Assignment – 2

Submitted by ~ Ishika Dutta (CT_CSI_DV_5572)

Task 2.1
Deploy Linux and Windows virtual machine and access them using SSH and RDP

To create a Linux VM (Ubuntu)

1. Search **"Virtual Machines"** → Click **+ Create**
2. Fill in:
    - **Name:** linux-vm
    - **Region:** Choose your region
    - **Image:** Ubuntu Server 22.04 LTS
    - **Size:** Standard B1s
    - **Authentication type:** SSH public key
    - **Username:** ishika
    - **Generate new key pair** or upload your own public key
3. **Allow selected ports** → Choose **SSH (22)**
4. **Review + Create** → **Create**

To connect to Linux VM

- On your terminal (Linux/Mac) or with PuTTY (Windows)

```
ssh azureuser@<public-ip>
```

- You can get <public-ip> from the VM's Overview page.

Step 3: Create a Windows VM

1. **+ Create VM** again
2. Fill in:
    - **Name:** windows-vm
    - **Image:** Windows Server 2022
    - **Size:** Standard B2s or similar
    - **Username/password**
3. **Allow selected ports** → Choose **RDP (3389)**

4. **Review + Create → Create**

Step 4: Connect to Windows VM

- Get public IP from VM Overview
- Open **Remote Desktop Connection** on your PC
    - **Computer:** <public-ip>
    - **Username:** As set during creation
- Enter password → Connect

Task 2.2
Create an App Service Plan Provision a Web App in the existing App Service Plan and deploy a simple welcome page on it.

Create an App Service Plan

```
az appservice plan create \
  --name MyAppServicePlan \
  --resource-group MyResourceGroup \
  --sku FREE \
  --is-linux
```

Provision a Web App in the App Service Plan

```
az webapp create \
  --resource-group MyResourceGroup \
  --plan MyAppServicePlan \
  --name my-unique-webapp-name \
  --runtime "PYTHON|3.10"  # Change runtime as needed
```

Deploy a Simple Welcome Page

Deploy a static HTML page using Local Git

Create a folder with an index.html

```
<!-- index.html -->
<!DOCTYPE html>
<html>
  <head><title>Welcome</title></head>
  <body>
    <h1>Welcome to My Web App!</h1>
  </body>
</html>
```

Initialize Git and Push to Azure

```
cd my-webapp-folder
git init
git add .
git commit -m "Initial commit"

az webapp deployment source config-local-git \
  --name my-unique-webapp-name \
  --resource-group MyResourceGroup
```

Push to the Web App

```
git remote add azure https://<your-app-name>.scm.azurewebsites.net/<your-app-name>.git
git push azure master
```

Welcome page will be live at your provided URL

Task 2.3
Create ACR and pull image from ACR and Create a container from it

Create Azure Container Registry (ACR)

```
az acr create \
  --resource-group MyResourceGroup \
  --name myUniqueRegistryName \
  --sku Basic \
  --admin-enabled true
```

Log in to ACR from Docker

```
az acr login --name myUniqueRegistryName
```

Tag and Push a Docker Image to ACR

Tag Docker image with the ACR login server address

```
az acr list --resource-group MyResourceGroup --query "[].loginServer" --output tsv
```

Pushing image to webapp

```
docker tag mywebapp:v1 myuniqueregistryname.azurecr.io/mywebapp:v1
docker push myuniqueregistryname.azurecr.io/mywebapp:v1
```

Create a Container Instance from the ACR Image

```
az container create \
  --resource-group MyResourceGroup \
  --name mywebappcontainer \
  --image myuniqueregistryname.azurecr.io/mywebapp:v1 \
  --registry-login-server myuniqueregistryname.azurecr.io \
  --registry-username <ACR_USERNAME> \
  --registry-password <ACR_PASSWORD> \
  --dns-name-label mywebappdnsname \
  --ports 80
```

To get your ACR credentials:

```
az acr credential show --name myUniqueRegistryName
```

