TASK-01 TERRAFORM

Q.1 Make a note on:

a. What is Terraform?
b. Why Terraform?
c. Benefits of Terraform

Terraform:

Terraform is an open-source infrastructure as code (IaC) software tool created by HashiCorp. It allows users to define and provision a data center infrastructure using a high-level configuration language known as HashiCorp Configuration Language (HCL), or optionally JSON. Terraform manages external resources (such as public cloud infrastructure, private cloud infrastructure, network appliances, software as a service, and platform as a service) through its configuration files, which describe the components needed to run a single application or your entire data center.

Terraform is chosen for several reasons:

- **Declarative Syntax:** Users define the desired state of infrastructure, and Terraform figures out how to achieve that state, managing dependencies and sequencing steps automatically.
- **Multi-Cloud Capability:** Terraform supports multiple cloud providers, allowing the management of hybrid or multi-cloud environments using a single tool.
- Infrastructure as Code: This approach allows infrastructure to be versioned and treated as code, making it easier to track changes, collaborate, and automate deployments.
- **Resource Graph:** Terraform builds a graph of all resources, which enables it to plan changes efficiently, detect dependencies, and create or modify resources in the correct order.

- **Execution Plans:** Before making any changes, Terraform provides an execution plan which describes what will happen if the code is applied, giving a clear overview of the impact of any changes.
- **Community and Ecosystem:** A large and active community contributes to a rich ecosystem of modules and plugins, extending Terraform's capabilities and making it easier to manage various types of infrastructure.

Benefits of Terraform:

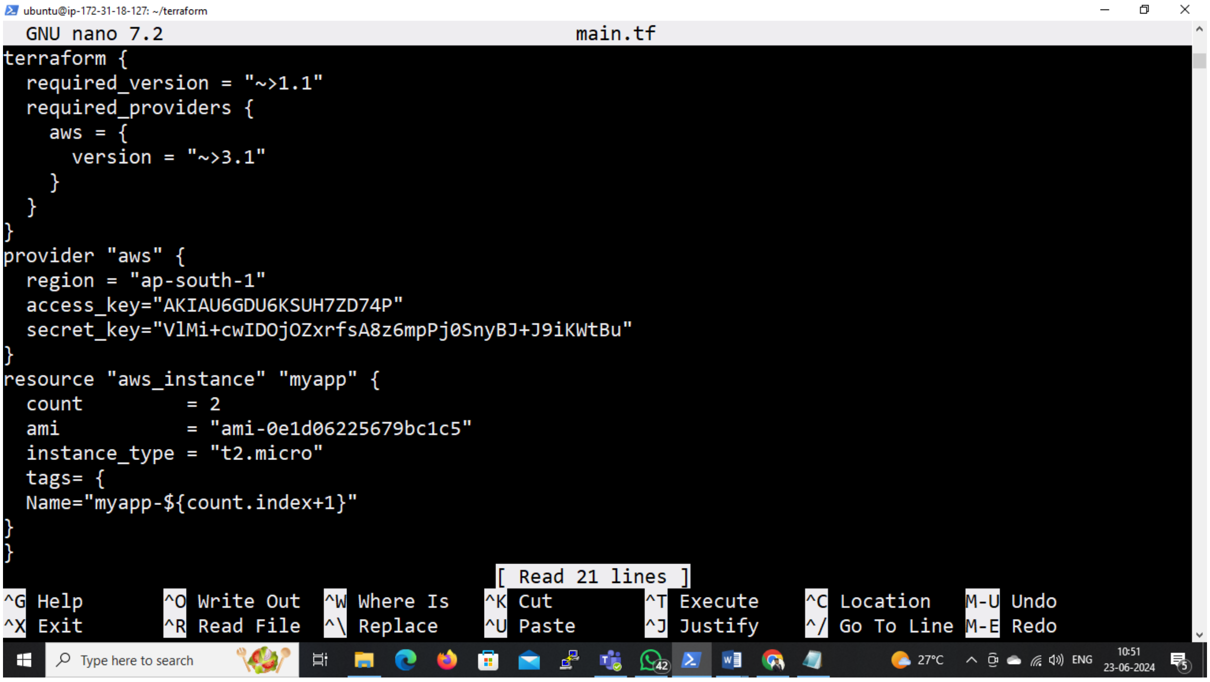
- **Consistency:** With Terraform, infrastructure can be consistently provisioned and configured, reducing human errors and ensuring that the infrastructure state is predictable.
- **Scalability:** Terraform can manage infrastructures of varying sizes, from small environments to large, complex infrastructures across multiple cloud providers.
- **Version Control:** Infrastructure changes can be versioned and tracked, just like application code, providing a clear history of changes and the ability to roll back to previous states if needed.
- **Automation:** Automating infrastructure provisioning with Terraform saves time and reduces manual effort, allowing teams to focus on other critical tasks.
- **Collaboration:** Terraform’s configuration files can be shared and collaborated on, facilitating better teamwork and coordination among different teams and stakeholders.
- **Efficiency:** The resource graph and execution plans enable Terraform to make efficient changes, only modifying what is necessary and providing clear visibility into proposed changes before they are applied.
- **Multi-Provider Support:** Terraform’s ability to support multiple providers means it can be used to manage a diverse range of resources,

from different cloud providers to various on-premises services, providing a unified toolset for infrastructure management.

Q.2 Launch two EC2 instances with names as “myapp-1” and “myapp-2” using Amazon-Linux OS in ‘ap-south-1’ region.

