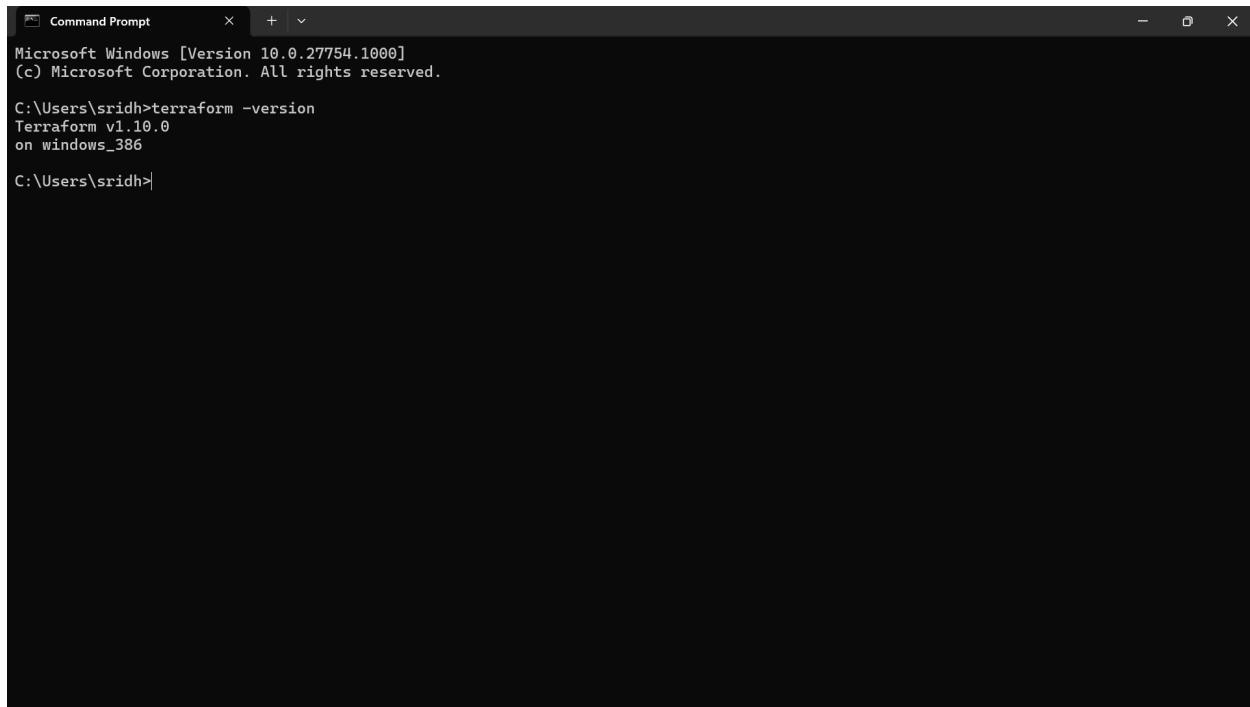


Install Terraform from the link (<https://developer.hashicorp.com/terraform/install>)



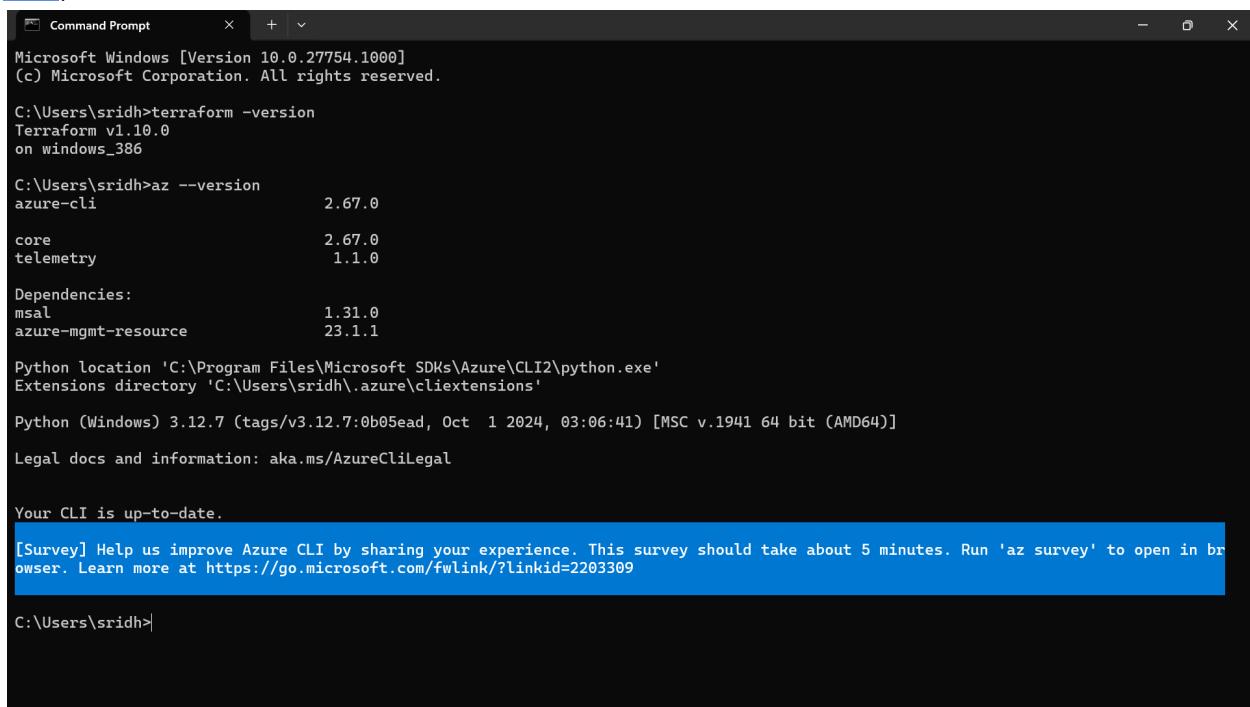
```
Microsoft Windows [Version 10.0.27754.1000]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sridh>terraform -version
Terraform v1.10.0
on windows_386

C:\Users\sridh>
```