Access the Container

```
http://mywebappdnsname.<region>.azurecontainer.io
```

Task 2.4
Create Container Instance and deploy a simple docker application on it. Create Container Groups and test functionality

Deploy a Simple Docker App in Azure Container Instance (ACI)

```
az container create \
  --resource-group MyResourceGroup \
  --name mycontainer \
  --image nginx \
  --ports 80 \
  --dns-name-label mynginxdemo \
  --location eastus
```

Test the Deployment

```
http://mynginxdemo.eastus.azurecontainer.io
```

Create a Container Group (Multiple Containers)

```
apiVersion: 2019-12-01
location: eastus
name: mycontainergroup
properties:
  containers:
    - name: webapp
      properties:
        image: nginx
        ports:
          - port: 80
        resources:
          requests:
            cpu: 1.0
            memoryInGb: 1.5
    - name: logsidecar
      properties:
        image: busybox
        command:
          - /bin/sh
          - -c
          - while true; do echo logging from container; sleep 10; done
        resources:
          requests:
            cpu: 0.5
            memoryInGb: 0.5
  osType: Linux
  ipAddress:
    type: Public
    dnsNameLabel: mycontainergroupdemo
    ports:
      - port: 80
tags: null
type: Microsoft.ContainerInstance/containerGroups
```

Deploy It:

```
az container create \
  --resource-group MyResourceGroup \
  --file container-group.yaml
```

Test Container Group

Visit:

```
http://mycontainergroupdemo.eastus.azurecontainer.io
```

Monitor logs:

```
az container logs --resource-group MyResourceGroup --name mycontainergroup --container-name logsidecar
```

Functionality Testing

```
az container exec --resource-group MyResourceGroup \
  --name mycontainergroup --container-name webapp \
  --exec-command "/bin/sh"
```

Task 2.5

A. Create a Vnet, 2 Subnets Subnet-1: Linux VM, WindowsVM Subnet-2: SQL DB, B. Create 4 VNets 1. Management Vnet (HUB) 2. Production Vnet 3. Testing Vnet 4. Developing Vnet And Configure Hub and Spoke Architecture and verify it's working by launching VM in each VNet and ping from Managemnent VM to every other VM

Create 1 VNet with 2 Subnets

- *Subnet-1: Linux VM & Windows VM*
- *Subnet-2: Azure SQL Database (simulated)*

Create Resource Group

```
az group create --name NetInfraRG --location eastus
```

Create VNet and Subnets

```
az network vnet create \
  --resource-group NetInfraRG \
  --name MyVNet \
  --address-prefix 10.0.0.0/16 \
  --subnet-name Subnet-1 \
  --subnet-prefix 10.0.1.0/24
```

Add Subnet-2

```
az network vnet subnet create \
  --resource-group NetInfraRG \
  --vnet-name MyVNet \
  --name Subnet-2 \
  --address-prefix 10.0.2.0/24
```

Create Linux and Windows VMs in Subnet-1

```
az vm create \
  --resource-group NetInfraRG \
  --name LinuxVM \
  --vnet-name MyVNet \
  --subnet Subnet-1 \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys

az vm create \
  --resource-group NetInfraRG \
  --name WindowsVM \
  --vnet-name MyVNet \
  --subnet Subnet-1 \
  --image Win2022Datacenter \
  --admin-username azureuser \
  --admin-password P@ssw0rd1234
```

Create Hub-and-Spoke VNet Architecture

Create All VNets

```
# HUB - Management VNet
az network vnet create \
  --resource-group NetInfraRG \
  --name ManagementVNet \
  --address-prefix 10.1.0.0/16 \
  --subnet-name mgmt-subnet \
  --subnet-prefix 10.1.0.0/24

# SPOKE 1 - Production
az network vnet create \
  --resource-group NetInfraRG \
  --name ProductionVNet \
  --address-prefix 10.2.0.0/16 \
  --subnet-name prod-subnet \
  --subnet-prefix 10.2.0.0/24

# SPOKE 2 - Testing
az network vnet create \
  --resource-group NetInfraRG \
  --name TestingVNet \
  --address-prefix 10.3.0.0/16 \
  --subnet-name test-subnet \
  --subnet-prefix 10.3.0.0/24

# SPOKE 3 - Development
az network vnet create \
  --resource-group NetInfraRG \
  --name DevVNet \
  --address-prefix 10.4.0.0/16 \
  --subnet-name dev-subnet \
  --subnet-prefix 10.4.0.0/24
```

