**Terraform Task-1**

Q.1 Make a note on:

* 1. What is Terraform?
  2. Why Terraform?
  3. 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



**Provider Block:**

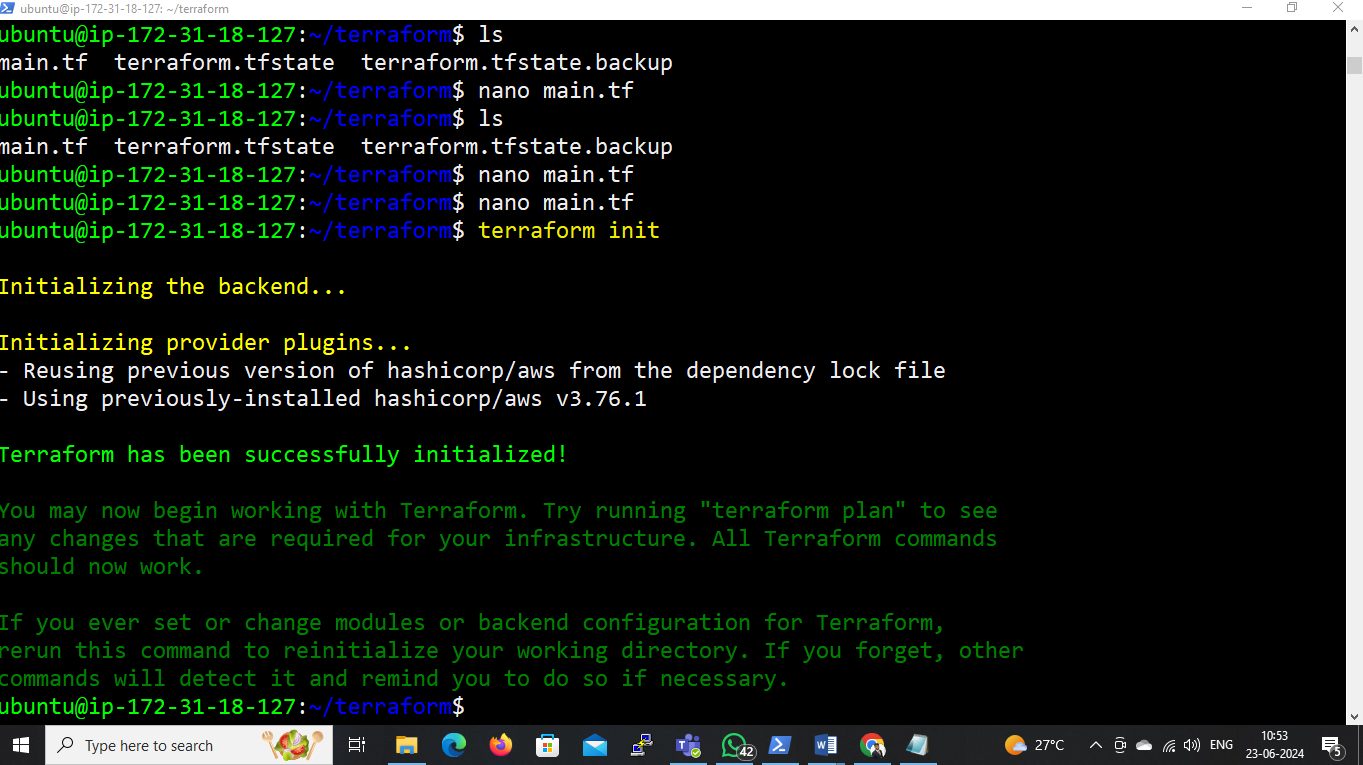
Specifies the AWS provider and the region where the resources will be created.