Install Azure CLI

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



```
Microsoft Windows [Version 10.0.27754.1000]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sridh>terraform -version
Terraform v1.10.0
on windows_386

C:\Users\sridh>az --version
azure-cli          2.67.0
core               2.67.0
telemetry          1.1.0

Dependencies:
msal                1.31.0
azure-mgmt-resource 23.1.1

Python location 'C:\Program Files\Microsoft SDKs\Azure\CLI2\python.exe'
Extensions directory 'C:\Users\sridh\.azure\cliextensions'

Python (Windows) 3.12.7 (tags/v3.12.7:0b05ead, Oct 1 2024, 03:06:41) [MSC v.1941 64 bit (AMD64)]
Legal docs and information: aka.ms/AzureCliLegal

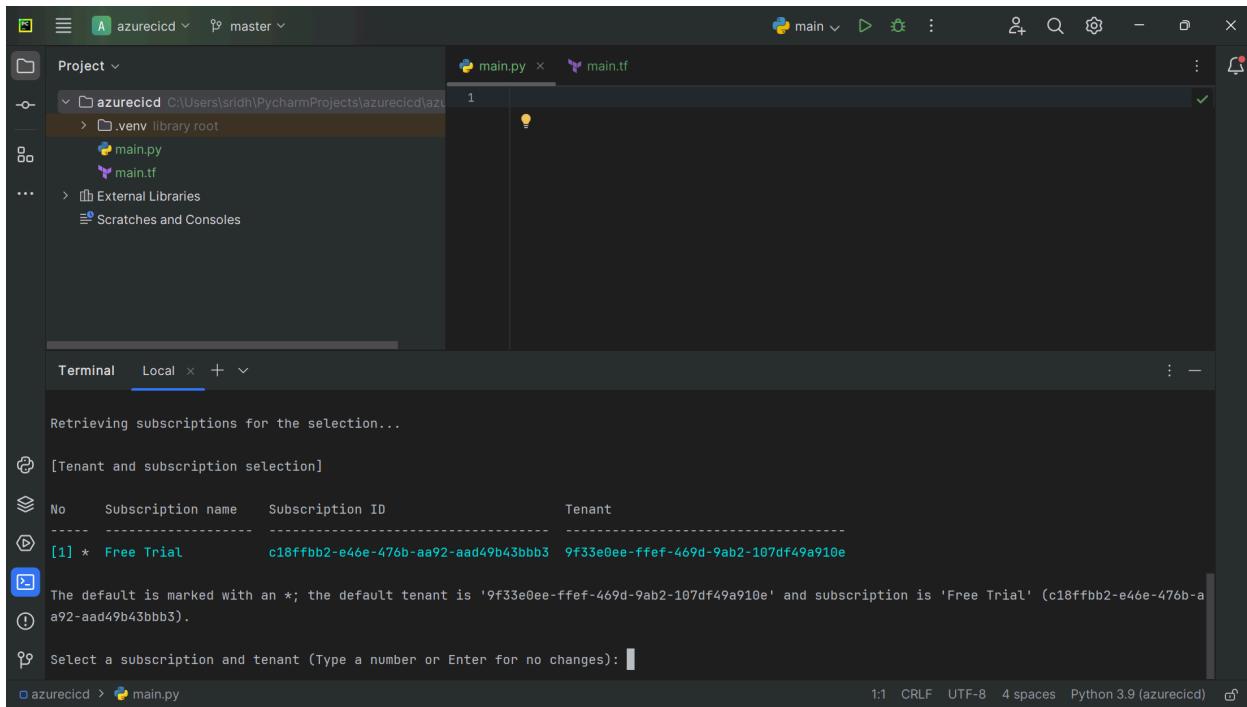
Your CLI is up-to-date.

[Survey] Help us improve Azure CLI by sharing your experience. This survey should take about 5 minutes. Run 'az survey' to open in browser. Learn more at https://go.microsoft.com/fwlink/?linkid=2203309

C:\Users\sridh>
```

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>
```



The screenshot shows a PyCharm interface with a terminal window open. The terminal output is as follows:

```
Retrieving subscriptions for the selection...
[Tenant and subscription selection]
No Subscription name Subscription ID Tenant
-----
[1] * Free Trial c18ffbb2-e46e-476b-aa92-aad49b43bbb3 9f33e0ee-ffef-469d-9ab2-107df49a910e
The default is marked with an *; the default tenant is '9f33e0ee-ffef-469d-9ab2-107df49a910e' and subscription is 'Free Trial' (c18ffbb2-e46e-476b-aa92-aad49b43bbb3).
Select a subscription and tenant (Type a number or Enter for no changes):
```

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.

