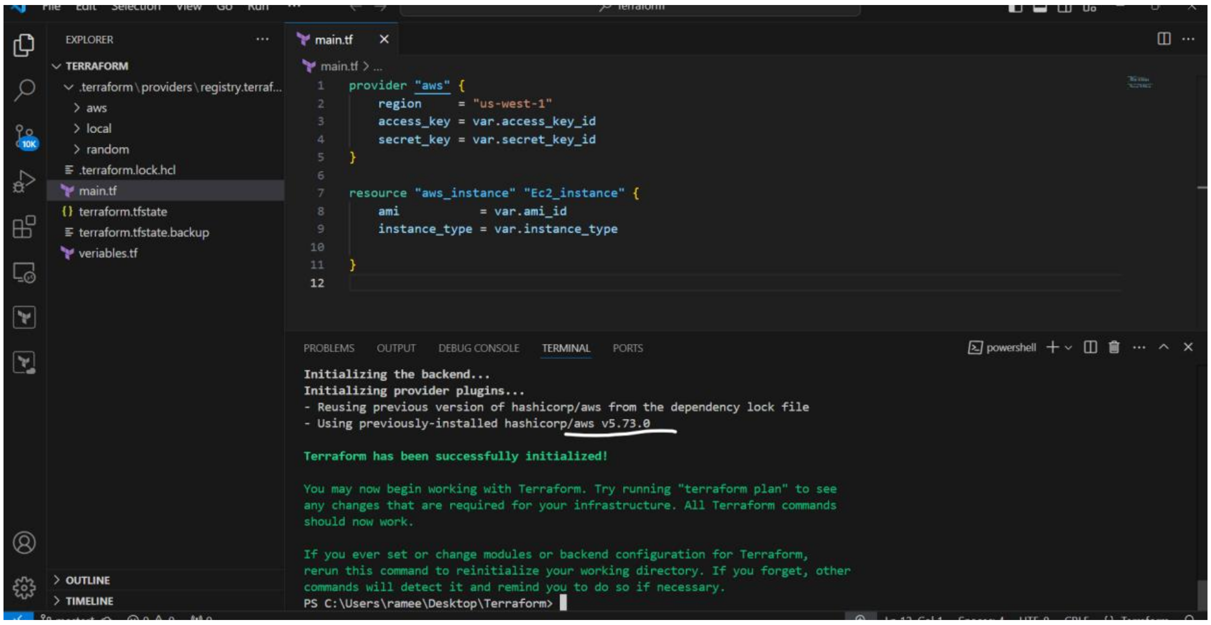
1) Execute the script shown in video.

```
Version Constraints:
====================
```

- By default, Terraform downloads the latest provider version during terraform init.
- To avoid compatibility issues, it's best to specify a provider version in the configuration.
- Locking versions ensures stable deployments and prevents unexpected changes.

  Here see the below image while enter the terraform init ---- it will download the latest version of provider.

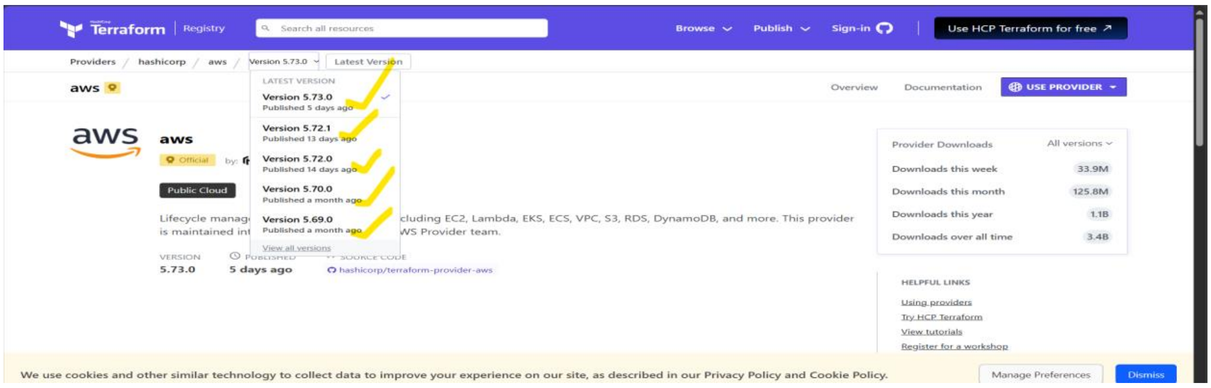


```
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}

resource "aws_instance" "Ec2_instance" {
    ami           = var.ami_id
    instance_type = var.instance_type

}
```

```
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.73.0

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:\Users\ramee\Desktop\Terraform>
```

Now go to the terraform registry there I am selecting the aws provider.

- Here check the versions the versions are changed in frequently.
- So in the middle of the project versions changed that time we face compatibility issue.
- So To avoid compatibility issues, it's best to specify a provider version in the configuration.



Providers / hashicorp / aws / Version 5.73.0 Latest Version

aws

LATEST VERSION
Version 5.73.0 ✓
Published 5 days ago
Version 5.72.1
Published 13 days ago
Version 5.72.0
Published 14 days ago
Version 5.70.0
Published a month ago
Version 5.69.0
Published a month ago
View all versions

Lifecycle manag... cluding EC2, Lambda, EKS, ECS, VPC, S3, RDS, DynamoDB, and more. This provider is maintained int... WS Provider team.

VERSION 5.73.0 PUBLISHED 5 days ago SOURCE CODE hashicorp/terraform-provider-aws

Provider Downloads	All versions
Downloads this week	33.9M
Downloads this month	125.8M
Downloads this year	1.1B
Downloads over all time	3.4B

We use cookies and other similar technology to collect data to improve your experience on our site, as described in our Privacy Policy and Cookie Policy.

Version Specification Guide

Use the below specifications

version = "5.73.0" --> download the exact version

version = "!= 5.72.1" --> will not use the mentioned version

version = "< 5.72.0" --> lesses than the mention version

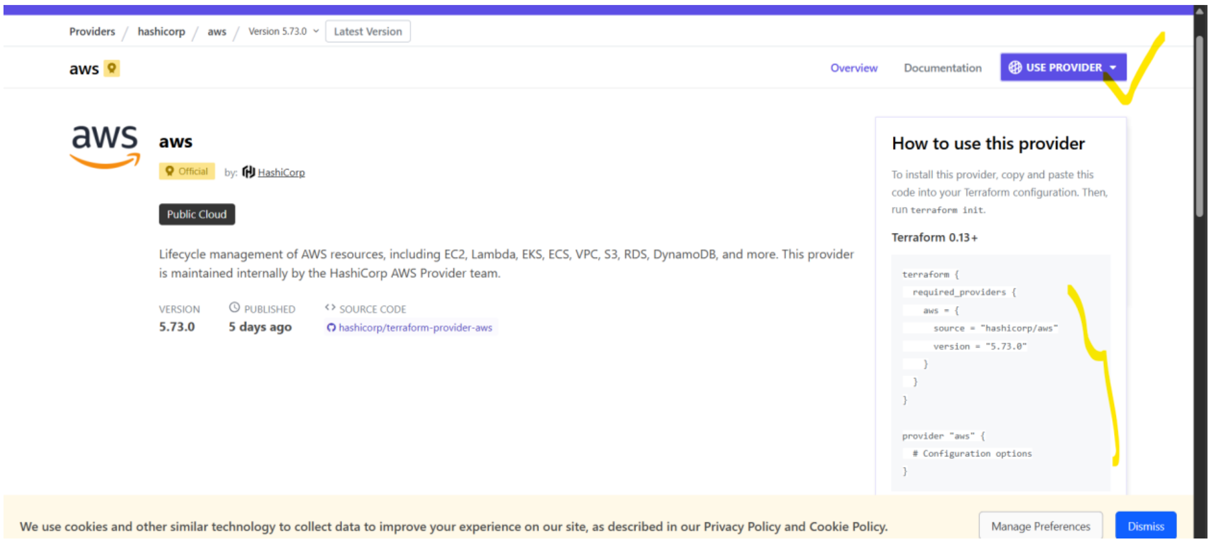
version = "> 5.73.0" --> greater than the given version

version = "~> 5.72.1" --> specific version or higher version.

