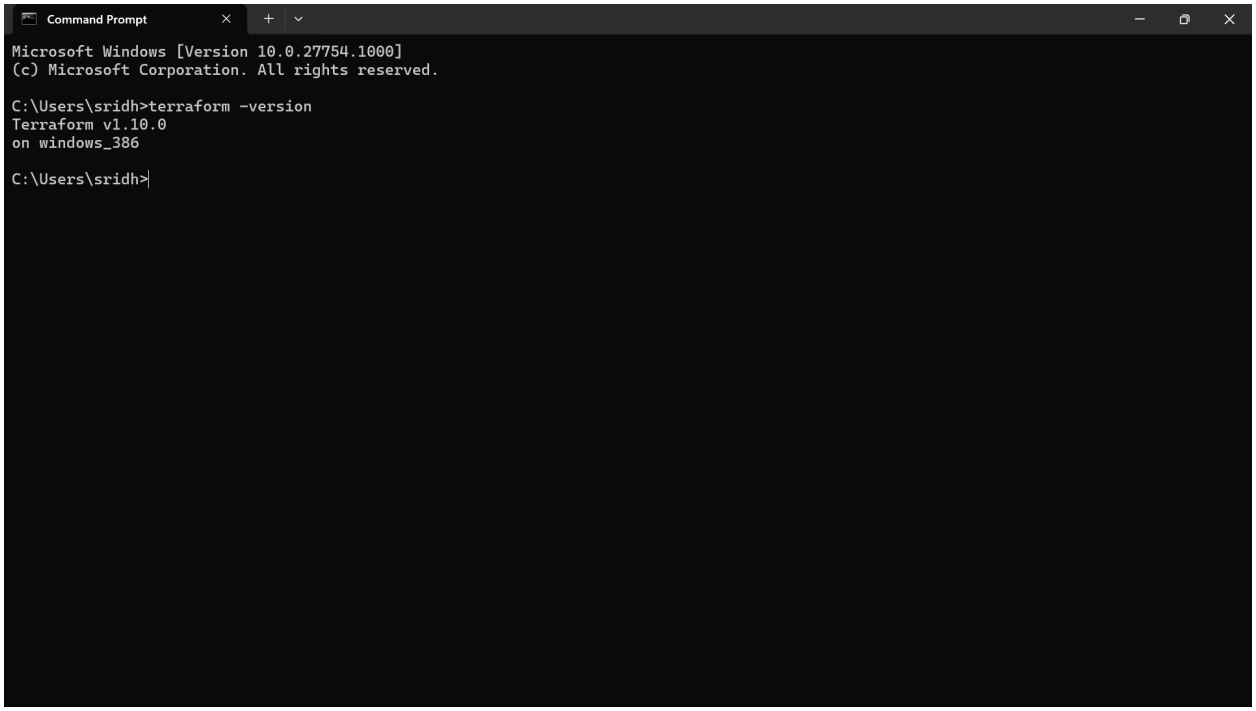
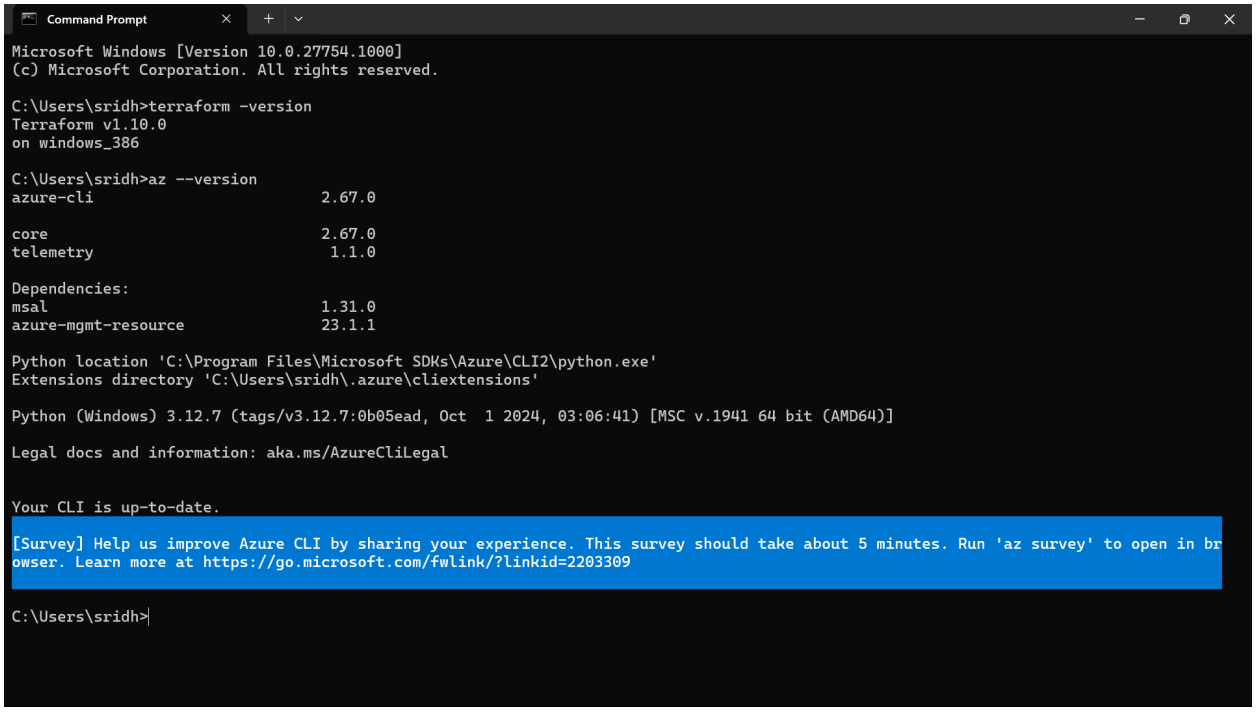
Install Terraform from the link (https://developer.hashicorp.com/terraform/install)

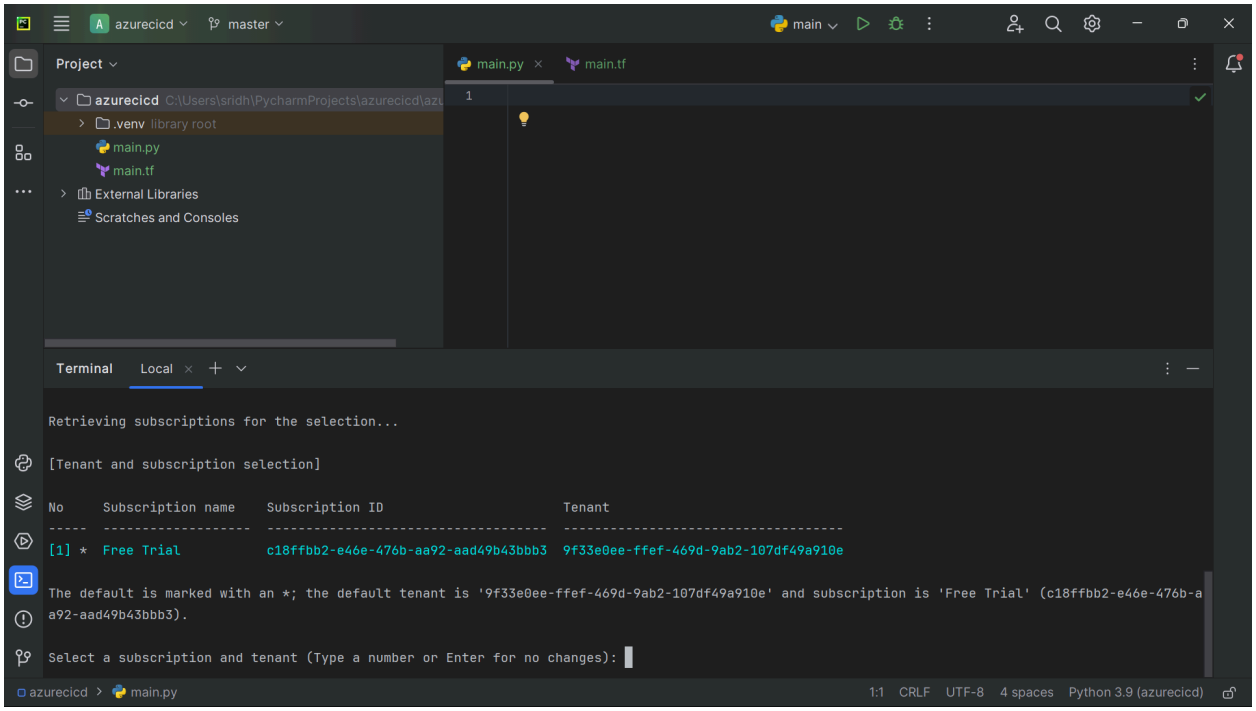


Install Azure CLI (https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?tabs=azure-cli#install-or-update)



I created a new project in the PyCharm IDE, Then I linked Azure CLI to the existing azure subscription (free trial), I did this by running the below commands in terminal