```

1 terraform {
2     required_providers {
3         source = "hashicorp/azurerm"
4             version = "~> 3.0.2"
5     }
6
7     required_version = ">= 1.1.0"
8 }
9
10 }
11
12 provider "azurerm" {
13     features {}
14 }

```

Terminal Local × +

- Using previously-installed hashicorp/azurerm v3.0.2

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.

(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>

azurecicd > main.tf

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.

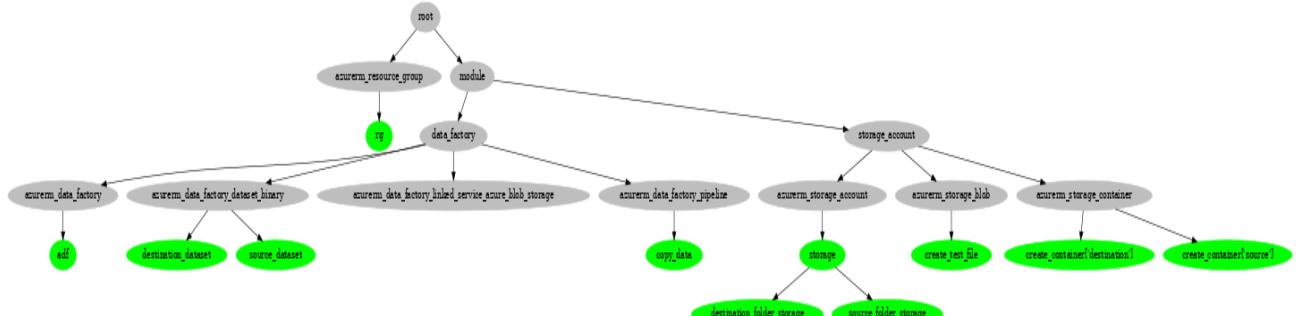
provider "azurerm":

- 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

```

main.py
main.py
main.tf
main.py
main.tf

Project
  - azurecicd
    - .terraform
    - .venv
      - library root
        - .terraform.lock.hcl
    - main.py
    - main.tf
    - External Libraries
    - Scratches and Consoles

main.py
main.tf

Terminal
  - Using previously-installed hashicorp/azurerm v3.0.2
  - 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.
  (.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd> []
  azurecicd > main.tf
  
```

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"
}
  
```

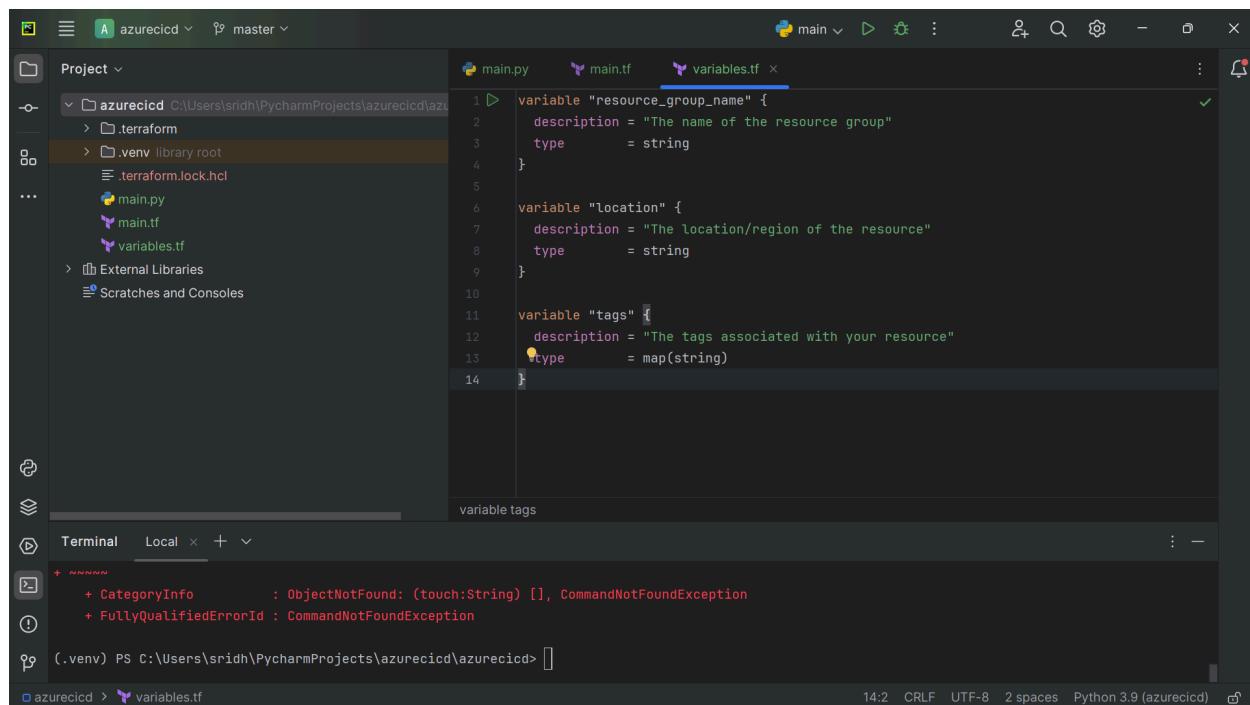
```

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)
}
```