We have to mention specific version before Provider Black.

Just click on the below Link you will see the below interface and click on the use provider.

hashicorp/aws | Terraform Registry



Lifecycle management of AWS resources, including EC2, Lambda, EKS, ECS, VPC, S3, RDS, DynamoDB, and more. This provider is maintained internally by the HashiCorp AWS Provider team.

VERSION 5.73.0 · PUBLISHED 5 days ago · SOURCE CODE hashicorp/terraform-provider-aws

How to use this provider

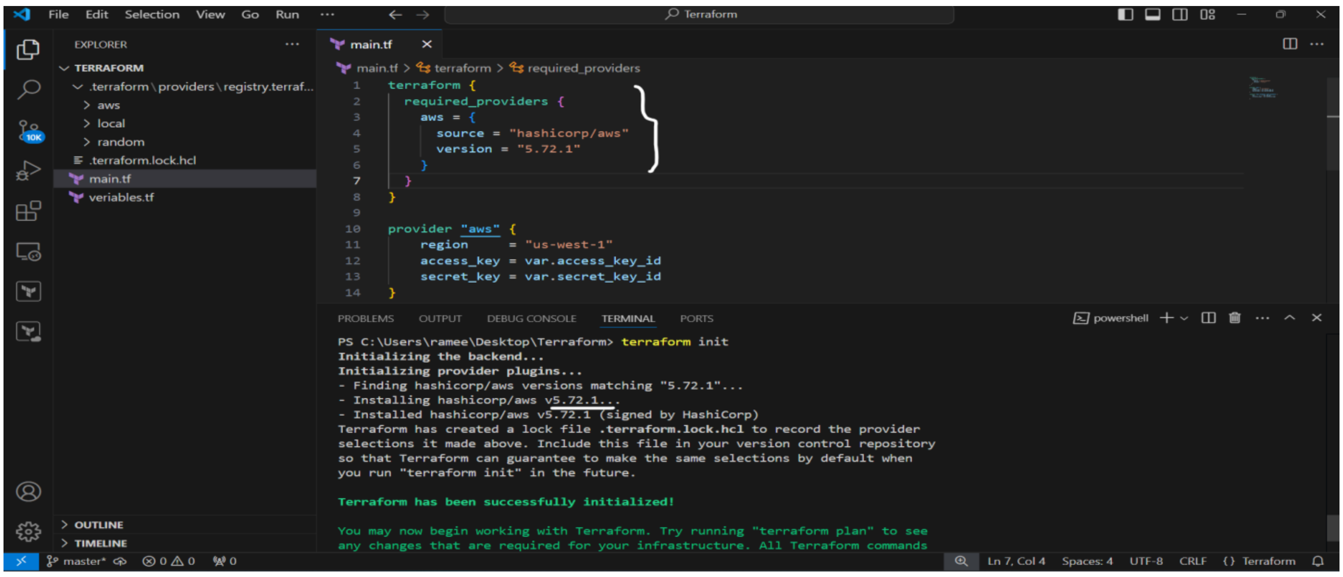
To install this provider, copy and paste this code into your Terraform configuration. Then, run `terraform init`.

Terraform 0.13+

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.73.0"
    }
  }
}

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

We use cookies and other similar technology to collect data to improve your experience on our site, as described in our Privacy Policy and Cookie Policy.

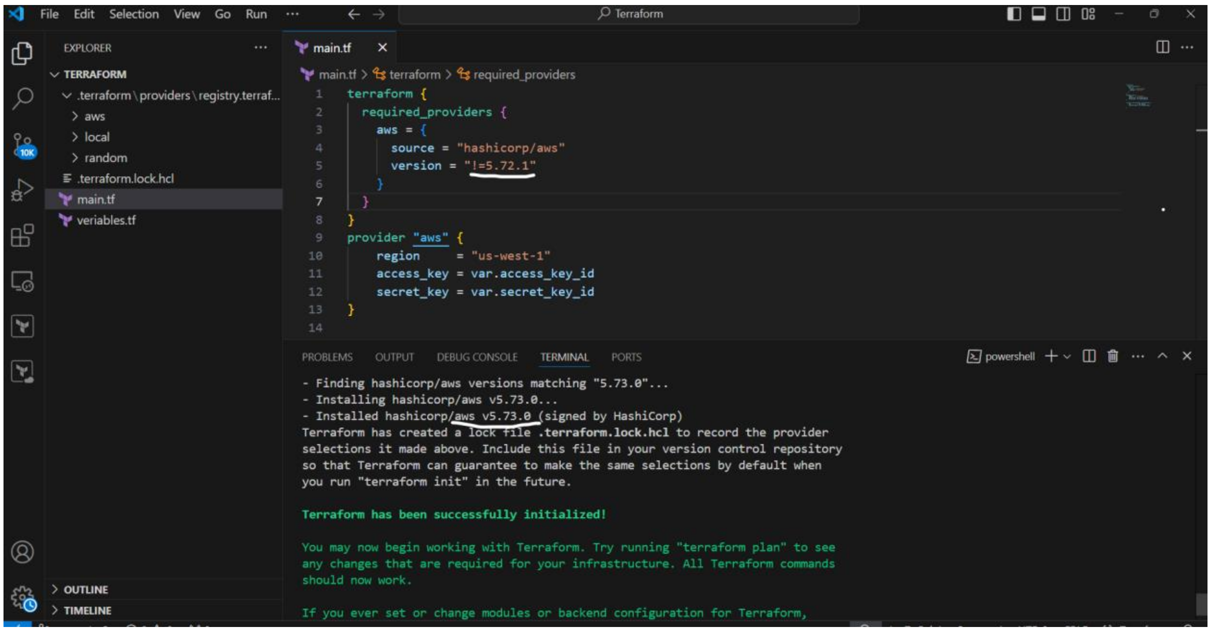
- Here I am mention the provider specific version. – to download the exact version
- Actually before while entering the terraform init it will downloaded the latest version as 5.73.0. but now it will downloaded the 5.72.1 version downloaded.


File Edit Selection View Go Run ...
Terraform
EXPLORER
TERRAFORM
.terraform \ providers \ registry.terraf...
aws
local
random
.terraform.lock.hcl
main.tf
veriables.tf
main.tf
main.tf > terraform > required_providers
OUTLINE
TIMELINE
master*
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Ln 7, Col 4 Spaces: 4 UTF-8 CRLF Terraform

```
terraform {
    required_providers {
        aws = {
            source = "hashicorp/aws"
            version = "5.72.1"
        }
    }
}

provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
PS C:\Users\ramee\Desktop\Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "5.72.1"...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (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
```

To avoid the specific Version of provider use the version = "!=specific version "