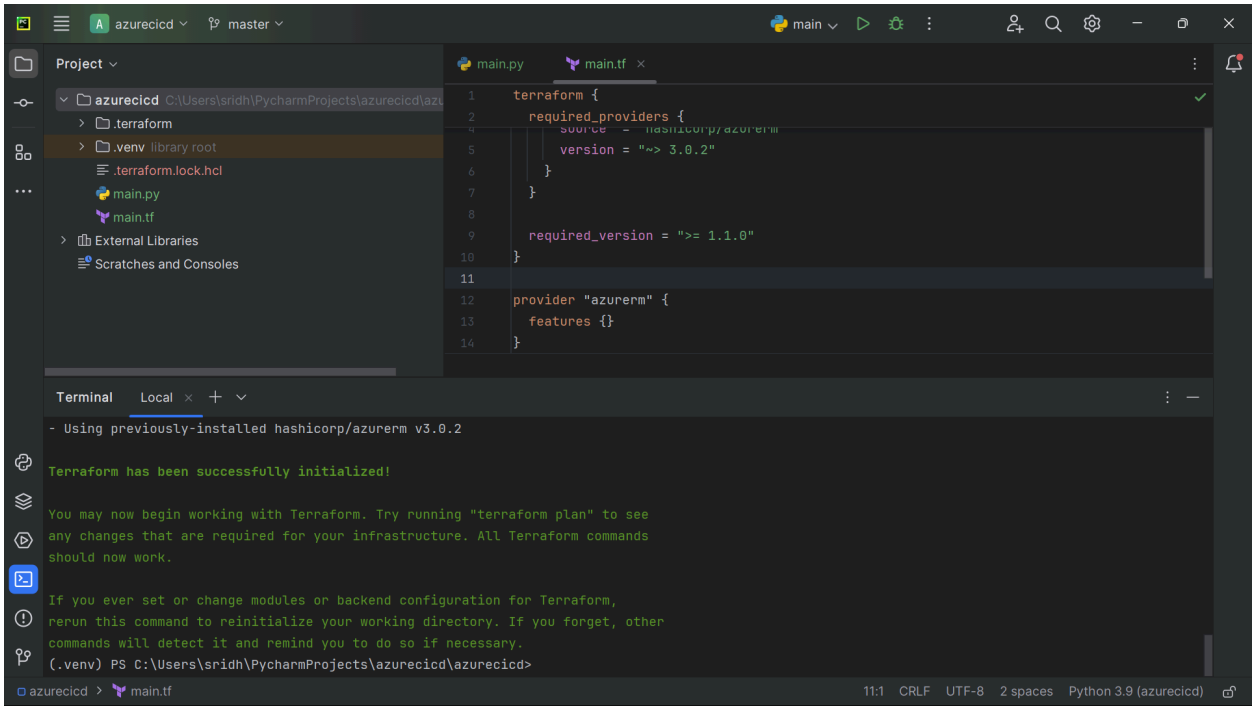
az login tenant <TENANT_ID>



Once I run terraform init by executing the below code

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0.2"
    }
  }

  required_version = ">= 1.1.0"
}

provider "azurerm" {
  features {}
}
```

I can see the below screen. “Terraform init” gets the image of terraform and installs it into the system.



Code Explanation:

required_providers:

- This block specifies the providers Terraform configuration requires. Providers are plugins that allow Terraform to interact with cloud services like Azure, AWS, or GCP.
- Here, the provider is azurerm, which is the Azure Resource Manager provider.
- **source**:
  - Specifies the namespace (hashicorp) and the name (azurerm) of the provider.
  - It ensures Terraform downloads the correct provider plugin.
- **version**:
  - Specifies the version constraint for the provider.
  - ~> 3.0.2 means any version from 3.0.2 to 3.1.x (but not 4.x.x).
  - This ensures compatibility with your configuration.

required_version:

- Specifies the minimum version of Terraform required to execute this configuration.
- **>= 1.1.0** ensures that any Terraform version starting from 1.1.0 or higher is used.

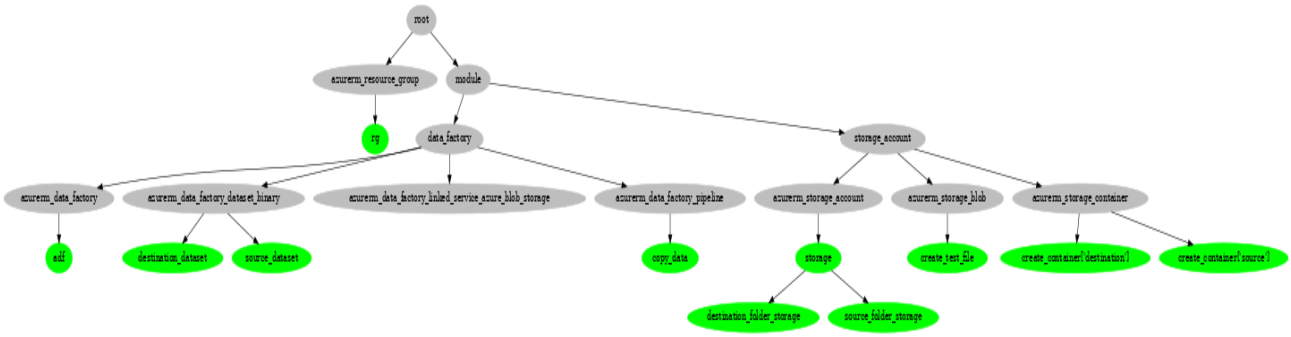
provider "azurerm":

- Declares the Azure Resource Manager provider for Terraform. This is essential to interact with Azure resources.

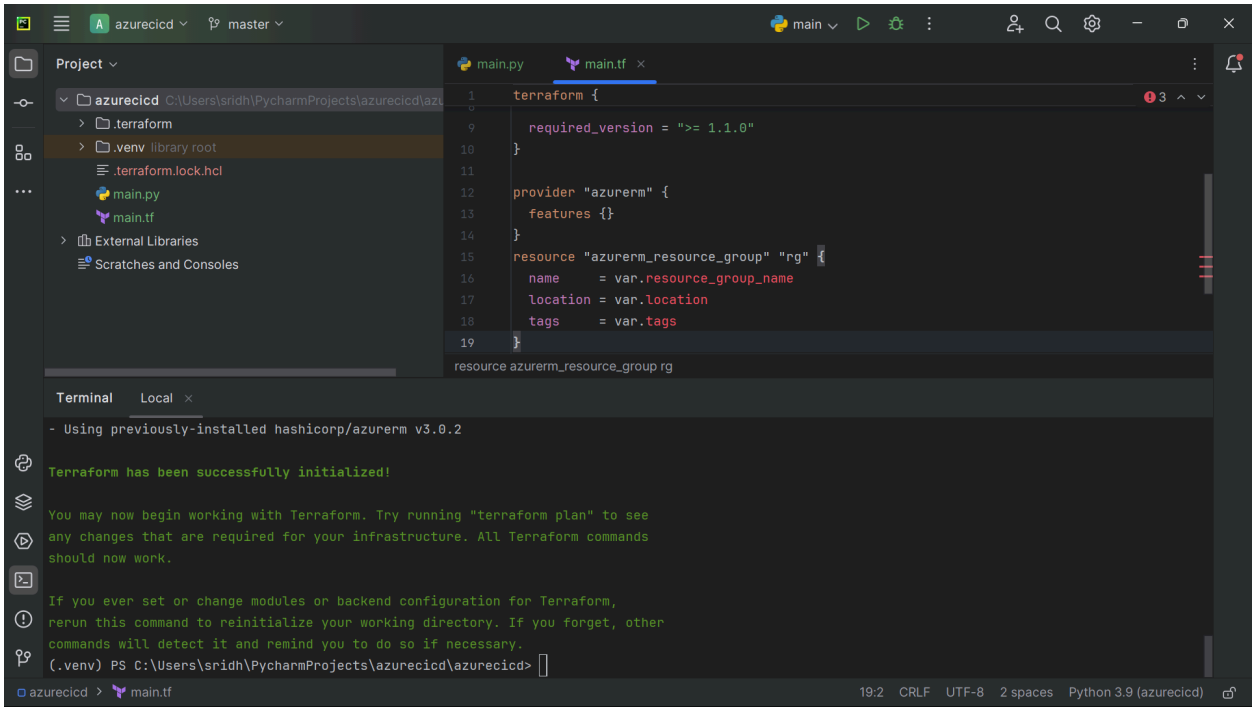
features {}:

- This is a placeholder for Azure-specific configuration settings.
- Since Terraform v2.0.0 of azurerm, this block is **mandatory**, even if it's empty.
- It's used to configure optional provider features like **resource graph** or **advanced networking features**.

System Architecture:



From the system architecture, I have a resource group and module. I will start with the resource group

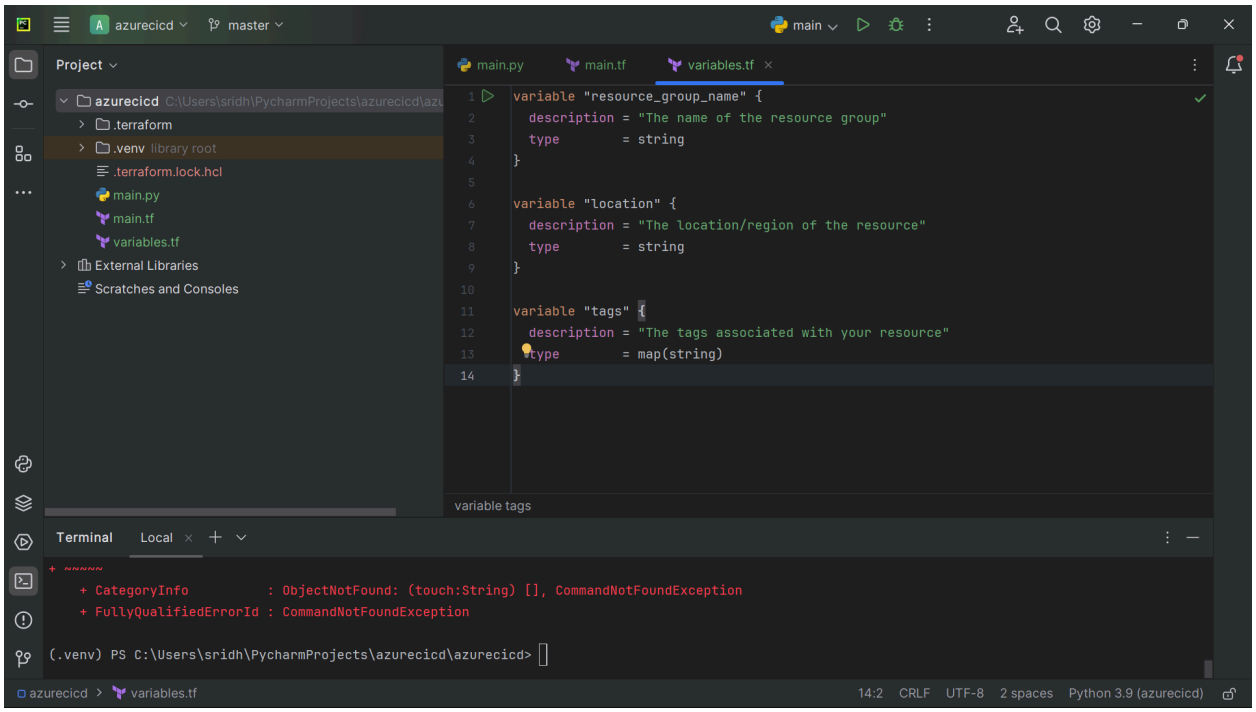


I am still seeing the errors because I don't have any variables installed. So, I created another file variables.tf as below

Code:

```
variable "resource_group_name" {
 description = "The name of the resource group"
```

```
 type        = string
}

variable "location" {
 description = "The location/region of the resource"
 type        = string
}

variable "tags" {
 description = "The tags associated with your resource"
 type        = map(string)
}
```



Code Explanation:

The code snippet defines three **input variables** in a Terraform configuration file. These variables allow you to parameterize your infrastructure configuration, making it reusable and flexible. Here's an explanation of each variable block:

1. Variable: resource_group_name

```
variable "resource_group_name" {
  description = "The name of the resource group"
  type        = string
}
```

Purpose: Defines a variable named resource_group_name to store the name of a resource group.

Key Attributes:

- description: Provides a description of what this variable represents (useful for documentation or when running terraform plan).
- type: Specifies that this variable must be a string.

Usage: This variable allows users to pass the name of the resource group as a parameter when executing the Terraform configuration.

2. Variable: location

```
variable "location" {
  description = "The location/region of the resource"
  type        = string
}
```

Purpose: Specifies the location/region where resources will be deployed.
Key Attributes:

- description: Clarifies that this variable determines the geographical region for deployment (e.g., "East US" or "West Europe").
- type: Specifies that the value for this variable must also be a string.

3. Variable: tags

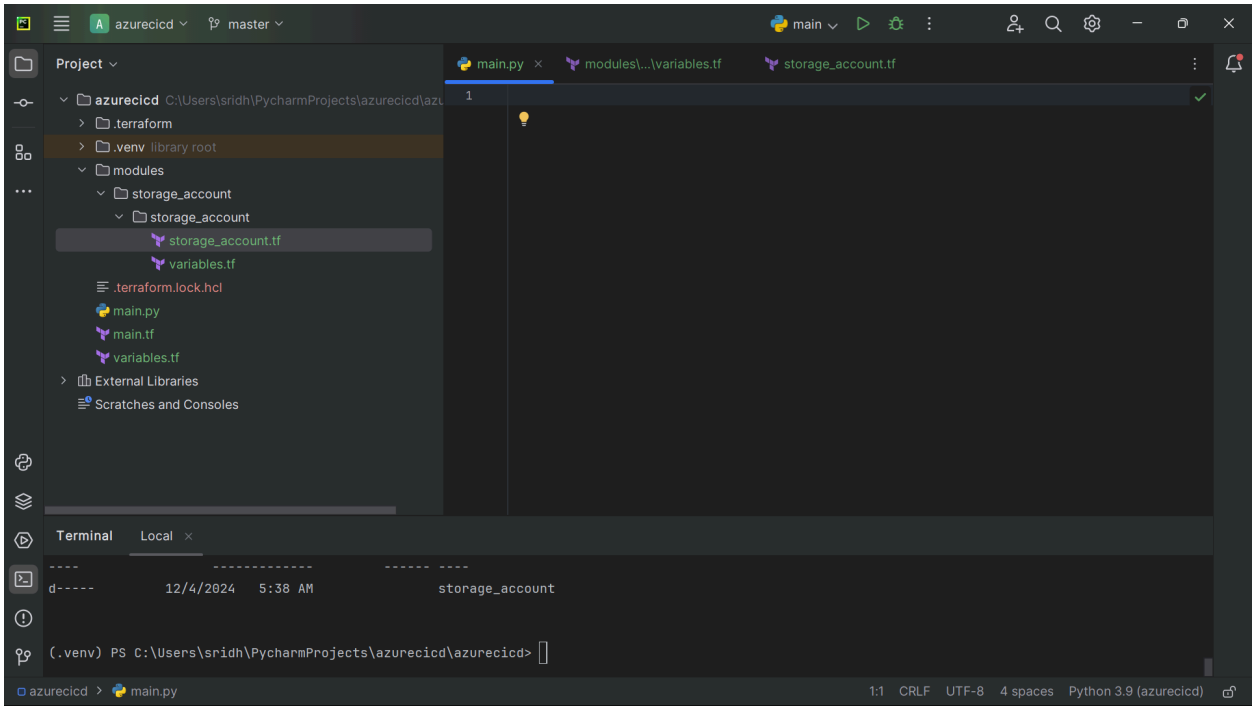
```
variable "tags" {
  description = "The tags associated with your resource"
  type        = map(string)
}
```

Purpose: Allows the definition of tags (key-value pairs) to associate metadata with resources.
Key Attributes:

- description: Explains that this variable will contain tags for resources, typically used for categorization, cost tracking, or organization.
- type: Specifies that the value should be a **map** with keys and values as strings

For the module, I have a data factory and storage account. I will start with storage account as the data factory is having dependency on storage account

I started by creating a directory for storage account in the PyCharm IDE and I created two terraform files namely storage_account.tf and variables.tf as below



From the system architecture, I have storage account, storage blob and storage container. To get these 3 into the system, I used the below code

For storage.tf file

```
resource "azurerm_storage_account" "storage" {

name = var.storage_account_name

resource_group_name = var.resource_group_name

location = var. location

account_tier = "Standard"

account_replication_type = "LRS"

tags = {

 environment = "development"

}

}

resource "azurerm_storage_container" "create_container" {

 for_each = {
```

```
   source      = var.source_folder_name,
   destination = var.destination_folder_name
 }


 name                  = each.key
 storage_account_name  = azurerm_storage_account.storage.name
 container_access_type = var.container_access_type
}


resource "azurerm_storage_blob" "create_test_file" {
 name                   = "test.txt"
 storage_account_name   = azurerm_storage_account.storage.name
 storage_container_name =
azurerm_storage_container.create_container["source"].name
 type                   = "Block"
 source_content         = "Hello Word!"
}
output "storage_account_key" {
 value = azurerm_storage_account.storage.primary_access_key
}
```

Code Explanation:

1. Azure Storage account

```
resource "azurerm_storage_account" "storage" {
  name                     = var.storage_account_name
  resource_group_name      = var.resource_group_name
  location                 = var.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
```

```
  tags = {
    environment = "development"
  }
}
```

Explanation:

- Creates an Azure Storage Account.
- **`name`**: Uses the value from the `var.storage_account_name` variable to set the account name.
- **`resource_group_name`**: Specifies the Azure Resource Group in which the storage account will be created, sourced from `var.resource_group_name`.
- **`location`**: Specifies the Azure region for the storage account, sourced from `var.location`.
- **`account_tier`**: Defines the storage performance tier, here set as `"Standard"`.
- **`account_replication_type`**: Specifies the replication type. `"LRS"` (Locally Redundant Storage) keeps multiple copies of your data within a single data center.
- **`tags`**: Adds metadata tags for organization and categorization (e.g., the environment is `"development"`).

2. Azure Storage Containers