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

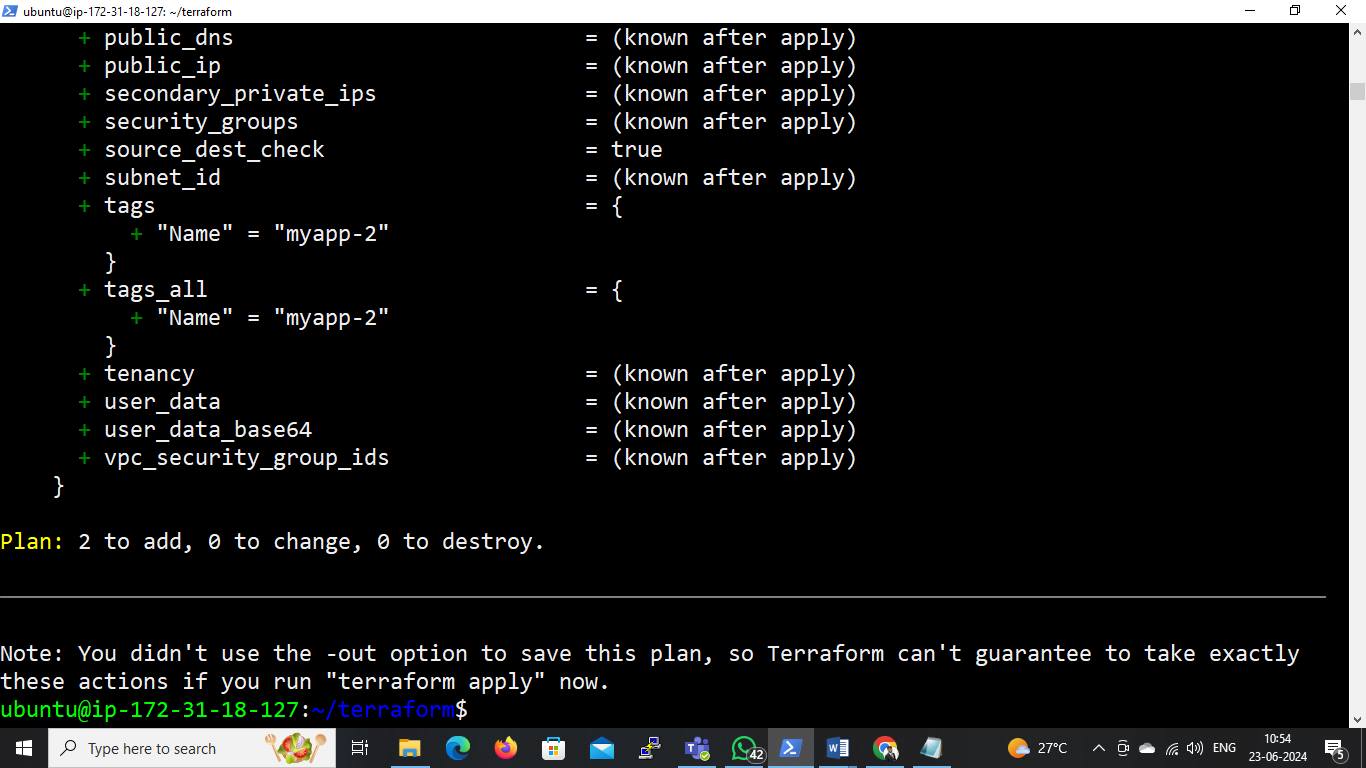
**Resource Block:**

Defines the aws\_instance resource to launch two EC2 instances.

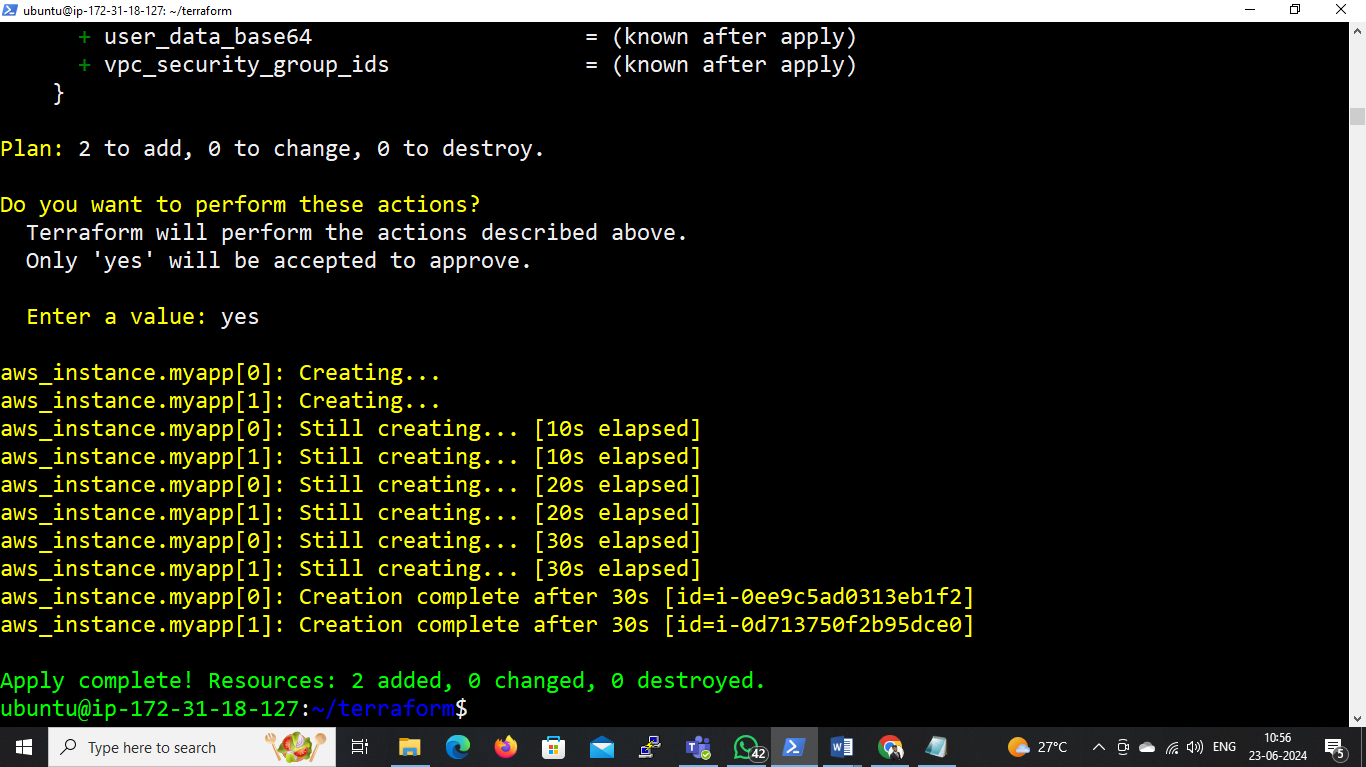
* **count:** Used to create multiple instances. In this case, we set count = 2 to create two instances.
* **ami:** The Amazon Machine Image (AMI) ID for Amazon Linux 2 in the ap-south-1 region. We need to mentioned " ami-0e1d06225679bc1c5" with the latest Amazon Linux 2 AMI ID for the ap-south-1 region.
* **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.