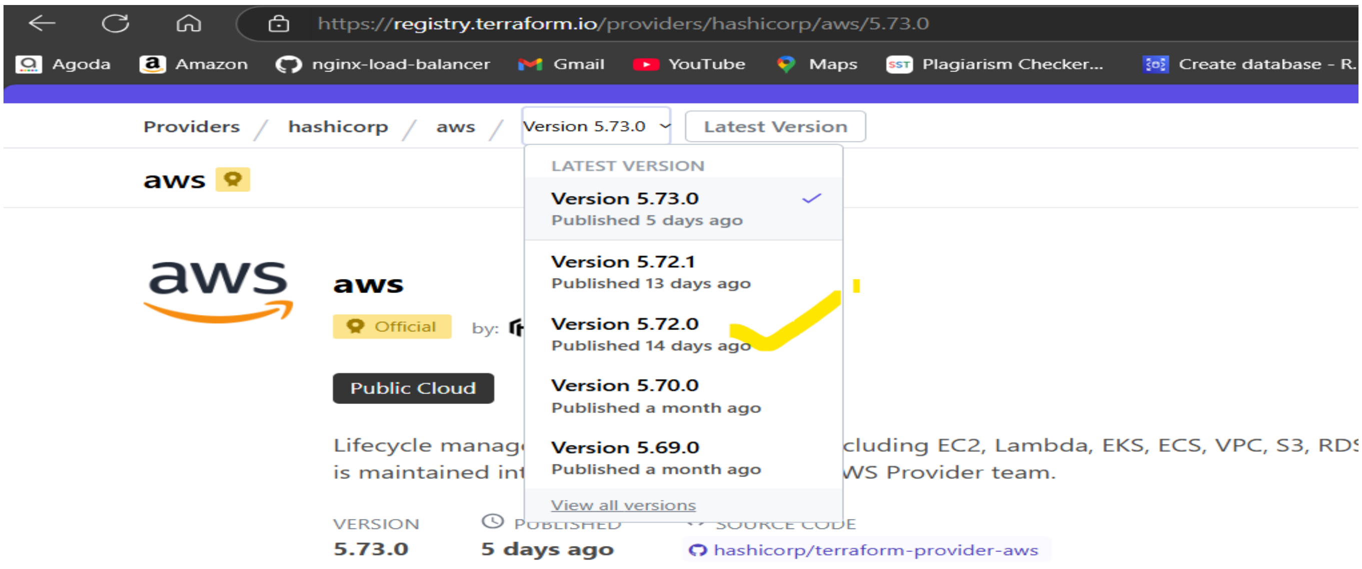
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "!=5.72.1"
    }
  }
}
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
- Finding hashicorp/aws versions matching "5.73.0"...
- Installing hashicorp/aws v5.73.0...
- Installed hashicorp/aws v5.73.0 (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!

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,
```

So we avoid the 5.72.1 version so it will download the latest version from terraform registry.

less than the mention version

< "specific version" below will be download.



https://registry.terraform.io/providers/hashicorp/aws/5.73.0

Providers / hashicorp / aws / Version 5.73.0 Latest Version

LATEST VERSION

Version 5.73.0 ✓
Published 5 days ago

Version 5.72.1
Published 13 days ago

Version 5.72.0
Published 14 days ago

Version 5.70.0
Published a month ago

Version 5.69.0
Published a month ago

View all versions

aws

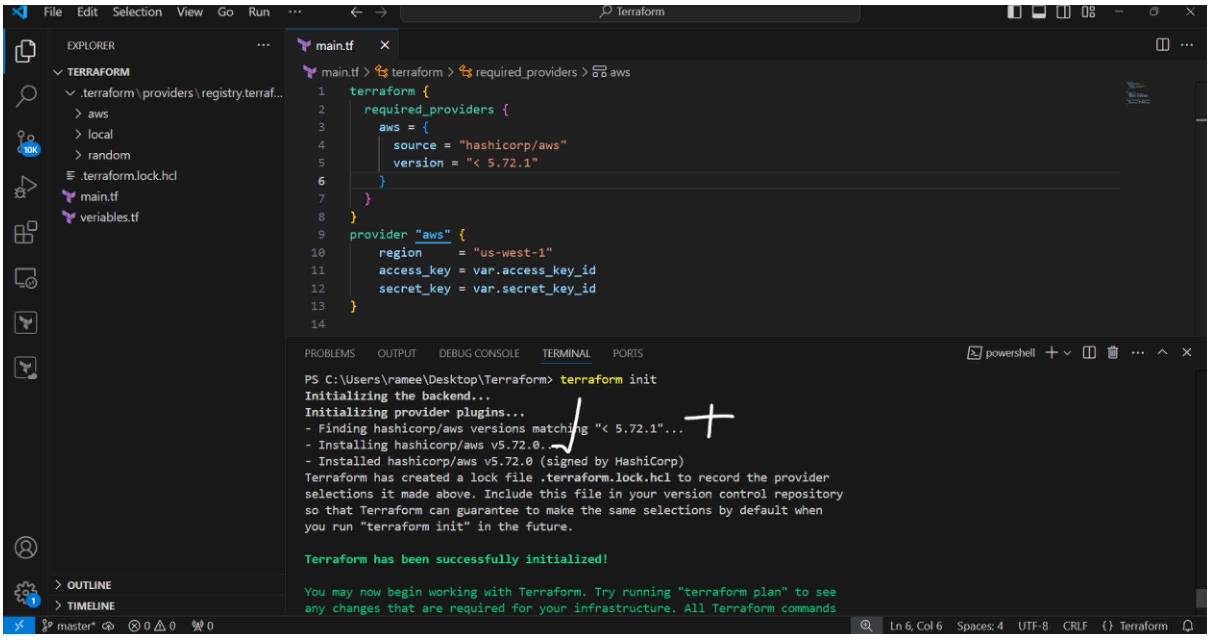
Official by:

Public Cloud

Lifecycle manag cluding EC2, Lambda, EKS, ECS, VPC, S3, RD
is maintained int WS Provider team.

VERSION 5.73.0 PUBLISHED 5 days ago SOURCE CODE hashicorp/terraform-provider-aws

Here we given < 5.72.1 so it's downloaded the 5.72.0 version.

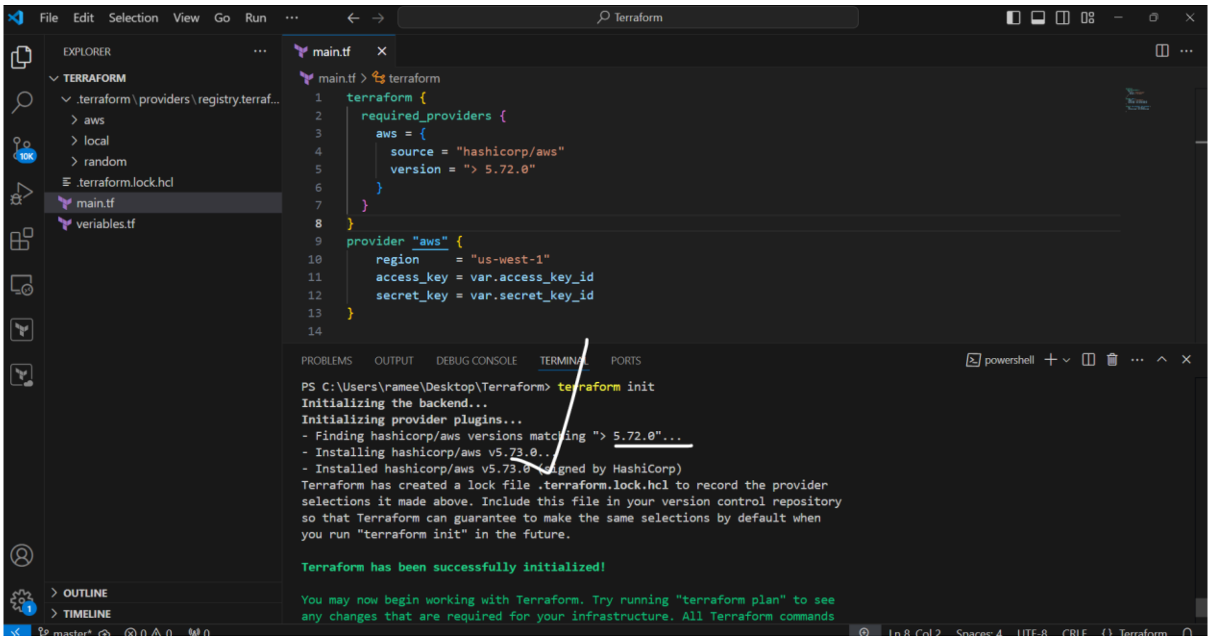


