TASK 9:

a. Managing resources using Terraform

b. Creating and running containers.

a. Managing resources using Terraform

Azure Cloud Shell is an interactive, browser-accessible shell for managing Azure resources. It provides a command-line interface (CLI) experience that you can use to manage your Azure resources. Azure Cloud Shell supports both Bash and PowerShell.

1. Access Azure Cloud Shell

Via Azure Portal:

1. Go to the [Azure Portal](https://portal.azure.com).
2. Click on the **Cloud Shell** icon in the top navigation bar (it looks like a command prompt **`>_`)**.
3. If prompted, choose either Bash or PowerShell as your shell environment.

2. Set up Storage for Azure Cloud Shell

Azure Cloud Shell requires an Azure storage account to persist files, scripts, and configurations across sessions.

1. When you launch Cloud Shell for the first time, you'll be prompted to create a storage account.
2. Choose a subscription and create a new resource group or use an existing one.
3. Select a region for your storage account.
4. Azure will create a file share in the storage account to store your Cloud Shell files.

- Using an Existing Storage Account:

If you already have a storage account, you can configure Cloud Shell to use it:

1. Launch Cloud Shell.
2. Click on the Show advanced settings link.
3. Select your subscription, storage account, and file share.

3. Configure Your Shell Environment

- Switch Between Bash and PowerShell
- Use the dropdown menu in the Cloud Shell toolbar to switch between Bash and PowerShell.

Persistent Home Directory:

- Your `$HOME` directory is persisted in the Azure file share, so any files or configurations you save here will be available in future sessions.

4. Upload and Download Files(Manage Files)

-Upload Files:

- Use the **Upload/Download** files button in the Cloud Shell toolbar to upload files to your Cloud Shell environment.

- Download Files:

- Use the same button to download files from your Cloud Shell environment to your local machine.

5. Customize Cloud Shell Settings

- Change Font Size:

- Use the settings icon (gear icon) in the Cloud Shell toolbar to adjust the font size.

- **Timeout Settings:**
  - Cloud Shell sessions time out after 20 minutes of inactivity.
  - You can reconnect by refreshing the browser or relaunching Cloud Shell.

Creating Virtual machine

Step1: Create a Terraform file named `main.tf` with following code

main.tf

```
# Configure the Azure provider
provider "azurerm" {
  features {}
  subscription_id = "5c9ff0c8-5a6c-4675-b0f0-a86e73d02d19" # Replace
with your subscription ID
}
# Create a resource group
resource "azurerm_resource_group" "rg" {
  name     = "myResourceGroup123"
  location = "Central India"
}

# Create a virtual network
resource "azurerm_virtual_network" "vnet" {
  name                = "myVNet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}
# Create a subnet
resource "azurerm_subnet" "subnet" {
  name                 = "mySubnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}
# Create a public IP address
resource "azurerm_public_ip" "publicip" {
  name                = "myPublicIP"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Static"
  sku            = "Standard"
}
```

```
# Create a network interface
resource "azurerm_network_interface" "nic" {
  name                = "myNIC"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "myNicConfiguration"
    subnet_id                     = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.publicip.id
  }
}
# Create a virtual machine
resource "azurerm_virtual_machine" "vm" {
  name                  = "myVM"
  location              = azurerm_resource_group.rg.location
  resource_group_name   = azurerm_resource_group.rg.name
  network_interface_ids = [azurerm_network_interface.nic.id]
  vm_size               = "Standard_DS1_v2"

  storage_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku       = "18.04-LTS"
    version   = "latest"
  }

  storage_os_disk {
    name              = "myOsDisk"
    caching           = "ReadWrite"
    create_option     = "FromImage"
    managed_disk_type = "Standard_LRS"
  }

  os_profile {
    computer_name  = "myvm"
    admin_username = "azureuser"
    admin_password = "Password1234!"
  }

  os_profile_linux_config {
    disable_password_authentication = false
```

```
  }
}
```

Step 3: Click on the **Upload** button in **Manage Files** Tab, select file **main.tf** file and upload it.

Step 4: Initialize Terraform

- The `terraform init` command initializes a Terraform working directory by downloading provider plugins, setting up the backend, and preparing the environment for execution.

```
$terraform init
```

Step 5: Plan the Deployment

- Generate an execution plan to preview the changes.

```
$terraform plan
```

Step 6: Apply the Configuration

- Deploy the resources by applying the configuration:

```
$terraform apply
```

Output:

- Creates a resource group named **myResourceGroup**.
- Creates a virtual network (myVNet) and a subnet (mySubnet).
- Creates a public IP address (myPublicIP) and a network interface (myNIC).
- Creates a VM (myVM) with the following specifications:
  - **OS**: Ubuntu 18.04-LTS
  - **VM Size**: Standard_DS1_v2
  - **Admin Credentials**: Username azureuser and password Password1234!
- Outputs the public IP address of the VM after deployment.

b. Creating and running containers.

