Practical Task 1: Install, Configure, and Manage Terraform State in Azure

Requirements:

- Install Terraform on your local machine.
- Verify the installation by checking the Terraform version.
- Authenticate with Azure using **az login** and configure Terraform for Azure authentication.
- Create a **Terraform backend** configuration using an **Azure Storage Account** to store the Terraform state remotely:
  - Define a storage account, a container, and a blob in **Terraform configuration**.
  - Use **terraform init** to initialize the backend.
  - Run **terraform apply** to deploy the storage account for state management.
  - Verify that the Terraform state file is stored in the Azure Storage Account.
  - Implement basic **state locking** using Azure blob storage.
  - Destroy the storage account (after confirming the state behavior).

Practical Task 2: Deploy an Azure Virtual Machine with a Custom Network and Security Rules

Requirements:

Extend the Terraform configuration to deploy:

- An **Azure Virtual Network (VNet)** with a **custom subnet**.
- A **Network Security Group (NSG)** with the following rules:
  - Allow **SSH (port 22) inbound** for a specific IP range.
  - Allow **HTTP (port 80) inbound** for all users.
  - Deny all other inbound traffic.
- A **Public IP Address** assigned to the VM.
- An **Azure Virtual Machine (VM)** using an Ubuntu image, attached to the subnet and NSG.
- A **Terraform output variable** to display the public IP of the VM after deployment.
- Use **Provisioners** to run a startup script that installs and starts an Nginx web server on the VM.
- Verify:
  - That SSH access works for the specified IP range.
  - That the Nginx web page is accessible via the VM's public IP.
- Destroy the infrastructure when complete.

Practical Task 3: Implement a Scalable Infrastructure with Load Balancer and Auto Scaling

Requirements:

Extend the Terraform configuration to create a **highly available infrastructure** by deploying:

- A **Virtual Network (VNet)** with multiple subnets across **two Azure Availability Zones**.
- An **Azure Load Balancer** with:
  - A **backend pool** of multiple **Virtual Machines (VMs)**.

    - A **health probe** for HTTP on port 80.
    - A **load-balancing rule** to distribute traffic across VMs.
- A **Virtual Machine Scale Set (VMSS)** with:
    - At least **two VM instances** that auto-scale based on CPU usage.
    - A startup script to install **Apache** and deploy a sample website.
- A **Storage Account** to store Terraform state remotely.
- Verify that:
    - The **Load Balancer IP** distributes traffic between VM instances.
    - Auto-scaling works when CPU usage spikes.
- Implement **Terraform modules** to modularize networking, compute, and security configurations.
- Destroy the infrastructure when testing is complete.

Practical Task 4: Install and Configure Ansible for Azure

Requirements:

- Install **Ansible** on your local machine.
- Verify the installation by checking the Ansible version.
- Install the **Azure Ansible Collection**.
- Authenticate Ansible with Azure using **service principal authentication**:
    - Create an **Azure Active Directory Service Principal** with the necessary permissions.
    - Retrieve the **client ID, tenant ID, and secret key**.
    - Store credentials securely in an **Ansible Vault**.
- Write a basic **Ansible inventory file** that defines Azure as the target environment.
- Create a simple **Ansible playbook** that:
    - Retrieves a list of all Azure resource groups.
    - Prints the result as output.
    - Execute the playbook and verify the output.

Practical Task 5: Deploy an Azure Virtual Machine with Ansible

Requirements:

Extend the Ansible configuration to **deploy an Azure Virtual Machine (VM)**:

- Create an **Ansible playbook** that:
    - Defines an **Azure Virtual Network (VNet)** and a **Subnet**.
    - Creates a **Network Security Group (NSG)** with rules:
        - Allow **SSH (port 22) inbound** from a specified IP.
        - Allow **HTTP (port 80) inbound** for all users.
    - Deploys an **Ubuntu VM** in the subnet with an attached **public IP address**.
    - Uses **SSH key-based authentication** for the VM.
    - Write an **Ansible role** to configure the VM by:

  - Installing **Nginx** and starting the service.
  - Copying a custom **HTML file** to serve as the default web page.
- Run the playbook and verify:
  - The VM is successfully deployed in Azure.
  - Nginx is running and the custom webpage is accessible via the public IP.
- Implement **idempotency** by running the playbook multiple times and ensuring no unintended changes occur.

