PRACTICAL TEST 1

Title:

Creating and Configuring a Virtual Machine (VM) on Google Cloud for Internal Web Application

---

AIM:

To create, configure, and secure a Virtual Machine (VM) instance in Google Cloud Platform (GCP) with high availability, scalability, and reliability to host an internal web-based tool for approximately 200 concurrent users.

REQUIREMENTS:

- Google Cloud Platform account
- Compute Engine enabled
- Internet connection
- Web browser (Chrome/Firefox)
- SSH access permission
- Basic knowledge of Linux commands

THEORY:

A **Virtual Machine (VM)** in cloud computing is a virtualized environment that behaves like a physical computer. In **Google Cloud**, VM instances are managed through **Compute Engine**, which provides scalable and flexible infrastructure-as-a-service (IaaS).

Key Concepts:

1. **Machine Type:** Defines the number of CPUs and memory resources.
2. **Image:** The base operating system used to boot the VM (e.g., Ubuntu, Debian).
3. **Region and Zone:** Determines where the VM resources are physically located. Choosing the right region ensures low latency and high availability.
4. **Firewall Rules:** Controls incoming and outgoing network traffic.
5. **Availability Policy:** Includes automatic restart and preemption settings to maintain uptime.

Creating a properly configured VM ensures that applications remain accessible even under heavy load, with optimal performance and security.

PROCEDURE:

Step 1: Login to Google Cloud Console

1. Visit https://console.cloud.google.com.
2. Log in using your Google account.
3. Ensure that the **Compute Engine API** is enabled.

Step 2: Create a New VM Instance

1. Navigate to **Compute Engine → VM Instances → Create Instance.**
2. Enter the instance name as internal-web-vm.
3. **Region:** Select us-central1 for reliability and availability.
4. **Zone:** Choose us-central1-a (physically isolated zone for fault tolerance).

Step 3: Configure Machine Type

- Choose **Machine Type:** e2-standard-4 (4 vCPUs, 16 GB RAM).
  This configuration supports approximately 200 concurrent employees.

Step 4: Select Operating System

- Under **Boot Disk**, click **Change** → Select **Ubuntu 22.04 LTS** (stable and lightweight).
- Disk Type: Standard Persistent Disk (default 20 GB).

Step 5: Configure Firewall and Network

- Check both **Allow HTTP traffic** and **Allow HTTPS traffic** options.
- Network: Default VPC (or a custom network if required).
- Subnetwork: Automatically selected.

Step 6: Configure Availability Policy

- Enable **Automatic Restart** to ensure the VM restarts after maintenance or unexpected shutdowns.
- Leave Preemptibility disabled for continuous uptime.

Step 7: Create the VM

- Review all configurations.
- Click **Create**.
- Wait for the VM instance to start.

Step 8: Connect to the VM via SSH

- Click **SSH** on the VM instance details page.
- Verify the connection by running:
- hostname
- uname -a

Output confirms the system is running.

Step 9: Install Web Server

Inside the VM terminal:

sudo apt update

sudo apt install apache2 -y

sudo systemctl start apache2

sudo systemctl enable apache2

Test by opening the external IP in a browser → displays “Apache2 Default Page.”

Step 10: Secure the VM

- Configure OS firewall:
- sudo ufw allow 'Apache Full'
- sudo ufw enable
- Disable root SSH login (for added security).

OUTPUT:

1. VM instance successfully created and running.
2. Apache web server hosted and reachable through HTTP.
3. Verified high availability through auto-restart configuration.
4. Secured using both GCP and OS-level firewalls.

ADVANTAGES:

- Scalable on demand.
- High availability through Google’s global infrastructure.
- Simplified deployment using prebuilt images.
- Secure access through IAM and firewall rules.

RESULT:

Successfully created, configured, and secured a Virtual Machine instance in Google Cloud for internal application hosting with high performance and availability.