To launch two EC2 instances named "myapp-1" and "myapp-2" using Amazon **Linux OS** in the **ap-south-1** region with Terraform, you need to create a **main.tf** configuration file with the necessary code. Here's a sample main.tf file to achieve this:

- nano main.tf



```
ubuntu@ip-172-31-18-127: ~/terraform
  GNU nano 7.2                                  main.tf
terraform {
  required_version = "~>1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  region = "ap-south-1"
  access_key="AKIAU6GDU6KSUH7ZD74P"
  secret_key="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"
}
resource "aws_instance" "myapp" {
  count         = 2
  ami           = "ami-0e1d06225679bc1c5"
  instance_type = "t2.micro"
  tags= {
  Name="myapp-${count.index+1}"
}
}
                                                  [ Read 21 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

Provider Block:

Specifies the AWS provider and the region where the resources will be created.

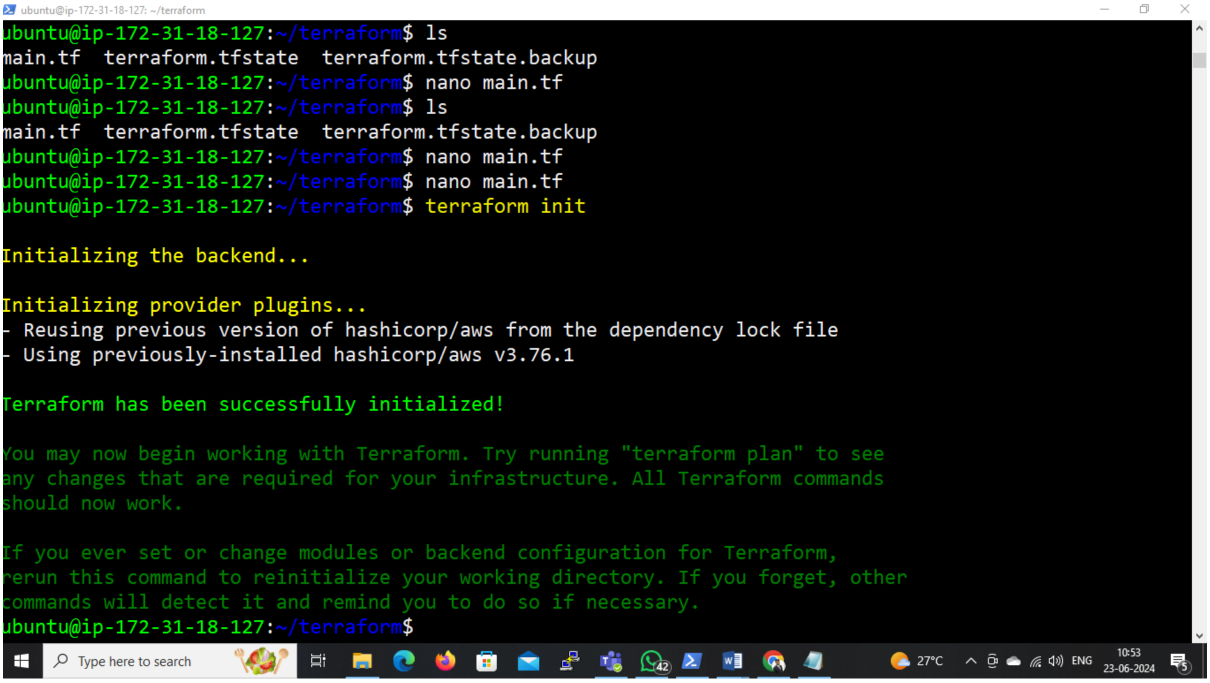
Also provide a access key and secret key of AWS account.

Resource Block:

Defines the aws_instance resource to launch two EC2 instances.

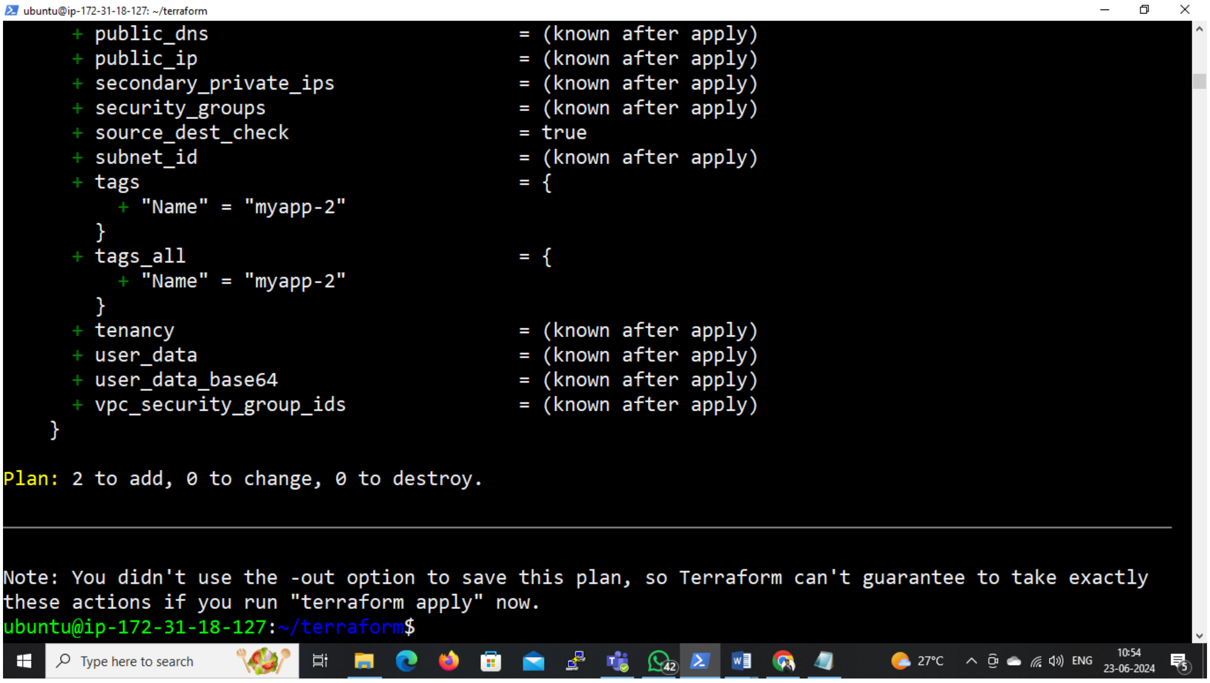
- **count:** Used to create multiple instances. In this case, we set count = 2 to create two instances.
- **ami:** The Amazon Machine Image (AMI) ID for Amazon Linux 2 in the ap-south-1 region. We need to mentioned " ami-0e1d06225679bc1c5" with the latest Amazon Linux 2 AMI ID for the ap-south-1 region.
- **instance_type:** Specifies the type of instance to be created. Here, t2.micro is used.
- **tags:** Assigns a name tag to each instance, using the count index to differentiate between "myapp-1" and "myapp-2".

➢ **Initialize Terraform**: Run **“terraform init”** to initialize the configuration and download the necessary provider plugins.