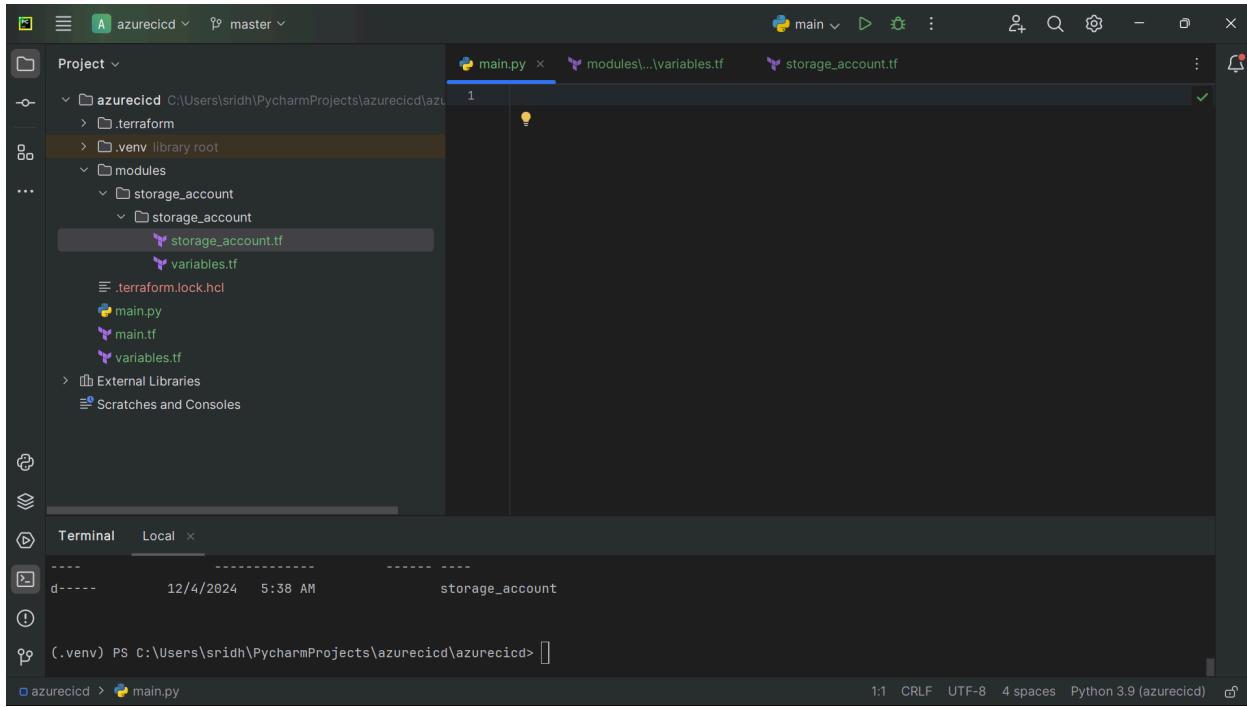
Purpose: Allows the definition of tags (key-value pairs) to associate metadata with resources.

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 = {
        for container in var.create_container : container
    }
}
```

```

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

The screenshot shows the PyCharm IDE interface with a dark theme. On the left, the Project tool window displays a file structure for a 'azurecid' project. Inside 'main.py', there is a code snippet for Terraform:

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

resource "azurerm_resource_group" "rg" {
    name      = var.resource_group_name
    location = var.location
    tags      = var.tags
}

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
}
```

The 'variables.tf' file is also visible in the Project view. Below the Project view, the Terminal tab shows the command-line interface:

```
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.
(.venv) PS C:\Users\sridh\PycharmProjects\azurecid\azurecid> terraform validate
Success! The configuration is valid.
```

The status bar at the bottom indicates the current environment is '.azurecid' and the file is 'main.tf'.

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

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar on the left lists files like 'main.py', 'main.tf', and 'variables.tfvars'. The 'variables.tfvars' file is currently selected and open in the main editor area. The code defines variables for a storage account:

```
resource_group_name = "dev-rg"
storage_account_name = "devstorage"
location = "uksouth"

tags = {
    Environment = "development"
}

source_folder_name = "source"
destination_folder_name = "destination"
```

In the bottom terminal window, the command `terraform validate` is run, resulting in the output:

```
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.
(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd> terraform validate
Success! The configuration is valid.
```

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

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar on the left lists files like 'main.py', 'main.tf', and 'variables.tfvars'. The terminal window at the bottom shows the output of the `terraform plan` command:

```
# module.storage_account.azurerm_storage_container.create_container["destination"] will be created
+ resource "azurerm_storage_container" "create_container" {
    + container_access_type  = "private"
    + has_immutability_policy = (known after apply)
    + has_legal_hold          = (known after apply)
    + id                      = (known after apply)
    + metadata                = (known after apply)
    + name                    = "destination"
    + resource_manager_id     = (known after apply)
    + storage_account_name    = "devstorage"
}

# module.storage_account.azurerm_storage_container.create_container["source"] will be created
+ resource "azurerm_storage_container" "create_container" {
    + container_access_type  = "private"
    + has_immutability_policy = (known after apply)
    + has_legal_hold          = (known after apply)
    + id                      = (known after apply)
    + metadata                = (known after apply)
    + name                    = "source"
```

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar at the top indicates the current project is 'azurecicd' and the branch is 'master'. Below the navigation bar, the 'Project' tool window shows a file structure with 'main.py', 'main.tf', and 'variables.tfvars' files. The 'main.tf' file is open in the editor, displaying Terraform code. The terminal tab at the bottom shows the command-line interface for running Terraform. The terminal output shows a 'Plan' command being run, which outputs 5 resources to be added. A note in the terminal states: '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.' The status bar at the bottom right shows the current time as 10:37, character count as 5 chars, and Python version as 3.9 (azurecicd).