VIVA QUESTIONS:

1. **What is a Virtual Machine in GCP?**
   A VM is a virtual server hosted in Google's data center that runs workloads like a physical computer.
2. **Why is the region and zone important?**
   They determine latency, fault tolerance, and physical resource location.
3. **What is the purpose of firewall rules?**
   They control network traffic to and from the VM, ensuring security.
4. **What does automatic restart do?**
   It ensures the VM restarts automatically after a system failure or maintenance.
5. **How can you connect securely to a VM?**
   By using SSH keys or Google-managed SSH connections.

PRACTICAL TEST 2

Title:

Deploying Load Balanced and Autoscaling Virtual Machines in Google Cloud

---

AIM:

To create and configure multiple identical VM instances behind an HTTP Load Balancer and enable Autoscaling in Google Cloud to efficiently handle web traffic based on demand.

REQUIREMENTS:

- Google Cloud account
- Compute Engine enabled
- Load Balancing and Monitoring APIs enabled
- Internet connection

THEORY:

Load Balancing distributes incoming network traffic across multiple backend instances, ensuring high availability and fault tolerance.
Autoscaling automatically adjusts the number of VM instances based on demand metrics (e.g., CPU utilization).

In Google Cloud, **Managed Instance Groups (MIGs)** work with load balancers to dynamically create or delete instances as traffic fluctuates, optimizing performance and cost.

Components Involved:

1. **Instance Template:** Blueprint of VM configuration.
2. **Managed Instance Group (MIG):** Group of identical VMs managed collectively.
3. **Load Balancer:** Distributes client requests across instances.
4. **Autoscaler:** Monitors and adjusts instance count automatically.

PROCEDURE:

Step 1: Create an Instance Template

1. Navigate to **Compute Engine → Instance Templates → Create Template.**
2. Name: web-template.
3. Machine Type: e2-medium.
4. Boot Disk: Ubuntu 22.04 LTS.
5. Firewall: Allow HTTP and HTTPS traffic.
6. Add **Startup Script:**
7. #! /bin/bash
8. apt update
9. apt install apache2 -y
10. systemctl start apache2
11. echo "Welcome to Cloud Engineering Load Balanced Server" > /var/www/html/index.html
12. Click **Create.**

Step 2: Create a Managed Instance Group

1. Navigate to **Instance Groups → Create Instance Group.**
2. Select **New Managed Instance Group.**
3. Name: web-mig.
4. Choose the previously created instance template.
5. Location Type: Single zone (us-central1-a).
6. Set **Autoscaling:**
    - Metric: CPU utilization
    - Target: 60%
    - Minimum instances: 1
    - Maximum instances: 5

7. Click **Create.**

Step 3: Configure Health Check

1. Go to **Network Services → Health Checks → Create Health Check.**
2. Type: HTTP
3. Port: 80
4. Path: /
5. Save and use this for load balancing backend.

Step 4: Create an HTTP Load Balancer

1. Navigate to **Network Services → Load Balancing → Create Load Balancer.**
2. Choose **HTTP(S) Load Balancer → Start Configuration.**
3. Backend Configuration:
    - Select the managed instance group (web-mig).
    - Add health check created previously.
4. Frontend Configuration:
    - Protocol: HTTP
    - Assign new static IP.
5. Review and click **Create.**

Step 5: Test Load Balancer

- Access the **frontend IP address** in a browser.
- Increase simulated load using:
- while true; do curl -s http://<load-balancer-ip>/ > /dev/null; done
- Observe new instances being created automatically under **Compute Engine → Instance Groups.**

OUTPUT:

1. Load balancer successfully distributes traffic among multiple VMs.
2. Autoscaler automatically creates or deletes instances.
3. Application maintains performance even under heavy load.

ADVANTAGES:

- Efficient traffic handling.
- Fault tolerance and high availability.
- Reduced cost through automatic scaling.
- Centralized management through MIG.

RESULT:

Successfully deployed a scalable and load-balanced web application using Managed Instance Group, HTTP Load Balancer, and Autoscaler in Google Cloud.

