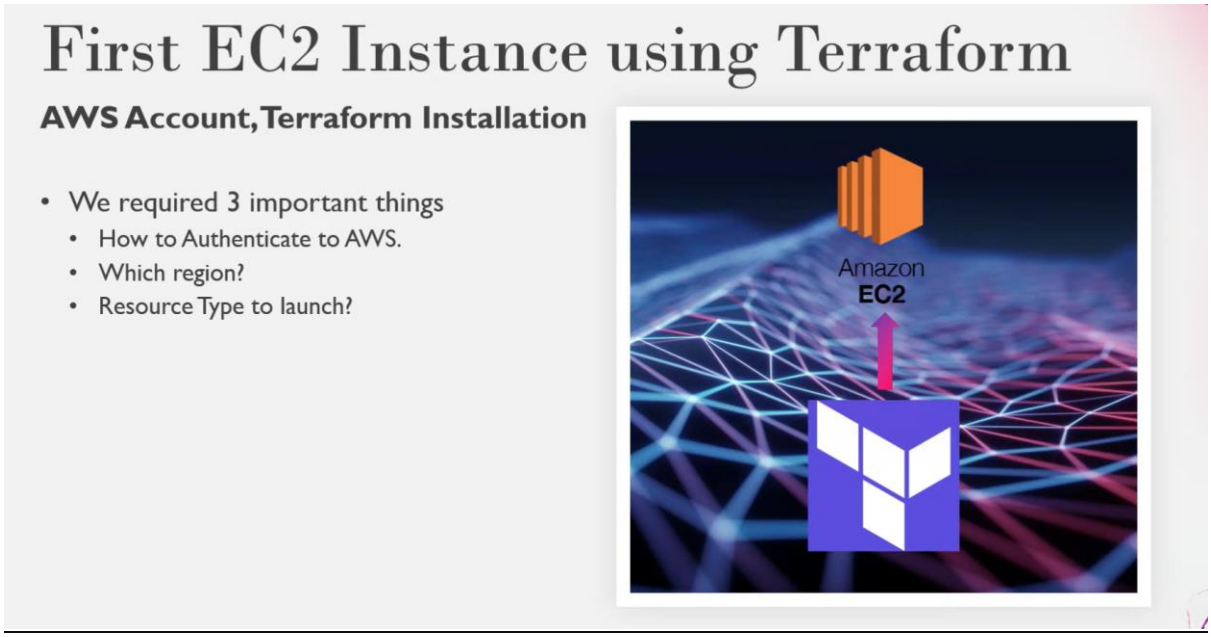
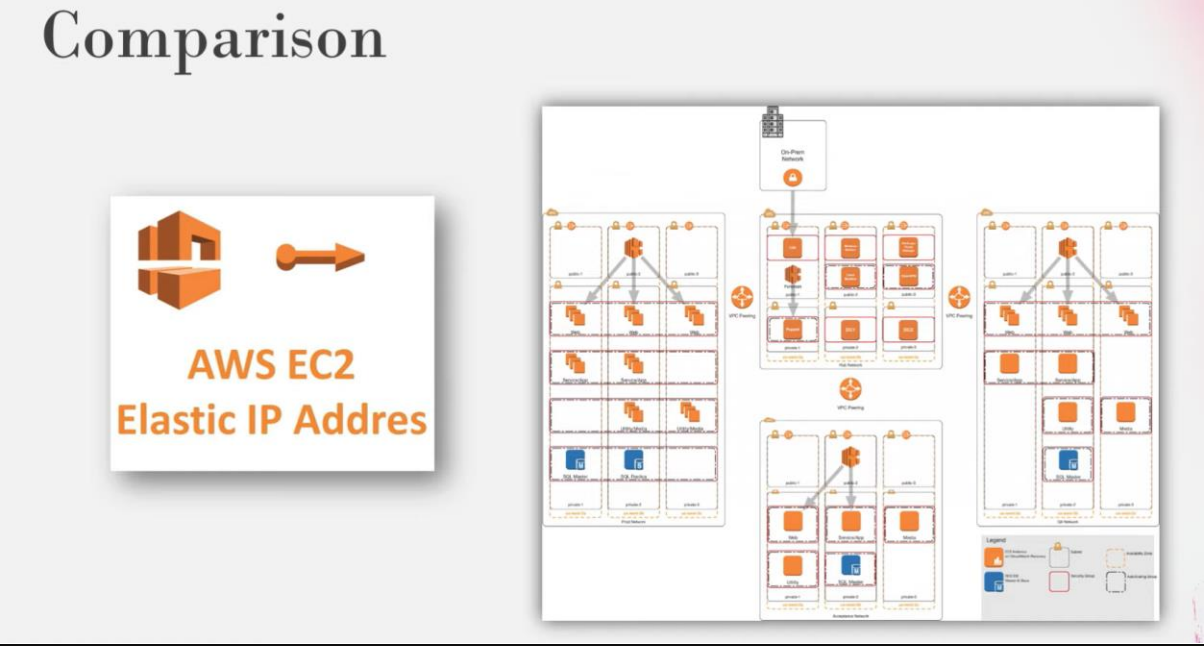
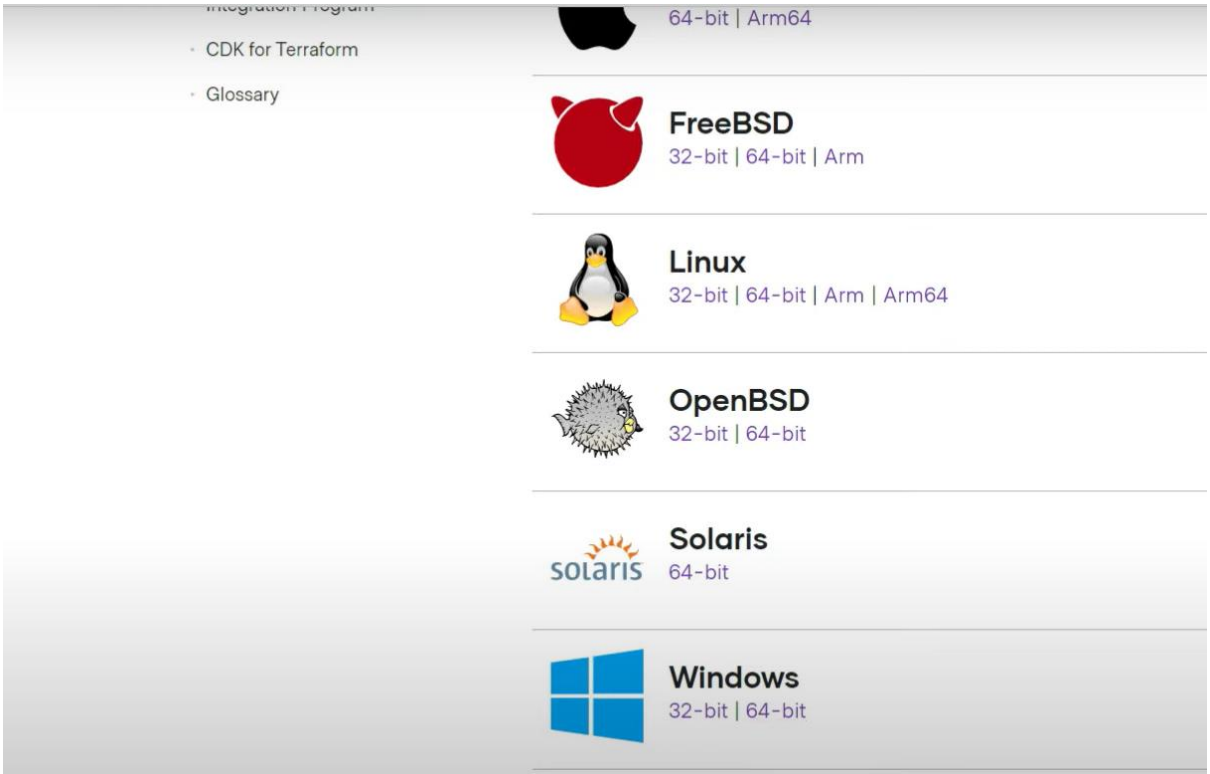
Basic Concept : Creating First EC2 Instance Terraform