```
ubuntu@ip-172-31-18-127:~/terraform$ ls
main.tf  terraform.tfstate  terraform.tfstate.backup
ubuntu@ip-172-31-18-127:~/terraform$ nano main.tf
ubuntu@ip-172-31-18-127:~/terraform$ ls
main.tf  terraform.tfstate  terraform.tfstate.backup
ubuntu@ip-172-31-18-127:~/terraform$ nano main.tf
ubuntu@ip-172-31-18-127:~/terraform$ nano main.tf
ubuntu@ip-172-31-18-127:~/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 v3.76.1

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.
ubuntu@ip-172-31-18-127:~/terraform$
```

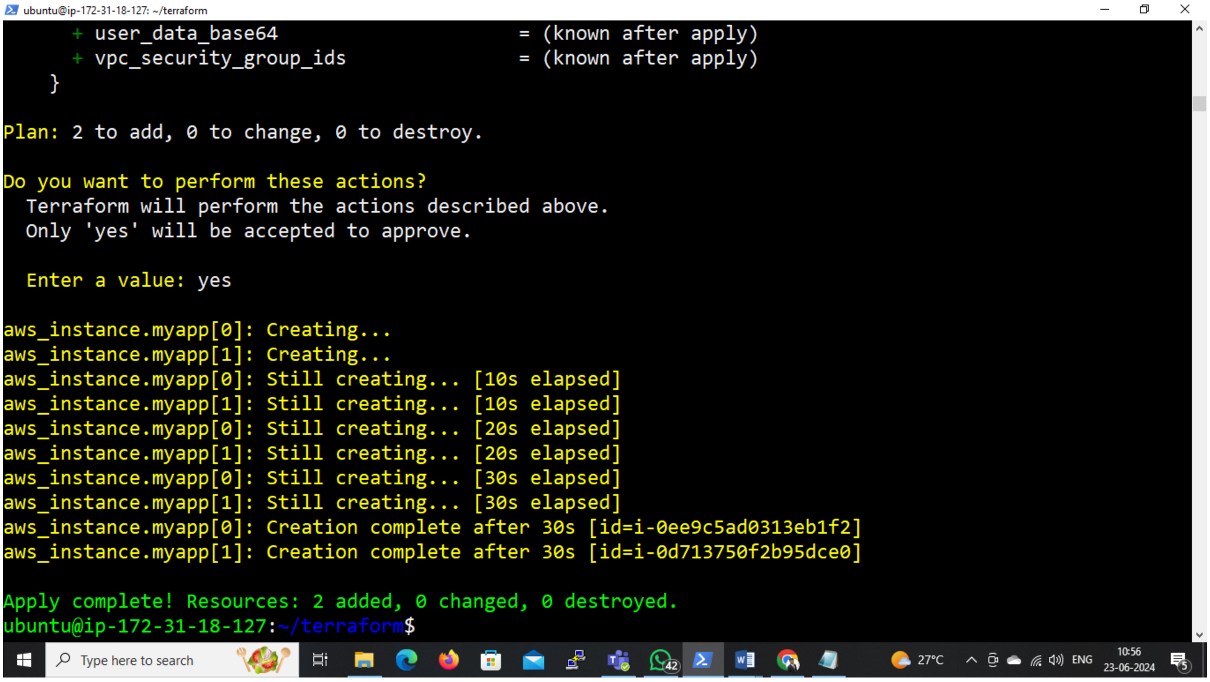
- **Plan the Changes**: Run **“terraform plan”** to preview the changes that will be applied.



```
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "myapp-2"
        }
      + tags_all                             = {
          + "Name" = "myapp-2"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + vpc_security_group_ids               = (known after apply)
    }