VIVA QUESTIONS:

1. What is a Managed Instance Group?
   → A collection of identical VMs managed as a single entity.
2. Why is load balancing needed?
   → To distribute workload evenly and prevent server overload.
3. What metrics can be used for autoscaling?
   → CPU utilization, request latency, or custom metrics.
4. What is a health check?
   → A periodic test to determine instance health and availability.
5. What is the main difference between static and dynamic scaling?
   → Static scaling is fixed; dynamic scaling adjusts automatically.

PRACTICAL TEST 3

Title:

Creating a Secure Cloud Storage Bucket for Web Application Image Uploads

---

AIM:

To create a secure Google Cloud Storage bucket for storing image uploads with restricted access permissions, ensuring that only authorized service accounts can read and write data.

REQUIREMENTS:

- Google Cloud account

- Cloud Storage API enabled
- gsutil CLI tool installed
- Web application service account

THEORY:

Cloud Storage provides scalable, durable, and secure object storage.
Buckets store data objects (like images or videos), and access can be controlled using **IAM roles** and **Access Control Lists (ACLs)**.

Security Features:

- **IAM roles:** Assign access permissions to users or service accounts.
- **Public Access Prevention:** Ensures no object is exposed to the internet.
- **gsutil CLI:** Allows management of buckets via command line.

PROCEDURE (Using Console):

1. Navigate to **Storage → Buckets → Create.**
2. Enter Bucket Name: secure-image-bucket.
3. Location: us-central1.
4. Storage Class: **Standard.**
5. Choose **Fine-grained Access Control.**
6. Disable **Public Access.**
7. Assign permissions:
    - Add **Service Account:** webapp@project-id.iam.gserviceaccount.com
    - Role: **Storage Object Admin.**
8. Click **Create.**

PROCEDURE (Using gsutil CLI):

Create bucket

gsutil mb -l us-central1 -c STANDARD gs://secure-image-bucket/

Prevent public access

gsutil iam ch allUsers:objectViewer- gs://secure-image-bucket/

Assign service account permission

gsutil iam ch serviceAccount:webapp@project-id.iam.gserviceaccount.com:objectAdmin gs://secure-image-bucket/

Verify configuration

gsutil iam get gs://secure-image-bucket/

OUTPUT:

1. Bucket created in us-central1.
2. Public access disabled.
3. Only service account can upload and retrieve files.

ADVANTAGES:

- Enhanced data security.
- Reliable storage for web apps.
- IAM-based role management.
- Easy access via gsutil or API.

RESULT:

Successfully created and configured a secure Cloud Storage bucket with restricted permissions and regional access.

VIVA QUESTIONS:

1. What is a Cloud Storage bucket?
   → A container for storing data objects in Google Cloud.
2. Why is IAM important in Cloud Storage?
   → To define who can access or modify bucket data.
3. How do you prevent public access?
   → Disable “Public Access” and avoid allUsers/allAuthenticatedUsers roles.
4. What command is used to create a bucket via CLI?
   → gsutil mb
5. What are the different storage classes?
   → Standard, Nearline, Coldline, and Archive.

PRACTICAL TEST 4

Title:

Setting up Cloud SQL MySQL Database and Securing Access

AIM:

To create a MySQL-based Cloud SQL instance and grant secure, restricted access to the application server using authorized networks and Secret Manager.

REQUIREMENTS:

- Google Cloud account
- Cloud SQL API enabled
- Secret Manager API enabled

THEORY:

Cloud SQL is a managed database service supporting MySQL, PostgreSQL, and SQL Server. It automates backups, replication, and patching.

To ensure security:

- Use **Private IPs** or **Authorized Networks** for connection.
- Manage credentials with **Secret Manager**.