Create VNet Peerings (Hub-and-Spoke)

```
# Peer Management → Production
az network vnet peering create \
  --name MgmtToProd \
  --resource-group NetInfraRG \
  --vnet-name ManagementVNet \
  --remote-vnet ProductionVNet \
  --allow-vnet-access

az network vnet peering create \
  --name ProdToMgmt \
  --resource-group NetInfraRG \
  --vnet-name ProductionVNet \
  --remote-vnet ManagementVNet \
  --allow-vnet-access

# Management → Testing
az network vnet peering create \
  --name MgmtToTest \
  --resource-group NetInfraRG \
  --vnet-name ManagementVNet \
  --remote-vnet TestingVNet \
  --allow-vnet-access

az network vnet peering create \
  --name TestToMgmt \
  --resource-group NetInfraRG \
  --vnet-name TestingVNet \
  --remote-vnet ManagementVNet \
  --allow-vnet-access

# Management → Development
az network vnet peering create \
  --name MgmtToDev \
  --resource-group NetInfraRG \
  --vnet-name ManagementVNet \
  --remote-vnet DevVNet \
  --allow-vnet-access

az network vnet peering create \
  --name DevToMgmt \
  --resource-group NetInfraRG \
  --vnet-name DevVNet \
  --remote-vnet ManagementVNet \
  --allow-vnet-access
```

Create a VM in Each VNet

```
# VM in Management VNet
az vm create \
  --resource-group NetInfraRG \
  --name MgmtVM \
  --vnet-name ManagementVNet \
  --subnet mgmt-subnet \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys

# Production VM
az vm create \
  --resource-group NetInfraRG \
  --name ProdVM \
  --vnet-name ProductionVNet \
  --subnet prod-subnet \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys

# Testing VM
az vm create \
  --resource-group NetInfraRG \
  --name TestVM \
  --vnet-name TestingVNet \
  --subnet test-subnet \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys

# Development VM
az vm create \
  --resource-group NetInfraRG \
  --name DevVM \
  --vnet-name DevVNet \
  --subnet dev-subnet \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys
```

Enable ICMP (ping) on All VMs

```
az network nsg rule create \
  --resource-group NetInfraRG \
  --nsg-name <VM-NIC-NSG> \
  --name AllowICMP \
  --protocol Icmp \
  --direction Inbound \
  --priority 1001 \
  --source-address-prefixes '*' \
  --destination-port-ranges '*' \
  --access Allow \
  --destination-address-prefixes '*'
```

Test Connectivity from MgmtVM to All VMs

```
ssh azureuser@<MgmtVM-Public-IP>
```

Inside MgmtVM

```
ping <Private-IP-of-ProdVM>
ping <Private-IP-of-TestVM>
ping <Private-IP-of-DevVM>
```

Task 2.6
Create a Internal and External Load Balancer

External Load Balancer

Create a Virtual Network and Subnet

```
az network vnet create \
  --resource-group LBDemoRG \
  --name LBVNet \
  --subnet-name LBSubnet
```

Create Public IP Address for the Load Balancer

```
az network public-ip create \
  --resource-group LBDemoRG \
  --name PublicLBIP
```

Create the Load Balancer

```
az network lb create \
  --resource-group LBDemoRG \
  --name MyPublicLB \
  --sku Standard \
  --frontend-ip-name PublicFrontend \
  --backend-pool-name PublicBackPool \
  --public-ip-address PublicLBIP \
  --vnet-name LBVNet
```