Plan: 2 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.
ubuntu@ip-172-31-18-127:~/terraform$
```

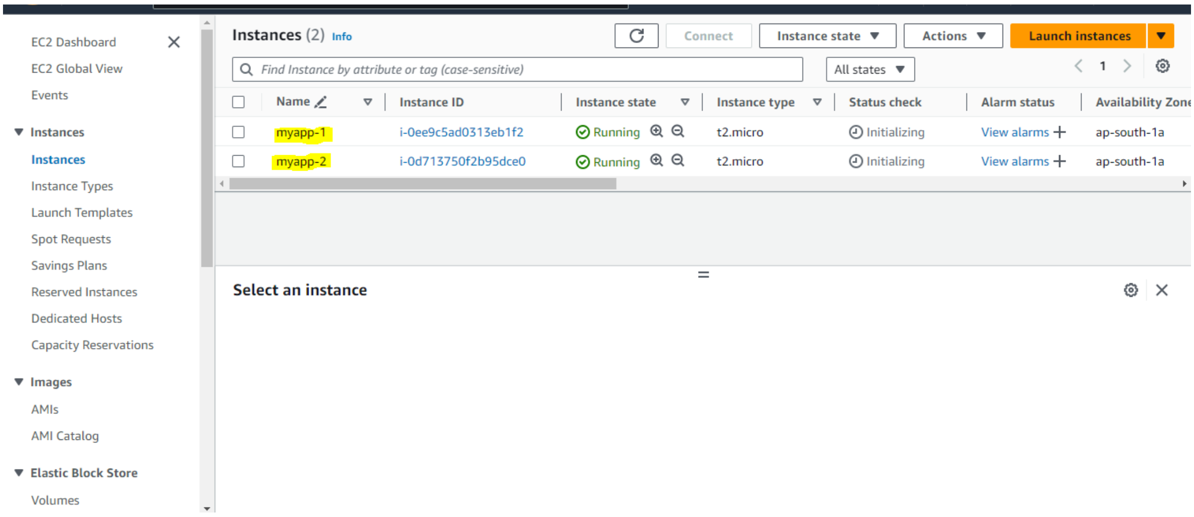
- **Apply the Configuration**: Run “**terraform apply**” to create the EC2 instances as defined in the main.tf file.

- **Confirm the Apply Step**: Type “**yes**” when prompted to confirm that we want to create the resources.



```
      + user_data_base64                     = (known after apply)
      + vpc_security_group_ids               = (known after apply)
    }