```
resource "azurerm_storage_container" "create_container" {
  for_each = {
    source      = var.source_folder_name,
    destination = var.destination_folder_name
  }

  name                  = each.key
  storage_account_name  = azurerm_storage_account.storage.name
  container_access_type = var.container_access_type
}
```

Explanation:

- **Creates multiple storage containers** dynamically using the `for_each` loop.
- **`for_each`**: Loops through a map where keys are container names (`source` and `destination` from the variable map).
- **`name`**: Each container's name is derived from the key in the map.
- **`storage_account_name`**: Links each container to the storage account created earlier (`azurerm_storage_account.storage.name`).

- **`container_access_type`**: Sets the access level for the container, defined in the `var.container_access_type` variable (e.g., `"private"` or `"blob"`).

3. Azure Storage Blob

```
resource "azurerm_storage_blob" "create_test_file" {
  name                   = "test.txt"
  storage_account_name   = azurerm_storage_account.storage.name
  storage_container_name = azurerm_storage_container.create_container["source"].name
  type                   = "Block"
  source_content         = "Hello Word!"
}
```

Explanation:

- Creates a **test file blob** (`test.txt`) inside the `"source"` storage container.
- **`name`**: Specifies the blob name, `"test.txt"`.
- **`storage_account_name`**: Links the blob to the storage account.
- **`storage_container_name`**: Specifies the container where the blob will be stored, linked dynamically to the `"source"` container from the earlier loop.
- **`type`**: Defines the blob type. `"Block"` blobs are optimized for streaming and storage of large data.
- **`source_content`**: Provides the file content, here a simple text saying `"Hello Word!"`.

4. Output the storage account key

```
output "storage_account_key" {
  value = azurerm_storage_account.storage.primary_access_key
}
```

Explanation:

- Outputs the primary access key for the storage account (`azurerm_storage_account.storage.primary_access_key`).
- This key can be used for authentication when accessing the storage account programmatically or via APIs.

To link them to the root module, I used the below code in main.tf

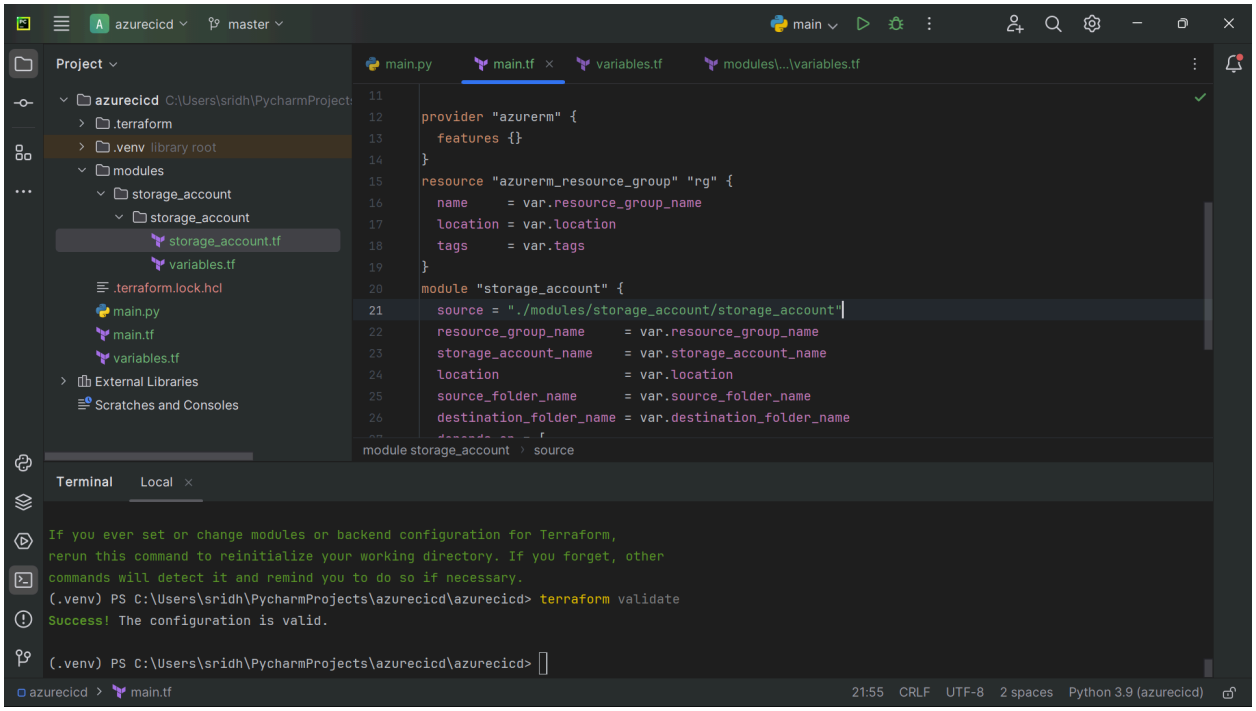
```
module "storage_account" {

 source = "./modules/storage_account/storage_account"

 resource_group_name     = var.resource_group_name

 storage_account_name    = var.storage_account_name

 location                = var.location

 source_folder_name      = var.source_folder_name

 destination_folder_name = var.destination_folder_name

 depends_on = [

   azurerm_resource_group.rg

 ]