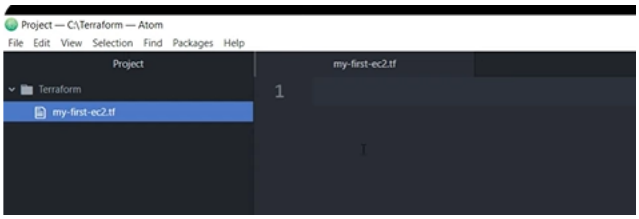




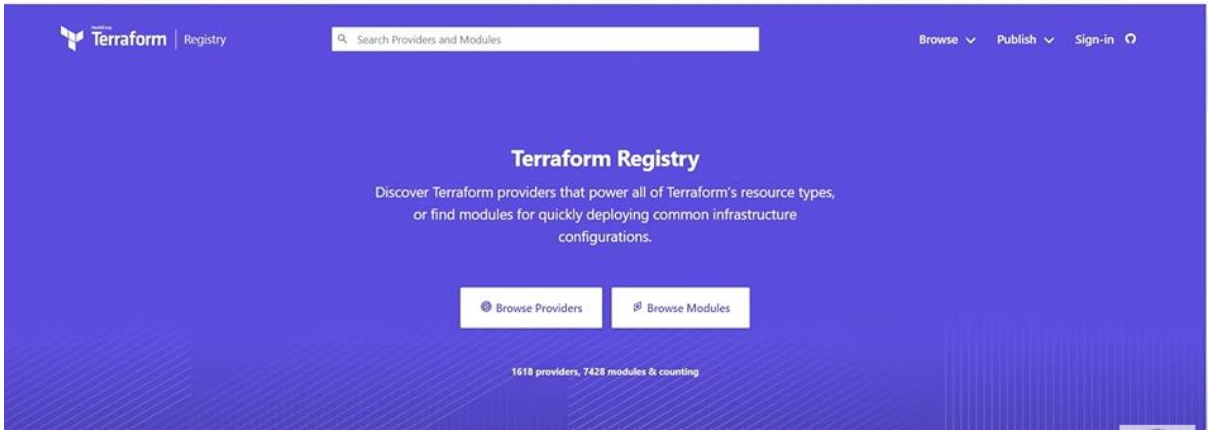
Terraform installation on windows

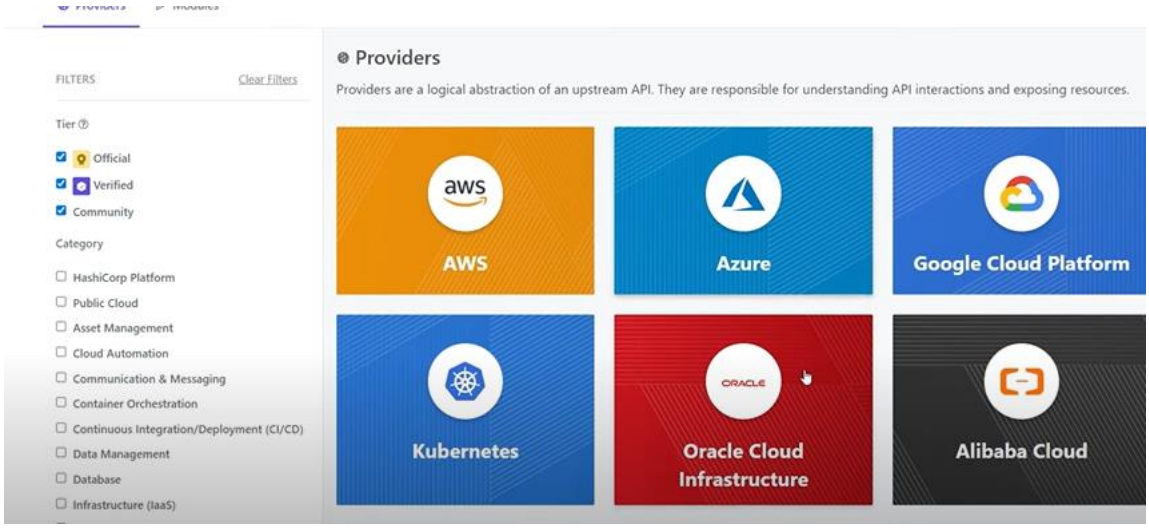


Using Editor:



Terraform Registry:





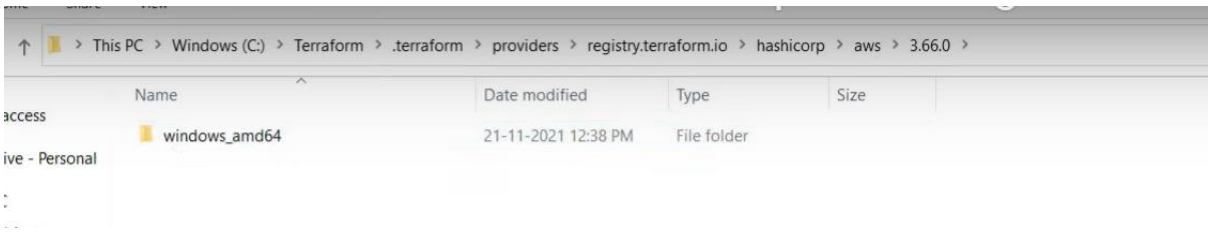
my-first-ec2.tf

Press Esc to exit full screen

```
provider "aws" {
  region = "ap-south-1"
  access_key = "AKIAX6W4IEODQT2PPO6N"
  secret_key = "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"
}

resource "aws_instance" "my-fisrt-ec2" {
  ami           = "ami-0108d6a82a783b352"
  instance_type = "t2.micro"

  tags = {
    Name = "My-First-EC2-Using-TF"
  }
}

```

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.66.0...
- Installed hashicorp/aws v3.66.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

As soon as terraform init. terraform file will get generated.



Creating one more provider and need to reinitialize

my-first-ec2.tf

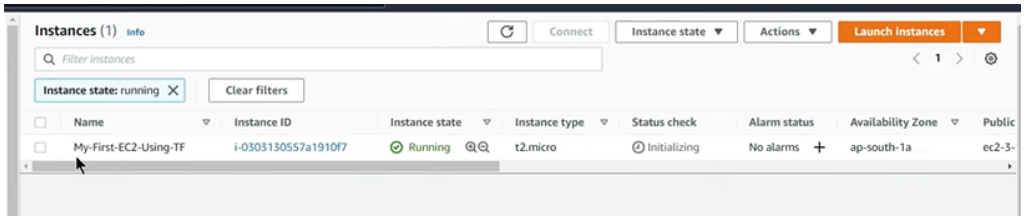
```
provider "aws" {
  region = "ap-south-1"
  access_key = "AKIAX6W4IEODQT2PPO6N"
  secret_key = "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"
}

provider "azurerm" {}

resource "aws_instance" "my-fisrt-ec2" {
  ami           = "ami-0108d6a82a783b352"
  instance_type = "t2.micro"

  tags = {
    Name = "My-First-EC2-Using-TF"
  }
}
```

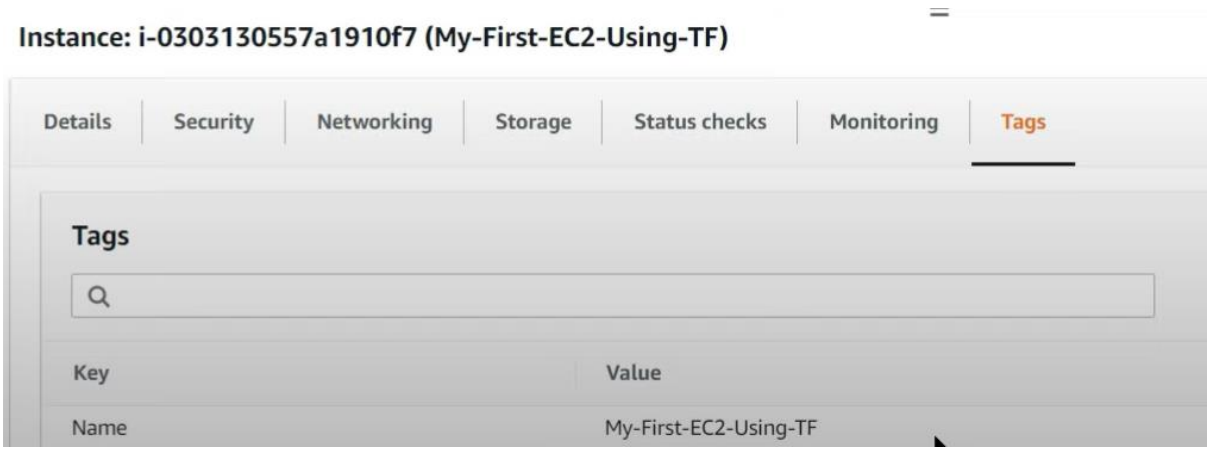
```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding latest version of hashicorp/azurerm...
- Using previously-installed hashicorp/aws v3.66.0
- Installing hashicorp/azurerm v2.86.0...
- Installed hashicorp/azurerm v2.86.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

```
PS C:\Terraform> 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_instance.my-fisrt-ec2 will be created
  + resource "aws_instance" "my-fisrt-ec2" {
      + ami                                  = "ami-0108d6a82a783b352"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
      + cpu_core_count                       = (known after apply)
      + cpu_threads_per_core                 = (known after apply)
      + disable_api_termination              = (known after apply)
      + ebs_optimized                        = (known after apply)
      + get_password_data                    = false
      + host_id                              = (known after apply)
      + id                                   = (known after apply)
      + instance_initiated_shutdown_behavior = (known after apply)
```

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply"
now.
PS C:\Terraform> terraform apply

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_instance.my-fisrt-ec2 will be created
  + resource "aws_instance" "my-fisrt-ec2" {
      + ami                                  = "ami-0108d6a82a783b352"
      + arn                                  = (known after apply)
      + associate_public_ip_address          = (known after apply)
      + availability_zone                    = (known after apply)
```