Create Two Virtual Machines in the Backend Pool

```
for i in 1 2; do
  az vm create \
    --resource-group LBDemoRG \
    --name VM$i \
    --vnet-name LBVNet \
    --subnet LBSubnet \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys \
    --no-wait
done
```

Add VMs to the Backend Pool

```
for i in 1 2; do
  nic_id=$(az vm show -g LBDemoRG -n VM$i --query 'networkProfile.networkInterfaces[0].id' -o tsv)
  az network nic ip-config address-pool add \
    --address-pool PublicBackPool \
    --ip-config-name ipconfig1 \
    --nic-name $(basename $nic_id) \
    --lb-name MyPublicLB \
    --resource-group LBDemoRG \
    --vnet-name LBVNet
done
```

Create a Health Probe and Load Balancing Rule

```
az network lb probe create \
  --resource-group LBDemoRG \
  --lb-name MyPublicLB \
  --name HealthProbe \
  --protocol Tcp \
  --port 80

az network lb rule create \
  --resource-group LBDemoRG \
  --lb-name MyPublicLB \
  --name HTTPRule \
  --protocol Tcp \
  --frontend-port 80 \
  --backend-port 80 \
  --frontend-ip-name PublicFrontend \
  --backend-pool-name PublicBackPool \
  --probe-name HealthProbe
```

Internal Load Balancer

Create the Internal Load Balancer

```
az network lb create \
  --resource-group LBDemoRG \
  --name MyInternalLB \
  --sku Standard \
  --frontend-ip-name InternalFrontend \
  --backend-pool-name InternalBackPool \
  --vnet-name LBVNet \
  --subnet LBSubnet \
  --private-ip-address 10.0.0.100
```

Add Same VMs to Internal Backend Pool

```
for i in 1 2; do
  nic_id=$(az vm show -g LBDemoRG -n VM$i --query 'networkProfile.networkInterfaces[0].id' -o tsv)
  az network nic ip-config address-pool add \
    --address-pool InternalBackPool \
    --ip-config-name ipconfig1 \
    --nic-name $(basename $nic_id) \
    --lb-name MyInternalLB \
    --resource-group LBDemoRG \
    --vnet-name LBVNet
done
```

Create Internal Health Probe and Load Balancing Rule

```
az network lb probe create \
  --resource-group LBDemoRG \
  --lb-name MyInternalLB \
  --name InternalHealthProbe \
  --protocol Tcp \
  --port 80

az network lb rule create \
  --resource-group LBDemoRG \
  --lb-name MyInternalLB \
  --name InternalHTTPRule \
  --protocol Tcp \
  --frontend-port 80 \
  --backend-port 80 \
  --frontend-ip-name InternalFrontend \
  --backend-pool-name InternalBackPool \
  --probe-name InternalHealthProbe
```

Task 2.7
Create and test Azure Application gateway

Create Virtual Network and Subnet

```
az network vnet create \
  --resource-group AppGwRG \
  --name AppGwVNet \
  --address-prefix 10.0.0.0/16 \
  --subnet-name AppGwSubnet \
  --subnet-prefix 10.0.1.0/24
```

Create a backend subnet for VMs

```
az network vnet subnet create \
  --resource-group AppGwRG \
  --vnet-name AppGwVNet \
  --name BackendSubnet \
  --address-prefix 10.0.2.0/24
```

Create 2 Backend VMs in BackendSubnet

```
for i in 1 2; do
  az vm create \
    --resource-group AppGwRG \
    --name WebVM$i \
    --vnet-name AppGwVNet \
    --subnet BackendSubnet \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys \
    --no-wait
done
```

Install NGINX on Each VM

```
az vm list-ip-addresses --resource-group AppGwRG -o table
ssh azureuser@<public-ip>
```

```
sudo apt update && sudo apt install nginx -y
echo "Welcome to WebVM1" | sudo tee /var/www/html/index.html
```