}
```

Explanation:

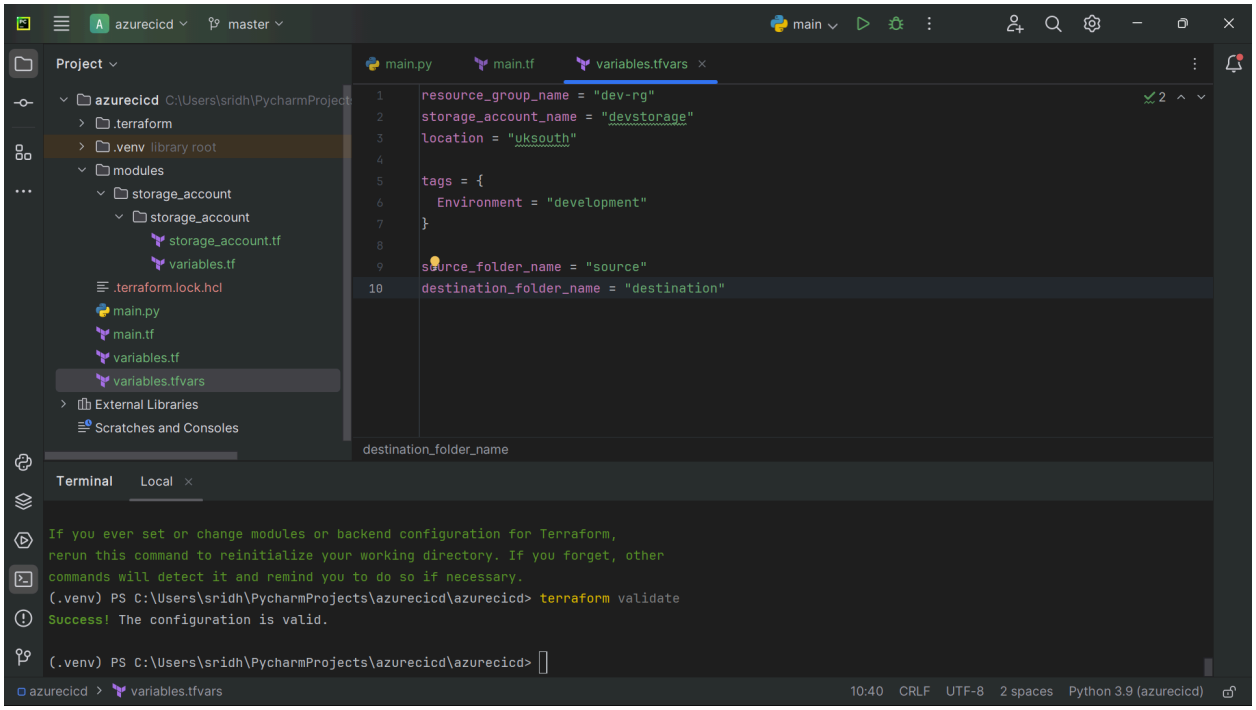
1. **module Block**:
    - The module block defines a reusable module that encapsulates the logic for creating the Azure Storage Account and related resources like containers and blobs.
    - **source**: Specifies the file path (./modules/storage_account/storage_account) where the module's code is located.
2. **Input Variables**:
    - The module is parameterized with variables to make it reusable and flexible for different configurations. These variables are passed to the module:
        - **resource_group_name**: The name of the Azure Resource Group where the storage account will reside.
        - **storage_account_name**: The name of the Azure Storage Account to be created.
        - **location**: The Azure region where the resources will be deployed.
        - **source_folder_name**: The name of the source folder (likely used in creating storage containers).
        - **destination_folder_name**: The name of the destination folder (likely used in creating storage containers).
3. **Dependency Management**:
    - **depends_on**: Ensures that the module execution waits until the azurerm_resource_group.rg resource (Azure Resource Group) has been fully

created. This is crucial for maintaining a valid dependency chain, as the storage account must be created within a resource group that already exists.

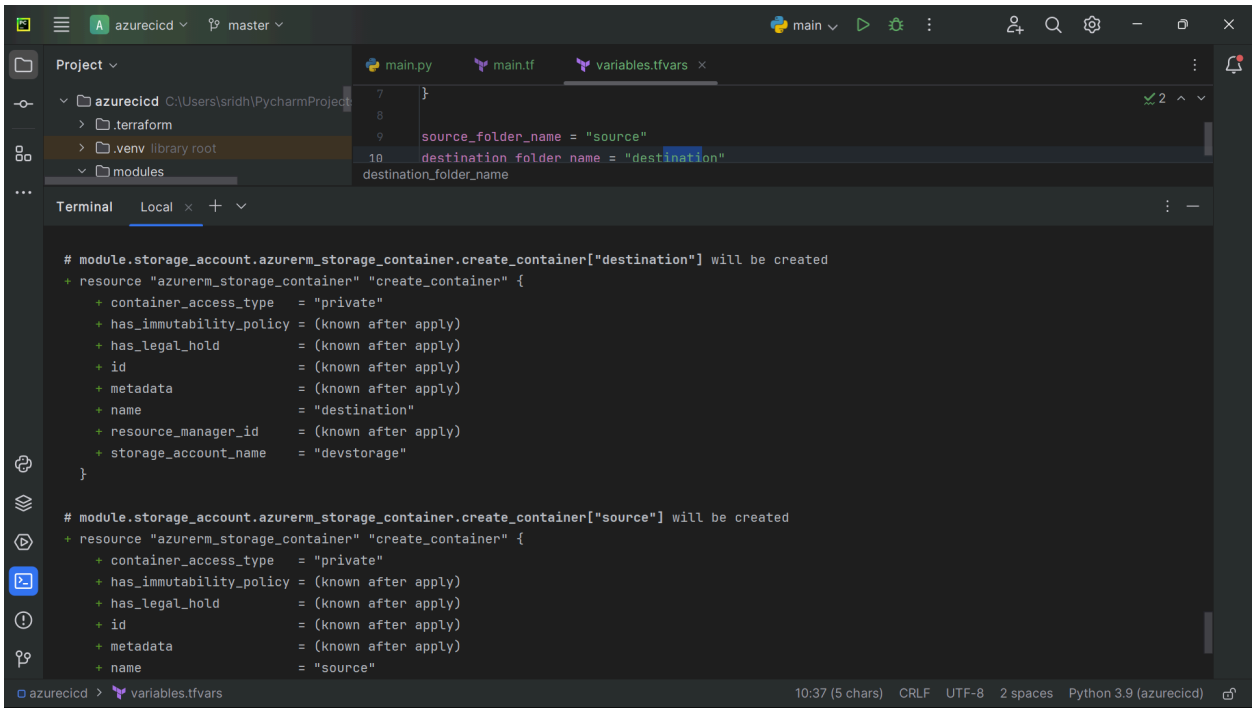
Then i initialized the terraform using “terraform init” and validated the terraform scripts using “terraform validate”, I can see the below screen

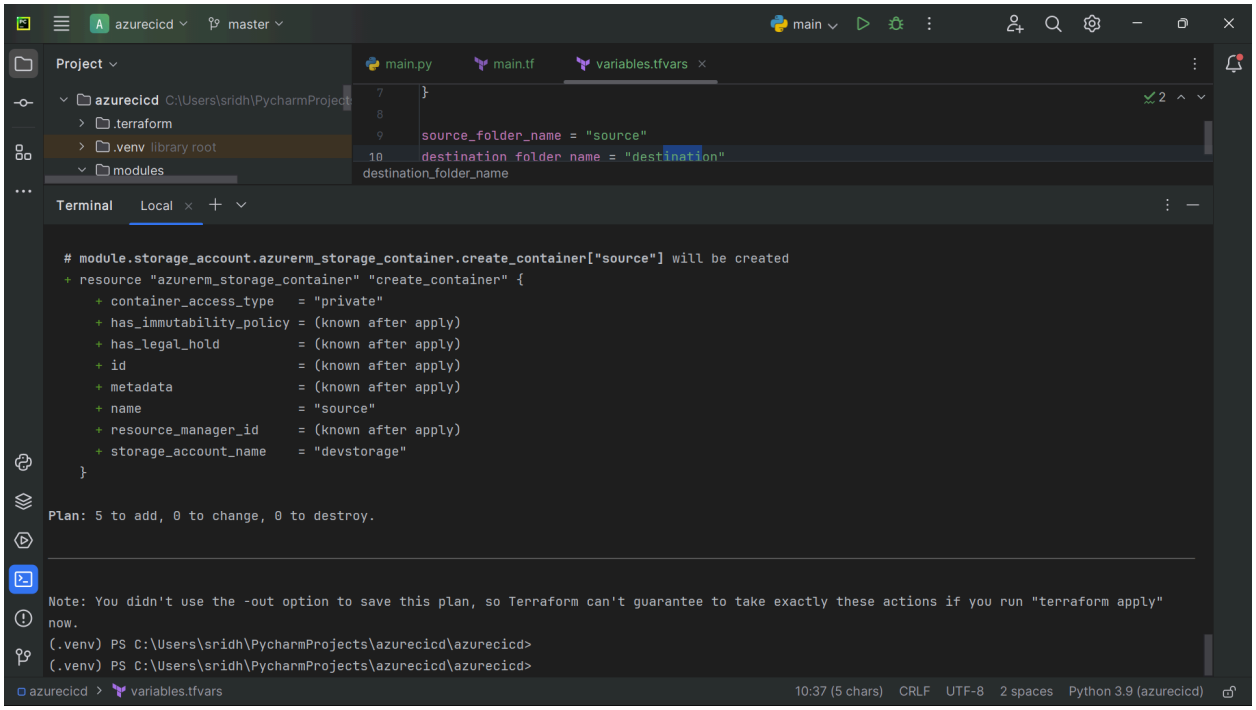