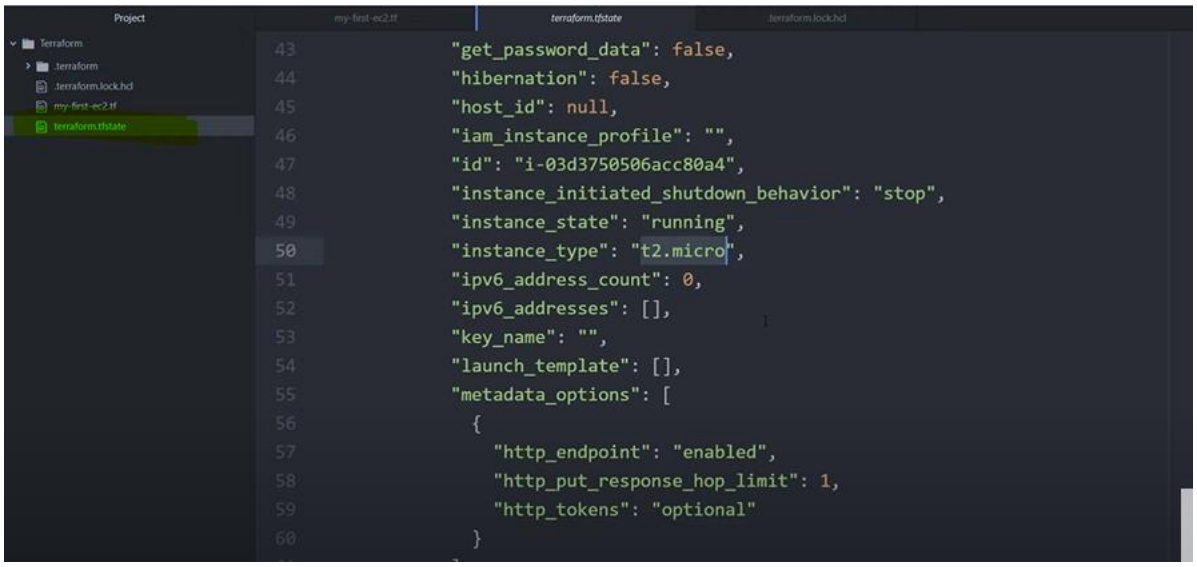
```
Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

aws_instance.my-fisrt-ec2: Creating...
aws_instance.my-fisrt-ec2: Still creating... [10s elapsed]
```



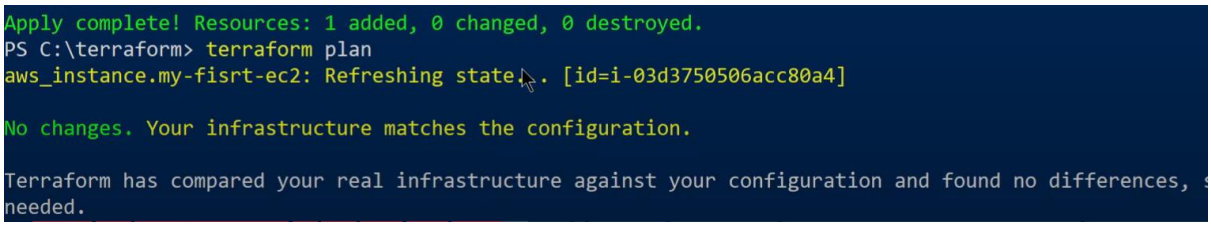


1 - Basic Concept - tfstate backup and destroy, target flag.

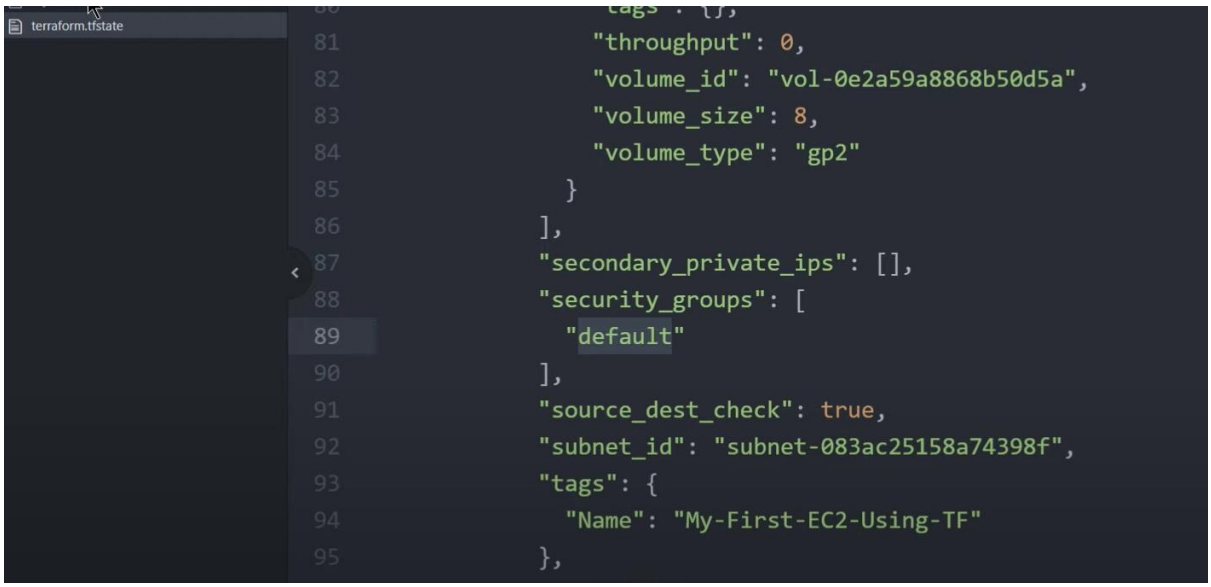
Tfstate file : Initial state of a file



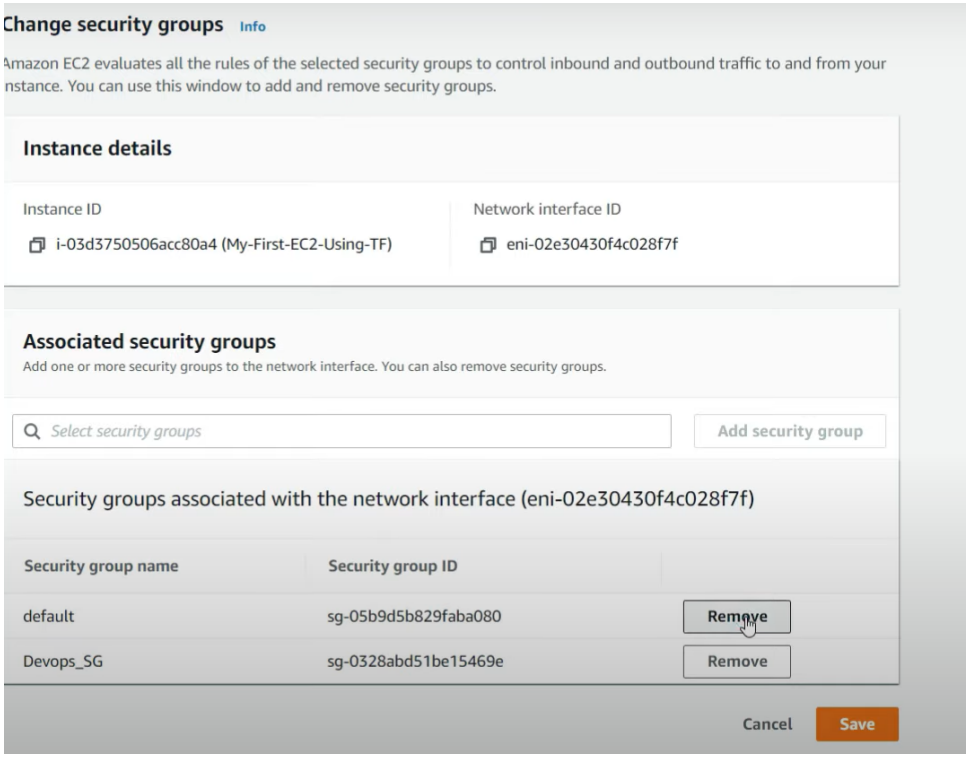
Tfsate backup: if we make any changes/modifications it will create a backup file. It is a step back configuration



From AWS console we are making changes not from terraform code. (terraform compare changes Actual state and desired state) Terraform state file /configuration file is consisting of **default** security group.



From console **making change** in security group.



It's showing SG is got changed.

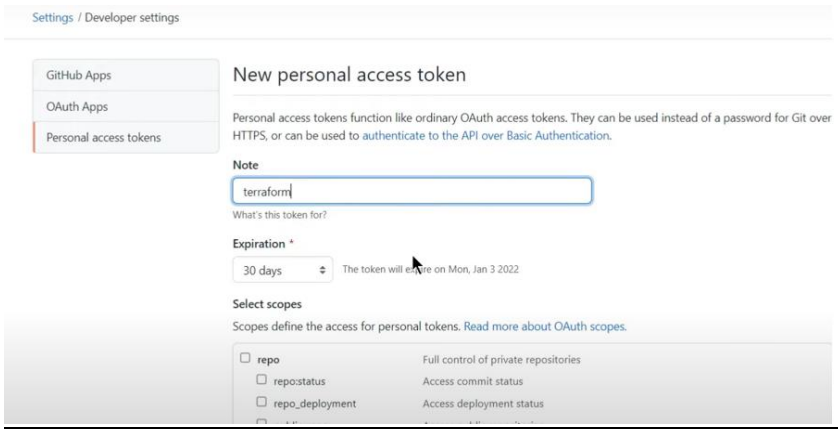
```
PS C:\terraform> terraform plan
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply":

  # aws_instance.my-fisrt-ec2 has been changed
  ~ resource "aws_instance" "my-fisrt-ec2" {
        id                                   = "i-03d3750506acc80a4"
      ~ security_groups                      = [
          + "Devops_SG",
          - "default",
        ]
        tags                                 = {
            "Name" = "My-First-EC2-Using-TF"
        }
      ~ vpc_security_group_ids               = [
          + "sg-0328abd51be15469e",
          - "sg-05b9d5b829faba080",
        ]
        # (26 unchanged attributes hidden)
```

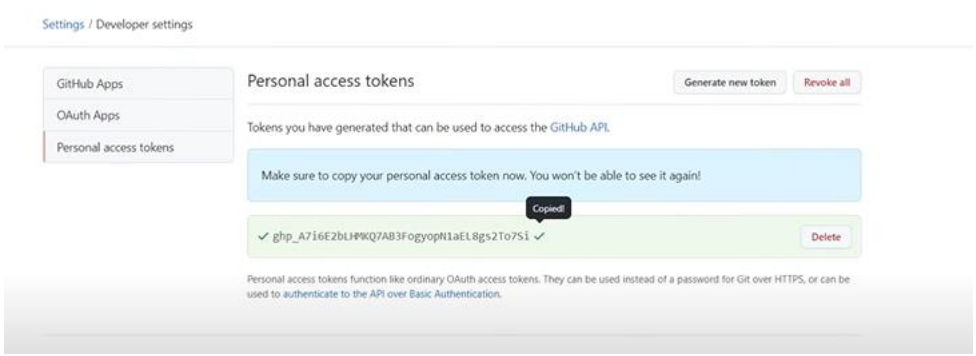
```
our configuration already matches the changes detected above. If you'd like
nd apply a refresh-only plan:
 terraform apply -refresh-only
S C:\terraform> terraform apply -refresh-only
ws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]
```

In state file we are having sg group: devOps_ sg

In backup file we are having sg group : default

Create the GIT repo using terraform:





github_repository

This resource allows you to create and manage repositories within your GitHub organization or personal account.