Activity 1: Installing and Verifying Docker

1. Check if Docker is installed:
   ```
   $docker --version
   ```
2. Verify installation by running a test container:
   ```
   $docker run hello-world
   ```

Activity 2: Managing Images:

1. **List all images**
   ```
   $docker image ls
   ```
2. **Pull an image from Docker Hub**
   ```
   $docker pull <image_name>:<tag>
   $docker pull ubuntu:latest
   ```
3. **Remove an image**
   ```
   $docker rmi <image_name>:<tag>
   $docker rmi ubuntu:latest
   ```
4. **Build an image from a Dockerfile**
   ```
   $docker build -t <image_name>:<tag> <path_to_Dockerfile>
   $docker build -t myapp:1.0 .
   ```

Activity 3: Docker Container Commands

1. Run a container from an image

```
$docker run <image_name>:<tag>
$docker run ubuntu:latest
```

2. Run a container in interactive mode

```
$docker run -it <image_name>:<tag> /bin/bash
$docker run -it ubuntu:latest /bin/bash
```

3. Run a container in detached mode (background)

```
$docker run -d <image_name>:<tag>
$docker run -d nginx:latest
```

4. List running containers

```
$docker ps
```

To list all containers (including stopped ones):

```
$docker ps -a
```

5. Stop a running container

```
$docker stop <container_id_or_name>
$docker stop my_container
```

6. Start a stopped container

```
$docker start <container_id_or_name>
$docker start my_container
```

7. Remove a container

```
$docker rm <container_id_or_name>
$docker rm my_container
```

8. Rename a container

```
$docker rename <old_name> <new_name>
```

Activity 4: Creating a Container to Test Java Version

1. Crate a Dockerfile in the Notepad and Save it Docker working directory

```
# Use an official base image with Java
FROM openjdk:17
# Set working directory
WORKDIR /app
# Copy application files (optional)
COPY . /app
# Command to keep container running (if needed)
CMD ["java", "-version"]
```

2. Create a Docker image from Dockerfile

```
$docker build -t my-image .
```

3. List Docker image files

```
$Docker image ls
REPOSITORY    TAG         IMAGE ID         CREATED               SIZE
my-image      latest      1d70afa1a90f    About an hour ago    727MB
```

4. Create a Docker container

```
Docker run my-image
Output:
openjdk version "17.0.2" 2022-01-18
OpenJDK Runtime Environment (build 17.0.2+8-86)
OpenJDK 64-Bit Server VM (build 17.0.2+8-86, mixed mode,
sharing)
```

5. List Containers

```
$Docker ps –a
CONTAINER ID     IMAGE        COMMAND          CREATED
13e095a45239     my-image    "java -version"   2 minutes ago
STATUS
Exited (0) 2 minutes ago
```

Activity 5: Creating a Container to run Java Application

1. Create a Java Application

MyApp.java

```
public class MyApp
{
     public static void main(String args[])
     {
          System.out.println("Welcome to Docker Container");
     }
}
```

- Compile and Run Java Application
- Copy **MyApp.class** file to Docker Working Directory

2. Create a Dockerfile

Dockerfile

```
# Use OpenJDK base image
FROM openjdk:17-jdk-slim

# Set the working directory
WORKDIR /app
```

```
# Copy the JAR file into the container
COPY MyApp.class  /app/MyApp.class

# Command to run the application
CMD ["java", "MyApp"]
```

3. **Create a Docker Image**

```
docker build –t my-java-app  .
```

4. **List all Image files**

```
docker image ls
```

5. **Create a Container and run Java Application using Container**

```
docker run my-java-app

Output:
Welcome to Docker Container
.
```

Activity 6: Creating a Container to run Python Application

1. **Write a Python Script**

```
app.py

print(“Welcome to Docker Container”)
```

2. **Create a Dockerfile**

Dockerfile2

```
# Use an official Python runtime as a base image
FROM python:3.10-slim

# Set the working directory
WORKDIR /app

# Copy the Python script into the container
COPY app.py /app/app.py

# Command to run the Python script
CMD ["python", "app.py"]
```

3. **Create a Docker Image**

```
docker build –t my-python-app –f Dockerfile2  .
```

4. **List all Image files**

```
docker image ls
```

5. **Create a Container and run Java Application using Container**

```
docker run my-python-app
```

Output:

```
Welcome to Docker Container
```

6. **Launching static website using Docker container and Apache web server**

```
docker pull httpd
```

```
docker run -dit --name my-apache -p 8080:80 httpd
```

open browser:

type: localhost:8080

Output:
It Works!