```
# module.storage_account.azurerm_storage_container.create_container["source"] will be created
+ resource "azurerm_storage_container" "create_container" {
    + container_access_type      = "private"
    + has_immutability_policy   = (known after apply)
    + has_legal_hold              = (known after apply)
    + id                           = (known after apply)
    + metadata                     = (known after apply)
    + name                         = "source"
    + resource_manager_id        = (known after apply)
    + storage_account_name       = "devstorage"
}
}
Plan: 5 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.

(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>
(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>
```

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

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar at the top indicates the current project is 'azurecicd' and the branch is 'master'. Below the navigation bar, the 'Project' tool window shows a file structure with 'main.py', 'main.tf', 'variables.tfvars', and 'storage_account.tf' files. The 'main.tf' file is open in the editor, displaying Terraform code. The terminal tab at the bottom shows the command-line interface for running Terraform. The terminal output shows a 'Plan' command being run, which outputs 5 resources to be added. A note in the terminal states: '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.' The status bar at the bottom right shows the current time as 27:18, character count as 15 chars, and Python version as 3.9 (azurecicd).

```
source_content          = "Hello World with Shree!"
```

```
+ storage_account_name = "devstorage"
}

Plan: 5 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.

(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>
(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd> terraform apply -var-file="variables.tfvars"
```

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

The screenshot shows the PyCharm IDE interface with a dark theme. The top navigation bar includes tabs for 'main.py', 'main.tf', 'variables.tfvars', and 'storage_account.tf'. The 'storage_account.tf' tab is active, displaying the following Terraform code:

```
source_content = "Hello World with Shree!"  
resource azurerm_storage_blob create_test_file > source_content  
  
+ resource_manager_id      = (known after apply)  
+ storage_account_name    = "devstorage"  
}  
  
# module.storage_account.azurerm_storage_container.create_container["source"] will be created  
+ resource "azurerm_storage_container" "create_container" {  
    + container_access_type      = "private"  
    + has_immutability_policy   = (known after apply)  
    + has_legal_hold             = (known after apply)  
    + id                         = (known after apply)  
    + metadata                   = (known after apply)  
    + name                       = "source"  
    + resource_manager_id        = (known after apply)  
    + storage_account_name       = "devstorage"  
}  
  
Plan: 5 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.
```

In the bottom status bar, it says '27:18 (15 chars) CRLF UTF-8 2 spaces Python 3.9 (azurecid)'.

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

The screenshot shows the PyCharm IDE interface with the following details:

- Project:** azurecidc (selected)
- File Structure:** C:\Users\sridh\PycharmProject\azurecidc
- Files:** main.py, main.tf, variables.tfvars, storage_account.tf
- Code Editor:** The main.tf file is open, showing Terraform code to create a storage container named "source".

```
+ resource "azurerm_storage_container" "create_container" {
    + container_access_type      = "private"
    + has_immutability_policy   = (known after apply)
    + has_legal_hold              = (Known after apply)
    + id                           = (known after apply)
    + metadata                     = (known after apply)
    + name                         = "source"
    + resource_manager_id         = (known after apply)
    + storage_account_name        = "devstorage"
}
```

Plan: 5 to add, 0 to change, 0 to destroy.
- Terminal:** Shows the Terraform plan output and a confirmation prompt:

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
```
- Console:** Shows the Terraform command-line interface output:

```
azurerm_resource_group.rg: Creating...
azurerm_resource_group.rg: Creation complete after 2s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg]
module.storage_account.azurerm_storage_account.storage: Creating...
```
- Status Bar:** azurecidc > modules > storage_account > storage_account > storage_account.tf, 27:18 (15 chars), CRLF, UTF-8, 2 spaces, Python 3.9 (azurecidc)

```

resource_group_name = "dev-rg"
storage_account_name = "storage989796"
location = "uksouth"
tags = {
  Environment = "development"
}
source_folder_name = "source"
destination_folder_name = "destination"

```

Module output:

```

module.storage_account.azurerm_storage_container.create_container["source": Creating...
module.storage_account.azurerm_storage_container.create_container["destination": Creation complete after 0s [id=https://storage989796.blob.core.windows.net/destination]
module.storage_account.azurerm_storage_container.create_container["source": Creation complete after 1s [id=https://storage989796.blob.core.windows.net/source]
module.storage_account.azurerm_storage_blob.create_test_file: Creating...
module.storage_account.azurerm_storage_blob.create_test_file: Creation complete after 0s [id=https://storage989796.blob.core.windows.net/source/test.txt]
Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

```

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

Name	Type	Location
storage989796	Storage account	UK South

The screenshot shows the Microsoft Azure Storage account containers page. The left sidebar lists various storage services: Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Storage Mover, Partner solutions, and Data storage. Under Data storage, the Containers option is selected, showing sub-options for File shares, Queues, and Tables. The main content area displays a table of containers. The table has columns for Name, Last modified, Anonymous access level, and Lease state. Two containers are listed: 'destination' and 'source'. Both were created on 12/4/2024, 8:07:12 AM, and are set to Private with Available lease states.

Name	Last modified	Anonymous access level	Lease state
destination	12/4/2024, 8:07:12 AM	Private	Available
source	12/4/2024, 8:07:12 AM	Private	Available

The screenshot shows the Microsoft Azure Storage account source container blobs page. The left sidebar shows the container structure: Overview, Diagnose and solve problems, Access Control (IAM), and Settings. The main content area displays a table of blobs. The table has columns for Name, Modified, Access tier, Archive status, Blob type, and Size. One blob named 'test.txt' is listed, created on 12/4/2024, 8:07:13 AM, in the Hot (Inferred) access tier, with Block blob type and 23 B size. The authentication method is set to Access key.