Example Usage

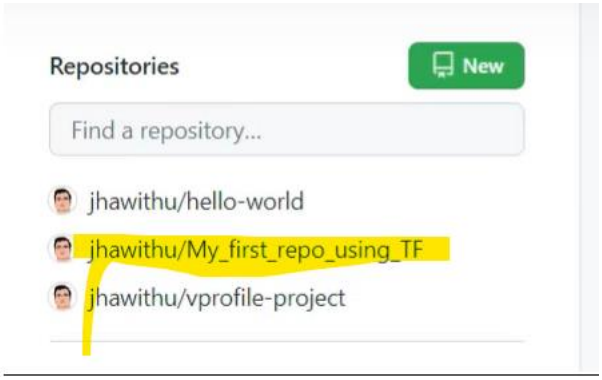
```
resource "github_repository" "example" {
  name        = "example"
  description = "My awesome codebase"

  visibility = "public"

  template {
    owner      = "github"
    repository = "terraform-module-template"
  }
}
```

```
my-first-ec2.tf    github.tf    .terraform.lock.hcl

terraform {
  required_providers {
    github = {
      source  = "integrations/github"
      version = "~> 4.0"
    }
  }
}

# Configure the GitHub Provider
provider "github" {
  token = "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"
}

resource "github_repository" "example" {
  name        = "My_first_repo_using_TF"
  visibility = "private"
}
```

```
PS C:\terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding integrations/github versions matching "~> 4.0"...
- Using previously-installed hashicorp/aws v3.68.0
```

```
PS C:\terraform> terraform plan
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]

Terraform used the selected providers to generate the following execution plan.
following symbols:
  + create

Terraform will perform the following actions:
```

```
PS C:\terraform> terraform apply
aws_instance.my-fisrt-ec2: Refreshing state... [id=i-03d3750506acc80a4]
```



Now tf state file consist of info of both ec2+GIT repo

Terraform target:

If I want to destroy on GITHUB repo provider not ec2 provider in that case we will use terraform target:

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
PS C:\terraform> terraform destroy -target github_repository.example
```

Terraform destroy -target resourcename. Localname

```
github_repository.example: Refreshing state... [id=My_first_repo_using_TF]

Terraform used the selected providers to generate the following execution plan.
following symbols:
  - destroy

Terraform will perform the following actions:

  # github_repository.example will be destroyed
  - resource "github_repository" "example" {
```

After removing git hub provider state file consist of information about only for ec2 provider and backup file consist of both ec2 provider + git hub provider info

3 - Terraform Resource Attribute and Output Values

Crate s3 and Elastic IP and we will fetch it and display in the output.

Single EIP associated with an instance

```
resource "aws_eip" "lb" {
  instance = aws_instance.web.id
  vpc      = true
}
```

Private Bucket w/ Tags

```
resource "aws_s3_bucket" "b" {
  bucket = "my-tf-test-bucket"
  acl    = "private"

  tags = {
    Name        = "My bucket"
    Environment = "Dev"
  }
}
```

Declaring an Output Value

Each output value exported by a module must be declared using an `output` block:

```
output "instance_ip_addr" {
  value = aws_instance.server.private_ip
}
```

Attributes Reference

In addition to all arguments above, the following attributes are exported:

- `allocation_id` - ID that AWS assigns to represent the allocation of the Elastic IP address for use with instances in a VPC.
- `association_id` - ID representing the association of the address with an instance in a VPC.
- `carrier_ip` - Carrier IP address.
- `customer_owned_ip` - Customer owned IP.
- `domain` - Indicates if this EIP is for use in VPC (`vpc`) or EC2 Classic (`standard`).
- `id` - Contains the EIP allocation ID.
- `private_dns` - The Private DNS associated with the Elastic IP address (if in VPC).
- `private_ip` - Contains the private IP address (if in VPC).
- `public_dns` - Public DNS associated with the Elastic IP address.
- `public_ip` - Contains the public IP address.
- `tags_all` - A map of tags assigned to the resource, including those inherited from the provider `default_tags` configuration block.

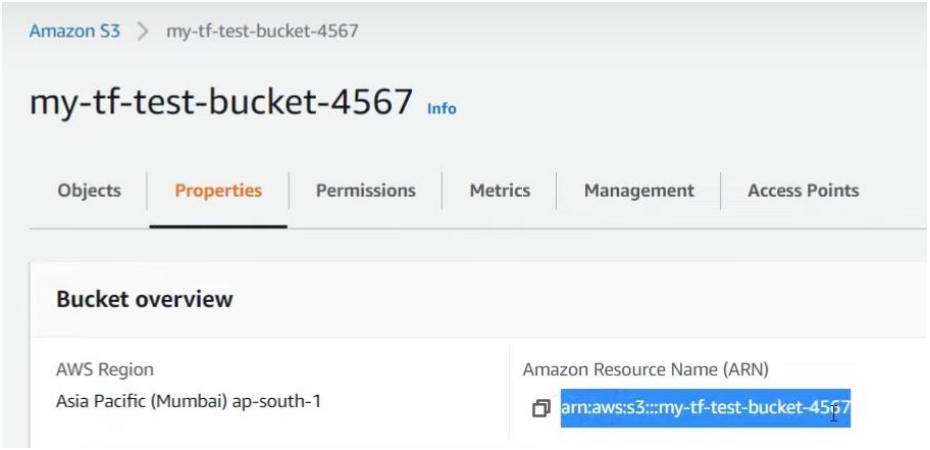
```
provider "aws" {
  region = "ap-south-1"
  access_key = "AKIAX6W4IEODYTWVDQ55"
  secret_key = "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"
}

resource "aws_eip" "lb" {
  vpc      = true
}

output "eip_output" {
  value = aws_eip.lb.public_ip
}

resource "aws_s3_bucket" "b" {
  bucket = "my-tf-test-bucket-4567"
}

output "s3_arn" {
  value = aws_s3_bucket.b.arn
}
```

Value = resource name. local name. attribute reference name

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

Changes to Outputs:
  + eip_output = (known after apply)
  + s3_arn     = (known after apply)
```

```
PS C:\Terraform> terraform apply
```

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

eip_output = "3.111.106.127"
s3_arn = "arn:aws:s3:::my-tf-test-bucket-4567"
```



If we have not given attribute reference name ,it will fetch all the values

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

eip_output = {
  "address" = tostring(null)
  "allocation_id" = "eipalloc-0e3965ce6413b8979"
  "associate_with_private_ip" = tostring(null)
  "association_id" = ""
  "carrier_ip" = ""
  "customer_owned_ip" = ""
  "customer_owned_ipv4_pool" = ""
  "domain" = "vpc"
  "id" = "eipalloc-0e3965ce6413b8979"
  "instance" = ""
  "network_border_group" = "ap-south-1"
  "network_interface" = ""
  "private_dns" = tostring(null)
  "private_ip" = ""
  "public_dns" = "ec2-3-111-104-11.ap-south-1.compute.amazonaws.com"
  "public_ip" = "3.111.104.11"
  "public_ipv4_pool" = "amazon"
  "tags" = tomap(null) /* of string */
  "tags_all" = tomap({})
  "timeouts" = null /* object */
  "vpc" = true
}
```

```
s3_ar = {
  "acceleration_status" = ""
  "acl" = "private"
  "arn" = "arn:aws:s3:::my-tf-test-bucket-4567"
  "bucket" = "my-tf-test-bucket-4567"
  "bucket_domain_name" = "my-tf-test-bucket-4567.s3.amazonaws.com"
  "bucket_prefix" = tostring(null)
  "bucket_regional_domain_name" = "my-tf-test-bucket-4567.s3.ap-south-1.amazonaws.com"
  "cors_rule" = tolist([])
  "force_destroy" = false
  "grant" = toset([])
  "hosted_zone_id" = "Z11RGJOFQNVJUP"
  "id" = "my-tf-test-bucket-4567"
  "lifecycle_rule" = tolist([])
  "logging" = toset([])
  "object_lock_configuration" = tolist([])
  "policy" = tostring(null)
  "region" = "ap-south-1"
  "replication_configuration" = tolist([])
  "request_payer" = "BucketOwner"
  "server_side_encryption_configuration" = tolist([])
  "tags" = tomap(null) /* of string */
  "tags_all" = tomap({})
  "versioning" = tolist([
    {
      "enabled" = false
      "mfa_delete" = false
    },
  ])
  "website" = tolist([])
  "website_domain" = tostring(null)
  "website_endpoint" = tostring(null)
}
```

4- Terraform Provider Version handling.