Create Public IP for Application Gateway

```
az network public-ip create \
  --resource-group AppGwRG \
  --name AppGwPublicIP \
  --sku Standard
```

Create Application Gateway

```
az network application-gateway create \
  --name MyAppGateway \
  --location eastus \
  --resource-group AppGwRG \
  --capacity 2 \
  --sku Standard_v2 \
  --frontend-port 80 \
  --vnet-name AppGwVNet \
  --subnet AppGwSubnet \
  --public-ip-address AppGwPublicIP
```

Add VMs to Backend Pool

Get private IPs of VMs

```
az vm list-ip-addresses --resource-group AppGwRG --query "[].virtualMachine.network.privateIpAddresses[0]" -o tsv
```

Add backend pool with these IPs:

```
az network application-gateway address-pool create \
  --gateway-name MyAppGateway \
  --resource-group AppGwRG \
  --name AppGwBackPool \
  --addresses <IP1> <IP2>
```

Add HTTP Settings, Listener, and Rule

```
az network application-gateway http-settings create \
  --gateway-name MyAppGateway \
  --resource-group AppGwRG \
  --name AppGwHttpSettings \
  --port 80 \
  --protocol Http \
  --cookie-based-affinity Disabled

az network application-gateway frontend-port create \
  --resource-group AppGwRG \
  --gateway-name MyAppGateway \
  --name FrontendPort \
  --port 80

az network application-gateway listener create \
  --resource-group AppGwRG \
  --gateway-name MyAppGateway \
  --name AppGwListener \
  --frontend-port FrontendPort \
  --frontend-ip MyAppGatewayFrontend

az network application-gateway rule create \
  --resource-group AppGwRG \
  --gateway-name MyAppGateway \
  --name AppGwRule \
  --http-listener AppGwListener \
  --rule-type Basic \
  --address-pool AppGwBackPool \
  --http-settings AppGwHttpSettings
```

Test the Application Gateway

```
az network public-ip show \
  --resource-group AppGwRG \
  --name AppGwPublicIP \
  --query "ipAddress" -o tsv
```

Task 2.8

Set up a domain, setup a server on a VM and use the DNS server for traffic

Provision a VM and Install Web Server

```
# Create resource group
az group create --name DNSDemoRG --location eastus

# Create VM with public IP
az vm create \
  --resource-group DNSDemoRG \
  --name WebVM \
  --image UbuntuLTS \
  --admin-username azureuser \
  --generate-ssh-keys
```

Allow HTTP traffic

```
az vm open-port --resource-group DNSDemoRG --name WebVM --port 80
```

SSH into VM and install nginx

```
ssh azureuser@<public-ip>
```

```
sudo apt update
sudo apt install nginx -y
echo "Hello from Azure DNS demo" | sudo tee /var/www/html/index.html
```

Buy or Use a Custom Domain

Create a DNS Zone in Azure

```
az network dns zone create \
  --resource-group DNSDemoRG \
  --name yourdomain.com
```

Add an A Record (DNS Mapping)

Get your VM's public IP:

```
az vm list-ip-addresses \
  --name WebVM \
  --resource-group DNSDemoRG \
  --query "[].virtualMachine.network.publicIpAddresses[0].ipAddress" \
  -o tsv
```

Map your domain to the VM's public IP:

```
az network dns record-set a add-record \
  --resource-group DNSDemoRG \
  --zone-name yourdomain.com \
  --record-set-name www \
  --ipv4-address <VM_PUBLIC_IP>
```

Update Domain Registrar's Nameservers

Get Azure-assigned nameservers:

```
az network dns zone show \
  --resource-group DNSDemoRG \
  --name yourdomain.com \
  --query nameServers \
  -o tsv
```

Test Your Setup

```
http://www.yourdomain.com
```

Task 2.9

Create a Storage account and explore all the option while creating Upload and access the blob Go through different Auth. Tech and test the same Try Azure storage explorer Provision Access keys and use them for connection with storage account Create a Shared access signature and check access scope Create a stored access policy over shared access signature and check access scope Learn about different access tiers and their use cases Apply lifecycle policy over objects and test the same test object replication in blob Create a file share and test functionality Create a Azure File sync