Before I perform terraform plan, I need the variables files which contains all the variables in key-value pairs, So I created another file with the name variables.tfvars with the below variables



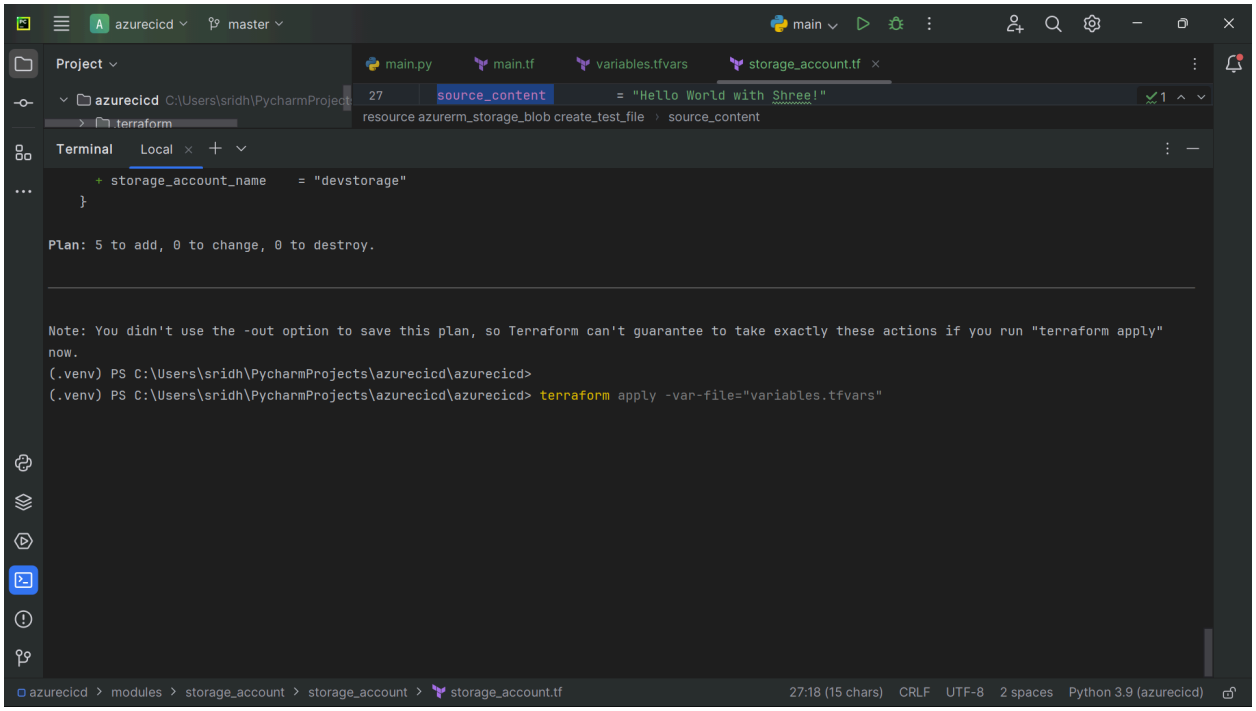
I executed “ terraform plan -var-file="variables.tfvars" ” in the terminal and I can see that resources are created



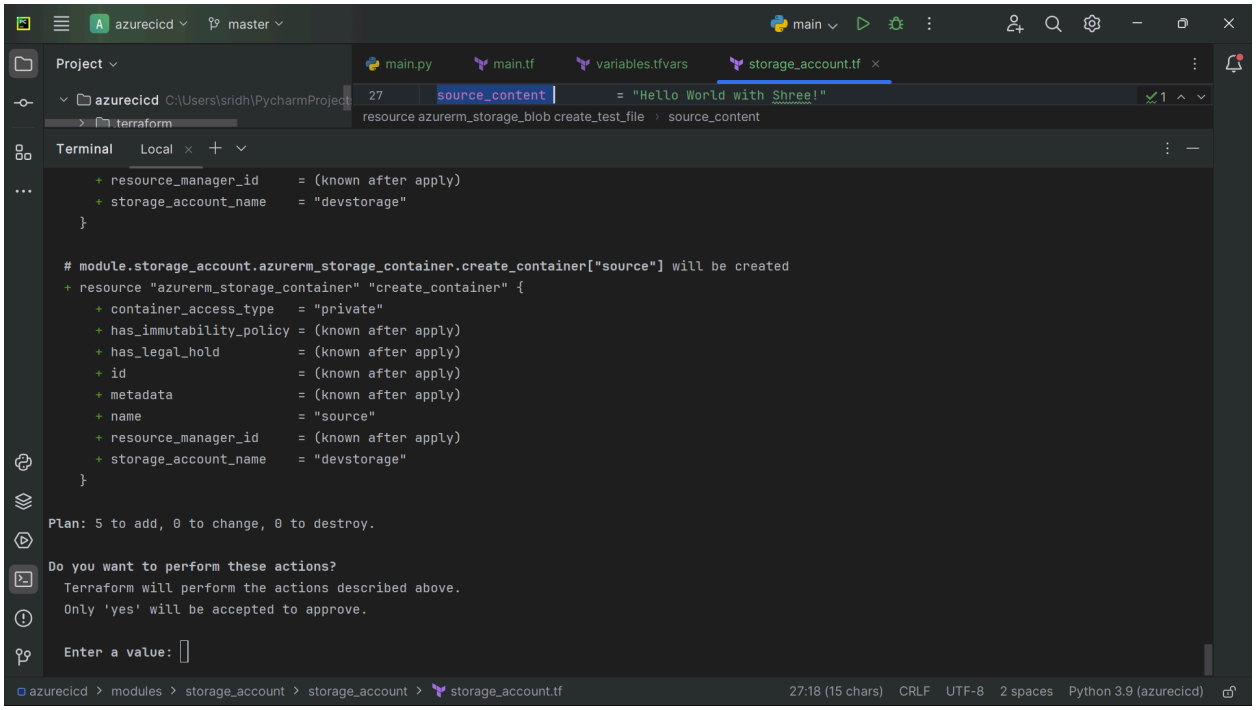


From the output, I can see that 5 resources are going to be added.

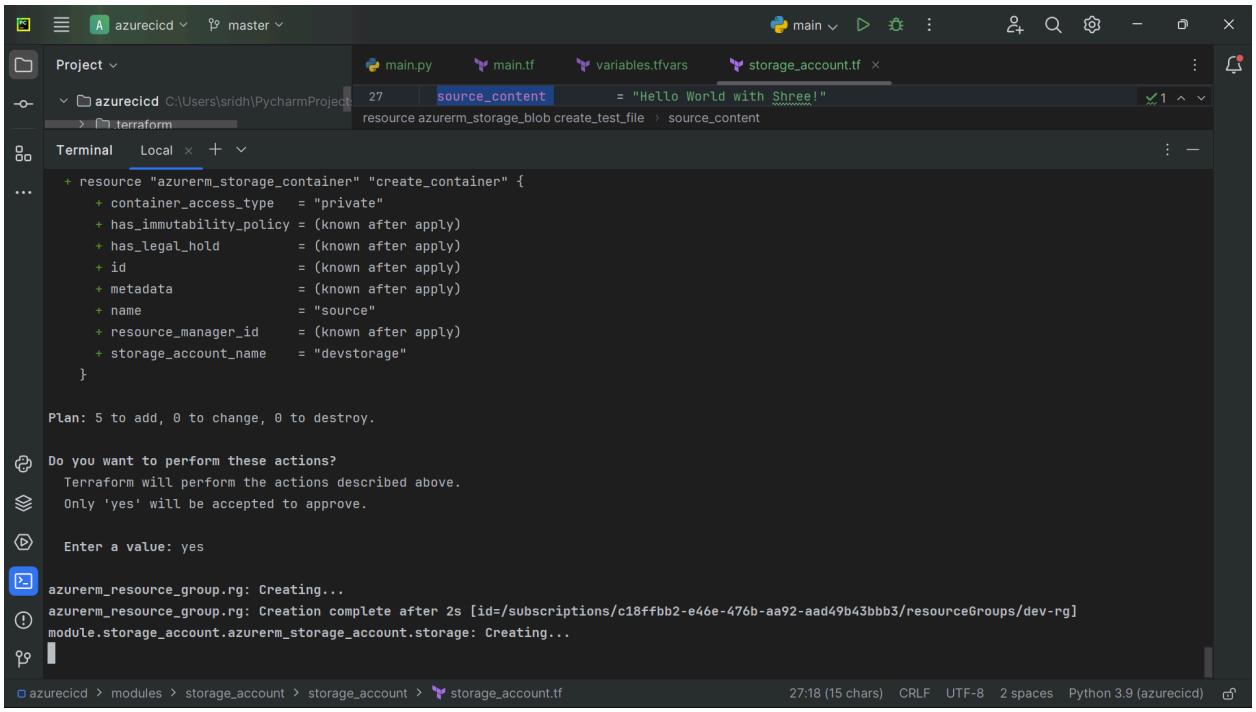
Now, If I change the "source_content" from "Hello world" to "Hello world with shree" and rerun the script using " terraform apply -var-file = variables.tfvars" as below

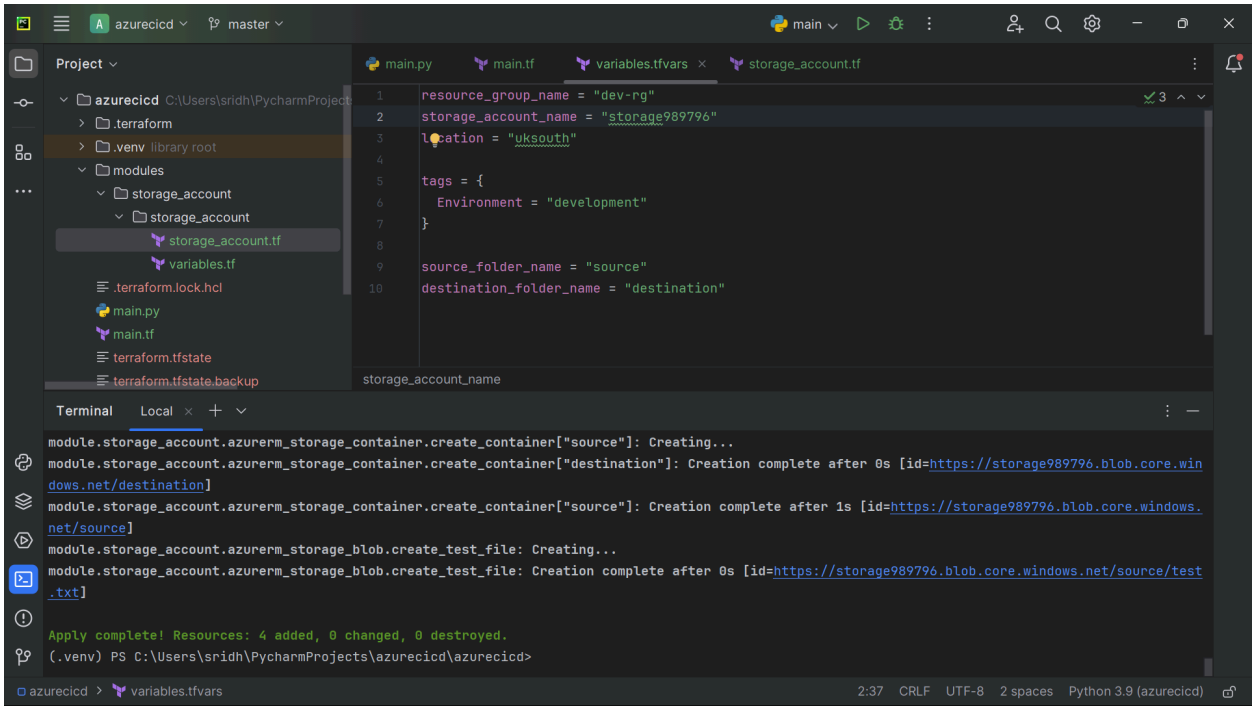


I was prompted to make a choice to create the resources in the subscription linked to azure

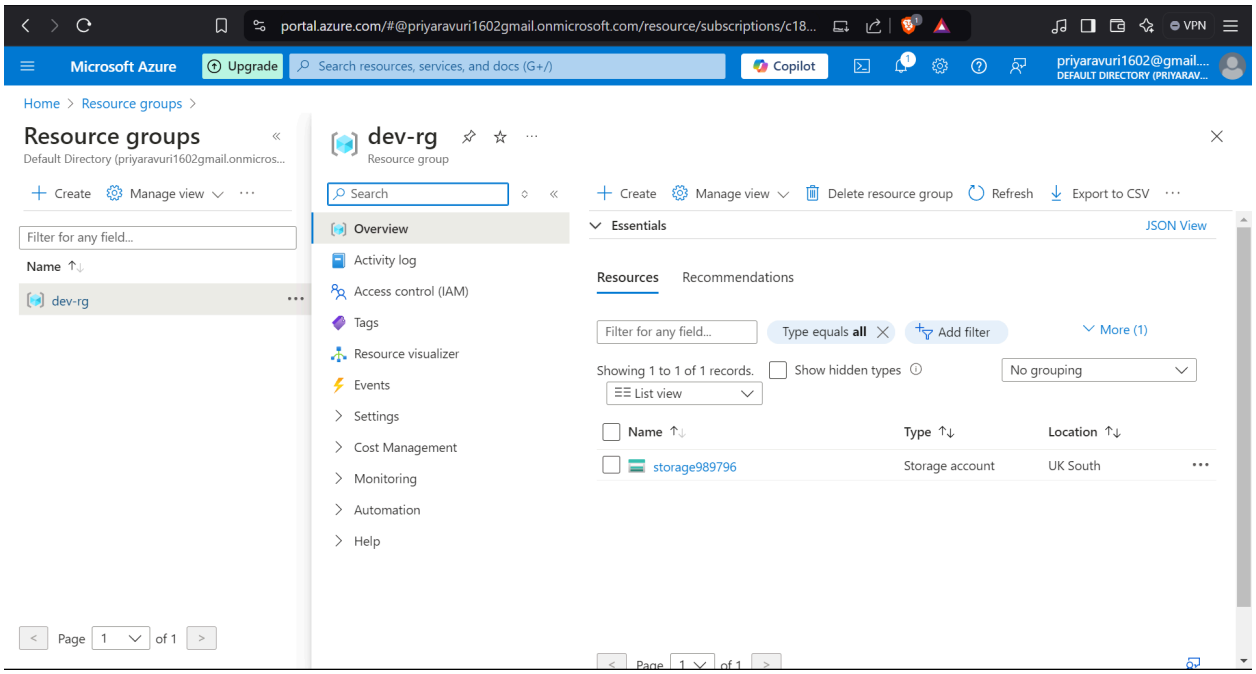


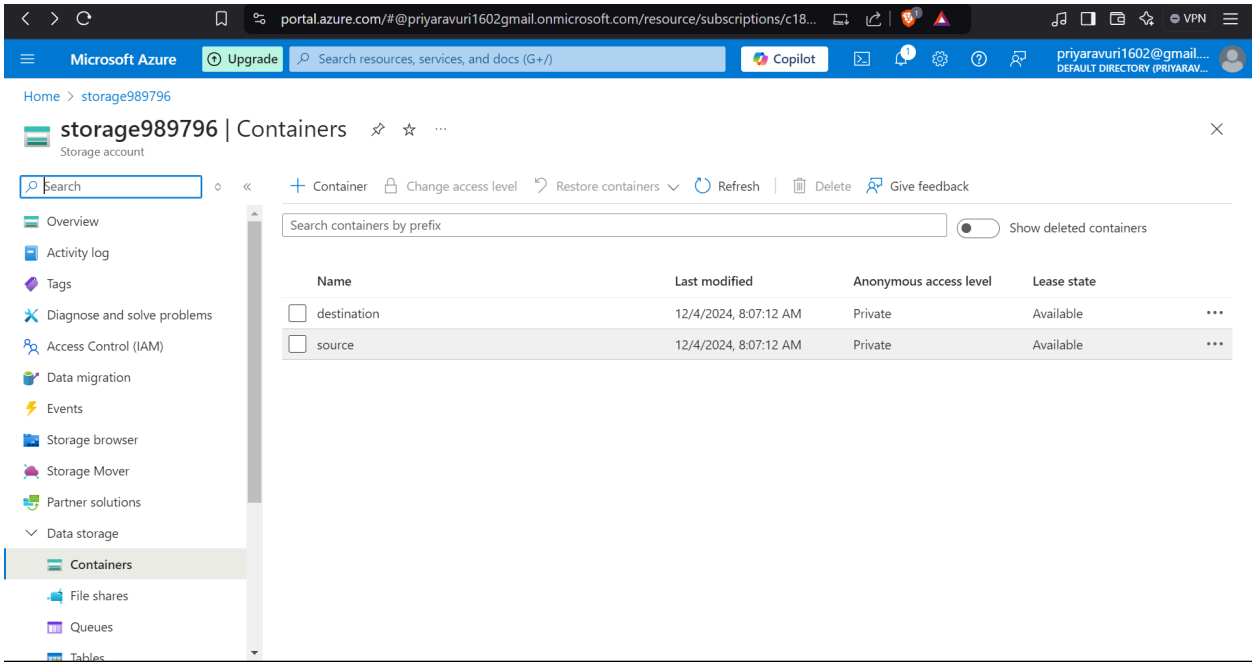
Once I approved, I can see the resources are being provisioned in azure

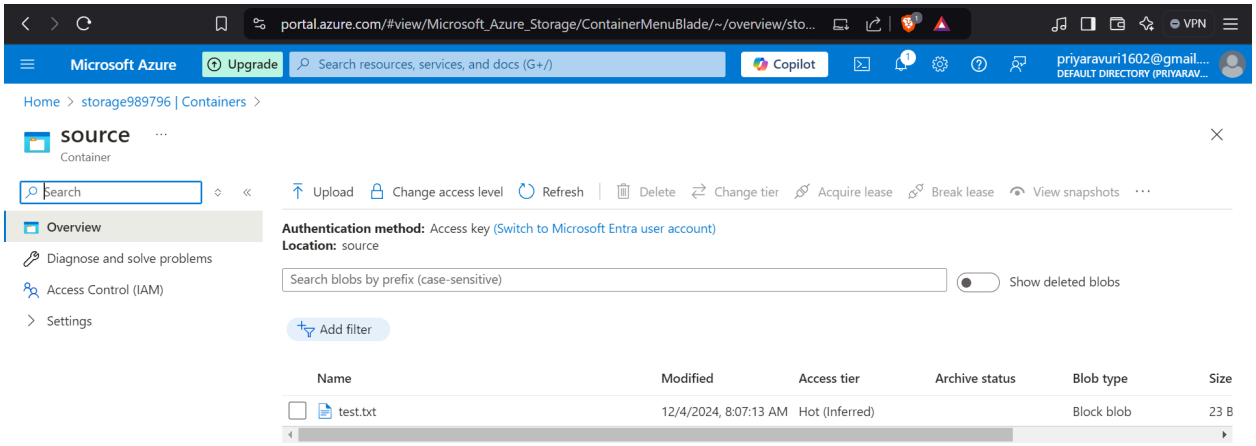




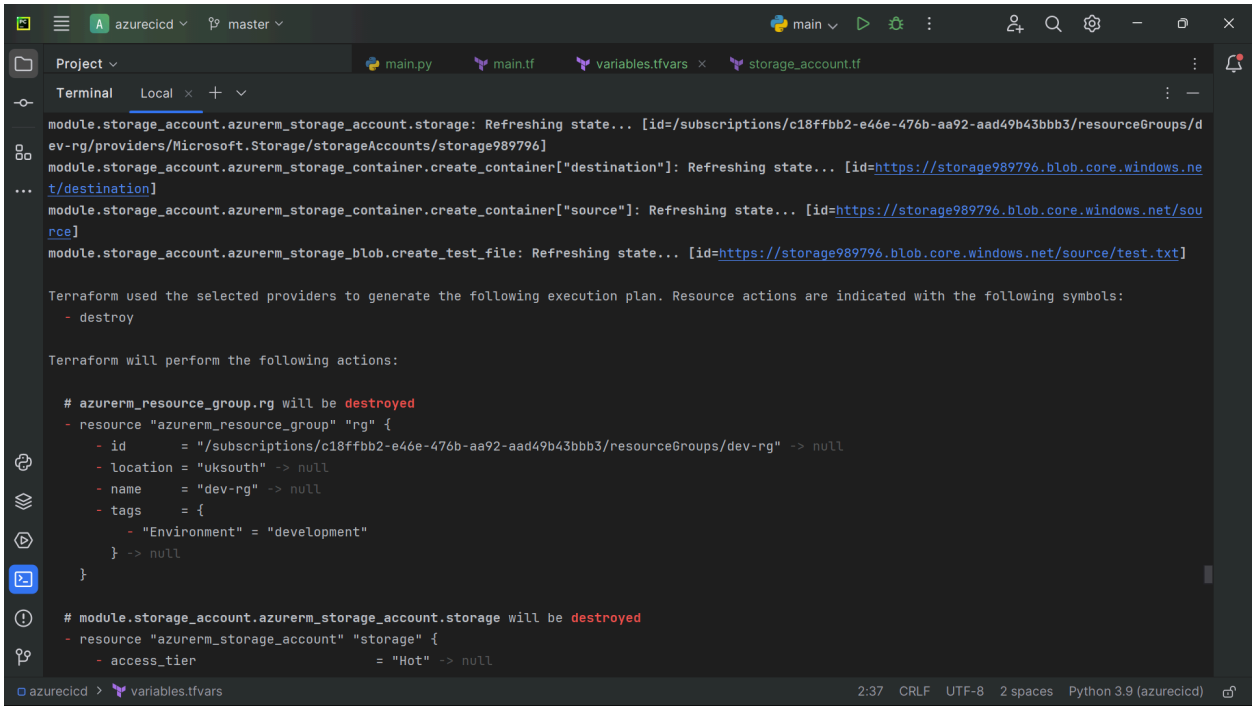
I can also verify the resources from the actual azure portal as below



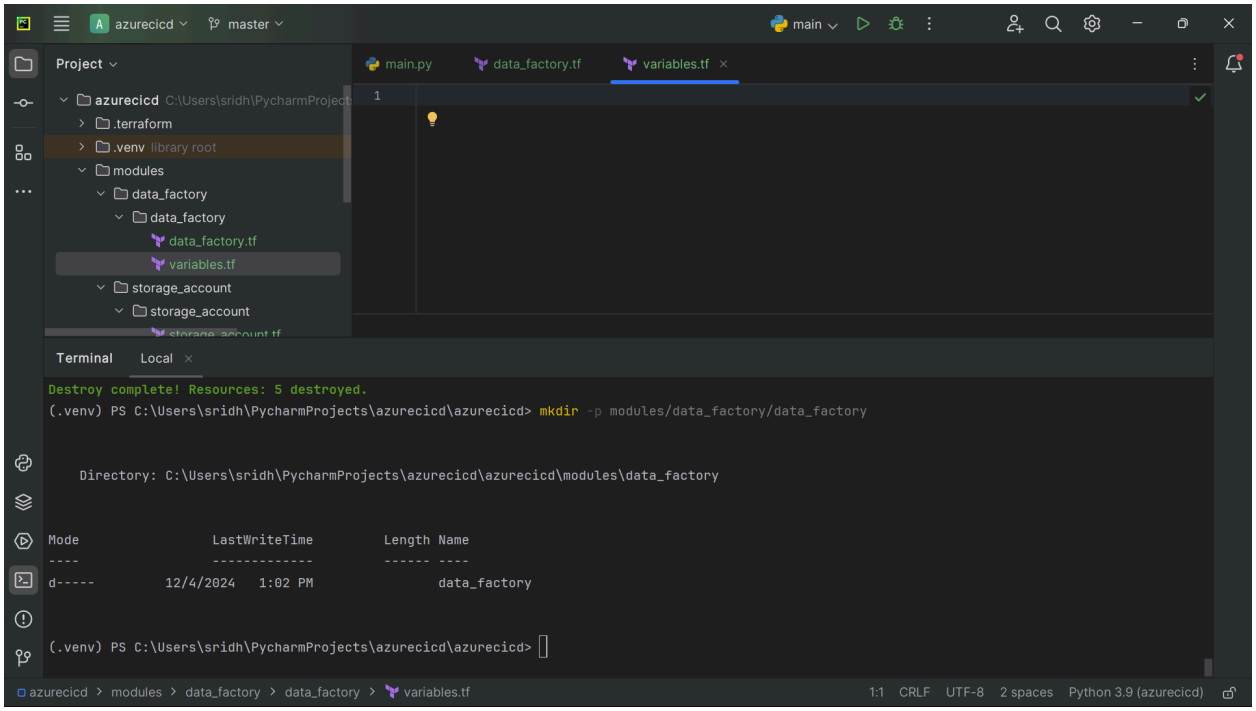




For now, I can destroy the resources as they will be created much later. I did all the above to check if my code is working as expected . I destroyed the resources by using “terraform destroy -var-file="variables.tfvars" “ in the terminal as below



From the system architecture, the next step is to create azure data factory. I created a directory with the name data_factory and two files with the names data_factory.tf and variables.tf as below



I used the below code in data_factory.tf