Name	Modified	Access tier	Archive status	Blob type	Size
test.txt	12/4/2024, 8:07:13 AM	Hot (Inferred)		Block blob	23 B

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

A screenshot of the Visual Studio Code interface. The top bar shows the workspace name 'azurecidc' and file names 'main.py', 'main.tf', 'variables.tfvars', and 'storage_account.tf'. The left sidebar has a 'Project' view and a terminal tab. The main area displays the Terraform execution plan. It starts with a note about providers generating an execution plan, followed by a list of actions:

- destroy

Terraform will perform the following actions:

```
# azurerm_resource_group.rg will be destroyed
- resource "azurerm_resource_group" "rg" {
    - id      = "/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg"
    - location = "ukouth"
    - name     = "dev-rg"
    - tags     = {
        - "Environment" = "development"
    }
}

# module.storage_account.azurerm_storage_account.storage will be destroyed
- resource "azurerm_storage_account" "storage" {
    - access_tier      = "Hot"
}
```

The bottom status bar shows the current time as 2:37, file encoding as CRLF, character set as UTF-8, and workspace size as 2 spaces. It also indicates Python 3.9 (azurecidc) is active.

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

The screenshot shows the PyCharm IDE interface with a Terraform project open. The left sidebar displays the project structure under 'azurecicd' with 'modules' expanded, showing 'data_factory' and 'storage_account'. The 'variables.tf' file is currently selected in the main editor tab. The terminal tab at the bottom shows the following output:

```
Destroy complete! Resources: 5 destroyed.
(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd> mkdir -p modules/data_factory/data_factory

Directory: C:\Users\sridh\PycharmProjects\azurecicd\azurecicd\modules\data_factory

Mode LastWriteTime Length Name
---- ---

d---- 12/4/2024 1:02 PM data_factory

(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>
```

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_string
}

#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

The screenshot shows the PyCharm IDE interface with the following details:

- Project Structure:** The left sidebar shows a project named "azurecidc" with files like "storage_account.tf", "variables.tf", ".terraform.lock.hcl", "main.py", "main.tf", "terraform.tfstate", "terraform.tfstate.backup", "variables.tf", and "variables.tfvars".
- Code Editor:** The main editor window displays a Terraform configuration file ("variables.tf") with the following code:

```

variable "df_name" {
  description = "The name of the Data Factory."
  type        = string
}

```
- Terminal:** The bottom terminal window shows the command-line output of the Terraform validate command:

```

(.venv) PS C:\Users\sridh\PycharmProjects\azurecidc\azurecidc> terraform validate
Success! The configuration is valid.

```
- Status Bar:** The bottom right corner shows the status bar with "31:1 CRLF UTF-8 2 spaces Python 3.9 (azurecidc)".

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar at the top indicates the current project is 'azurecicd' and the branch is 'master'. Below the navigation bar is a file tree showing files like 'main.tf', 'variables.tf', 'variables.tfvars', 'storage_account.tf', and 'storage_account/storage_account.tf'. The main editor window displays the 'variables.tfvars' file with the following content:

```
resource_group_name = "dev-rg"
storage_account_name = "storage959796"
location = "uksouth"
tags = {
  Environment = "development"
}
source_folder_name = "source"
destination_folder_name = "destination"
df_name = "shree-df"
```

Below the editor is a terminal window titled 'Local' showing the results of a 'terraform plan' command:

```
+ resource_manager_id      = (known after apply)
+ storage_account_name     = "storage959796"
}
```

The terminal also displays a note about saving the plan:

```
Plan: 11 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.

Terminal status bar: 2:34 CRLF UTF-8 2 spaces Python 3.9 (azurecicd)

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

The screenshot shows the PyCharm IDE interface with a dark theme. The project navigation bar at the top indicates the current project is 'azurecicd' and the branch is 'master'. Below the navigation bar is a file tree showing files like 'main.tf', 'variables.tf', 'variables.tfvars', 'storage_account.tf', and 'storage_account/storage_account.tf'. The main editor window displays the 'variables.tfvars' file with the same content as before.

The terminal window shows the results of a 'terraform apply' command:

```
+ name                  = "source"
+ resource_manager_id    = (known after apply)
+ storage_account_name   = "storage959796"
}
```

The terminal also displays a confirmation message:

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

The terminal prompt asks for user input:

```
Enter a value: 11
```

Terminal status bar: 2:34 CRLF UTF-8 2 spaces Python 3.9 (azurecicd)