```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "< 5.72.1"
    }
  }
}
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
PS C:\Users\ramee\Desktop\Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "< 5.72.1"...
- Installing hashicorp/aws v5.72.0...
- Installed hashicorp/aws v5.72.0 (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
```

version = "> 5.72.0" --> greater than the given version

so it will download the greater than version.



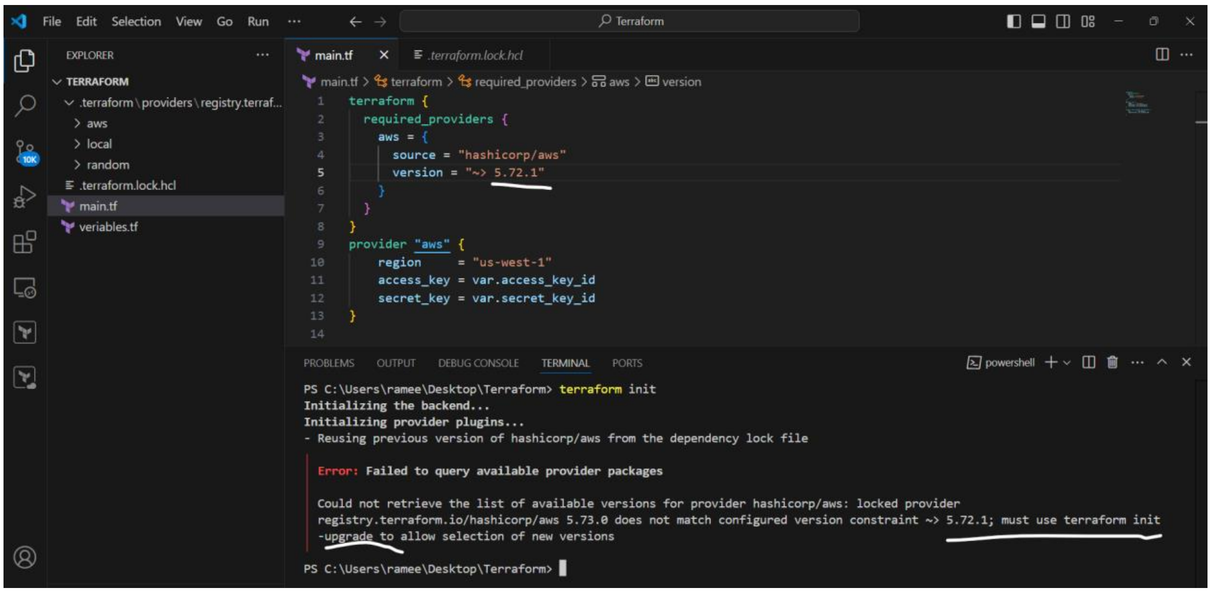
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "> 5.72.0"
    }
  }
}
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
PS C:\Users\ramee\Desktop\Terraform> terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "> 5.72.0"...
- Installing hashicorp/aws v5.73.0...
- Installed hashicorp/aws v5.73.0 (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!

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

version = "~> 5.72.1" --> specific version.

Here I am getting error below bcz before it downloaded the 5.72.1 version.

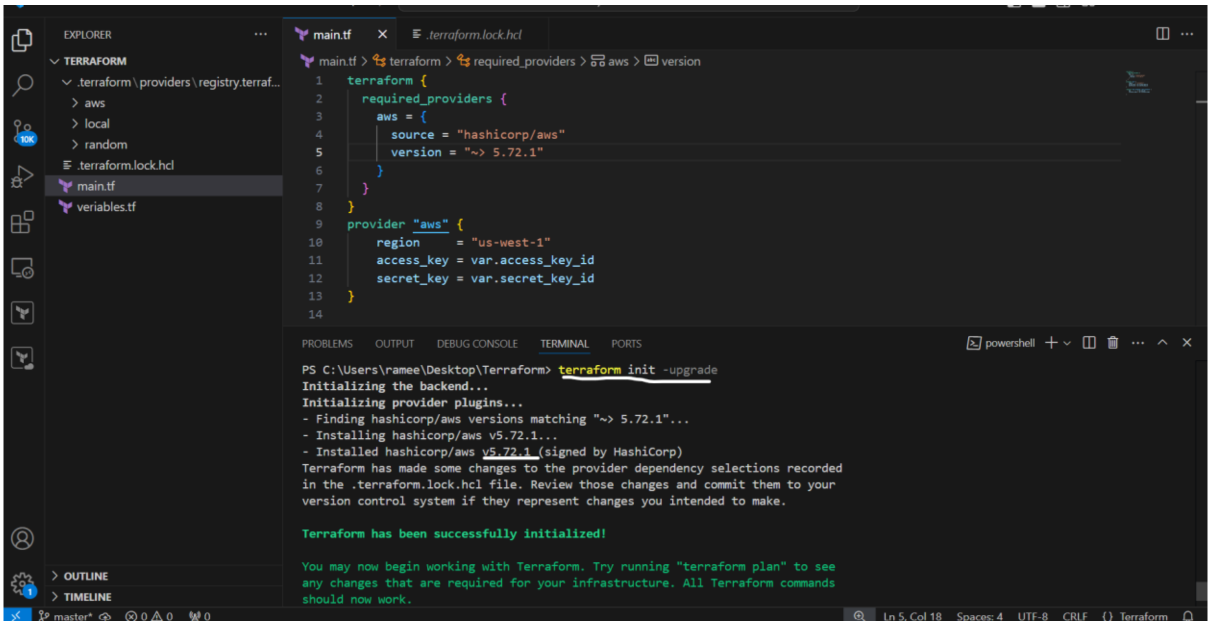
To change the version we need give the –upgrade command.



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 5.72.1"
    }
  }
}
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
PS C:\Users\ramee\Desktop\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/hashicorp/aws 5.73.0 does not match configured version constraint ~> 5.72.1; must use terraform init
 -upgrade to allow selection of new versions

PS C:\Users\ramee\Desktop\Terraform>
```

This command will downloaded the specific version what we given.



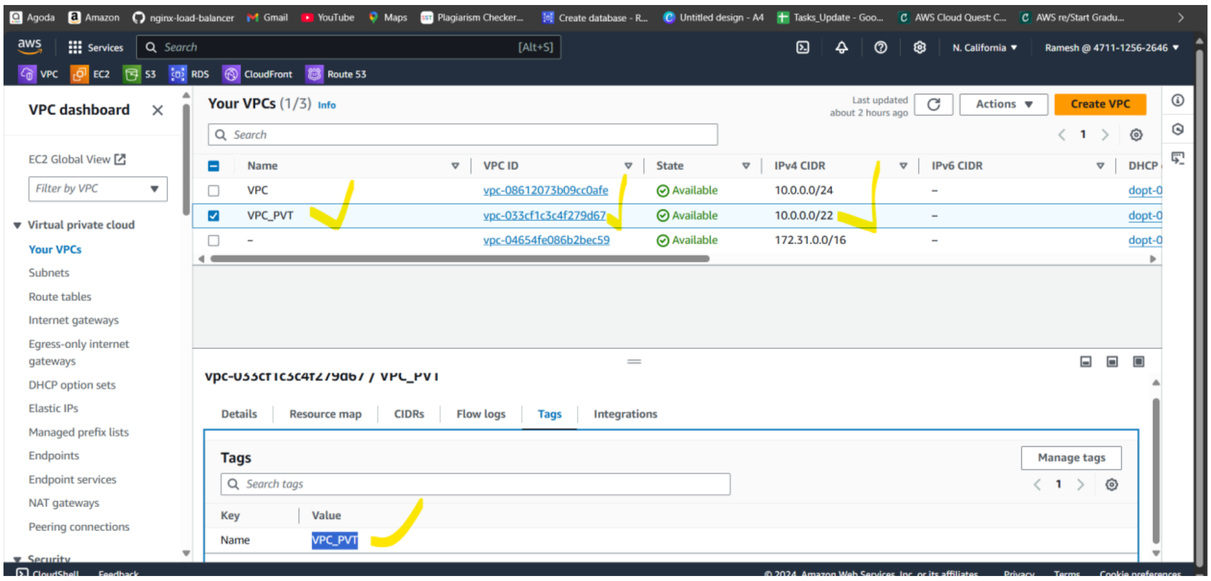
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 5.72.1"
    }
  }
}
provider "aws" {
    region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
```

```
PS C:\Users\ramee\Desktop\Terraform> terraform init -upgrade
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.72.1"...
- Installing hashicorp/aws v5.72.1...
- Installed hashicorp/aws v5.72.1 (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!