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.myapp[0]: Creating...
aws_instance.myapp[1]: Creating...
aws_instance.myapp[0]: Still creating... [10s elapsed]
aws_instance.myapp[1]: Still creating... [10s elapsed]
aws_instance.myapp[0]: Still creating... [20s elapsed]
aws_instance.myapp[1]: Still creating... [20s elapsed]
aws_instance.myapp[0]: Still creating... [30s elapsed]
aws_instance.myapp[1]: Still creating... [30s elapsed]
aws_instance.myapp[0]: Creation complete after 30s [id=i-0ee9c5ad0313eb1f2]
aws_instance.myapp[1]: Creation complete after 30s [id=i-0d713750f2b95dce0]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
ubuntu@ip-172-31-18-127:~/terraform$
```

- This will create two EC2 instances in the **ap-south-1** region, with names "**myapp-1**" and "**myapp-2**" using the specified Amazon Linux AMI.



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
myapp-1	i-0ee9c5ad0313eb1f2	Running	t2.micro	Initializing	View alarms	ap-south-1a
myapp-2	i-0d713750f2b95dce0	Running	t2.micro	Initializing	View alarms	ap-south-1a

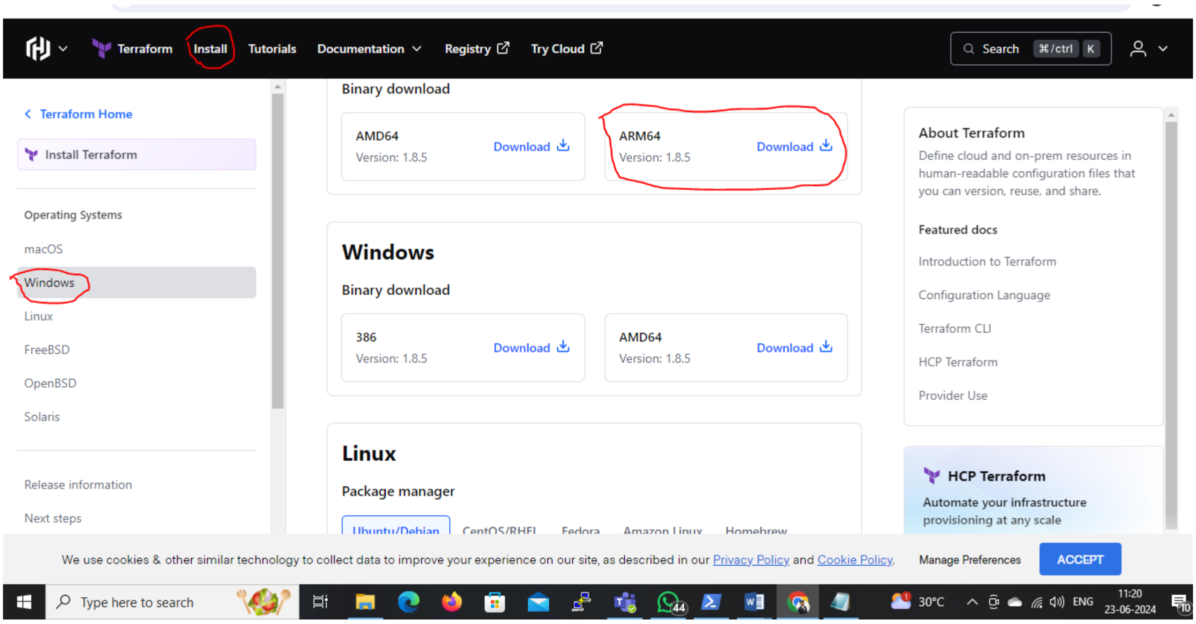
Q.3 Install Terraform on local machine (Laptop), integrate aws and terraform with VS code. Using VS code launch an EC2 instances with name ‘myserver’ using Windows OS in ‘ap-south-1’ region.

Step 1: Install Terraform

On Windows:

1. **Download Terraform:**

- Go to the Terraform download page.
- Download the appropriate package for your operating system.



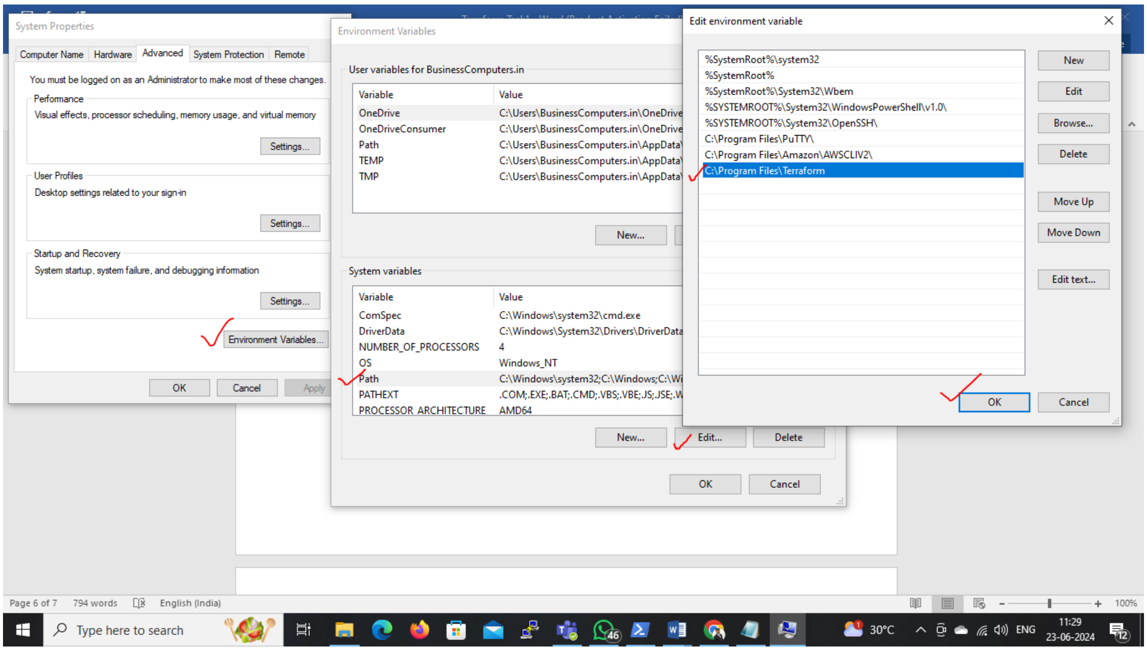
Define cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share.

Automate your infrastructure provisioning at any scale