```

Project  main.tf  variables.tf  variables.tfvars  storage_account.tf  storage_account\storage_acco...
Terminal Local × + 
module.data_factory.data.azurerm_storage_account.destination_folder_storage: Read complete after 1s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.Storage/storageAccounts/storage959796]
...
module.data_factory.data.azurerm_storage_account.source_folder_storage: Read complete after 1s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.Storage/storageAccounts/storage959796]
module.data_factory.azurerm_data_factory.adf: Still creating... [10s elapsed]
module.data_factory.azurerm_data_factory.adf: Creation complete after 10s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df]
module.data_factory.azurerm_data_factory.linked_service_azure_blob_storage.destination: Creating...
module.data_factory.azurerm_data_factory.linked_service_azure_blob_storage.source: Creating...
module.data_factory.azurerm_data_factory.linked_service_azure_blob_storage.destination: Creation complete after 3s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df/linkedservices/destination-storage]
module.data_factory.azurerm_data_factory.dataset_binary.destination_dataset: Creating...
module.data_factory.azurerm_data_factory.linked_service_azure_blob_storage.source: Creation complete after 3s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df/linkedservices/source-storage]
module.data_factory.azurerm_data_factory.dataset_binary.source_dataset: Creating...
module.data_factory.azurerm_data_factory.dataset_binary.destination_dataset: Creation complete after 2s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df/datasets/destination_dataset]
module.data_factory.azurerm_data_factory.dataset_binary.source_dataset: Creation complete after 2s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df/datasets/source_dataset]
module.data_factory.azurerm_data_factory.pipeline.copy_data: Creating...
module.data_factory.azurerm_data_factory.pipeline.copy_data: Creation complete after 2s [id=/subscriptions/c18ffbb2-e46e-476b-aa92-aad49b43bbb3/resourceGroups/dev-rg/providers/Microsoft.DataFactory/factories/shree-df/pipelines/copy_data_pipeline]
Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
(.venv) PS C:\Users\sridh\PycharmProjects\azurecicd\azurecicd>

```

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

Name	Type	Location	Actions
shree-df	Data factory (V2)	UK South	...
storage959796	Storage account	UK South	...

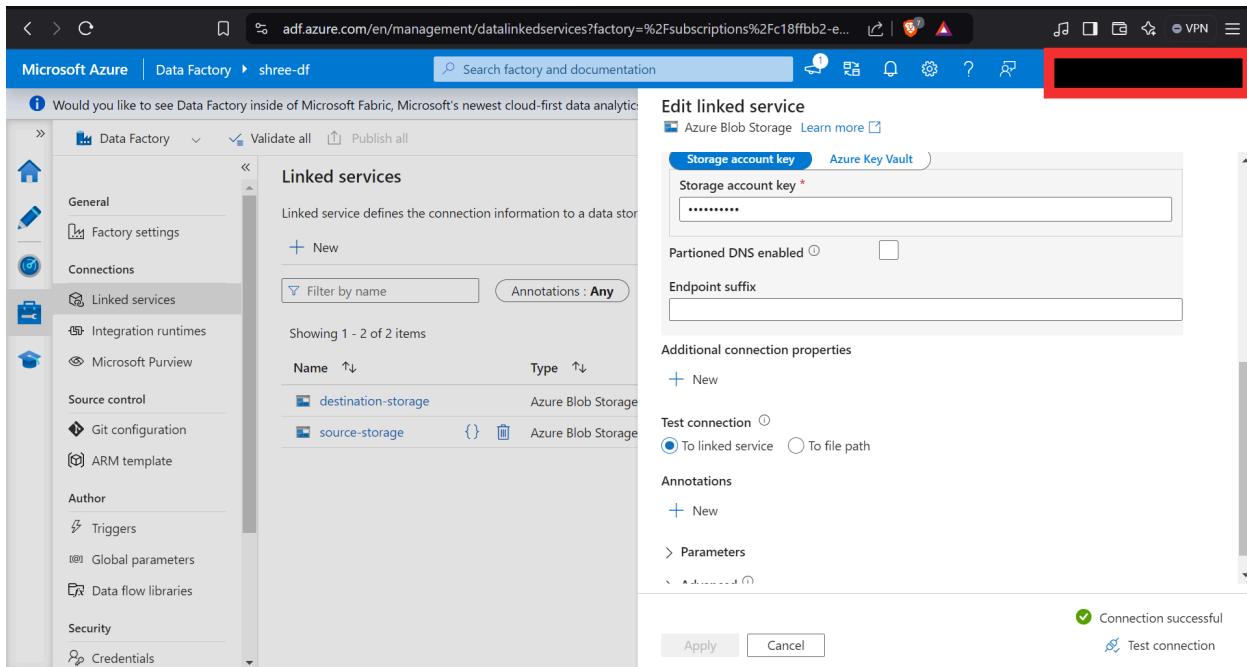
I can also see that a pipeline is created with all the source and destination datasets created as below

The screenshot shows the Microsoft Azure Data Factory pipeline editor. On the left, the 'Factory Resources' sidebar lists 'Pipelines' (1 item: 'copy_data_pipeline'), 'Datasets' (2 items: 'destination_dataset' and 'source_dataset'), and other sections like 'Data flows' and 'Power Query'. The main workspace displays the 'copy_data_pipeline' pipeline. It contains a single 'Copy data' activity named 'CopyFromSourceToDestination'. The 'Activities' pane on the right lists various options such as 'Move and transform', 'Synapse', and 'Machine Learning'. Below the pipeline view are tabs for 'Parameters', 'Variables', 'Settings', and 'Output', with a '+ New' button.

The screenshot shows the Microsoft Azure Data Factory management interface. The left sidebar includes sections like 'General' (Factory settings, Connections), 'Linked services' (selected), 'Integrations runtimes', 'Source control' (Git configuration, ARM template), 'Author' (Triggers, Global parameters), 'Data flow libraries', 'Security' (Credentials), and 'Data factories'. The main area is titled 'Linked services' and describes it as defining connection information to a data store or compute. It shows two entries: 'destination-storage' (Azure Blob Storage) and 'source-storage' (Azure Blob Storage). A 'New' button is available for creating new linked services.

