Practical Task 1: Linux Virtual Machine Setup and NSG Configuration

Create and configure a Linux Virtual Machine (VM) on Azure and secure it with a Network Security Group (NSG).

Requirements:

1. Create a Linux VM (Ubuntu or CentOS) in Azure using the free tier.
2. Connect to the VM via SSH using a public-private key pair.
3. Install and configure an Nginx web server on the VM.
4. Create and configure a Network Security Group (NSG) to allow only HTTP (port 80) and SSH (port 22) traffic.
5. Test access to the Nginx web server from a browser.
6. Verify that any other ports are blocked by the NSG.

Practical Task 2: Windows Virtual Machine and RDP Access Setup

Set up a Windows Virtual Machine (VM) on Azure and configure access via Remote Desktop Protocol (RDP).

Requirements:

1. Create a Windows VM (e.g., Windows Server 2019) in Azure using the free tier.
2. Enable and configure Remote Desktop Protocol (RDP) for secure access to the VM.
3. Connect to the VM via RDP using Azure credentials.
4. Install a web server role (IIS) and deploy a simple test HTML page.
5. Verify access to the test page from a browser.
6. Ensure that unnecessary ports are closed, allowing only RDP (port 3389) and HTTP (port 80).

Practical Task 3: Configuring an Azure Load Balancer

Create and configure a **Basic Azure Load Balancer** to distribute traffic across multiple virtual machines.

Requirements:

1. Create two Linux or Windows virtual machines in the same region and virtual network using the Azure Free Tier.
2. Install and configure a web server (e.g., Nginx on Linux or IIS on Windows) on both VMs with unique content for testing.

3. Create a **Basic Load Balancer** in Azure (included in the free tier) and configure it to balance HTTP (port 80) traffic between the two virtual machines.
4. Configure a health probe to monitor the availability of the VMs.
5. Test the Load Balancer by accessing its public IP address from a browser and verify that traffic is routed to both VMs (by observing the unique content from each server).
6. Verify that the Load Balancer removes unavailable VM from the traffic pool when it fails the health probe.

Practical Task 4: Configuring a Basic Load Balancer with Virtual Machine Scale Sets (VMSS)

Set up a **Basic Azure Load Balancer** to distribute traffic across a Virtual Machine Scale Set (VMSS).

Requirements:

1. Create a Virtual Machine Scale Set (VMSS) in Azure using Linux or Windows instances within the free tier (for example **B1s size**). Limit the scale set to two VM instances to avoid exceeding the free-tier 750-hour limit.
2. Deploy the scale set with a custom configuration to install and configure a web server (e.g., Nginx on Linux or IIS on Windows) on each VM instance.
3. Configure the **Basic Load Balancer** to distribute HTTP (port 80) traffic across the VM instances in the scale set.
4. Add a health probe to monitor the availability of instances in the VMSS.
5. Scale the VMSS manually by increasing the number of instances to verify the Load Balancer routes traffic to the newly added VMs.
6. Test the setup by accessing the Load Balancer's public IP address and verifying traffic distribution across multiple VM instances.
7. Verify that the Load Balancer removes an unavailable instance from the traffic pool when it fails the health probe.

Practical Task 5: Deploying a Web Application Using Azure App Services

Set up and deploy a simple web application using **Azure App Services**.

Requirements:

1. Create an Azure App Service (Web App) using the Azure Free Tier.
2. Select the runtime stack of your choice (e.g., .NET, Python, Node.js) during the setup.
3. Develop or use a sample web application (e.g., a "Hello World" app) and deploy it to the App Service using:

- Azure Portal
- Azure CLI
- Or direct deployment from a GitHub repository.

4. Test the deployed application by accessing its URL provided by Azure App Services.
5. Enable **App Service Logs** and verify that application logs are being generated.
6. Clean up resources after completion to avoid unnecessary usage.

Practical Task 6: Creating and Deploying an Azure Function to Process HTTP Requests

Set up and deploy an Azure Function that processes HTTP requests directly from the Azure portal.

Requirements:

1. **Set Up the Function App**:
   - Access the Azure portal and navigate to **Azure Functions**.
   - Create a new **Function App** using the Consumption (Serverless) plan within the free tier.
2. **Create a Function**:
   - Select the **HTTP trigger** template.
   - Choose a language of your choice (e.g., C#, JavaScript, or Python).
3. **Customize the Function**:
   - Modify the default code to return the following response:

     Hello, [name]! Welcome to Azure Functions.
   - If no name parameter is provided in the query string or request body, the response should be:

     Hello! Please provide your name.
4. **Test the Function**:
   - Use the **Test/Run** feature in the Azure portal to send HTTP requests.
   - Verify the Function responds appropriately with and without the name parameter.
5. **Verify External Access**:
   - Retrieve the public URL of the Azure Function.
   - Test the Function using a browser or a tool like Postman to ensure it's externally accessible.

6. **Monitor and Inspect**:
    - Navigate to the **Monitor** tab of the Function App.
    - Check metrics like execution count, response time, and errors