Practical Task 6: Deploy a Scalable Azure Infrastructure with Ansible and Dynamic Inventory (optional)

Requirements:
Extend the Ansible configuration to deploy a **highly available infrastructure** with:

- **Azure Load Balancer** to distribute traffic across multiple VMs.
- **Virtual Machine Scale Set (VMSS)** with at least **three VM instances** running Ubuntu.
- A **custom Ansible role** to configure each VM in the scale set:
  - Install **Docker** and run a containerized web application.
  - Configure **UFW (Uncomplicated Firewall)** to allow necessary ports.
- **Azure Dynamic Inventory** to automatically manage VM instances in Ansible.
- Implement a **rolling update strategy** with Ansible:
  - Update the web application without downtime.
  - Ensure only **one VM updates at a time** using serial and delay settings.
- Implement **Ansible Tower/AWX** for centralized playbook execution and monitoring.
- Verify that:
  - The load balancer distributes traffic evenly.
  - The scale set auto-scales based on CPU usage.
  - The rolling update mechanism works correctly.
- Implement **Ansible Vault** for secure credential storage.
- Tear down the infrastructure using Ansible after verification.

Practical Task 7: Deploy a Resource Group Using an ARM Template

Requirements:

- Create a **JSON-based ARM Template** that defines a **resource group** named ARMResourceGroup in the **East US** region.
- Use **az deployment sub create** to deploy the resource group using the ARM template.
- Verify the deployment in the Azure Portal or using the Azure CLI (**az group list**).
- Modify the template to add **tags** to the resource group and redeploy it.
- Remove the resource group after verification.

Practical Task 8: Deploy an Azure Storage Account Using an ARM Template

Requirements:

Extend the ARM template to define an **Azure Storage Account** with:

- A **unique name** and **StorageV2** account type.
- **Standard_LRS** as the replication type.
- Disable public access to the storage account by setting the public access level to Private for all containers.
- Add **parameters** for the storage account name and location.
- Deploy the template using **az deployment group create**.
- Validate the deployment using **az storage account list**.
- Modify the template to enable **Blob soft delete**, then redeploy it.
- Delete the storage account when done.

Practical Task 9: Terraform: Deploy a Production-Ready AKS Cluster with GitOps & Secret Management & Monitoring

1. Deploy an AKS cluster with Terraform, ensuring production readiness.
2. Integrate ArgoCD for GitOps-based application deployment.
3. Set up an Ingress Controller for ArgoCD access.
4. Enable Azure Key Vault CSI Driver for secure secrets management.
5. Deploy an application via ArgoCD, using a Git repository as the source.
6. Implement application health checks in ArgoCD using Kubernetes readiness/liveness probes.
7. Configure automated sync policies in ArgoCD to enable self-healing and pruning of outdated resources.
8. Enable Monitoring and Logging using:

- Azure Monitor & Log Analytics for cluster-wide observability
- Prometheus & Grafana for in-depth Kubernetes metrics
- Container Insights for real-time pod and node monitoring

Practical Task 10: Deploy a Virtual Machine with Networking Using an ARM Template

Requirements:
Create an ARM template that deploys:

- An **Azure Virtual Network (VNet)** with a **custom subnet**.
- A **Network Security Group (NSG)** allowing SSH and HTTP traffic.
- A **Virtual Machine (VM)** running **Ubuntu 20.04**.
- A **Public IP Address** assigned to the VM.
- Use **parameters** for VM name, admin username, and authentication type.
- Deploy the template and verify:
  - The VM is accessible via SSH.
  - The public IP is assigned correctly.
- Modify the template to enable **boot diagnostics** and redeploy.
- Delete the VM and associated resources after verification.

Azure Bicep Tasks

Convert an ARM Template to Bicep

Requirements:

Take an existing **ARM Template** (e.g., from Task 8) and convert it to **Azure Bicep** format using:

- **az bicep decompile --file <arm-template.json>**
- Refactor the Bicep template to:
  - Remove unnecessary metadata.
  - Use **variables** instead of hardcoded values.
  - Use **parameters** for the storage account name, SKU, and location.
- Deploy the Bicep file using **az deployment group create**.
- Validate the deployment and compare it to the original ARM deployment.
- Delete the storage account after testing.

Practical Task 11: Deploy a Multi-Resource Azure Infrastructure Using Bicep

Requirements:
Create an **Azure Bicep** file that deploys:

- A **Virtual Network (VNet)** with multiple **subnets**.