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.
```

Terraform Data sources:

In Terraform, data sources are used to read existing infrastructure and start with the keyword data. They allow you to access resources outside your configuration, avoiding duplication and enabling smooth integration.

Here I have one Vpc already existed so I want to read the all details of Vpc_id, Vpc_cidr Like using the data sources.


Your VPCs (1/3)

	Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR	DHCP
☐	VPC	vpc-08612073b09cc0afe	Available	10.0.0.0/24	–	dopt-0
☑	VPC_PVT	vpc-033cf1c3c4f279d67	Available	10.0.0.0/22	–	dopt-0
☐	-	vpc-04654fe086b2bec59	Available	172.31.0.0/16	–	dopt-0

vpc-033cf1c3c4f279d67 / VPC_PVT

Tags

Key	Value
Name	VPC_PVT

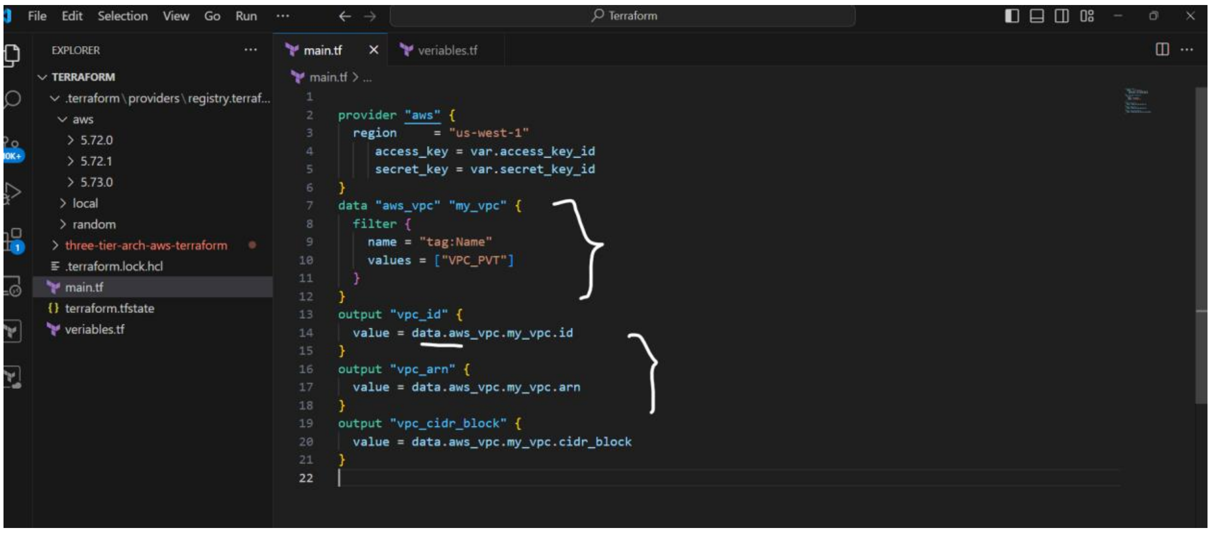
In main.tf I written the below script to get the details of VPC.

Start with – data, resource name --- aws_vpc, logical name --- my_vpc

Filter – it means I am filtering the specific VPC.

Using the output block this all details we can see in the output block.

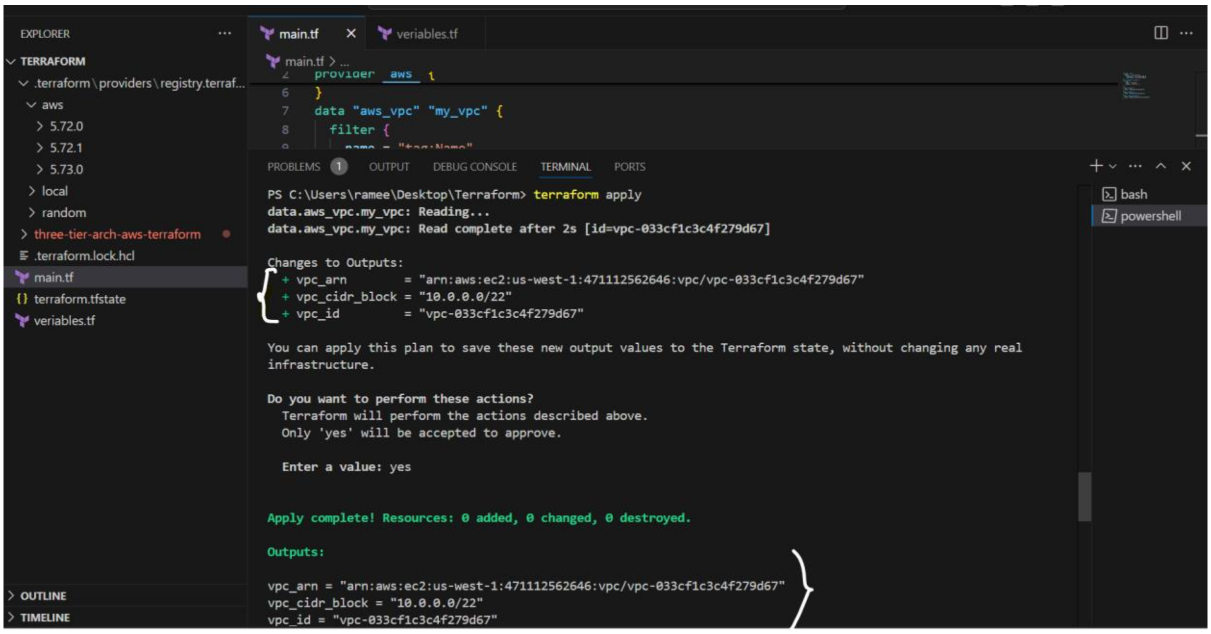
```
data "aws_vpc" "my_vpc" {
  filter {
    name = "tag:Name"
    values = ["VPC_PVT"]
  }
}
output "vpc_id" {
  value = data.aws_vpc.my_vpc.id
}
output "vpc_arn" {
  value = data.aws_vpc.my_vpc.arn
}
output "vpc_cidr_block" {
  value = data.aws_vpc.my_vpc.cidr_block }
```



```
provider "aws" {
  region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
data "aws_vpc" "my_vpc" {
  filter {
    name = "tag:Name"
    values = ["VPC_PVT"]
  }
}
output "vpc_id" {
  value = data.aws_vpc.my_vpc.id
}
output "vpc_arn" {
  value = data.aws_vpc.my_vpc.arn
}
output "vpc_cidr_block" {
  value = data.aws_vpc.my_vpc.cidr_block
}
```

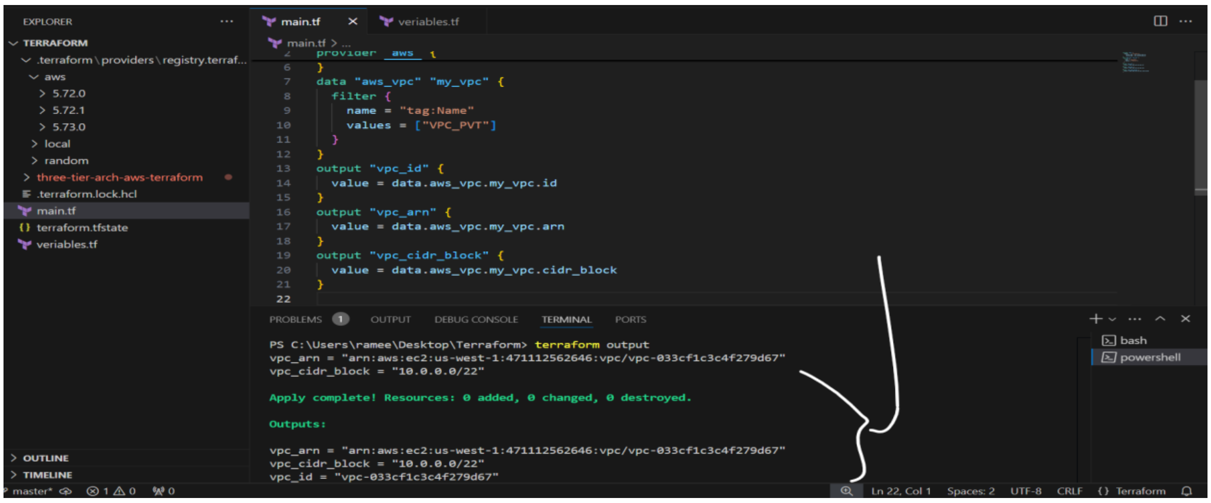
After that you need to initialize the all dependency's – terraform init

Terraform apply ---- to apply the changes.