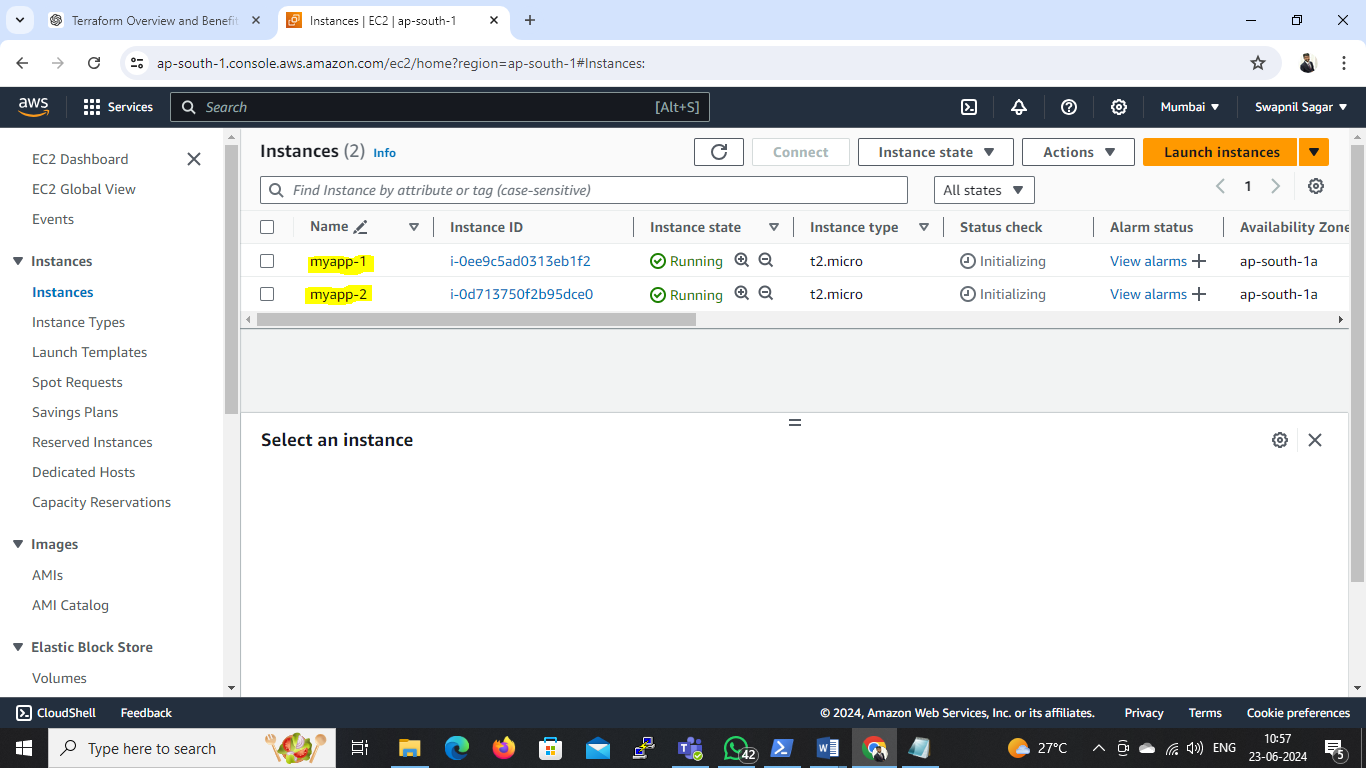
* **Plan the Changes**: Run **“terraform plan”** to preview the changes that will be applied.



* **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.



* 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.



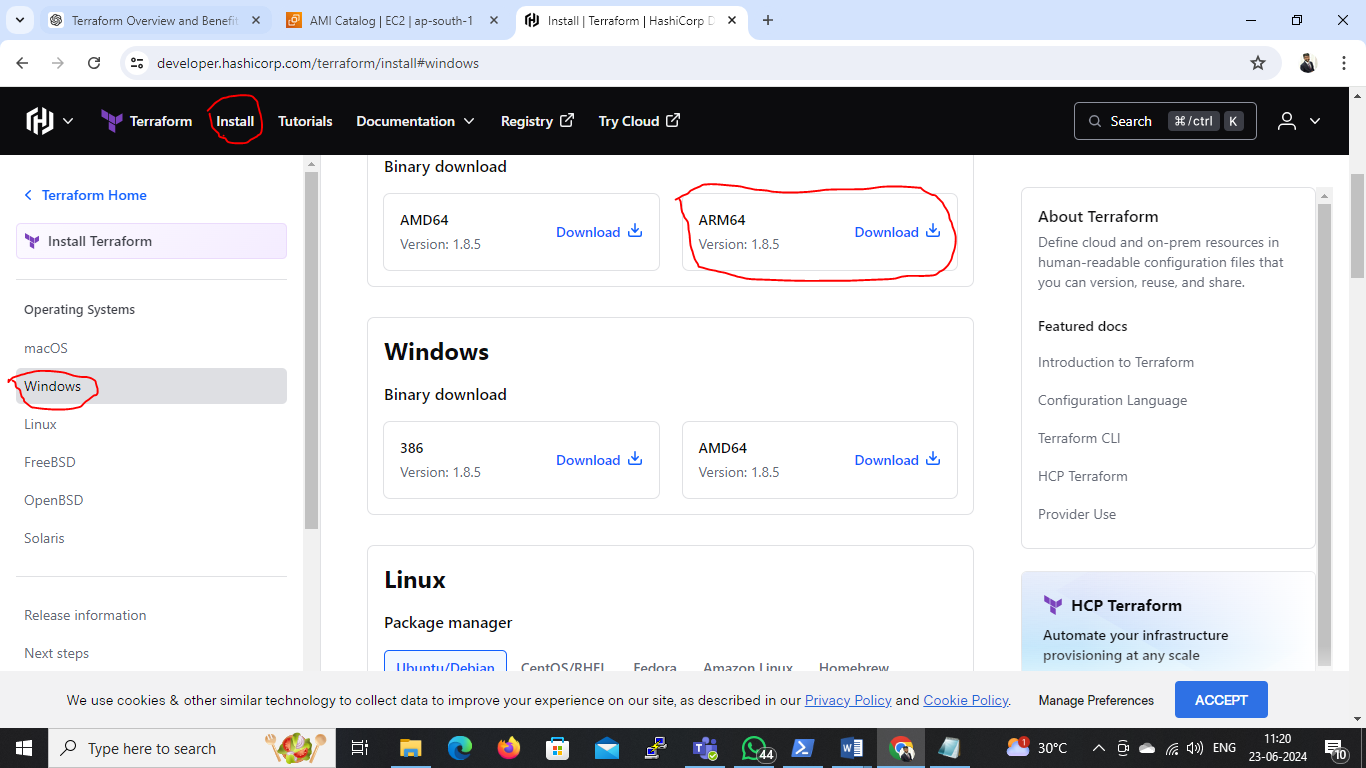
**Q.3 Install Terraform on local machine (Laptop), integrate aws and terraform with VS code. Using VS code launch an EC2 instances with name ‘myserver’ using Windows OS in ‘ap-south-1’ region.**