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

2. **Install Terraform:**

- Unzip the downloaded file.
- Move the terraform.exe file to a directory included in our system's PATH. Typically, we can place it in **C:\Program Files\Terraform**
- Search system variables in laptop & click on edit syetem variables.
- Open a window which is shiown below.
- a. Click on Envirnonment Variables
- b. In system variable drop down list select path and click on Edit
- c. Open a new window click on new button
- d. Add a path which is shown as **C:\Program Files\Terraform**



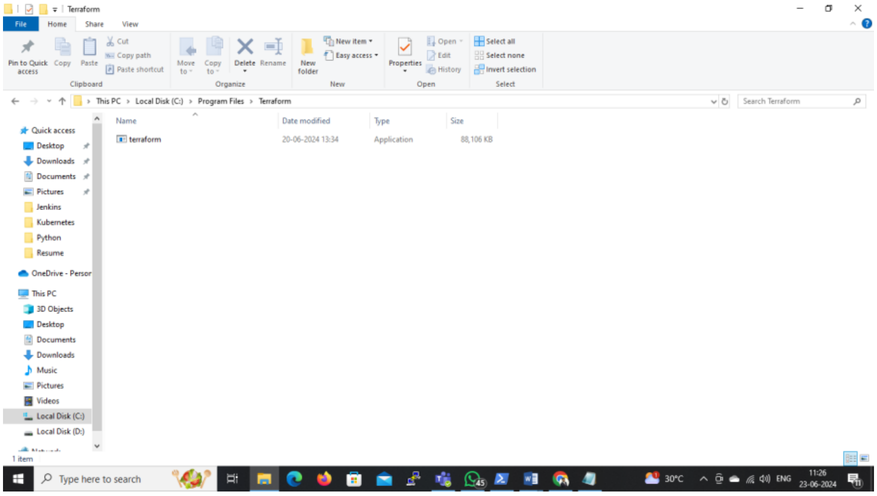
User variables for BusinessComputers.in

Variable	Value
OneDrive	C:\Users\BusinessComputers.in\OneDrive
OneDriveConsumer	C:\Users\BusinessComputers.in\OneDrive
Path	C:\Users\BusinessComputers.in\AppData\
TEMP	C:\Users\BusinessComputers.in\AppData\
TMP	C:\Users\BusinessComputers.in\AppData\

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Wi
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.W
PROCESSOR_ARCHITECTURE	AMD64

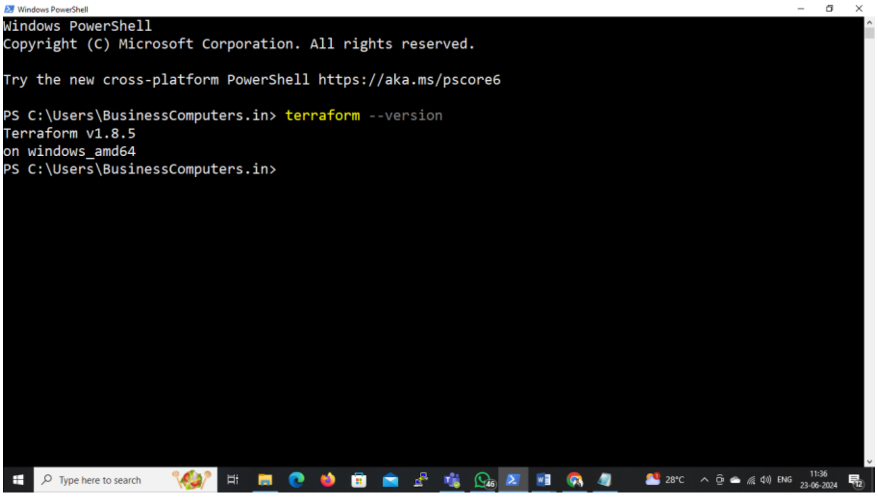
```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\PuTTY\
C:\Program Files\Amazon\AWSCLIV2\
C:\Program Files\Terraform
```



Name	Date modified	Type	Size
terraform	20-06-2024 13:34	Application	88,106 KB

3. Verify Installation:

- Open a command prompt and type:



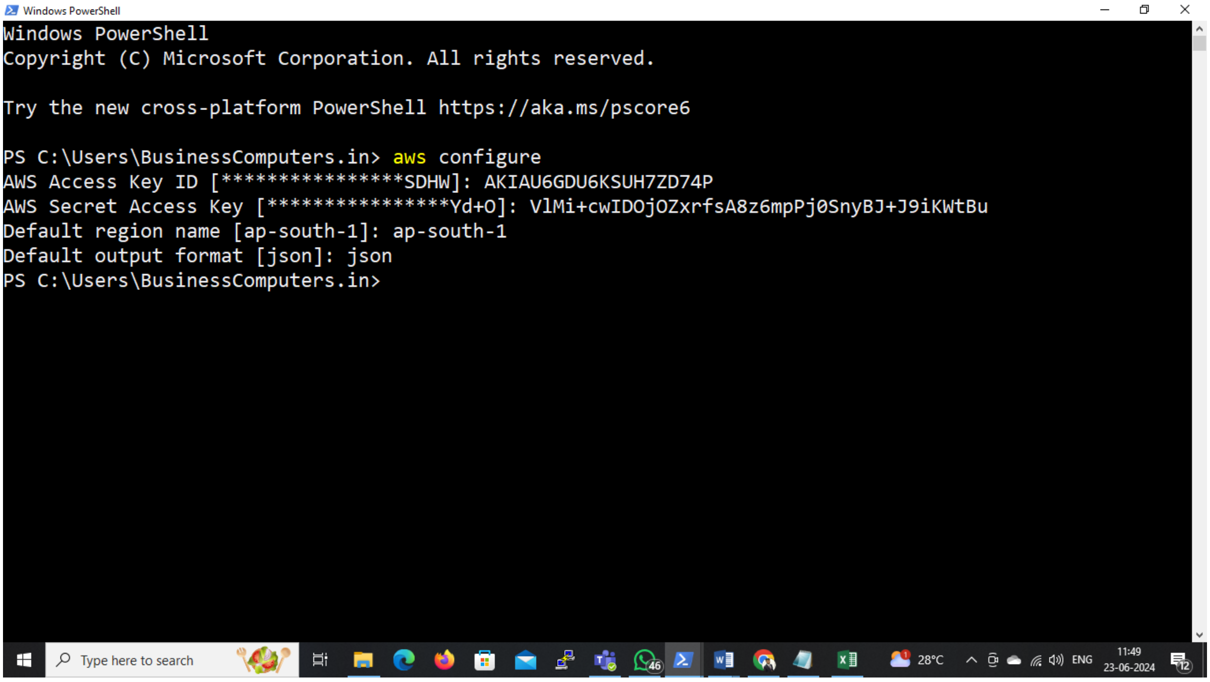
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\BusinessComputers.in> terraform --version
Terraform v1.8.5
on windows_amd64
PS C:\Users\BusinessComputers.in>
```