```
PS C:\Users\ramee\Desktop\Terraform> terraform apply
data.aws_vpc.my_vpc: Reading...
data.aws_vpc.my_vpc: Read complete after 2s [id=vpc-033cf1c3c4f279d67]

Changes to Outputs:
  + vpc_arn        = "arn:aws:ec2:us-west-1:471112562646:vpc/vpc-033cf1c3c4f279d67"
  + vpc_cidr_block = "10.0.0.0/22"
  + vpc_id         = "vpc-033cf1c3c4f279d67"

You can apply this plan to save these new output values to the Terraform state, without changing any real
infrastructure.

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


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

Outputs:

vpc_arn = "arn:aws:ec2:us-west-1:471112562646:vpc/vpc-033cf1c3c4f279d67"
vpc_cidr_block = "10.0.0.0/22"
vpc_id = "vpc-033cf1c3c4f279d67"
```

Now by entering the – terraform output you can see the directly



```
PS C:\Users\ramee\Desktop\Terraform> terraform output
vpc_arn = "arn:aws:ec2:us-west-1:471112562646:vpc/vpc-033cf1c3c4f279d67"
vpc_cidr_block = "10.0.0.0/22"

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

Outputs:

vpc_arn = "arn:aws:ec2:us-west-1:471112562646:vpc/vpc-033cf1c3c4f279d67"
vpc_cidr_block = "10.0.0.0/22"
vpc_id = "vpc-033cf1c3c4f279d67"
```

Terraform resource meta arguments:

Terraform resource Meta-Arguments can be useful while setting up your cloud infrastructure. The resource arguments depends_on, count, for_Each, provider, lifecycle has some features such as –

count: Creates multiple instances of a resource by specifying how many you want.

for_each: Iterates over a list or map to create multiple resources from the same block.

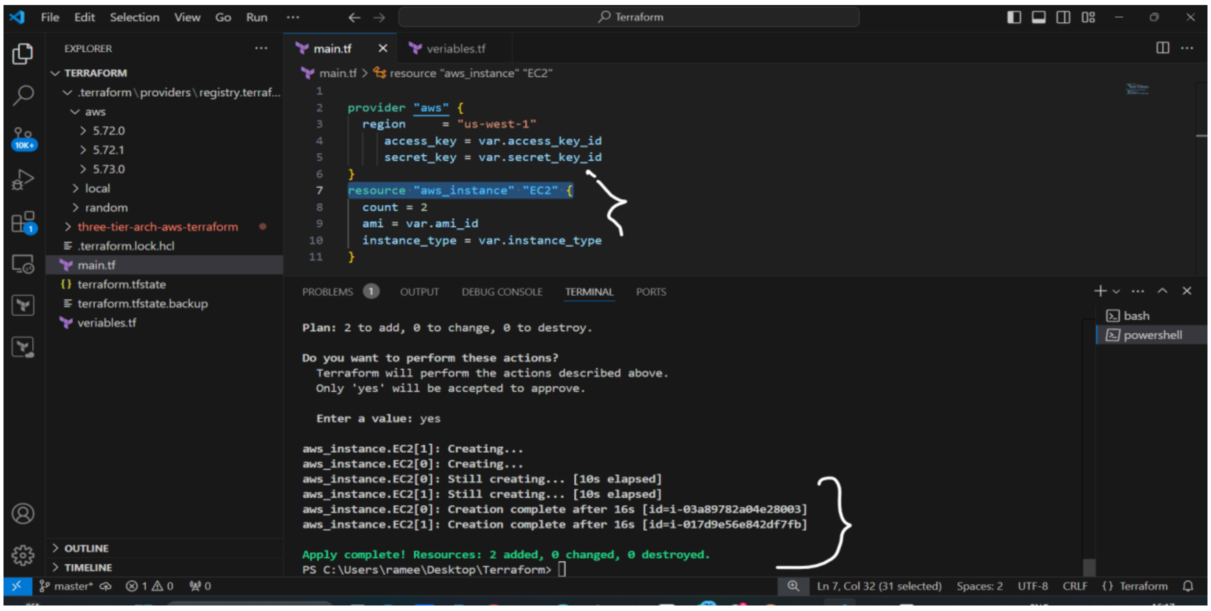
provider: Overrides the default provider settings, allowing you to specify which provider to use for a resource.

lifecycle: Controls resource management by preventing destruction, creating resources after others are deleted, and ignoring specific changes to the state.

depends_on: Specifies dependencies between resources to ensure they are created in the correct order.

Count: As the name suggests count can be used inside the aws_instance block to specify how many resources you would like to create.

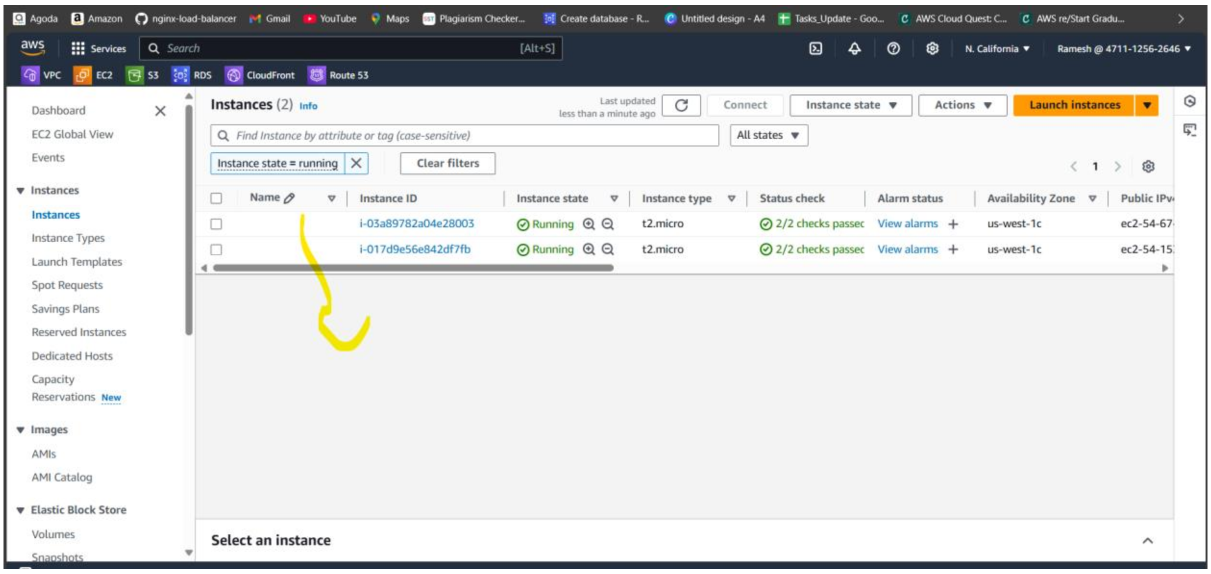
Here I am mention count = 2



```
provider "aws" {
  region     = "us-west-1"
    access_key = var.access_key_id
    secret_key = var.secret_key_id
}
resource "aws_instance" "EC2" {
  count = 2
  ami = var.ami_id
  instance_type = var.instance_type
}
```