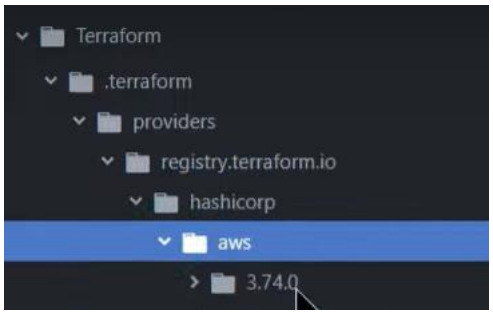
```
PS C:\Terraform> terraform version
Terraform v1.1.4
on windows_386
```

Provider.tf .terraform.lock.hcl

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.74.0"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

→If we will have not mentioned the version it will fetch the latest version.
→If we give different version in provider.tf file (configuration file) it will throw an error because it will check for lock file as well for the -upgrade without removing the lock file.
→If any version in mentioned, First will check the tf file if not will check the lock file

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
- Installed hashicorp/aws v3.74.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```



If want to downgrade the version we can do it but we will get an error in lock file

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file

  Error: Failed to query available provider packages

  Could not retrieve the list of available versions for provider hashicorp/aws: locked provider registry.terraform.io/
  not match configured version constraint 3.73.0; must use terraform init -upgrade to allow selection of new versions
```

Solution1→You can delete the lock file

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.73.0"...
- Installing hashicorp/aws v3.73.0...
- Installed hashicorp/aws v3.73.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
```

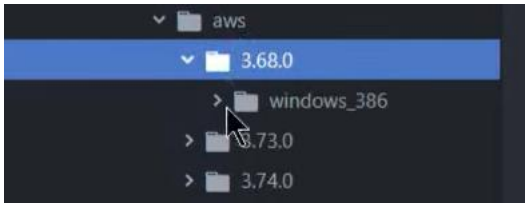
Solution2→Instead deleting lock file mention below mention command

>terraform init upgrade→It will unlock the lock file and mark the required information

Provider.tf

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "3.68.0"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

```
PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.68.0"...
- Installing hashicorp/aws v3.68.0...
- Installed hashicorp/aws v3.68.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!
```



Solution 3: If we remove particular version line from provider.tf

Terraform will check from provider.tf and than check it in .terraform.lock.hcl

```
# This file is maintained automatically by "terrafor
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version     = "3.68.0"
  constraints = "3.68.0"
  hashes = [
    "h1:TiCcru286vmz5wB9zPx/B4GGD9zhhrfUZ2pmLJKqIig=
    "zh:05a43a7dbd409451c08a958610234619d7e0d102e601
    "zh:0d195fa738a348e511550de39caec3f10cfb9afe8d69
    "zh:3d88a19b2a810559bc6953fe92b7a7c6e3251c550186
```

It will update the version mentioned in constraints.

There are multiple ways for specifying the version of a provider

Version Argument	Description
>=3.0	Greater than equal to the version
<=3.0	Less than equal to the version
~>2.0	Any version in 2.X range
>=3.10,<=3.50	Any version between 3.10 and 3.50

Provider.tf .terraform.lock.hcl

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~>2.0"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

It will download maximum into 2 series i.e one is 2 version and another one is 3version

```
PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 2.0"...
- Installing hashicorp/aws v2.70.1...
- Installed hashicorp/aws v2.70.1 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!
```

```
PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching ">= 3.10.0, <= 3.50.0"...
- Installing hashicorp/aws v3.50.0...
- Installed hashicorp/aws v3.50.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!
```

If we give not mentioned version it will throw an error.

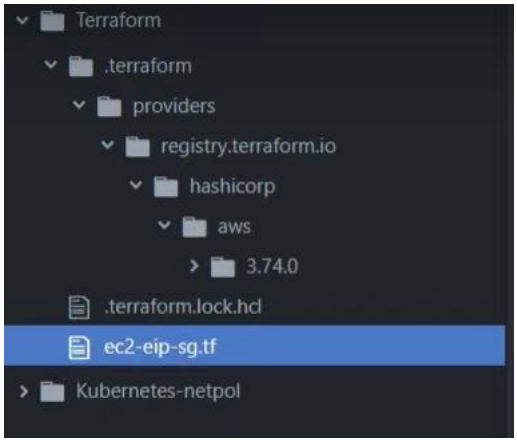
```
PS C:\Terraform> terraform init -upgrade

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "2.70.2"...

Error: Failed to query available provider packages

Could not retrieve the list of available versions for provider hashicorp/aws: no available releases match the given constraints 2.70.2
```

5 - Terraform Format, Validate, EIP

Terraform Format:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.74.0"
    }
  }
}

provider "aws" {
  region     = "ap-south-1"
  access_key = "AKIAX6W4IEODXH3GQO4I"
  secret_key = "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"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0f1fb91a596abf28d"
  instance_type = "t2.micro"
}

resource "aws_eip" "lb" {
```

```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
```




```
PS C:\Terraform> terraform fmt
ec2-eip-sg.tf
PS C:\Terraform>
```

All the spaces, syntax get aligned in terraform file

Terraform Validate:

If we add any attributes which are not exit. Terraform validate and throw an error like **"Unsupported arguments"**

```
provider "aws" {
  region     = "ap-south-1"
  access_key = "AKIAX6W4IEODXH3GQO4I"
  secret_key = "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"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0f1fb91a596abf28d"
  instance_type = "t2.micro"
  mymachine="Dev"
}
```

```
PS C:\Terraform> terraform validate

Error: Unsupported argument

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
  19:   mymachine="Dev"

An argument named "mymachine" is not expected here.
```

```
PS C:\Terraform> terraform plan

Error: Unsupported argument

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
  19:   mymachine="Dev"

An argument named "mymachine" is not expected here.
```

```
PS C:\Terraform> terraform validate

Error: Unsupported argument

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
  19:   mymachine="Dev"

An argument named "mymachine" is not expected here.

PS C:\Terraform> terraform plan

Error: Unsupported argument

  on ec2-eip-sg.tf line 19, in resource "aws_instance" "myec2":
  19:   mymachine="Dev"

An argument named "mymachine" is not expected here.
```

TERRAFORM APPLY will always ask for the confirmation instead we can use auto approve

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

 you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: no
```

```
PS C:\Terraform> terraform apply --auto-approve
```

```
Plan: 2 to add, 0 to change, 0 to destroy.
aws_eip.lb: Creating...
aws_instance.myec2: Creating...
aws_eip.lb: Creation complete after 0s [id=eipalloc-0844c78c64e6afbdd]
```

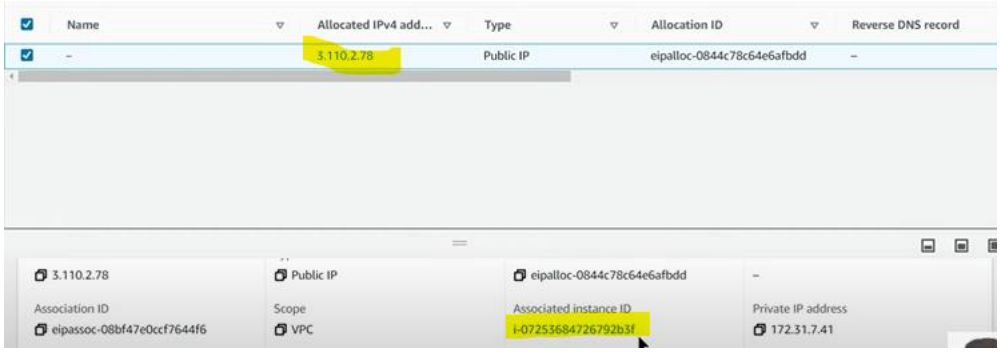
Terraform EIP: Association (Linking) EIP with EC2 instance.

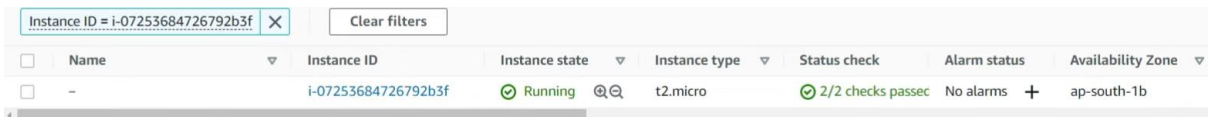
```
resource "aws_eip_association" "eip_assoc" {
  instance_id   = aws_instance.web.id
  allocation_id = aws_eip.example.id
}

resource "aws_instance" "web" {
  ami               = "ami-21f78e11"
  availability_zone = "us-west-2a"
  instance_type     = "t2.micro"

  tags = {
    Name = "HelloWorld"
  }
}

resource "aws_eip" "example" {
  vpc = true
}
```