**Step 1: Install Terraform**

**On Windows:**

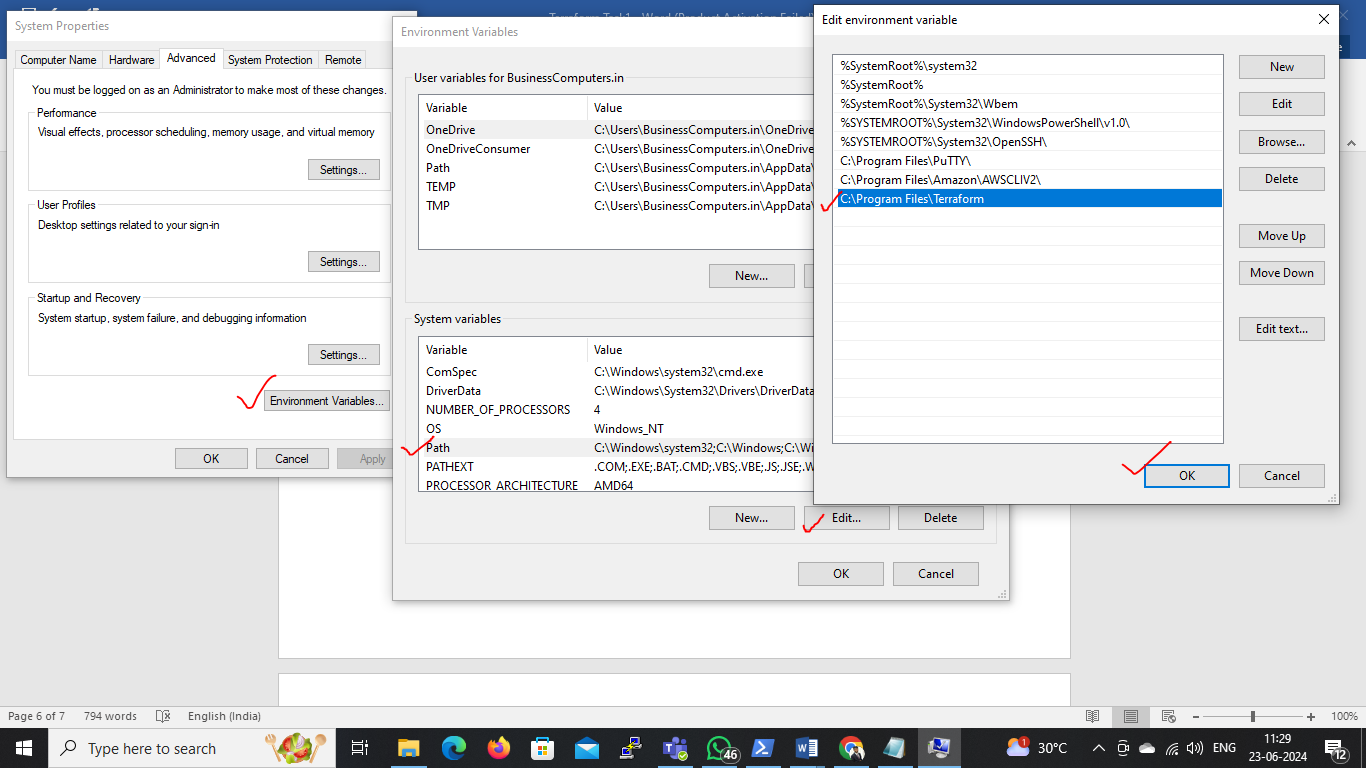
1. **Download Terraform:**

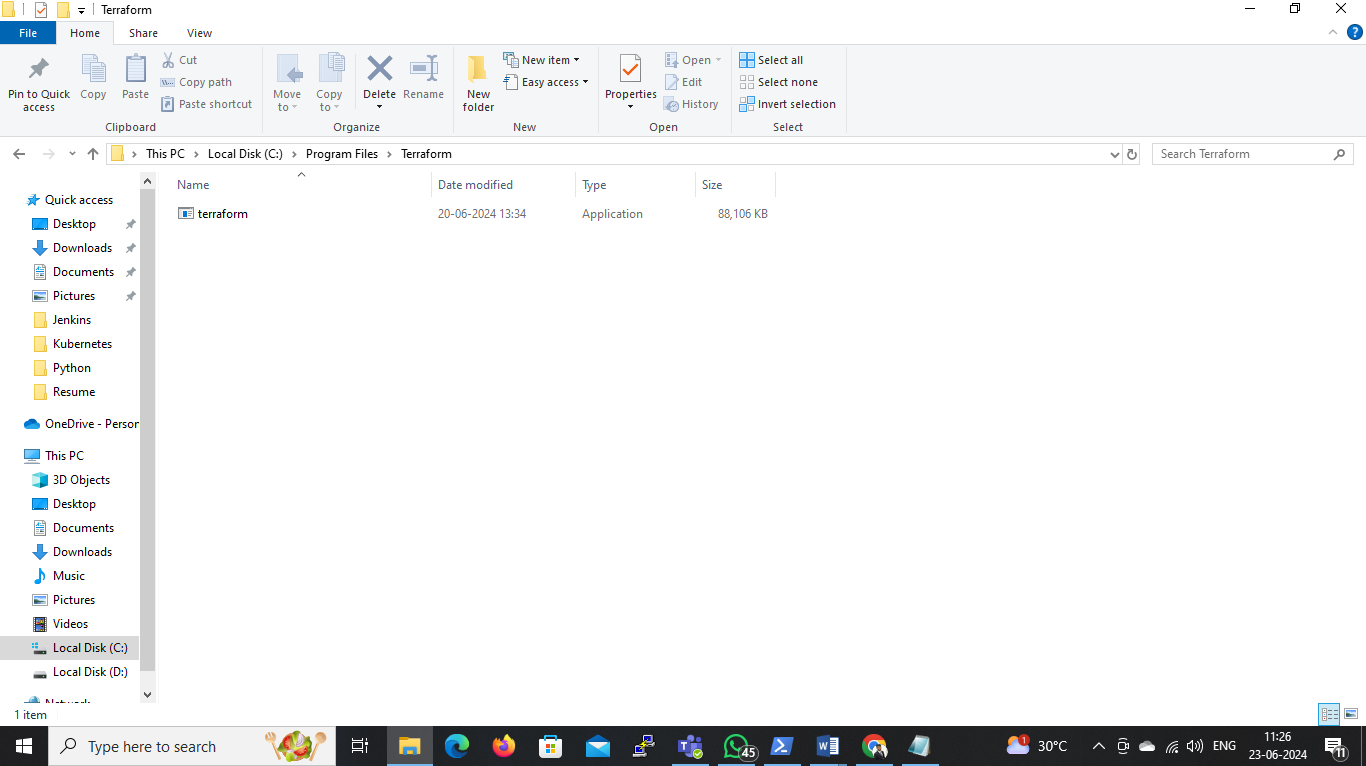
* Go to the Terraform download page.
* Download the appropriate package for your operating system.



1. **Install Terraform:**

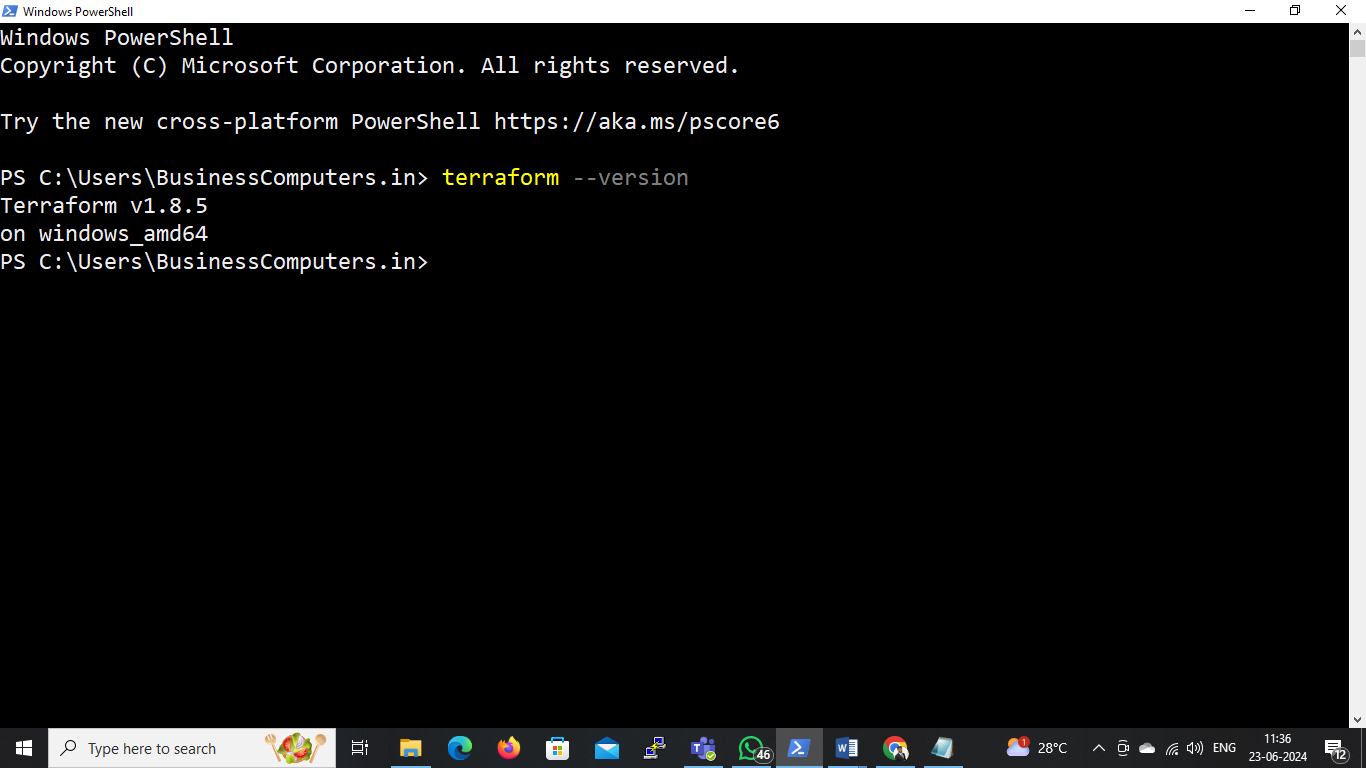
* Unzip the downloaded file.
* Move the terraform.exe file to a directory included in our system's PATH. Typically, we can place it in **C:\Program Files\Terraform**
* Search system variables in laptop & click on edit syetem variables.
* Open a window which is shiown below.
* a. Click on Envirnonment 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**





1. **Verify Installation:**

* Open a command prompt and type:

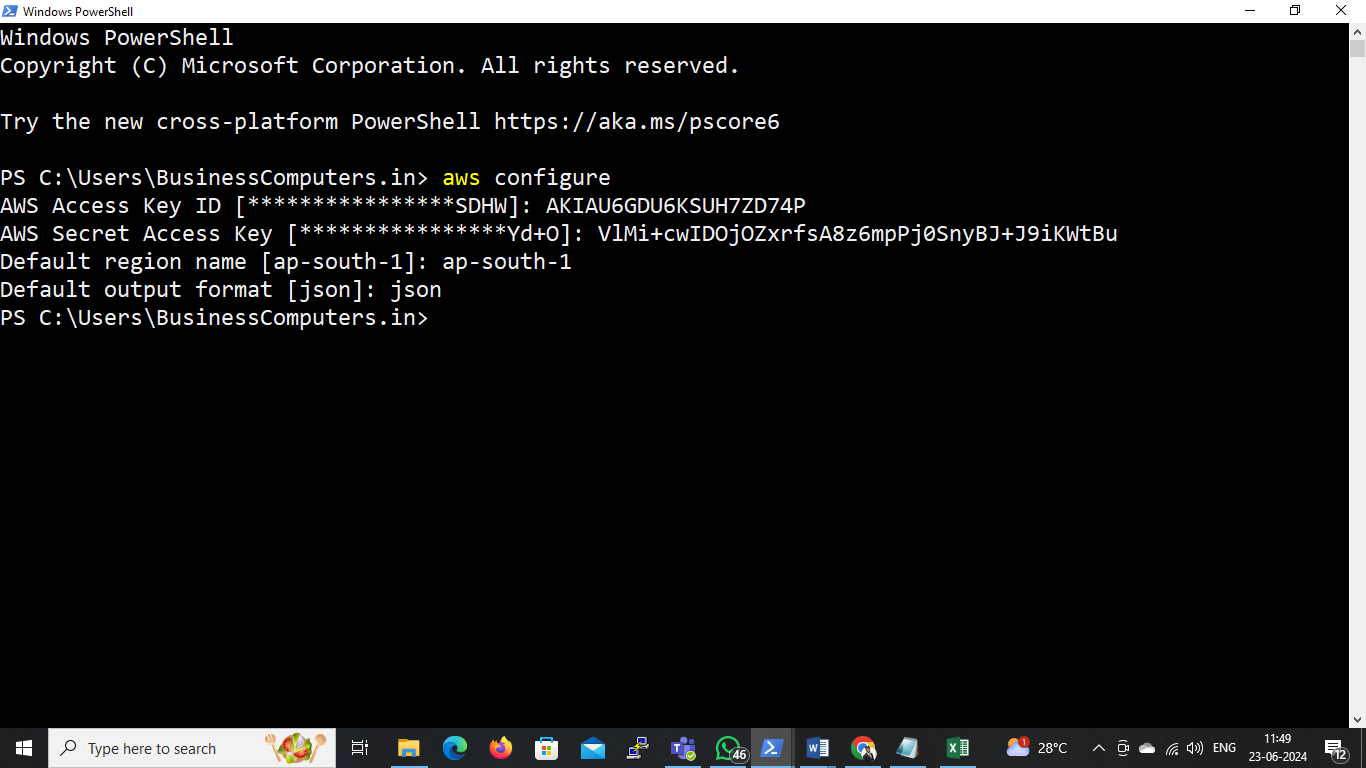


**Step 2: Set Up AWS CLI**

* 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).



**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.

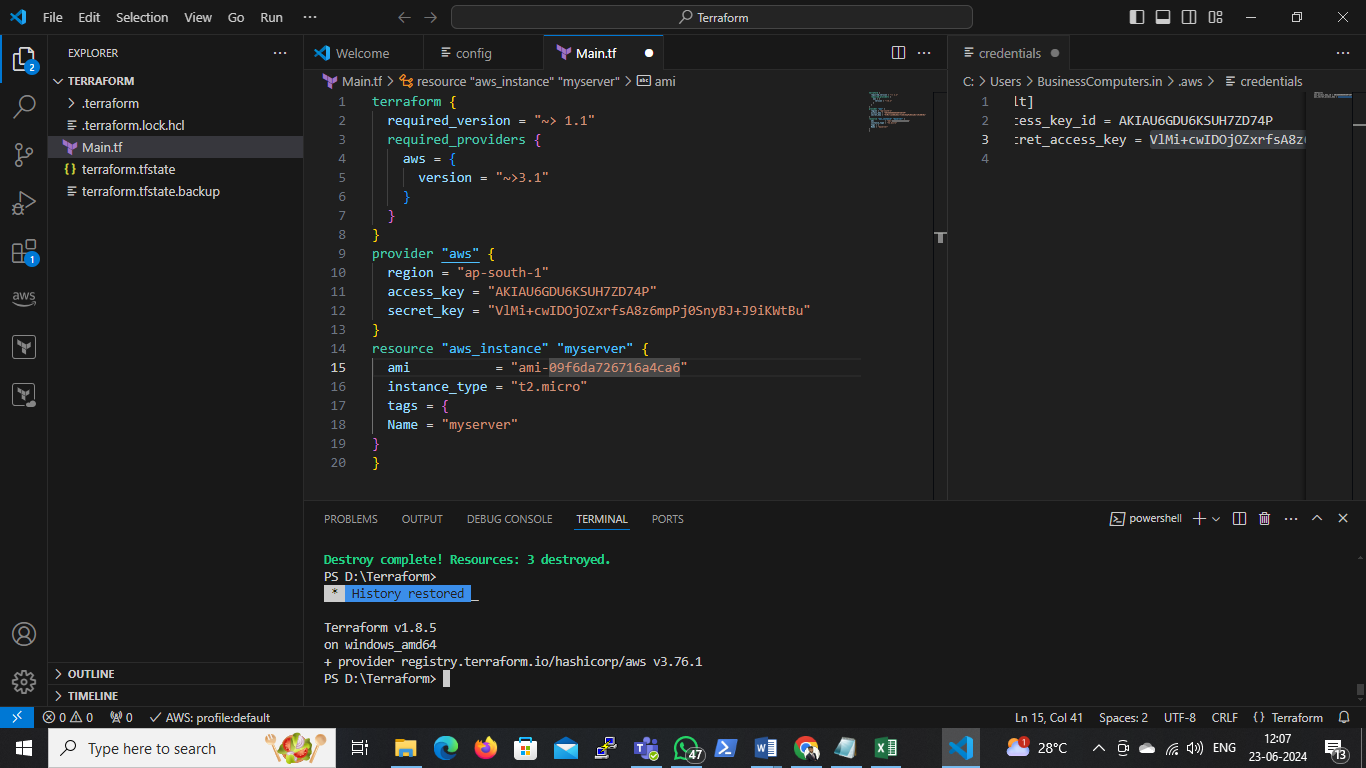
**Step 4: Create Terraform Configuration in VS Code**