```
resource "azurerm_data_factory" "adf" {

 name                = var.df_name

 resource_group_name = var.resource_group_name

 location            = var.location



 identity {

   type = "SystemAssigned"

 }

}



data "azurerm_storage_account" "source_folder_storage" {

 name                = var.storage_account_name

 resource_group_name = var.resource_group_name

}



data "azurerm_storage_account" "destination_folder_storage" {

 name                = var.storage_account_name

 resource_group_name = var.resource_group_name

}



resource "azurerm_data_factory_linked_service_azure_blob_storage" "source" {

 name              = "source-storage"

 data_factory_id   = azurerm_data_factory.adf.id

 connection_string =
data.azurerm_storage_account.source_folder_storage.primary_connection_string

}
```

```
resource "azurerm_data_factory_linked_service_azure_blob_storage" "destination"
{

 name              = "destination-storage"

 data_factory_id   = azurerm_data_factory.adf.id

 connection_string =
data.azurerm_storage_account.destination_folder_storage.primary_connection_stri
ng

}



#source and sink dataset to blob storage

resource "azurerm_data_factory_dataset_binary" "source_dataset" {



 name                = "source_dataset"

 data_factory_id     = azurerm_data_factory.adf.id

 linked_service_name =
azurerm_data_factory_linked_service_azure_blob_storage.source.name



 sftp_server_location {

   filename = "test.txt"

   path     = "source"

 }

}



resource "azurerm_data_factory_dataset_binary" "destination_dataset" {



 name                = "destination_dataset"

 data_factory_id     = azurerm_data_factory.adf.id
```

```
 linked_service_name =
azurerm_data_factory_linked_service_azure_blob_storage.destination.name


 sftp_server_location {

   filename = "test-${formatdate("YYYY-MM-DD-hh-mm-ss", timestamp() )}.txt"

   path     = "destination"

 }

}


resource "azurerm_data_factory_pipeline" "copy_data" {


 name            = "copy_data_pipeline"

 data_factory_id = azurerm_data_factory.adf.id


 activities_json = <<JSON

[

 {

   "name": "CopyFromSourceToDestination",

   "type": "Copy",

   "typeProperties": {

     "source": {

       "type": "BinarySource",

       "recursive": true

     },

     "sink": {

       "type": "BinarySink"
```

```
    },
    "enableStaging": false
  },
  "policy": {
    "timeout": "7.00:00:00",
    "retry": 0,
    "retryIntervalInSeconds": 30,
    "secureInput": false,
    "secureOutput": false
  },
  "scheduler": {
    "frequency": "Day",
    "interval": 1
  },
  "external": true,
   "inputs": [
    {
      "referenceName": "source_dataset",
      "type": "DatasetReference"
    }
  ],
  "outputs": [
    {
      "referenceName": "destination_dataset",
      "type": "DatasetReference"
    }
```

```
   ]

 }

]

JSON



 depends_on = [

   azurerm_data_factory_dataset_binary.source_dataset,

   azurerm_data_factory_dataset_binary.destination_dataset,

 ]

}
```

1. **Azure Data Factory (ADF)**:
    - A Data Factory instance is created with a system-assigned managed identity for secure access.
2. **Storage Account Integration**:
    - The script retrieves existing source and destination Azure Blob Storage accounts using their names and resource groups.
    - Creates **linked services** for connecting ADF to the source and destination storage accounts via connection strings.
3. **Datasets**:
    - **Source Dataset**: Points to the source folder in the source storage, targeting the file test.txt.
    - **Destination Dataset**: Targets the destination folder in the destination storage, dynamically appending a timestamp to the file name (e.g., test-YYYY-MM-DD-hh-mm-ss.txt).
4. **Pipeline**:
    - A **Copy Activity** is defined in JSON, specifying:
        - **Source**: Reads binary data recursively from the source dataset.
        - **Sink**: Writes binary data to the destination dataset.
        - **Execution Policy**: Includes timeouts and retry intervals.
    - Scheduled to run daily, ensuring regular updates between the source and destination.
5. **Dependencies**:
    - Ensures the pipeline waits for datasets and linked services to be fully configured before execution.

The script facilitates a seamless data transfer pipeline between Azure Blob Storage containers, leveraging Azure Data Factory's capabilities for automation and scheduling. It ensures:

- **Dynamic File Naming**: Timestamped destination files for uniqueness.
- **Scalability**: Handles binary data transfers recursively.
- **Ease of Configuration**: Modular setup for storage and datasets.

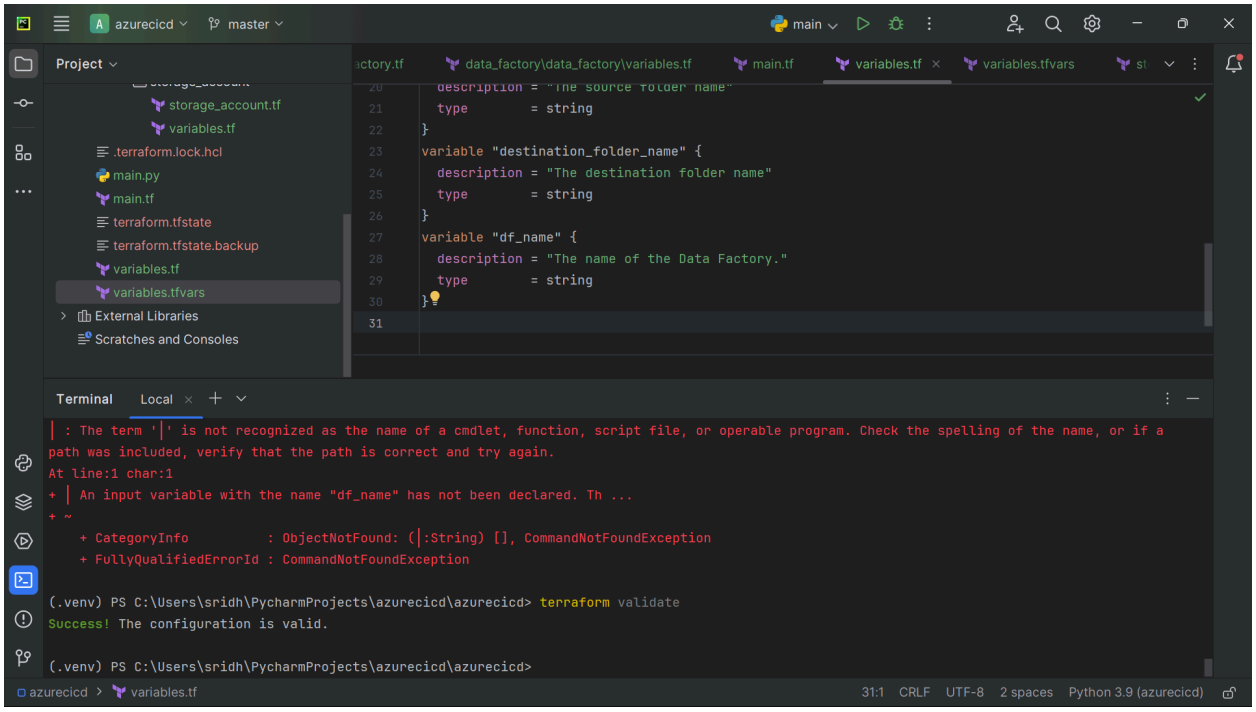
I created the following variables in variables.tf file

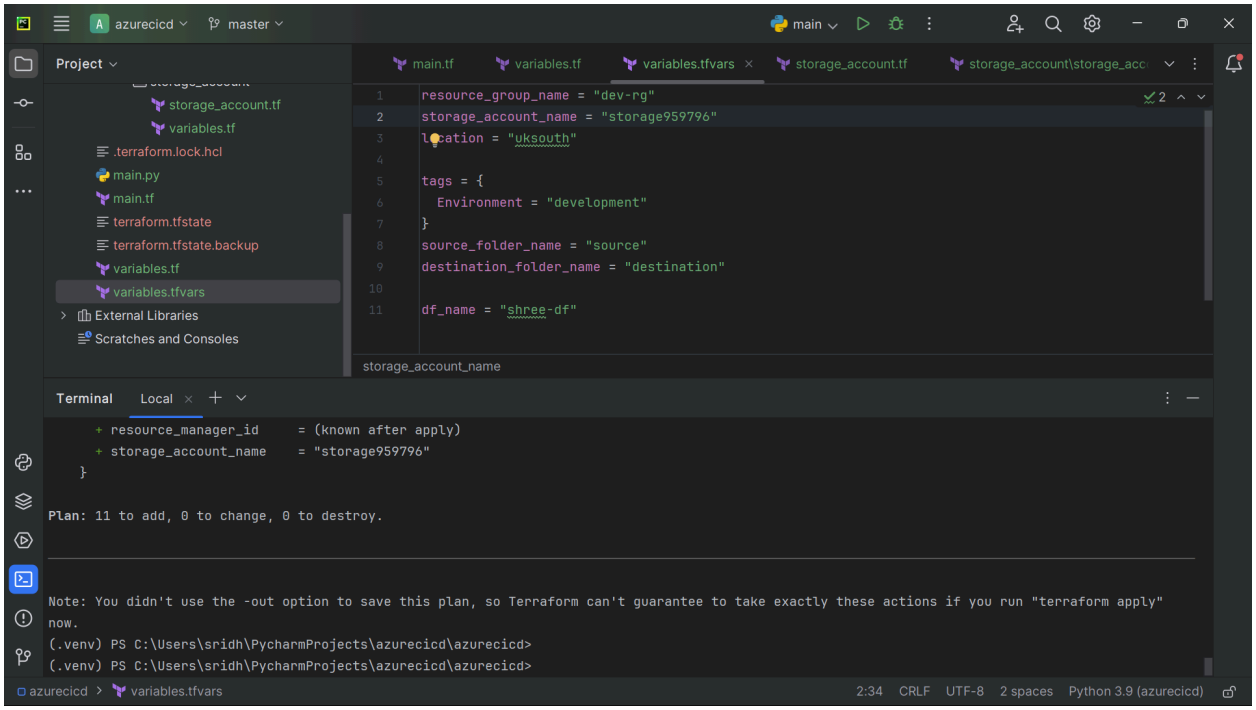
```
variable "storage_account_name" {

 description = "The storage account name"

 type        = string

}



variable "resource_group_name" {

 description = "The name of the resource group"

 type        = string

}



variable "location" {

 description = "The location/region of the resource"

 type        = string

}



variable "df_name" {

 description = "The data factory name"

 type        = string

}
```

To add the module to the root, I used the below code

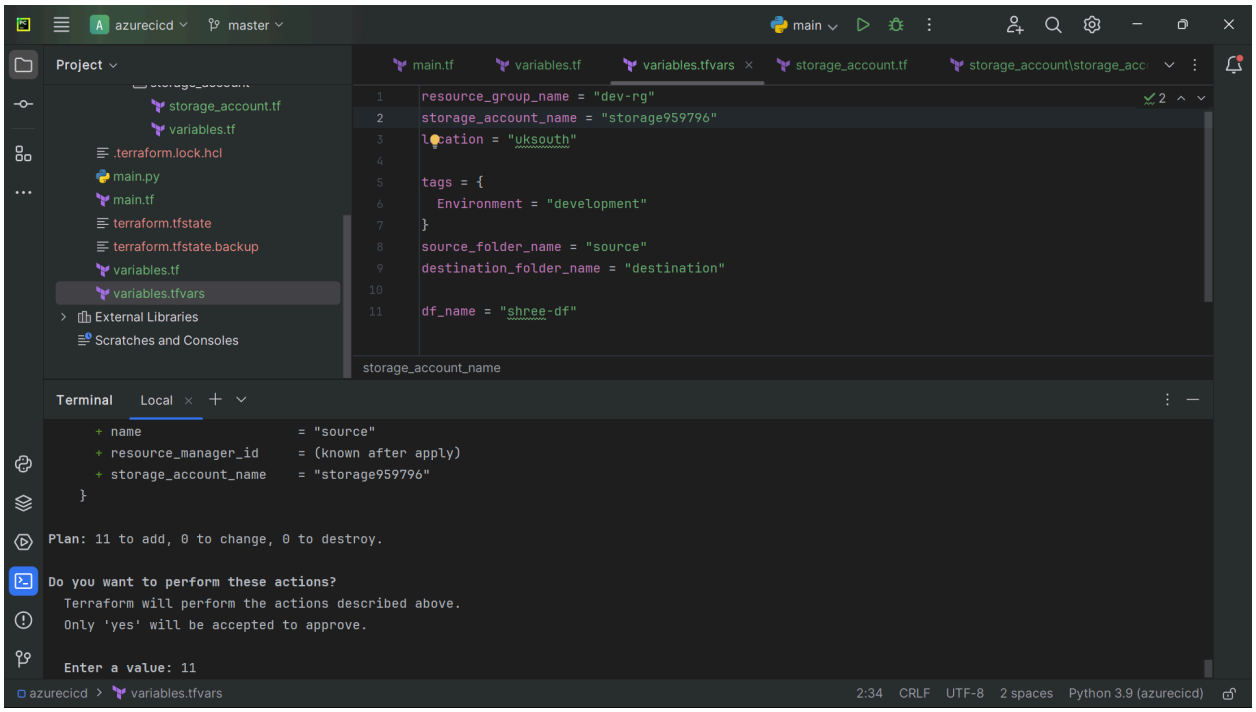
```
module "data_factory" {
  source = "./modules/data_factory/data_factory"

  df_name             = var.df_name
  location            = var.location
  resource_group_name  = var.resource_group_name
  storage_account_name = var.storage_account_name