Instance_id =resource name. local name. attribute name

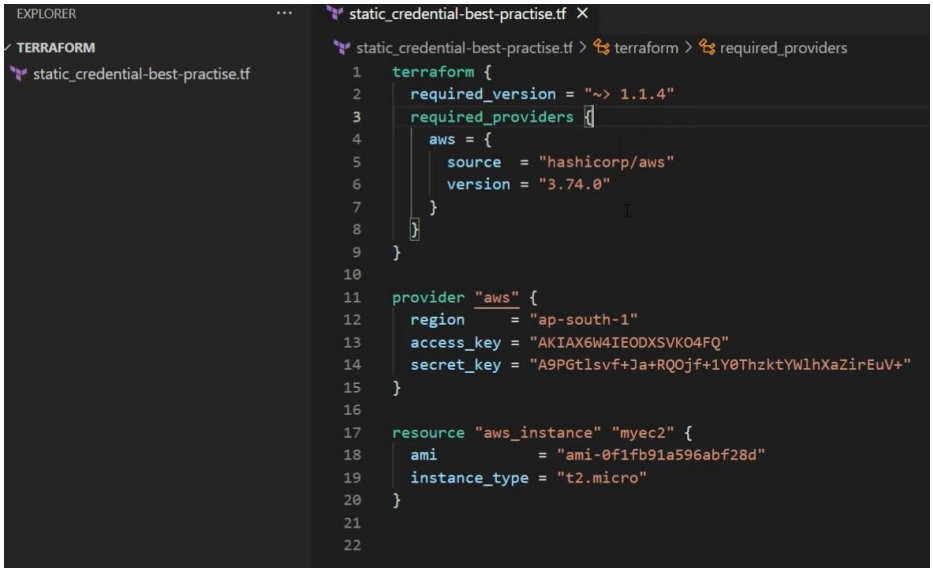
```
Terraform will perform the following actions:

  # aws_eip_association.eip_assoc will be created
  + resource "aws_eip_association" "eip_assoc" {
      + allocation_id        = "eipalloc-0844c78c64e6afbdd"
      + id                   = (known after apply)
      + instance_id          = "i-07253684726792b3f"
      + network_interface_id = (known after apply)
      + private_ip_address   = (known after apply)
      + public_ip            = (known after apply)
    }
```

```
PS C:\Terraform> terraform apply --auto-approve
```

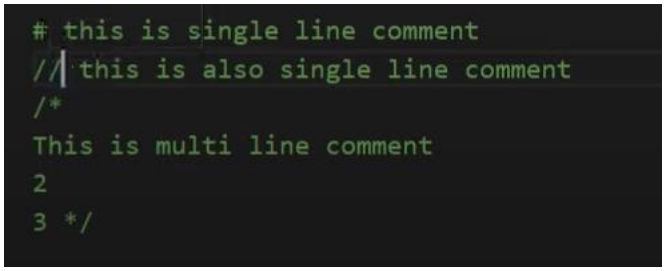




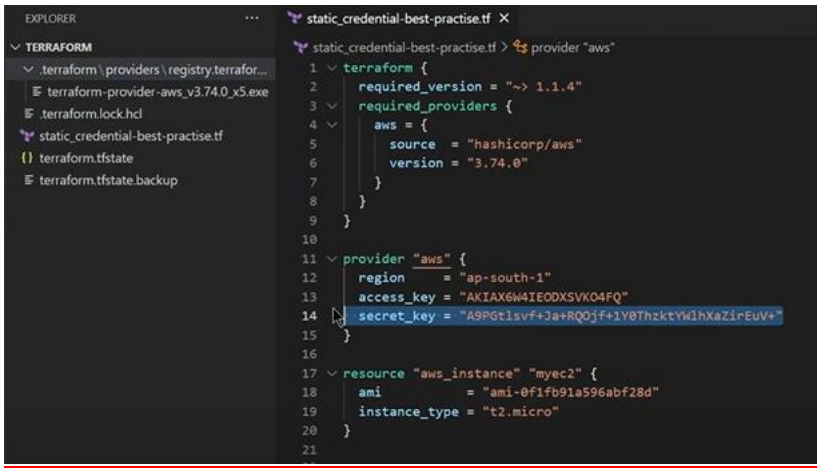
6 - Terraform Shared Credential, AWS CLI, Comments

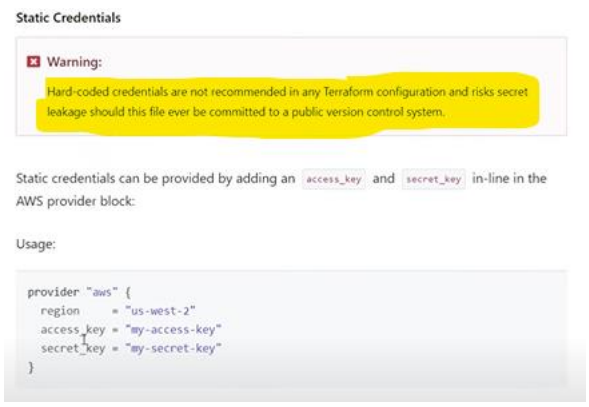


Comments :



```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
- Installed hashicorp/aws v3.74.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Terraform> terraform fmt
PS C:\Terraform> terraform validate
Success! The configuration is valid.
```

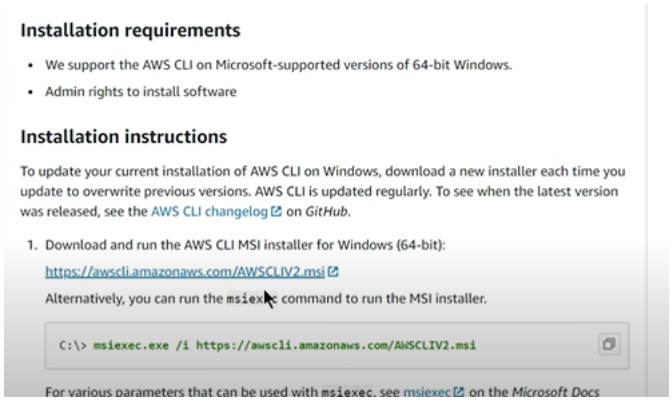




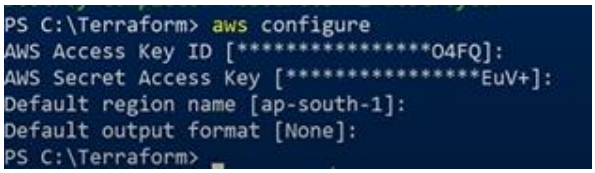
Instead of using Static Credentials we need to use Environment Variable, Shared credentials files.

AWS CLI (Command Line Interface):

Here we are store our credentials we are not using in terraform file.



To install AWS CLI



Location where the credentials and config files got installed in windows.

C:/Users/Username/.aws

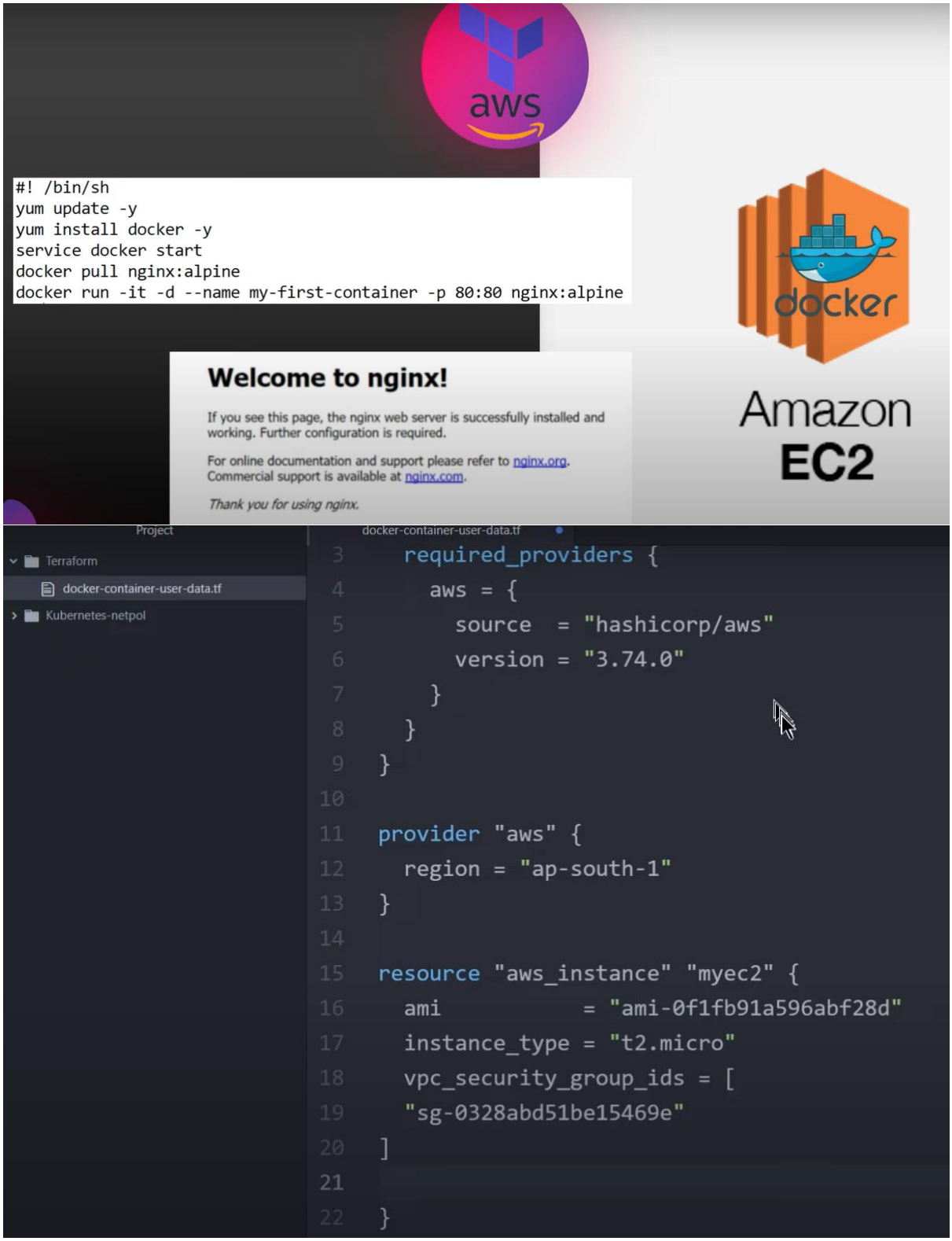
```
PS C:\Users\Alok\.aws> ls


    Directory: C:\Users\Alok\.aws


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        12-02-2022  10:51 AM             32 config
-a----        12-02-2022  10:51 AM            119 credentials


PS C:\Users\Alok\.aws> cat config
[default]
region = ap-south-1
PS C:\Users\Alok\.aws> cat .\credentials
[default]
aws_access_key_id = AKIAX6W4IEODXSVKO4FQ
aws_secret_access_key = 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
```

static_credential-best-practise.tf > ...

```
terraform {
  required_version = "~> 1.1.4"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.74.0"
    }
  }
}

provider "aws" {
  region     = "ap-south-1"
}

resource "aws_instance" "myec2" {
  ami           = "ami-0f1fb91a596abf28d"
  instance_type = "t2.micro"
}
```

7 Running Nginx from Docker Container using Terraform:

Create EC2 instance, create docker env on ec2 and pull docker image nginx.