**Open a New Folder:**

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

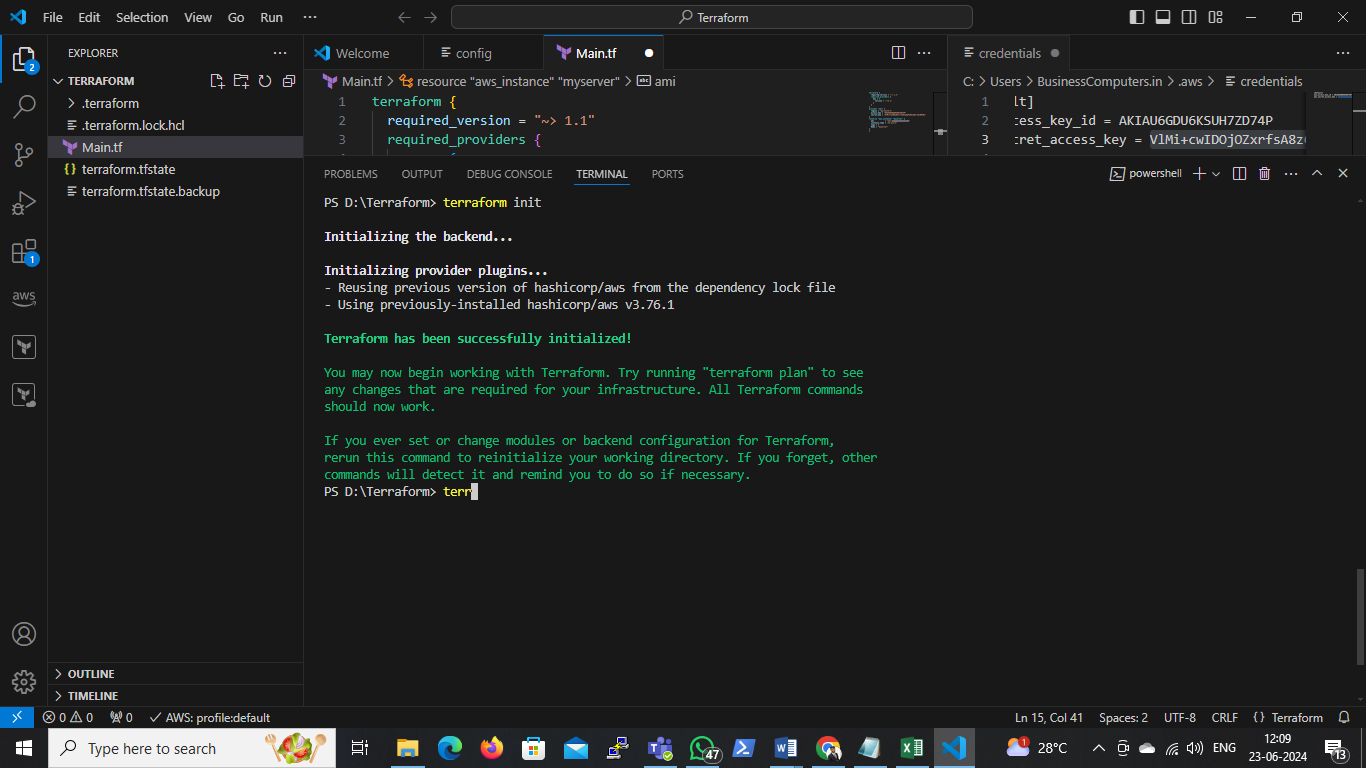
**Create “ main.tf ” File:**

* 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.