Create a Storage Account

```
az group create --name StorageDemoRG --location eastus

az storage account create \
  --name mystorageacc12345 \
  --resource-group StorageDemoRG \
  --location eastus \
  --sku Standard_LRS \
  --kind StorageV2 \
  --access-tier Hot
```

Upload and Access Blob

Create a container:

```
az storage container create \
  --account-name mystorageacc12345 \
  --name demo-container \
  --public-access container
```

Upload a file:

```
echo "Hello Azure Blob" > demo.txt

az storage blob upload \
  --account-name mystorageacc12345 \
  --container-name demo-container \
  --name demo.txt \
  --file demo.txt
```

Access the blob URL:

```
echo "https://mystorageacc12345.blob.core.windows.net/demo-container/demo.txt"
```

Explore Authentication Techniques

A. Access Keys (connection string)

```
az storage account keys list \
  --account-name mystorageacc12345 \
  --resource-group StorageDemoRG
```

B. SAS Token (Shared Access Signature)

```
az storage container generate-sas \
  --account-name mystorageacc12345 \
  --name demo-container \
  --permissions rwl \
  --expiry $(date -u -d "1 day" '+%Y-%m-%dT%H:%MZ') \
  --https-only \
  --output tsv
```

Use SAS token with:

```
https://<account>.blob.core.windows.net/<container>/<blob>?<SAS>
```

C. Stored Access Policy + SAS

Create policy:

```
az storage container policy create \
  --account-name mystorageacc12345 \
  --container-name demo-container \
  --name readPolicy \
  --permissions r \
  --expiry "2025-12-31T23:59Z"
```

Generate SAS using policy:

```
az storage blob generate-sas \
  --account-name mystorageacc12345 \
  --container-name demo-container \
  --name demo.txt \
  --policy-name readPolicy \
  --output tsv
```

5. Access Tiers and Use Cases

- *Hot: Frequently accessed data (default)*
- *Cool: Infrequent access, cheaper*
- *Archive: Long-term storage, must rehydrate*

Change tier:

```
az storage blob set-tier \
  --account-name mystorageacc12345 \
  --container-name demo-container \
  --name demo.txt \
  --tier Cool
```

Apply Lifecycle Management Policy

```
az storage account management-policy create \
  --account-name mystorageacc12345 \
  --resource-group StorageDemoRG \
  --policy '{
    "rules": [
      {
        "name": "moveToCool",
        "enabled": true,
        "type": "Lifecycle",
        "definition": {
          "filters": {
            "blobTypes": ["blockBlob"]
          },
          "actions": {
            "baseBlob": {
              "tierToCool": { "daysAfterModificationGreaterThan": 30 },
              "delete": { "daysAfterModificationGreaterThan": 90 }
            }
          }
        }
      }
    ]
  }'
```

Enable Object Replication

Create destination account:

```
az storage account create \
  --name mystorageaccdest \
  --resource-group StorageDemoRG \
  --location westus \
  --sku Standard_LRS \
  --kind StorageV2
```

Enable object replication in the Azure Portal:

- *Go to source → Data management → Object replication*
- *Add destination account*
- *Choose source & destination containers*

Test by uploading a blob to source; it replicates to the destination.

8. Create File Share and Test

```
az storage share create \
  --name demofileshare \
  --account-name mystorageacc12345
```

Upload file:

```
az storage file upload \
  --share-name demofileshare \
  --source demo.txt \
  --path demo.txt \
  --account-name mystorageacc12345
```

Configure Azure File Sync

1. *Install Azure File Sync Agent on the Windows Server*
2. *Register the server in Azure*
3. *Create a Sync Group in Azure Portal*
4. *Add Storage Sync Service*
5. *Create cloud endpoint (linked to file share)*
6. *Create server endpoint (local folder)*

Task 2.10
(Is repeated, Same as Task 2.9)