```
Plan: 2 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.EC2[1]: Creating...
aws_instance.EC2[0]: Creating...
aws_instance.EC2[0]: Still creating... [10s elapsed]
aws_instance.EC2[1]: Still creating... [10s elapsed]
aws_instance.EC2[0]: Creation complete after 16s [id=i-03a89782a04e28003]
aws_instance.EC2[1]: Creation complete after 16s [id=i-017d9e56e842df7fb]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\ramee\Desktop\Terraform>
```

So here two instances are created.

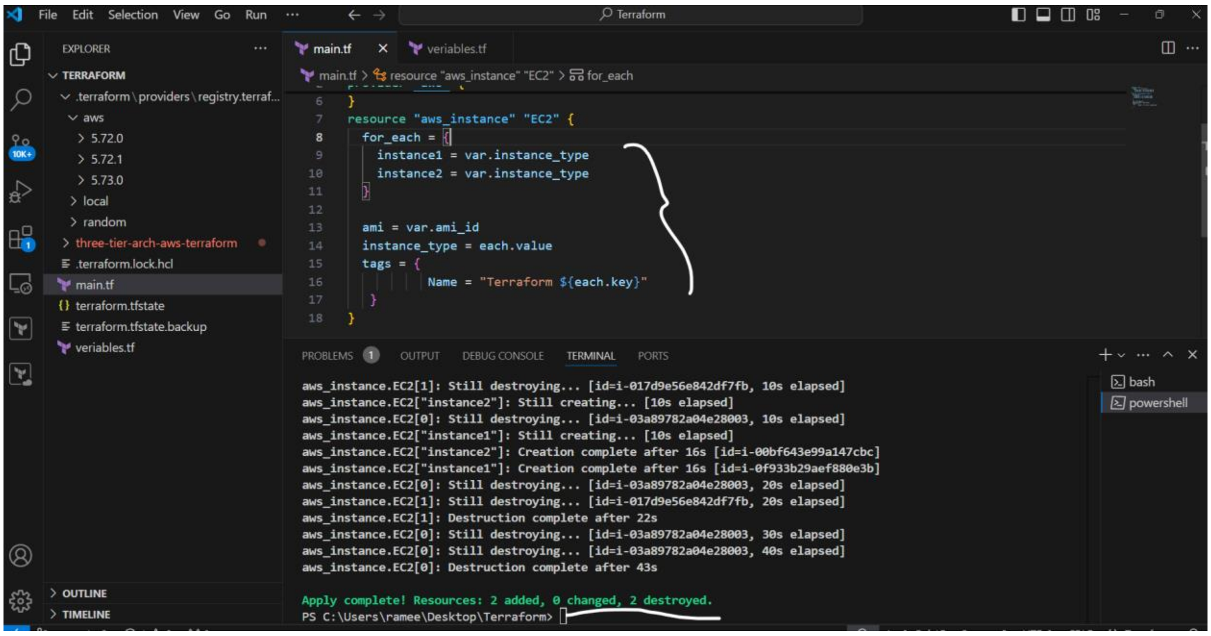


Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv
	i-03a89782a04e28003	Running	t2.micro	2/2 checks passec	View alarms	us-west-1c	ec2-54-67
	i-017d9e56e842df7fb	Running	t2.micro	2/2 checks passec	View alarms	us-west-1c	ec2-54-15

for_each:

Similar to the previous step 1 for_each can also be used for creating similar kinds of resources instead of creating a writing duplicate terraform block.

```
# Define an AWS EC2 instance resource
resource "aws_instance" "EC2" {
  # Use for_each to create multiple instances based on the provided map
  for_each = {
    instance1 = var.instance_type        # First instance using the specified instance type
    instance2 = var.instance_type        # Second instance using the same instance type
  }

  ami           = var.ami_id             # AMI ID from a variable to specify the base image
  instance_type = each.value             # Set the instance type using the current value from the for_each

  # Tags to identify the instances
  tags = {
    Name = "Terraform ${each.key}"        # Tag each instance with a unique name based on the key
  }
}
```



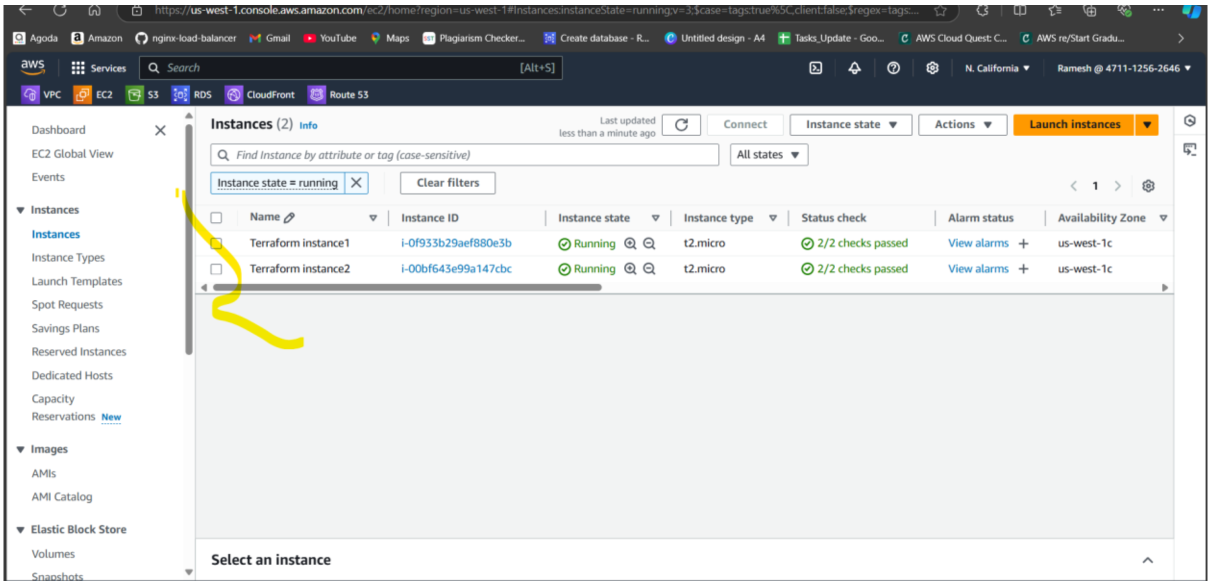
```
}
resource "aws_instance" "EC2" {
  for_each = {
    instance1 = var.instance_type
    instance2 = var.instance_type
  }

  ami = var.ami_id
  instance_type = each.value
  tags = {
          Name = "Terraform ${each.key}"
   }
}
```

```
aws_instance.EC2[1]: Still destroying... [id=i-017d9e56e842df7fb, 10s elapsed]
aws_instance.EC2["instance2"]: Still creating... [10s elapsed]
aws_instance.EC2[0]: Still destroying... [id=i-03a89782a04e28003, 10s elapsed]
aws_instance.EC2["instance1"]: Still creating... [10s elapsed]
aws_instance.EC2["instance2"]: Creation complete after 16s [id=i-00bf643e99a147cbc]
aws_instance.EC2["instance1"]: Creation complete after 16s [id=i-0f933b29aef880e3b]
aws_instance.EC2[0]: Still destroying... [id=i-03a89782a04e28003, 20s elapsed]
aws_instance.EC2[1]: Still destroying... [id=i-017d9e56e842df7fb, 20s elapsed]
aws_instance.EC2[1]: Destruction complete after 22s
aws_instance.EC2[0]: Still destroying... [id=i-03a89782a04e28003, 30s elapsed]
aws_instance.EC2[0]: Still destroying... [id=i-03a89782a04e28003, 40s elapsed]
aws_instance.EC2[0]: Destruction complete after 43s