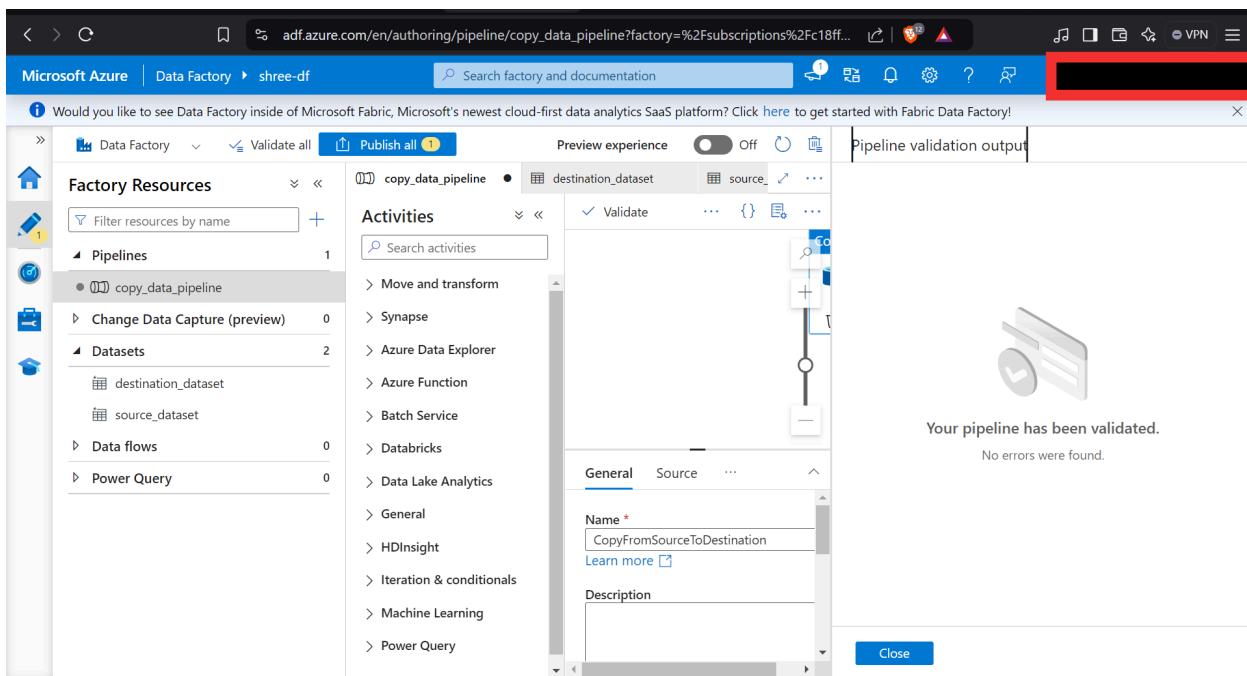
The screenshot shows the Microsoft Azure Data Factory interface. The left sidebar displays 'Factory Resources' with sections for Pipelines, Datasets, Data flows, and Power Query. Under Pipelines, 'copy_data_pipeline' is selected. Under Datasets, 'destination_dataset' is selected. The main workspace shows a preview of the 'destination_dataset' with the identifier '01'. Below the preview, the 'Connection' tab is active, showing the 'destination-storage' linked service and its connection status as successful. The 'Parameters' tab is also visible.

The screenshot shows the Microsoft Azure Data Factory interface, similar to the previous one but for the source dataset. The left sidebar shows 'Factory Resources' with 'source_dataset' selected under Datasets. The main workspace shows a preview of the 'source_dataset' with the identifier '01'. Below the preview, the 'Connection' tab is active, showing the 'source-storage' linked service and its connection status as successful. The 'Parameters' tab is also visible.



The screenshot shows the 'Edit linked service' dialog in the Microsoft Azure Data Factory interface. The left sidebar shows 'Data Factory' selected under 'General'. The main area displays 'Linked services' with two items listed: 'destination-storage' (Azure Blob Storage) and 'source-storage' (Azure Blob Storage). On the right, the 'Storage account key' tab is active, showing fields for 'Storage account key*' (containing '*****') and 'Partitioned DNS enabled' (unchecked). Below this are sections for 'Endpoint suffix', 'Additional connection properties', 'Test connection' (radio button selected for 'To linked service'), 'Annotations', and 'Parameters'. At the bottom are 'Apply' and 'Cancel' buttons, with a green checkmark icon and the text 'Connection successful' next to it.

I can see that pipeline is validated without any errors



The screenshot shows the 'Pipeline validation output' section in the Microsoft Azure Data Factory interface. The left sidebar shows 'Factory Resources' with 'Pipelines' expanded, showing 'copy_data_pipeline' selected. The main area displays the validation results for the 'copy_data_pipeline'. It shows a preview of the pipeline activities: 'Move and transform', 'Synapse', 'Azure Data Explorer', 'Azure Function', 'Batch Service', 'Databricks', 'Data Lake Analytics', 'General', 'HDInsight', 'Iteration & conditionals', 'Machine Learning', and 'Power Query'. A large green checkmark icon is displayed with the message 'Your pipeline has been validated.' and 'No errors were found.' Below the validation message is a 'Close' button.

Next, I debugged the pipeline

The screenshot shows the Microsoft Azure Data Factory pipeline editor. On the left, the 'Factory Resources' sidebar lists 'Pipelines' (copy_data_pipeline), 'Datasets' (destination_dataset, source_dataset), 'Data flows', and 'Power Query'. The main workspace displays the 'copy_data_pipeline' pipeline. Under 'Activities', a 'Copy data' activity named 'CopyFromSourceToDestination' is selected. The 'Output' tab shows a single row of data for the activity:

Activity name	Activity status	Activity type	Run start	Dura
CopyFromSourceToDestination	Succeeded	Copy data	12/4/2024, 8:50:48 PM	14s

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 (IaaS)