Step 2: Set Up AWS CLI

- Install AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
- Follow the installation instructions for our OS from the above link

Configure AWS CLI:

- Open a command prompt and configure AWS CLI with our credentials:
- Enter AWS Access Key ID, Secret Access Key, default region name (ap-south-1), and default output format (e.g., json).



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\BusinessComputers.in> aws configure
AWS Access Key ID [****************SDHW]: AKIAU6GDU6KSUH7ZD74P
AWS Secret Access Key [****************Yd+O]: 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
Default region name [ap-south-1]: ap-south-1
Default output format [json]: json
PS C:\Users\BusinessComputers.in>
```

Step 3: Integrate AWS and Terraform with VS Code

- Install Visual Studio Code:
- Download and install VS Code. https://code.visualstudio.com/docs?dv=win
- Install Terraform Extension in VS Code:
- Open VS Code.
- Go to the Extensions view by clicking on the square icon in the sidebar or pressing Ctrl+Shift+X.
- Search for “Terraform” and install the extension by HashiCorp.
- Install AWS Toolkit Extension in VS Code:
- In the Extensions view, search for “AWS Toolkit” and install the extension by AWS.

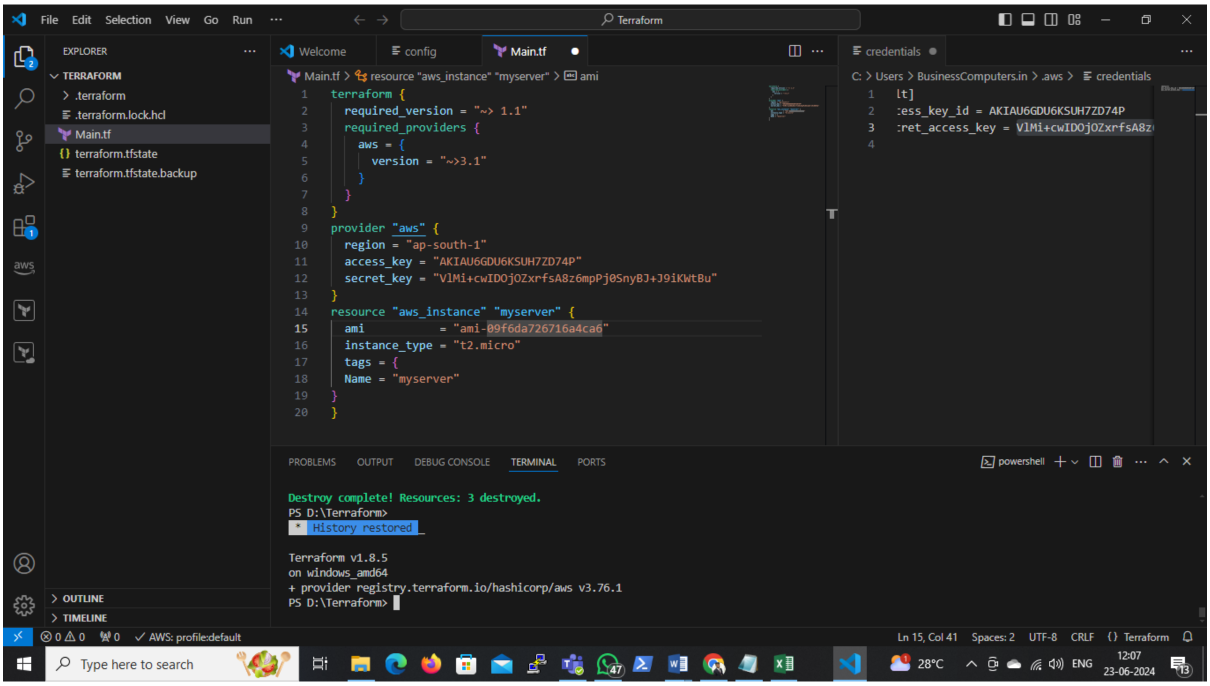
Step 4: Create Terraform Configuration in VS Code

Open a New Folder:

- In VS Code, open a new folder where we will store our Terraform configuration files.

Create “ main.tf ” File:

- Create a new file named main.tf in the folder and add the following configuration:
- Make sure to replace the AMI ID with the latest Windows Server AMI for the ap-south-1 region.



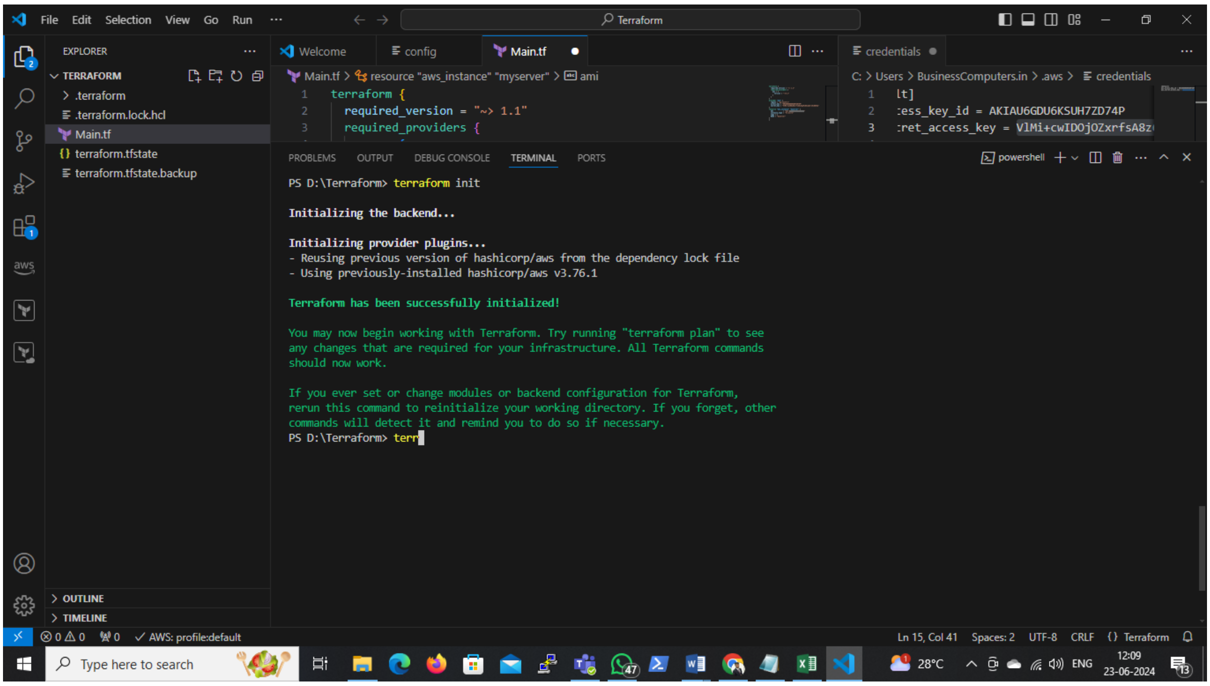
```
terraform {
  required_version = "~> 1.1"
  required_providers {
    aws = {
      version = "~>3.1"
    }
  }
}
provider "aws" {
  region = "ap-south-1"
  access_key = "AKIAU6GDU6KSUH7ZD74P"
  secret_key = "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"
}
resource "aws_instance" "myserver" {
  ami           = "ami-09f6da726716a4ca6"
  instance_type = "t2.micro"
  tags = {
  Name = "myserver"
}
}
```

```
lt]
:ess_key_id = AKIAU6GDU6KSUH7ZD74P
:ret_access_key = VlMi+cwIDOjOZxrfsA8z
```

```
Destroy complete! Resources: 3 destroyed.
PS D:\Terraform>
 *  History restored