Apply complete! Resources: 2 added, 0 changed, 2 destroyed.
PS C:\Users\ramee\Desktop\Terraform>
```

- As you can see in the above terraform block we have created 2 key-value pair instance1 = t2.micro and instance2 = t2.micro inside the for_each block.

The next question is ***how to use key-value pair defined inside for_each?***

- **The answer -** It is very simple ***you can just simply type each.value*** and it will iterate over the values.
- Here is a screenshot from aws after starting the **aws_instance**



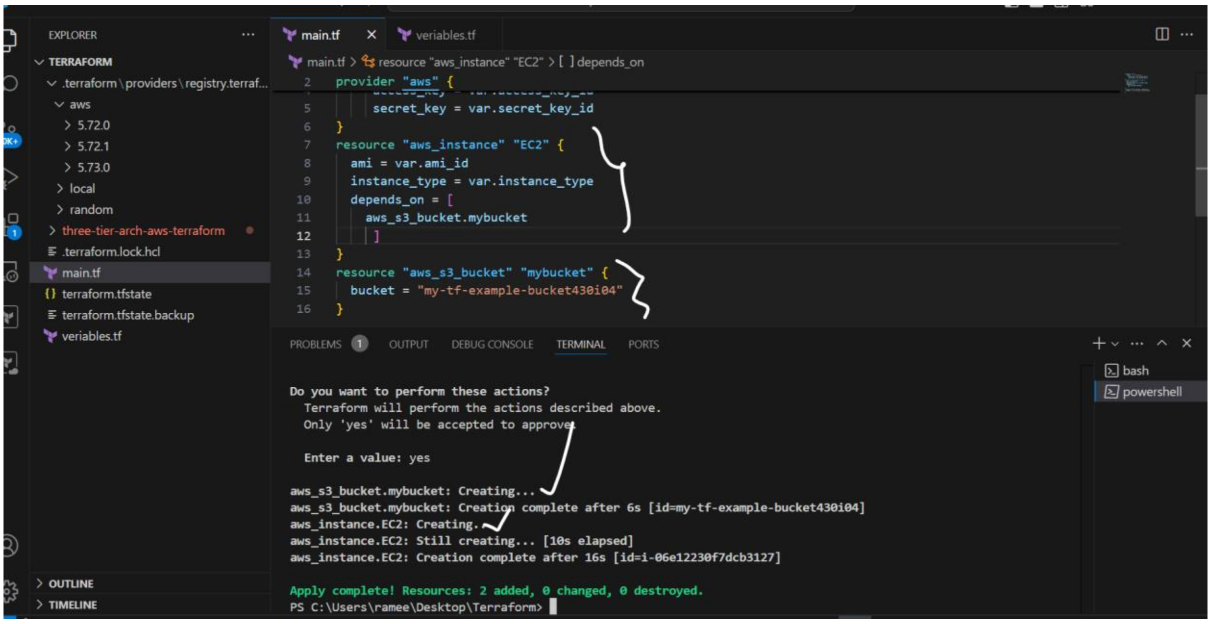
Instances (2)

Instance state = running

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
	Terraform instance1	i-0f933b29aef880e3b	Running	t2.micro	2/2 checks passed	View alarms	us-west-1c
	Terraform instance2	i-00bf643e99a147cbc	Running	t2.micro	2/2 checks passed	View alarms	us-west-1c

depends_on: depends_on makes sure one resource is created only after another is finished.

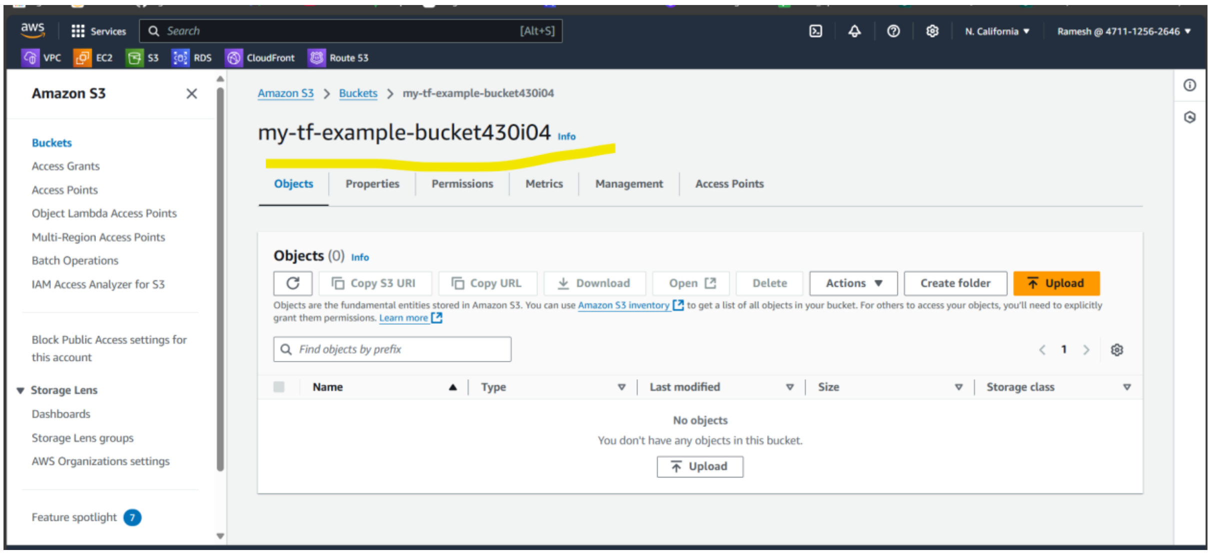
This Terraform script defines the EC2 instance resource first but uses the `depends_on` argument to ensure it is created only after the S3 bucket has been successfully provisioned.



```
provider "aws" {
    secret_key = var.secret_key_id
}
resource "aws_instance" "EC2" {
  ami = var.ami_id
  instance_type = var.instance_type
  depends_on = [
    aws_s3_bucket.mybucket
  ]
}
resource "aws_s3_bucket" "mybucket" {
  bucket = "my-tf-example-bucket430i04"
}
```

```
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_s3_bucket.mybucket: Creating...
aws_s3_bucket.mybucket: Creation complete after 6s [id=my-tf-example-bucket430i04]
aws_instance.EC2: Creating...
aws_instance.EC2: Still creating... [10s elapsed]
aws_instance.EC2: Creation complete after 16s [id=i-06e12230f7dcb3127]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
PS C:\Users\ramee\Desktop\Terraform>
```

Here bucket is crated.



Amazon S3 > Buckets > my-tf-example-bucket430i04

my-tf-example-bucket430i04

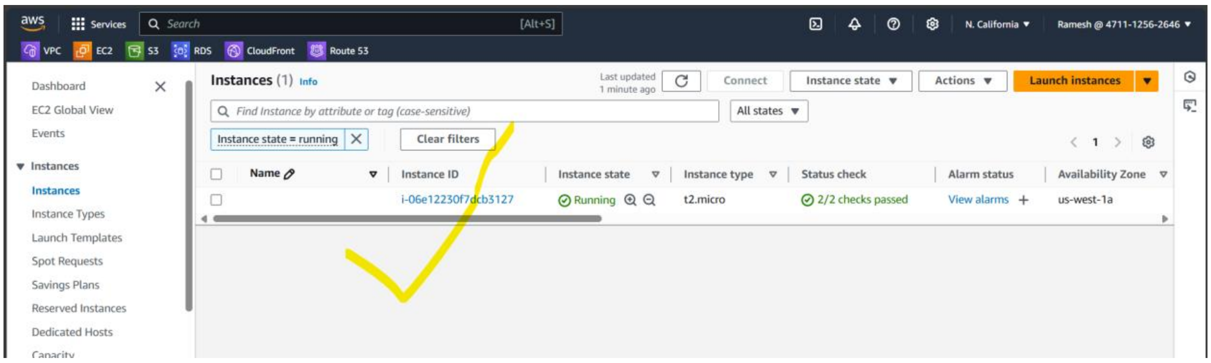
Objects (0)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

No objects

You don't have any objects in this bucket.

Here ec2 instance also created.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
	i-06e12230f7dcb3127	Running	t2.micro	2/2 checks passed	View alarms	us-west-1a

Done.

2) reate CICD pipeline for Nodejs Application.
https://github.com/betawins/Trading-UI.git

3) Explain 10 Maven commands.

1. **mvn clean** – Cleans the target directory by deleting any previously compiled or built files. Useful before starting a fresh build.

2. **mvn compile** - Compiles the project's source code, creating bytecode files from Java source files in the target/classes folder.

3. **mvn test** - Runs all test cases in the project using frameworks like JUnit or TestNG, ensuring the code's functionality.

4. **mvn package** - Compiles the code, runs tests, and packages the result into a JAR or WAR file, which is saved in the target directory.

5. **mvn install** - Builds and packages the project and then installs it into the local Maven repository (~/.m2/repository) for use in other projects.

6. **mvn deploy** - Uploads the packaged code to a remote repository, making it available to others.

7. **mvn site** - Generates project documentation and reports, creating a detailed site in the target/site directory.

8. **mvn dependency:tree** - Displays a tree of the project dependencies, helping to resolve version conflicts and understand transitive dependencies.

9. **mvn dependency:copy-dependencies** - Copies all dependencies to the specified directory, useful for external or customized deployments.

10. **mvn clean install -DskipTests** - Performs clean and install phases without running tests, often used when tests are not needed immediately.