- A **Storage Account** for VM diagnostics.
- A **Linux Virtual Machine** with SSH authentication.
- A **Network Security Group (NSG)** with restricted SSH access.
- Implement **modules** in Bicep for:
  - Networking
  - Virtual Machine Deployment
- Deploy the Bicep file and verify:
  - The VM is reachable via SSH.
  - The NSG allows the correct traffic.
- Modify the configuration to increase VM size and redeploy.
- Destroy the infrastructure after verification.

Practical Task 12: Implement Parameterization and Secrets Management in Bicep (optional)

Requirements:
Extend the **Bicep configuration** to include:

- **Parameter files (.bicepparam)** for different environments (e.g., Dev, Prod).
- A **Key Vault** to store sensitive values such as admin credentials.
- Securely retrieve the admin password from **Azure Key Vault** using @secure() annotation in Bicep.
- Deploy the infrastructure with environment-specific parameters.
- Implement **conditional logic** in Bicep to:
  - Deploy a different VM size for **Prod vs. Dev**.
  - Enable different security rules based on the environment.
- Verify that:
  - The correct configuration is applied for each environment.
  - The VM retrieves its credentials securely from Key Vault.
- Clean up the deployment.

Azure Monitor Tasks

Practical Task 13: Configure Azure Monitor to Track VM Metrics and Alerts

Requirements:

- Enable **Azure Monitor** for a Virtual Machine (VM).
- Configure monitoring for the following **metrics**:
  - CPU utilization
  - Disk read/write operations
  - Network In/Out
- Set up an **alert rule** that triggers when CPU utilization exceeds **80% for 5 minutes**.
- Configure the alert to send a **notification to an email address**.
- Verify the alert trigger by running a CPU-intensive process on the VM.

- Delete the alert rule after testing.

Practical Task 14: Create a Custom Dashboard in Azure Monitor (optional)

Requirements:

- Navigate to **Azure Monitor Dashboards**.
- Create a **custom dashboard** that includes:
  - A **line graph** for CPU usage of a Virtual Machine.
  - A **bar chart** for disk utilization across multiple VMs.
  - A **table** showing the top 5 VMs consuming the most network bandwidth.
- Customize the dashboard layout and apply filtering options.
- Share the dashboard with another Azure user (with Reader access).
- Save and export the dashboard configuration for reuse.

Application Insights Tasks

Practical Task 15: Enable Application Insights for a Web Application

Requirements:

- Enable **Application Insights** for an existing **Azure Web App**.
- Configure **automatic instrumentation** for the application (for .NET, Node.js, or Python).
- Monitor the following **performance metrics**:
  - Response time
  - Request count
  - Failed request rate
- Generate load on the application using Apache JMeter or a similar tool.
- View the performance metrics in **Application Insights**.
- Set up an **alert** for high response times (above 2 seconds).

Practical Task 16: Analyze Application Telemetry and Dependency Tracking

Requirements:
Use **Application Insights** to:

- Enable **Live Metrics Stream** for real-time monitoring.
- Capture **custom events and telemetry** from an application.
- Track **dependencies** (database calls, external API calls).
- Write a simple **Kusto Query** to retrieve and analyze:
  - The slowest 10 requests in the last **24 hours**.
  - The most frequently failing requests.
- Visualize the results in **Application Insights Workbooks**.

Log Analytics Tasks

Practical Task 17: Query and Analyze Azure Logs with Kusto Query Language (KQL)

Requirements:

- Connect **Azure Log Analytics** to a **Virtual Machine**.
- Use **Azure Monitor Logs** to ingest system logs.
- Write **basic KQL queries** to analyze logs:
  - Retrieve all logs from the last **3 hours**.
  - Find failed login attempts on the VM.
- Identify the top **processes consuming CPU resources**.
- Create a **scheduled query rule** to trigger an alert when a process exceeds **90% CPU usage**.
- Export the query results to a CSV file for reporting.

Practical Task 18: Implement Advanced Log Analytics Queries and Alerts (optional)

Requirements:

- Extend Log Analytics to monitor an **Azure Kubernetes Service (AKS) cluster**.
- Configure logs to capture:
  - **Pod crashes**
  - **Memory and CPU utilization per container**
  - **Network failures between services**
- Write **advanced KQL queries** to:
  - Detect trends in pod failures over time.
  - Identify the most resource-intensive services in AKS.
- Generate an **anomaly detection report** based on historical log patterns.
- Create an **alert rule** that triggers when the number of pod restarts exceeds **5 per minute**.
- Integrate the alert with **Azure Logic Apps** to automatically restart failing pods.