Terraform v1.8.5
on windows_amd64
+ provider registry.terraform.io/hashicorp/aws v3.76.1
PS D:\Terraform>
```

Open the integrated terminal in VS Code & Run the following command

- ➢ terraform init

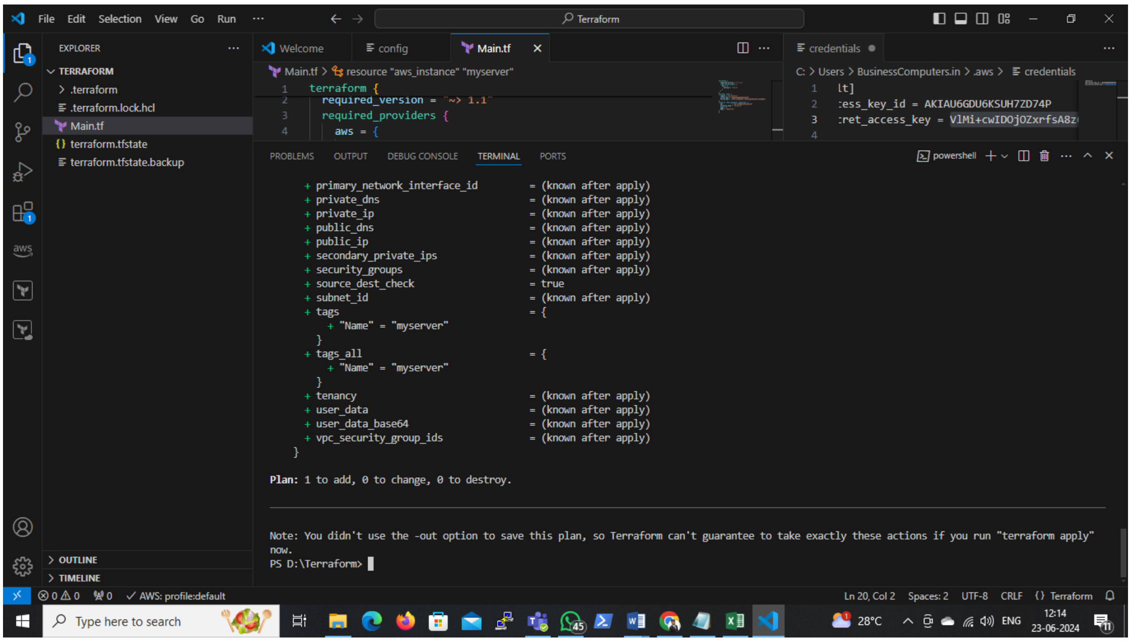


```
terraform {
  required_version = "~> 1.1"
  required_providers {
```

```
lt]
:ess_key_id = AKIAU6GDU6KSUH7ZD74P
:ret_access_key = VlMi+cwIDOjOZxrfsA8z
```

```
PS D:\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 v3.76.1

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 D:\Terraform> terr
```

- ➢ terraform plan

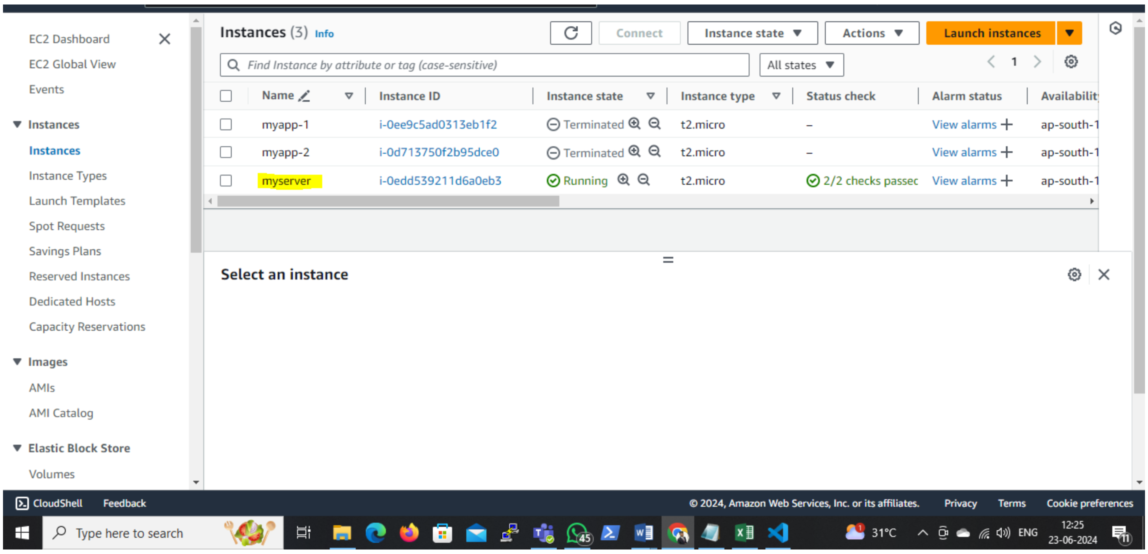


```
terraform {
  required_version = "~> 1.1"
  required_providers {
    aws = {
```

```
lt]
:ess_key_id = AKIAU6GDU6KSUH7ZD74P
:ret_access_key = VlMi+cwIDOjOZxrfsA8z
```

```
      + primary_network_interface_id         = (known after apply)
      + private_dns                          = (known after apply)
      + private_ip                           = (known after apply)
      + public_dns                           = (known after apply)
      + public_ip                            = (known after apply)
      + secondary_private_ips                = (known after apply)
      + security_groups                      = (known after apply)
      + source_dest_check                    = true
      + subnet_id                            = (known after apply)
      + tags                                 = {
          + "Name" = "myserver"
        }
      + tags_all                             = {
          + "Name" = "myserver"
        }
      + tenancy                              = (known after apply)
      + user_data                            = (known after apply)
      + user_data_base64                     = (known after apply)
      + vpc_security_group_ids               = (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 D:\Terraform>
```

- ➢ terraform apply



□	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availabilit
□	myapp-1	i-0ee9c5ad0313eb1f2	Terminated	t2.micro	–	View alarms +	ap-south-1
□	myapp-2	i-0d713750f2b95dce0	Terminated	t2.micro	–	View alarms +	ap-south-1
□	myserver	i-0edd539211d6a0eb3	Running	t2.micro	2/2 checks passec	View alarms +	ap-south-1

Created an EC2 instance named myserver with a Windows OS in the ap-south-1 region.