Add user data

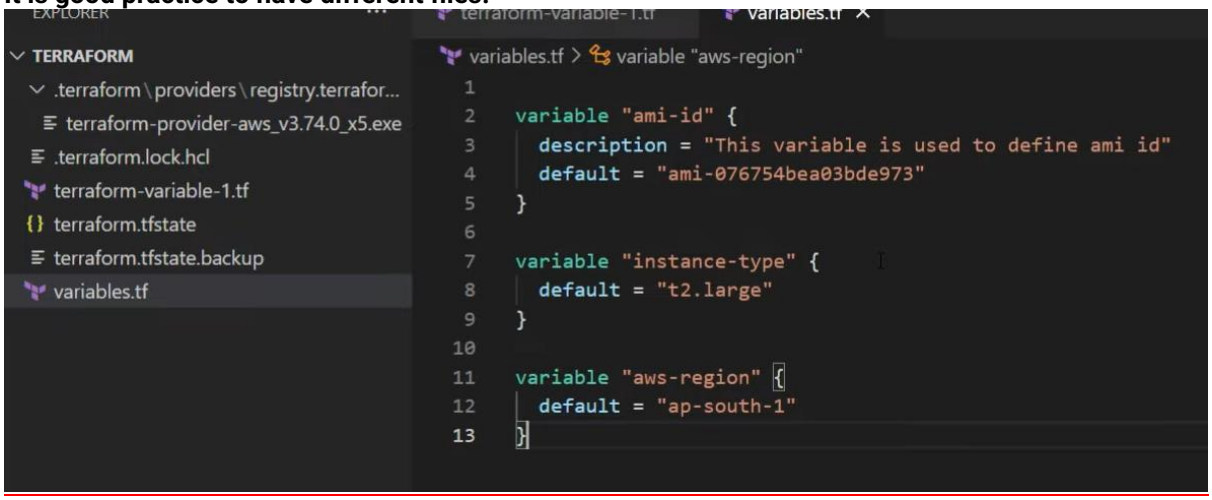
```
user_data = <<EOF
#! /bin/sh
yum update -y
yum install docker -y
service docker start
docker pull nginx:alpine
docker run -it -d --name my-first-container -p 80:80 nginx:alpine
EOF
}
```

```
PS C:\Terraform> aws configure
AWS Access Key ID [****************NOAI]:
AWS Secret Access Key [****************GI/3]:
Default region name [ap-south-1]:
Default output format [None]:
PS C:\Terraform> ls


    Directory: C:\Terraform


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        25-02-2022  07:52 PM            318 docker-container-user-data.tf


PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "3.74.0"...
- Installing hashicorp/aws v3.74.0...
```

```
PS C:\Terraform> terraform validate
Success! The configuration is valid.

PS C:\Terraform> terraform fmt
docker-container-user-data.tf
PS C:\Terraform> terraform plan
```

8 - Terraform Input Variable Part – 1

It is good practice to have different files.

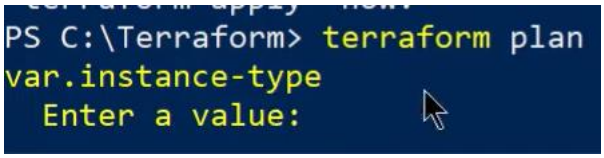


erraform-variable-1.tf > variable "aws-region" > default

```
terraform {
  required_version = "~> 1.1.4"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.74.0"
    }
  }
}

variable "ami-id" {
  description = "This variable is used to define ami id"
  default = "ami-076754bea03bde973"
}

variable "instance-type" {
  default = "t2.micro"
}

variable "aws-region" {
  default = "ap-south-1"
}
provider "aws" {
  region = "ap-south-1"
}

resource "aws_instance" "myec2-1" {
  ami           = "ami-076754bea03bde973"
  instance_type = "t2.micro"
}
```

```
provider "aws" {
  region = "ap-south-1"
}

resource "aws_instance" "myec2-1" {
  ami           = var.ami-id
  instance_type = var.instance-type
}

resource "aws_instance" "myec2-2" {
  ami           = var.ami-id
  instance_type = var.instance-type
}
```

Senario1 →If we define the variable but we have not given any default value then terraform plan will not given any error but it will ask for the entry



Senario2 →You do not have a access to the file to edit but all of sudden you have got requirement to make change default to t2.lagre than time you can override it.

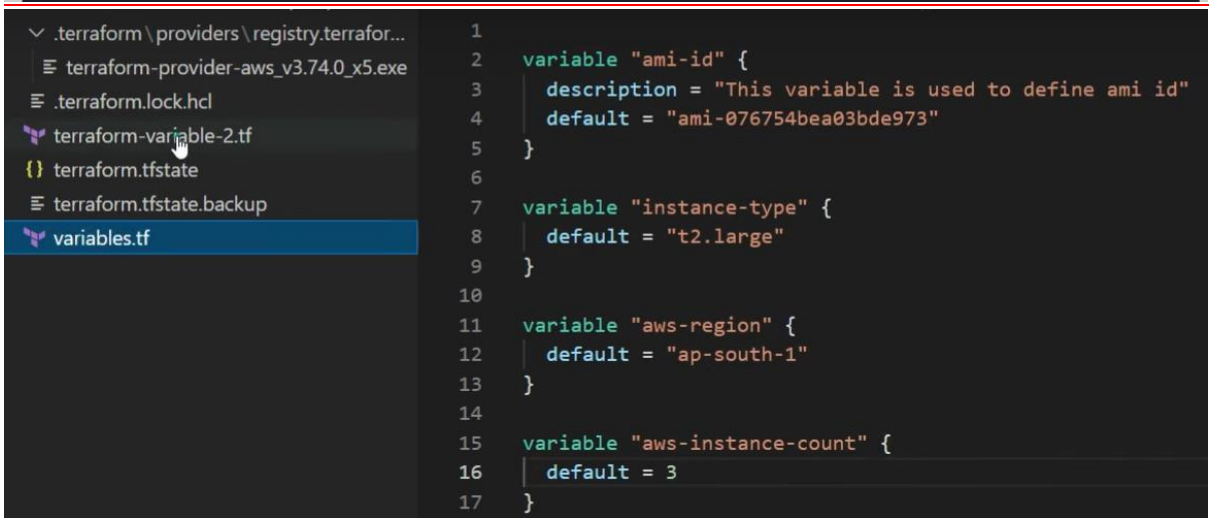
Even if default value is given already if you want to edit /modify any of the files you can use CLI and you can do **override**

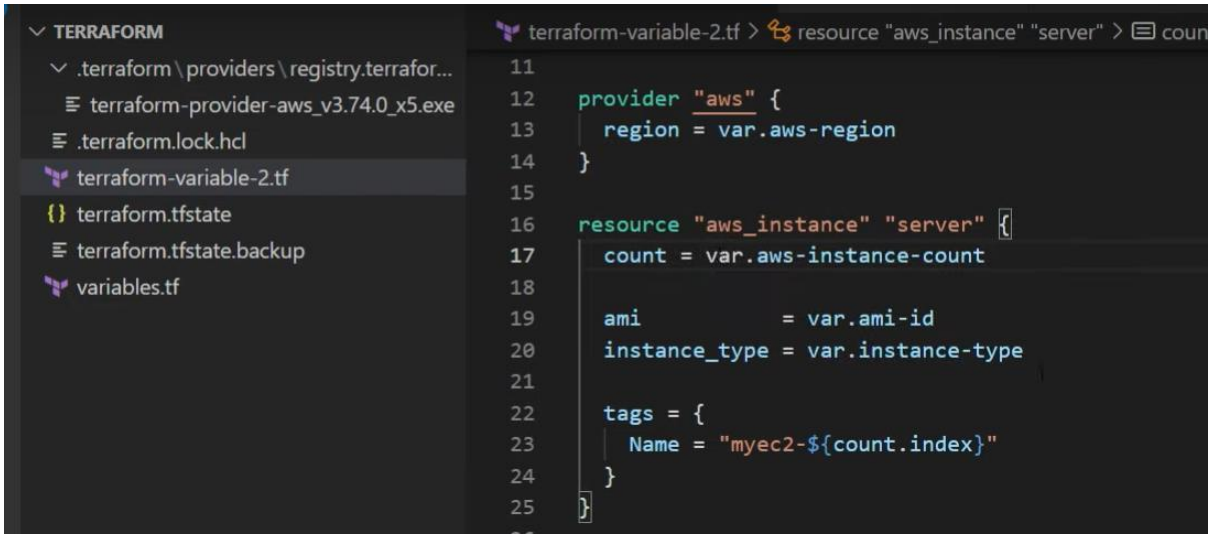
9 -Variable, Count and Generating and Applying TF Plan Part-2

`count` is a meta-argument defined by the Terraform language. It can be used with modules and with every resource type.