PROCEDURE:

1. Navigate to **SQL → Create Instance.**
2. Choose **MySQL** → Click **Next.**
3. Instance ID: prod-mysql-instance.
4. Region: us-central1.
5. Database Version: MySQL 8.0.
6. Machine Type: db-n1-standard-2.
7. Storage: 20 GB SSD.
8. Enable **Automatic Backups**.
9. Under **Connections → Authorized Networks**, add app server's IP.
10. Click **Create.**

After Creation:
11. Open **Cloud Shell** → connect to SQL:

gcloud sql connect prod-mysql-instance --user=root

12. Create a database and user:

CREATE DATABASE appdb;

CREATE USER 'appuser'@'%' IDENTIFIED BY 'securepass';

GRANT ALL PRIVILEGES ON appdb.* TO 'appuser'@'%';

FLUSH PRIVILEGES;

13. Store password securely:

echo "securepass" | gcloud secrets create db-password --data-file=-

14. Application retrieves credentials using IAM-secured access.

OUTPUT:

1. Cloud SQL instance running MySQL 8.0.
2. Database and user created.
3. Access restricted to app server.
4. Password stored securely in Secret Manager.

ADVANTAGES:

- Fully managed database service.
- Automatic backups and scaling.
- Strong network and data security.
- Simplified connection via private IP.

RESULT:

Successfully created and configured a secure Cloud SQL MySQL instance and integrated it with Secret Manager for credential protection.

VIVA QUESTIONS:

1. What is Cloud SQL?
   → A managed relational database service in GCP.

2. How to connect securely to Cloud SQL?
   → Using Private IPs or authorized networks.
3. What is Secret Manager used for?
   → Securely store and access sensitive data like passwords.
4. Why use MySQL 8.0?
   → It provides advanced performance and security features.
5. What command connects to Cloud SQL?
   → gcloud sql connect <instance-id> --user=root

PRACTICAL TEST 5

Title:

Designing a Private VPC Network for Internal-only Communication in Google Cloud

---

AIM:

To create a secure and isolated VPC network where internal Google Cloud resources can communicate privately without external internet access.

REQUIREMENTS:

- Google Cloud account
- VPC Network API enabled
- Internet access (for setup only)

THEORY:

A **Virtual Private Cloud (VPC)** allows private networking within Google Cloud.
It provides complete control over IP address ranges, routing, and firewall rules.

Private VPC Design:

- Custom Subnet (private IPs only)
- Disabled External IPs
- Internal Firewall rules
- Optional Private Google Access for API communication

PROCEDURE:

1. Navigate to **VPC Network → Create VPC Network.**
2. Name: internal-secure-vpc.
3. Subnet:
    - Name: private-subnet
    - Region: us-central1
    - IP Range: 10.0.0.0/16
4. Disable **Private Google Access** initially (optional).
5. Click **Create.**

Step 2: Create Firewall Rules

1. Allow internal communication:
2. gcloud compute firewall-rules create allow-internal \
3. --network=internal-secure-vpc \
4. --allow tcp,udp,icmp \
5. --source-ranges=10.0.0.0/16
6. Deny internet access:
7. gcloud compute firewall-rules create deny-internet \
8. --network=internal-secure-vpc \
9. --direction=EGRESS \
10. --action=DENY \
11. --destination-ranges=0.0.0.0/0

Step 3: Create VM Instances

1. Navigate to **Compute Engine → Create Instance.**
2. Name: private-vm1.
3. Network: internal-secure-vpc.
4. Subnet: private-subnet.
5. **External IP:** None (disabled).
6. Create another VM (private-vm2) with the same settings.

Step 4: Enable Private Google Access (Optional)

1. Go to **VPC → Subnets → private-subnet.**
2. Enable **Private Google Access.**
3. VMs can now reach Google APIs securely without internet exposure.

OUTPUT:

- Two VMs communicate via private IPs.
- No external access possible.
- Secure internal-only network established.

ADVANTAGES:

- Enhanced internal security.
- Prevents data leaks through external connections.
- Enables API access through Private Google Access.

RESULT:

Successfully implemented a private, secure VPC network that allows only internal communication with no internet exposure.

VIVA QUESTIONS:

1. What is a VPC network?
   → A virtual network providing isolated and secure communication in GCP.
2. What is the purpose of firewall rules?
   → To control inbound and outbound network traffic.
3. How to disable external internet access?
   → By not assigning external IPs and creating DENY egress rules.
4. What is Private Google Access?
   → Allows VMs without external IPs to access Google APIs internally.
5. Why prefer custom VPCs over default VPCs?
   → For better security, control, and IP management.