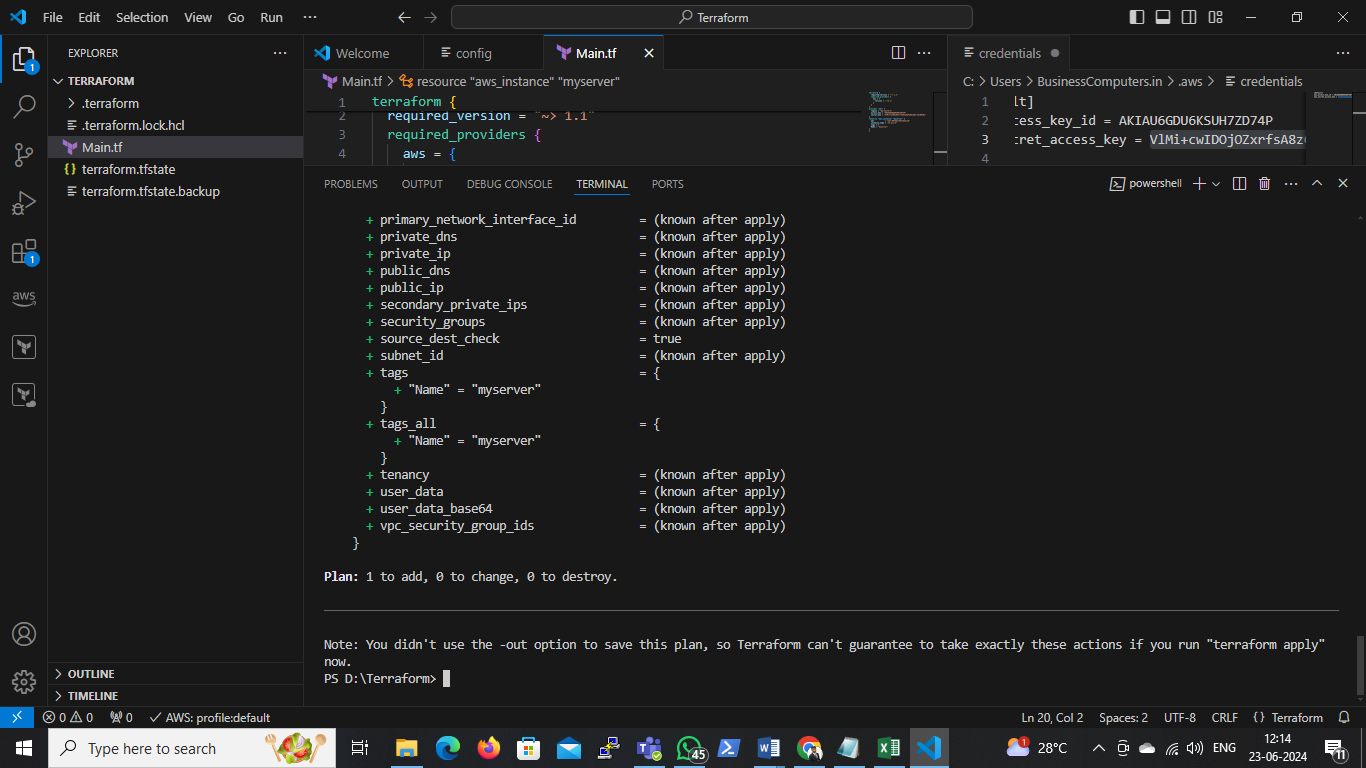


**Open the integrated terminal in VS Code & Run the following command**

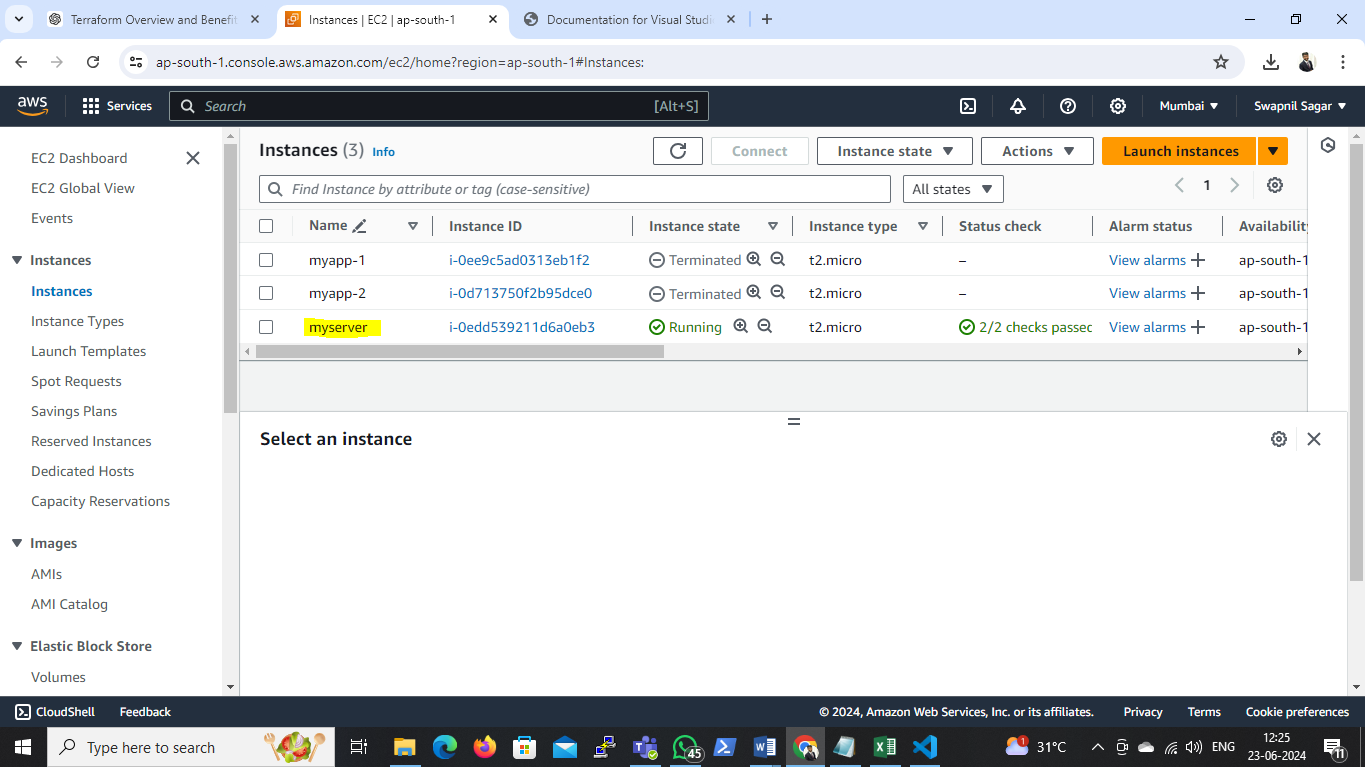
* terraform init



* terraform plan



* terraform apply



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