  depends_on = [
    module.storage_account
  ]
}
```

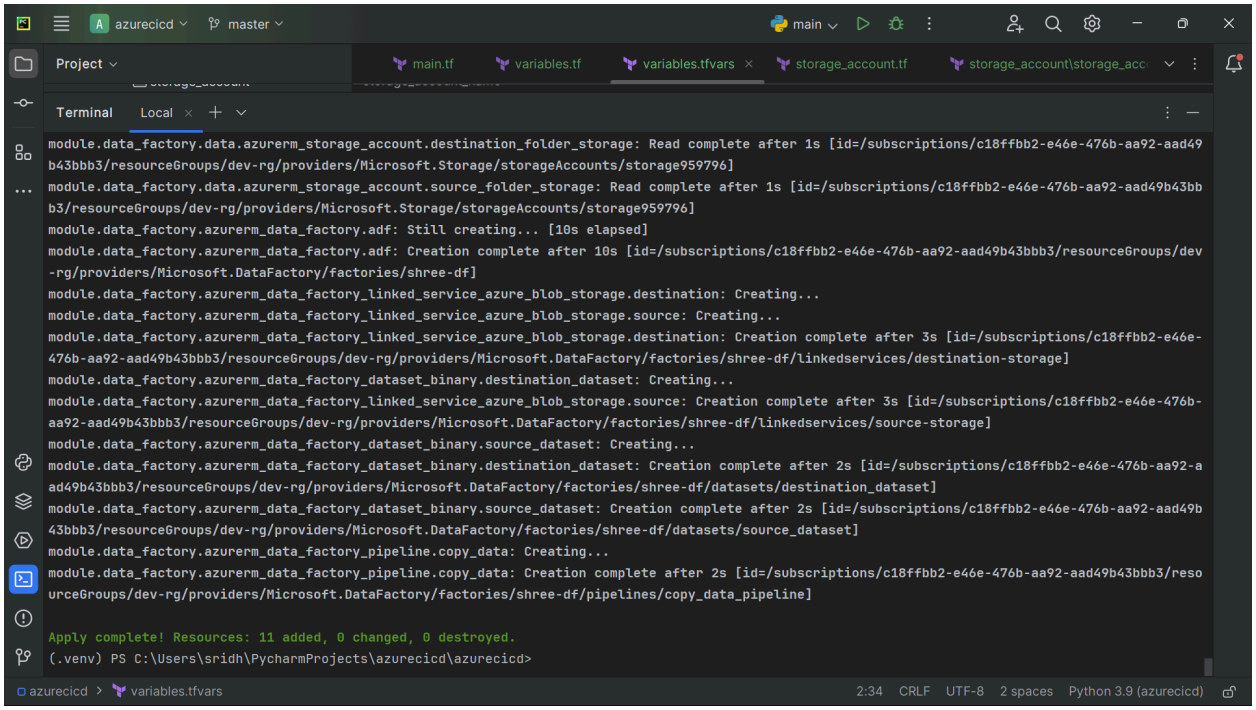
I initialized and validated the terraform script as below



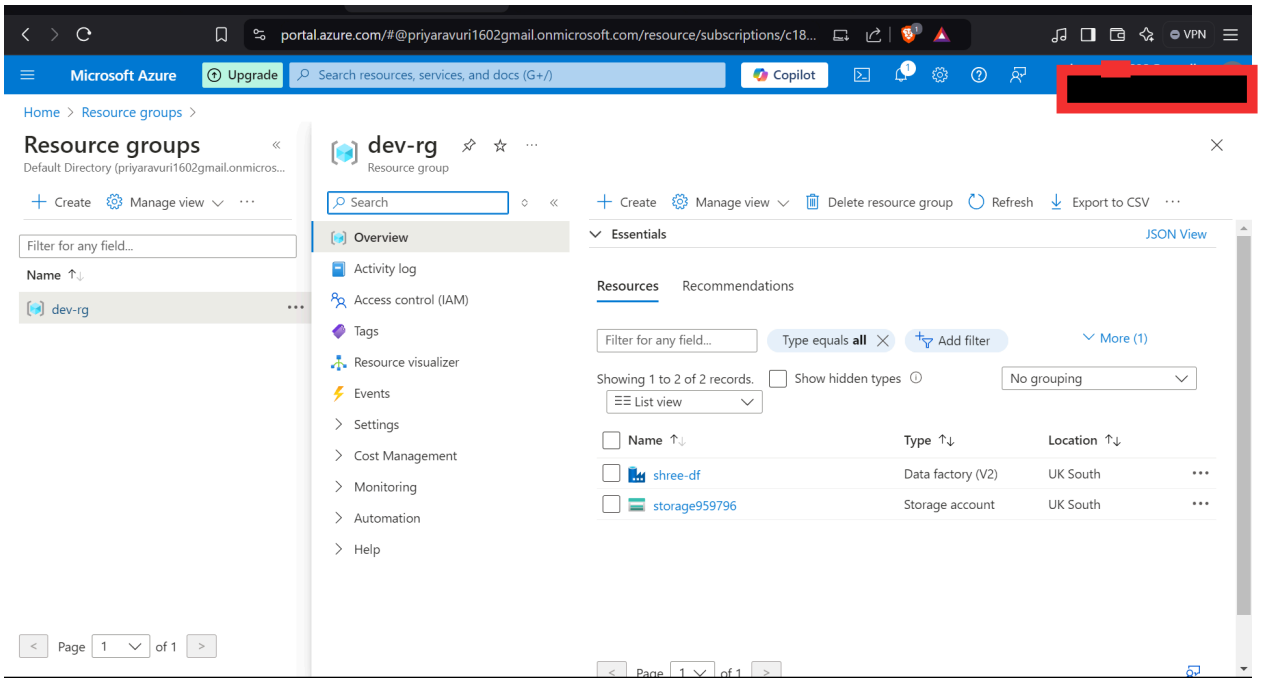


I can see that 11 resources are going to be added. Then, I applied the script

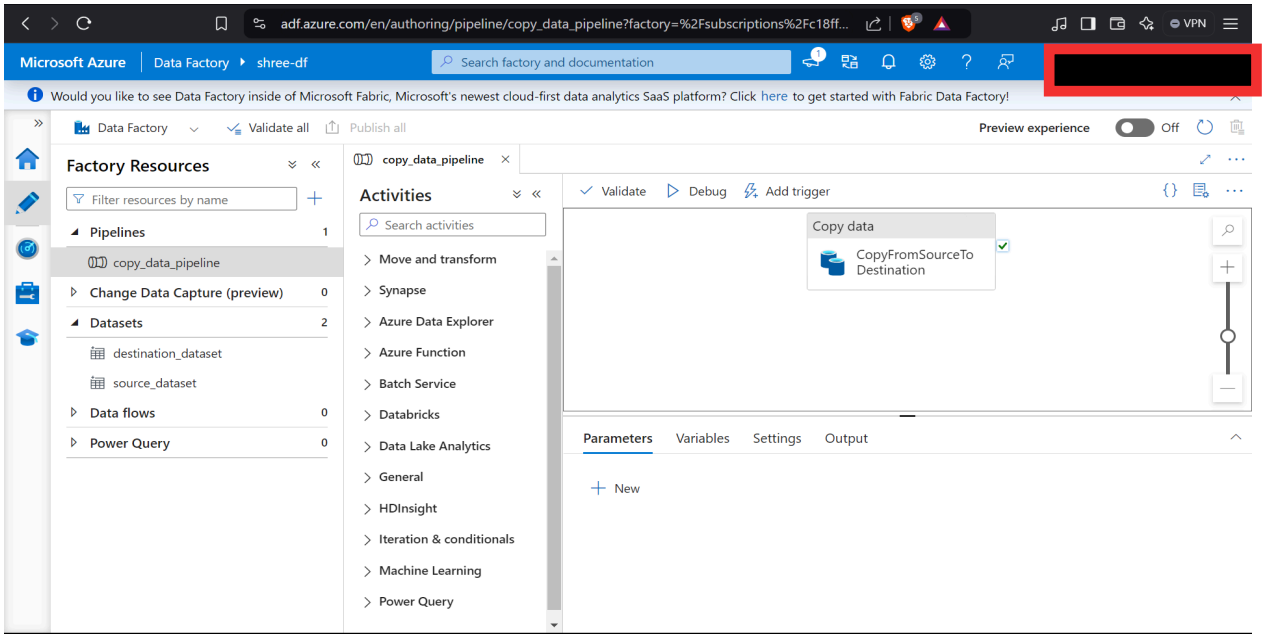


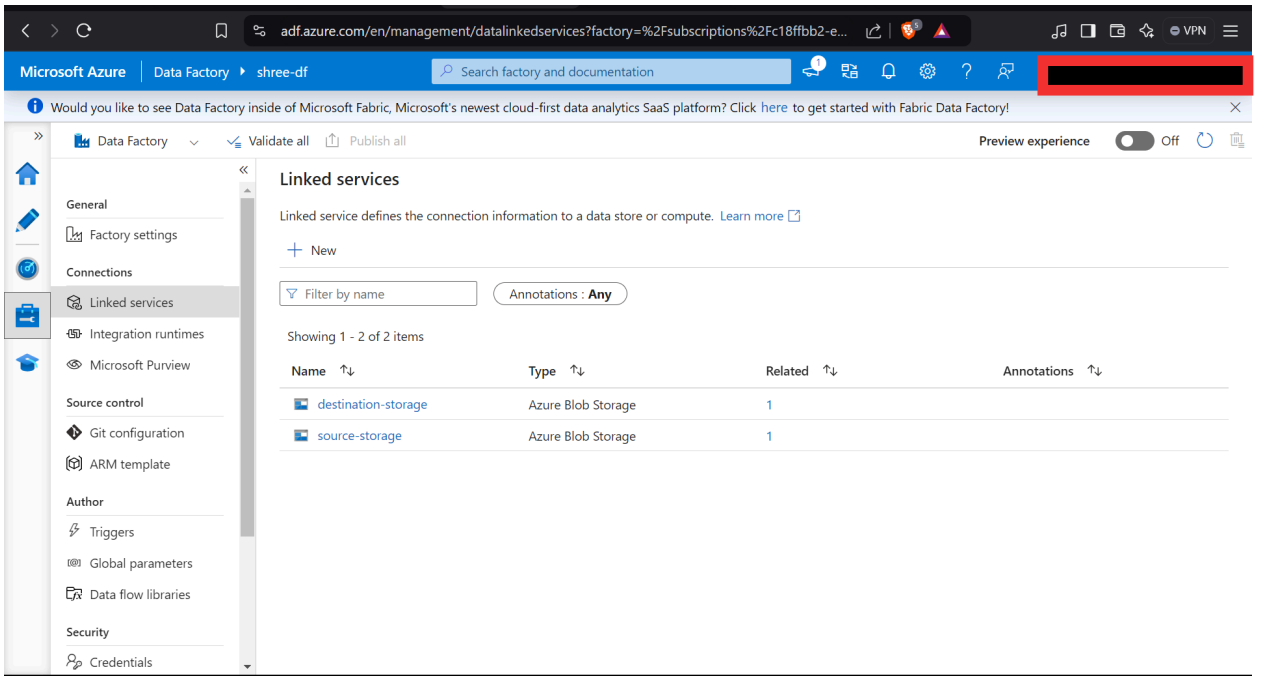


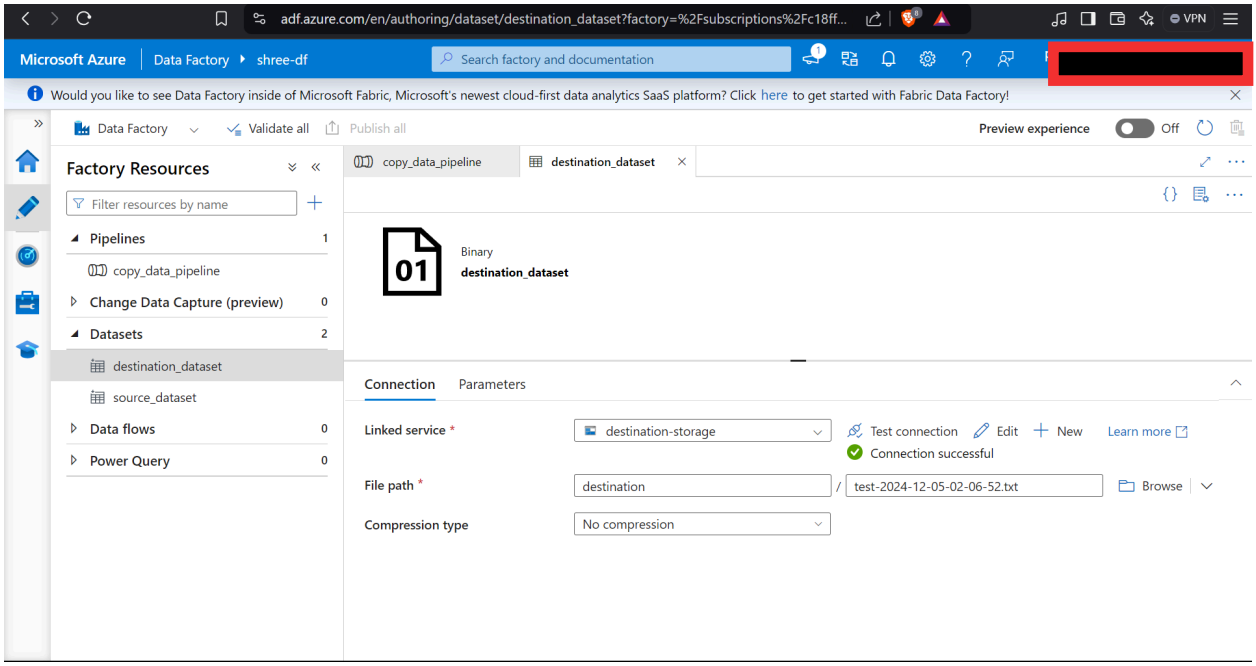
I can recheck if the resources are provisioned from Azure Portal as below. I can see both azure storage account and azure data factory

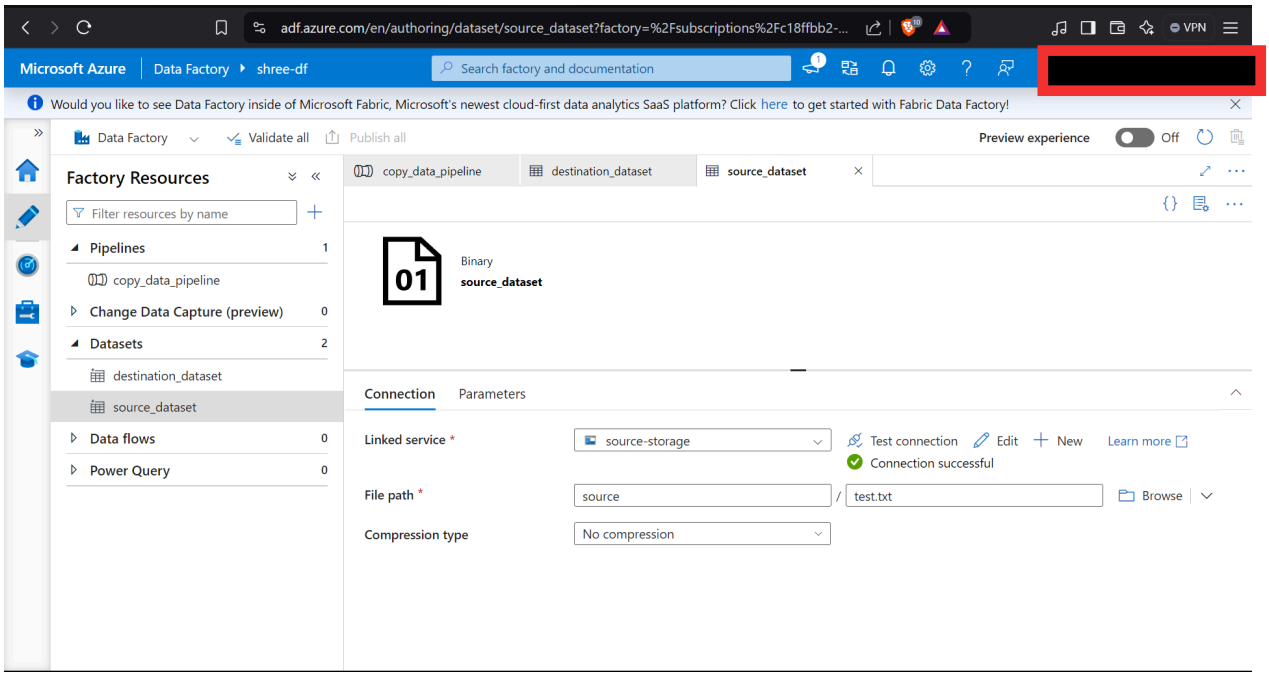


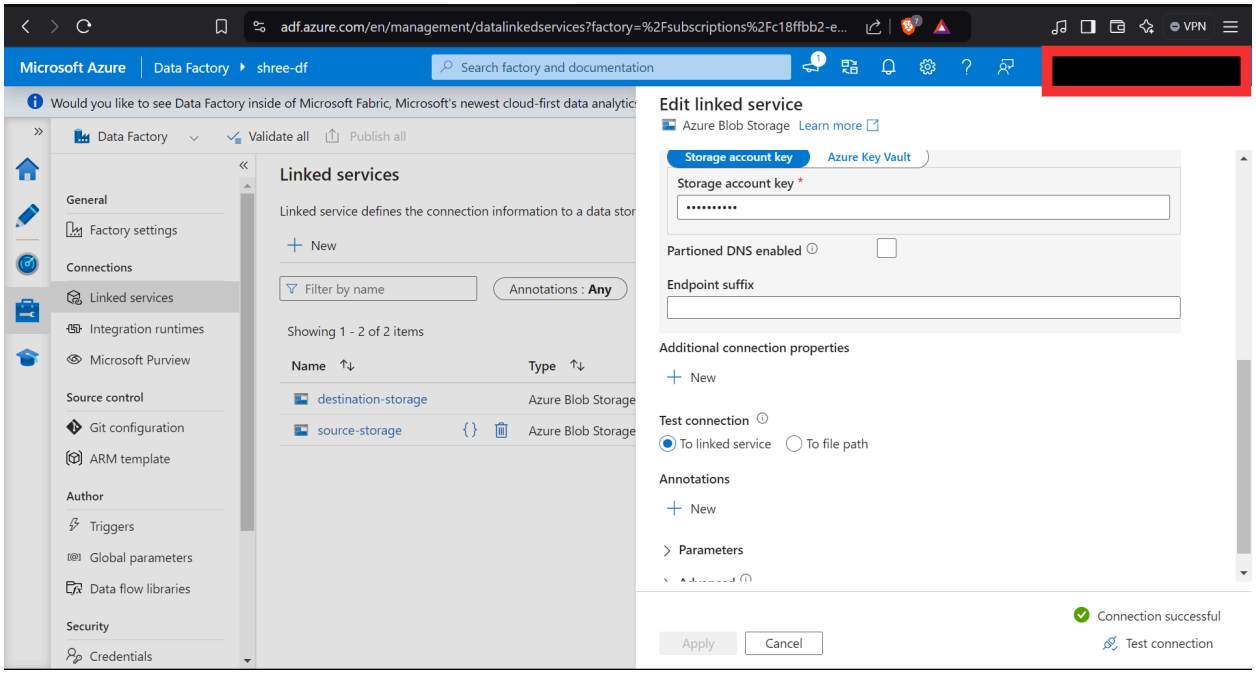
I can also see that a pipeline is created with all the source and destination datasets created as below


adf.azure.com/en/authoring/pipeline/copy_data_pipeline?factory=%2Fsubscriptions%2Fc18ff...
Microsoft Azure
Data Factory
shree-df
Search factory and documentation
Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click here to get started with Fabric Data Factory!
Data Factory
Validate all
Publish all
Preview experience
Off
Factory Resources
Filter resources by name
Pipelines 1
copy_data_pipeline
Change Data Capture (preview) 0
Datasets 2
destination_dataset
source_dataset
Data flows 0
Power Query 0
copy_data_pipeline
Activities
Search activities
Move and transform
Synapse
Azure Data Explorer
Azure Function
Batch Service
Databricks
Data Lake Analytics
General
HDInsight
Iteration & conditionals
Machine Learning
Power Query
Validate
Debug
Add trigger
Copy data
CopyFromSourceTo Destination
Parameters
Variables
Settings
Output
New

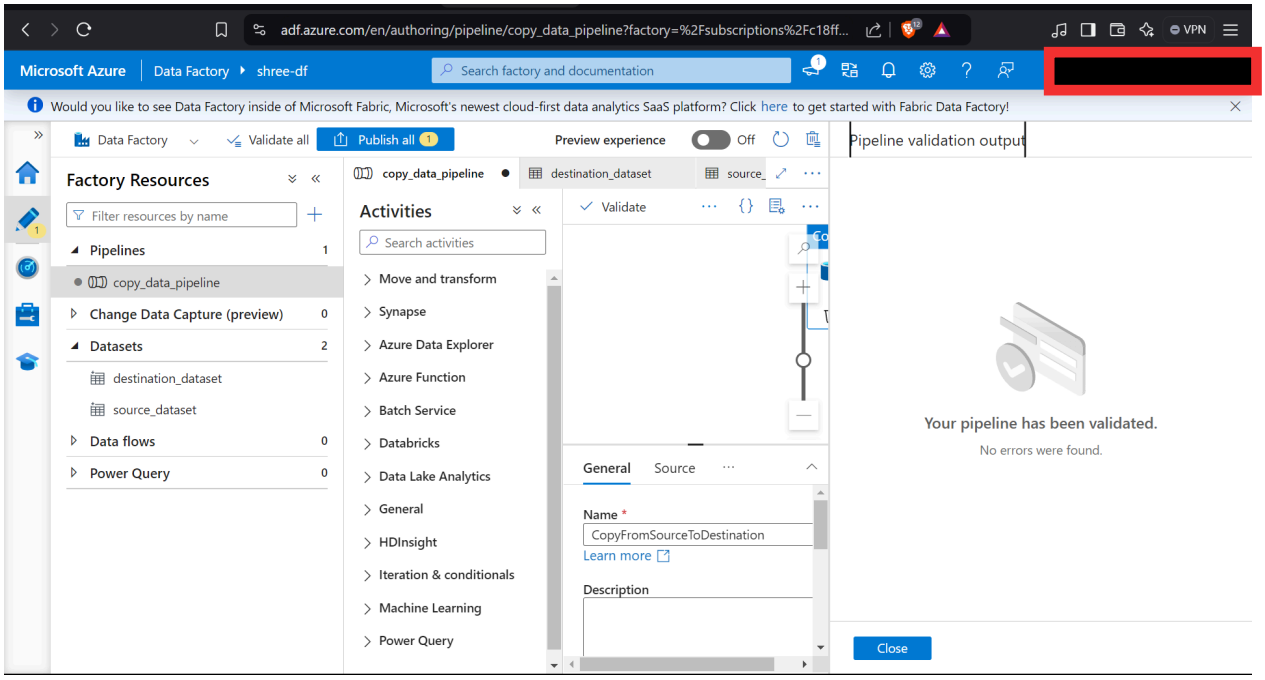

adf.azure.com/en/management/datalinkedservices?factory=%2Fsubscriptions%2Fc18ffbb2-e...
Microsoft Azure
Data Factory
shree-df
Search factory and documentation
Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click here to get started with Fabric Data Factory!
Data Factory
Validate all
Publish all
Preview experience
Off
General
Factory settings
Connections
Linked services
Integration runtimes
Microsoft Purview
Source control
Git configuration
ARM template
Author
Triggers
Global parameters
Data flow libraries
Security
Credentials
Linked services