The `count` meta-argument accepts a whole number, and creates that many instances of the resource or module. Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
}
```





Override the values.

```
PS C:\Terraform> terraform plan -var instance-type="t2.micro" -var aws-instance-count=1
```

Create customised file:

Scenario → To override multiple variables will use. **Generate a plan files** and put it in the myplan.plan file.

-out myplan.plan

```
Apply cancelled.
PS C:\Terraform> terraform plan -var instance-type="t2.micro" -var aws-instance-count=1 -out myplan.plan
```

```
Saved the plan to: myplan.plan

To perform exactly these actions, run the following command to apply:
    terraform apply "myplan.plan"
```


```
PS C:\Terraform> terraform apply myplan.plan
aws_instance.server[0]: Creating...
```

10 -. Input Var from *.tfvars, *.auto.tfvars & env var Part-3

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.)

The variable precedence starts from the below

1)-var and -var-file

2) *. auto. tfvars/*.auto.tfvars.json

3) terraform. tfvars. json

4)terraform.tfvars

5) Environment variable

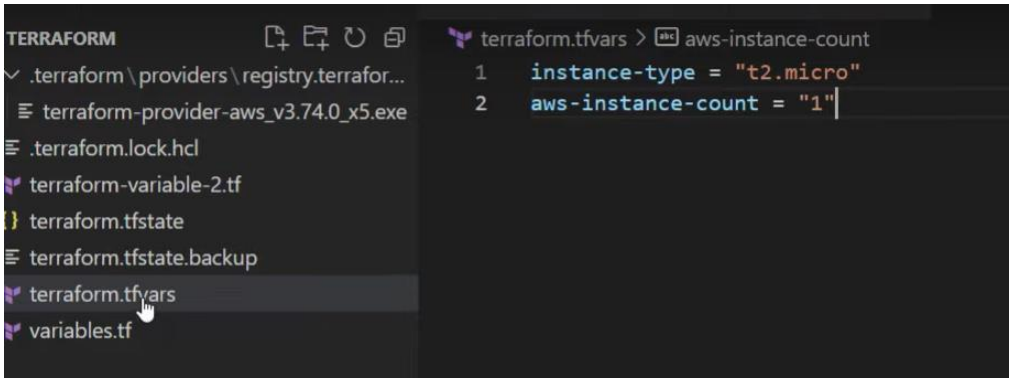
*Untitled - Notepad

File Edit Format View Help

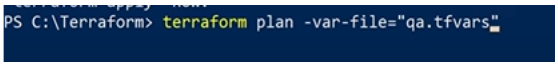
```
Set environment variable
==========================

export TF_VAR_instance-type="t2.medium"
export TF_VAR_aws-instance-count="10"

echo $TF_VAR_instance-type, $TF_VAR_aws-instance-count

unset TF_VAR_instance-type
unset TF_VAR_aws-instance-count
```


```
PS C:\Terraform> terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v3.74.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
PS C:\Terraform> terraform validate
Success! The configuration is valid.

PS C:\Terraform> terraform fmt
terraform-variable-2.tf
variables.tf
PS C:\Terraform>
```

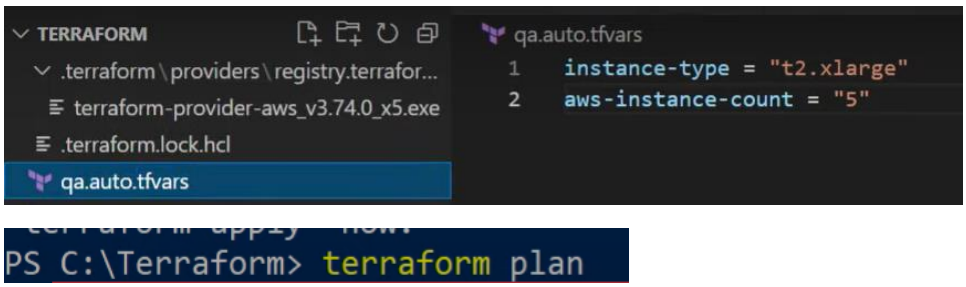
variable.tf file will be override by **terraform. tfvars** since its higher precision



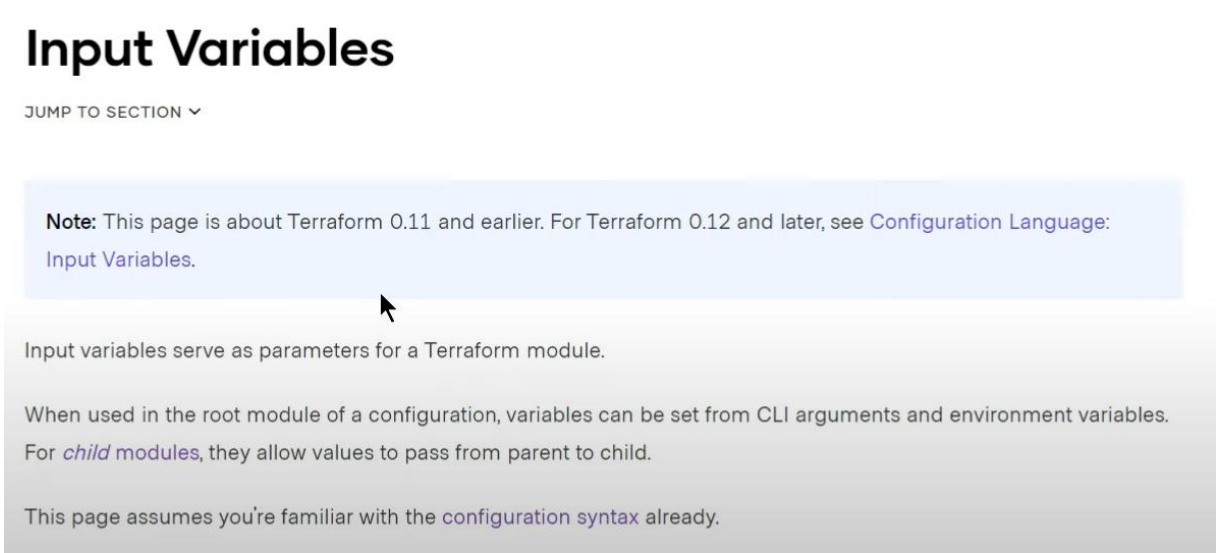
Secnario1→ We have to explicitly mention custom files while terraform plan **terraform plan -var-file= "qa.tfvars" – higher precidence**



Senario2 →You don't want to give every time i.e var-file name during plan for that you want to make it **default file** you need to rename it with the name**: qa.auto.tfvars [filename.auto.tfvars]**



11 - Implement Variable Type as Str, Num, List & Map. Part-4



Input variables can be defined as follows:

```
variable "key" {
  type = "string"
}

variable "images" {
  type = "map"

  default = {
    us-east-1 = "image-1234"
    us-west-2 = "image-4567"
  }
}

variable "zones" {
  type = "list"
  default = ["us-east-1a", "us-east-1b"]
}
```

Strings

String values are simple and represent a basic key to value mapping where the key is the variable name. An example is:

```
variable "key" {
  type    = "string"
  default = "value"
}
```

Copy

A multi-line string value can be provided using heredoc syntax.

```
variable "long_key" {
  type = "string"
  default = <<EOF
This is a long key.
Running over several lines.
EOF
}
```

Copy

Maps

A map value is a lookup table from string keys to string values. This is useful for selecting a value based on some other provided value.

A common use of maps is to create a table of machine images per region, as follows:

```
variable "images" {
  type    = "map"
  default = {
    "us-east-1" = "image-1234"
    "us-west-2" = "image-4567"
  }
}
```

Copy

LIST:

```
variable "ami-id" {
  description = "This variable is used to define ami id"
  default     = "ami-076754bea03bde973"
}

variable "instance-type" {
type = list(string)
default = ["t2.micro","t2.medium","t2.1arge"]
}

variable "aws-region" {
  default = "ap-south-1"
}

variable "aws-instance-count" {
  default = 1
}
```

terraform-variable-4.tf > resource "aws_instance" "ec2" > instance_type

```
terraform {
  required_version = "~> 1.1.4"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.74.0"
    }
  }
}


provider "aws" {
  region = var.aws-region
}

resource "aws_instance" "ec2" {
  count = var.aws-instance-count

  ami           = var.ami-id
  instance_type = var.instance-type[1]

  tags = {
    Name = "myec2-${count.index}"
  }
}
```

If we mention out of the default values will get an error.

```
Alok@Alok-DevOps MINGW64 /c/Terraform
$ terraform plan

Error: Invalid index

  on terraform-variable-4.tf line 20, in resource "aws_instance" "ec2":
  20:   instance_type = var.instance-type[3]
      ├────────────────
      │ var.instance-type is list of string with 3 elements

The given key does not identify an element in this collection value: the
given index is greater than or equal to the length of the collection.
```

MAP:

```
type = map(string)
default = {
  "dev" = "t2.micro"
  "qa" = "t2.medium"
  "uat" = "t2.large"
}

instance_type = var.instance-type["dev"] => t2.micro

*/
```

```
variable "instance-type" {
  type = map(string)
  default = {
    "dev" = "t2.micro"
    "qa" = "t2.medium"
    "uat" = "t2.large"
  }
}
```

```
resource "aws_instance" "ec2" {
  count = var.aws-instance-count

  ami           = var.ami-id
  instance_type = var.instance-type["qa"]

  tags = {
    Name = "myec2-${count.index}"
  }
}
```