Linked service defines the connection information to a data store or compute. Learn more
New
Filter by name
Annotations : Any
Showing 1 - 2 of 2 items
Name
Type
Related
Annotations
destination-storage
Azure Blob Storage
1
source-storage
Azure Blob Storage
1


adf.azure.com/en/authoring/dataset/destination_dataset?factory=%2Fsubscriptions%2Fc18ff...
Microsoft Azure
Data Factory
shree-df
Search factory and documentation
Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click here to get started with Fabric Data Factory!
Data Factory
Validate all
Publish all
Preview experience
Off
Factory Resources
Filter resources by name
Pipelines 1
copy_data_pipeline
Change Data Capture (preview) 0
Datasets 2
destination_dataset
source_dataset
Data flows 0
Power Query 0
copy_data_pipeline
destination_dataset
Binary
destination_dataset
Connection
Parameters
Linked service *
destination-storage
Test connection
Edit
New
Learn more
Connection successful
File path *
destination
test-2024-12-05-02-06-52.txt
Browse
Compression type
No compression

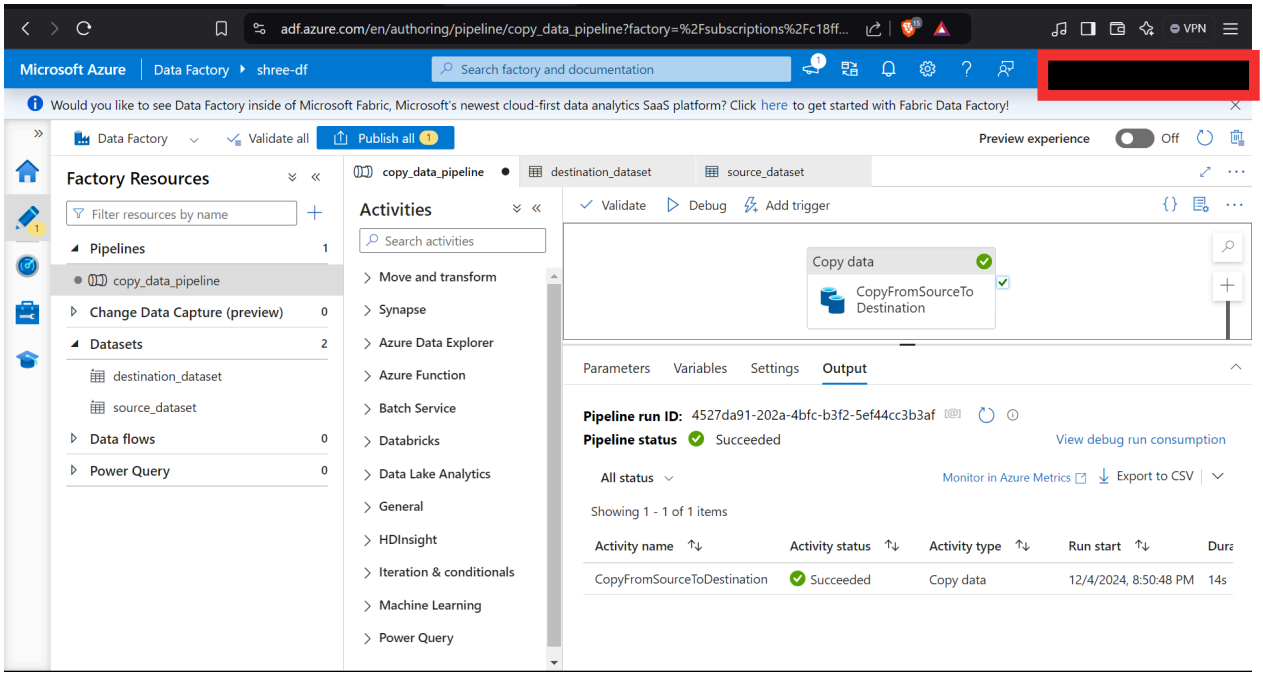

adf.azure.com/en/authoring/dataset/source_dataset?factory=%2Fsubscriptions%2Fc18ffbb2-...
Microsoft Azure
Data Factory
shree-df
Search factory and documentation
Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click here to get started with Fabric Data Factory!
Data Factory
Validate all
Publish all
Preview experience
Off
Factory Resources
Filter resources by name
Pipelines 1
copy_data_pipeline
Change Data Capture (preview) 0
Datasets 2
destination_dataset
source_dataset
Data flows 0
Power Query 0
copy_data_pipeline
destination_dataset
source_dataset
Binary
source_dataset
Connection
Parameters
Linked service *
source-storage
Test connection
Edit
New
Learn more
Connection successful
File path *
source
test.txt
Browse
Compression type
No compression



I can see that pipeline is validated without any errors



Next, I debugged the pipeline



Once the debug was successful, I can see the file in the destination storage container

By the above steps, I provisioned resources on Azure using Terraform which can be a simple POC for Infrastructure as code (IaaC)