EXPERIMENT NUMBER 1

Creating a Virtual Machine: Configure and deploy a virtual machine with specific CPU and memory requirements in Google Cloud

Step 1: Log in to Google Cloud Console

1. Go to Google Cloud Console.
2. Make sure you have a **Google Cloud account** and a **project** set up.

Step 2: Open Compute Engine

1. In the **Navigation Menu**, go to **Compute Engine** > **VM instances**.
2. Click **Create Instance**.

Step 3: Configure VM Instance

- **Name**: Give your VM a recognizable name.
- **Region & Zone**: Choose a region (e.g., `us-central1`) based on your needs.
- **Machine Configuration**:
  - Click **Machine type**.
  - Choose a predefined machine type or customize:
    - **Custom (vCPUs & Memory)**: Set the desired number of CPUs and RAM.
  - Example: If you need **4 vCPUs and 16 GB RAM**, select a **custom machine type** and set:
    - `vCPU: 4`
    - `Memory: 16 GB`

Step 4: Select Boot Disk

- Choose an OS (e.g., **Ubuntu, Debian, Windows Server**).
- Adjust **Boot disk size** if needed.

Step 5: Configure Networking (Optional)

- Allow HTTP/HTTPS traffic if needed.
- Modify firewall rules as required.

Step 6: Create and Deploy

- Click **Create**.
- Wait for the VM to be provisioned.

Step 7: Connect to the VM

- In the **VM instances list**, click **SSH** to connect directly from the browser.

Step 8: Verify Resources

Run the following command inside your VM to check CPU and memory:

```
sh
CopyEdit
lscpu   # Check CPU details
free -h # Check memory usage
```

Additional Configurations (Optional)

- **Install software**: Use `apt` or `yum` depending on the OS.
- **Enable auto-start**: Configure startup scripts if required.
- **Attach storage**: Add persistent disks if needed.

EXPERIMENT NUMBER 2

Getting Started with Cloud Shell and gcloud

What is Cloud Shell?

Cloud Shell is an interactive command-line environment in Google Cloud that lets you manage your resources using the `gcloud` command-line tool. It provides a pre-configured environment with essential Google Cloud tools.

Step 1: Open Cloud Shell

1. Go to the Google Cloud Console.
2. Click the **Cloud Shell icon** (⬛) in the top-right corner.
3. A terminal window will open at the bottom of the screen.

Step 2: Verify gcloud is Installed

Cloud Shell comes with `gcloud` pre-installed. Verify the installation with:

```
sh
CopyEdit
gcloud --version
```

You should see output similar to:

```
yaml
CopyEdit
Google Cloud SDK 123.0.0
bq 2.0.75
core 2023.10.01
gsutil 5.3
```

Step 3: Authenticate and Set Up a Project

Authenticate Google Cloud SDK

If not already authenticated, run:

```
sh
CopyEdit
gcloud auth login
```

This opens a browser window where you grant permissions.

Set the Active Project

To set your project, first list available projects:

```
sh
CopyEdit
gcloud projects list
```

Then, set a project:

```
sh
```

```
CopyEdit
gcloud config set project PROJECT_ID
```
Replace `PROJECT_ID` with your actual project ID.

Step 4: Common gcloud Commands

1. List Compute Engine Instances

```
sh
CopyEdit
gcloud compute instances list
```
This shows all virtual machines (VMs) in the current project.

2. Create a VM Instance

```
sh
CopyEdit
gcloud compute instances create my-vm \
    --machine-type=e2-medium \
    --image-family=debian-11 \
    --image-project=debian-cloud \
    --zone=us-central1-a
```

- `my-vm` → Name of the VM
- `--machine-type=e2-medium` → Defines CPU and memory
- `--image-family=debian-11` → OS Image
- `--zone=us-central1-a` → The zone where the VM is created

3. SSH into the VM

```
sh
CopyEdit
gcloud compute ssh my-vm --zone=us-central1-a
```

4. Delete a VM

```
sh
CopyEdit
gcloud compute instances delete my-vm --zone=us-central1-a
```

Step 5: Manage Storage Buckets

1. List Storage Buckets

```
sh
CopyEdit
gcloud storage buckets list
```

2. Create a New Storage Bucket

```
sh
CopyEdit
gcloud storage buckets create my-bucket --location=us-central1
```

3. Upload a File to a Bucket

```
sh
CopyEdit
gcloud storage cp myfile.txt gs://my-bucket/
```

4. List Files in a Bucket

```
sh
CopyEdit
gcloud storage ls gs://my-bucket/
```

5. Delete a Bucket

```
sh
CopyEdit
gcloud storage buckets delete my-bucket
```

Step 6: Managing IAM (Permissions)

1. List IAM Policies

```
sh
CopyEdit
gcloud projects get-iam-policy PROJECT_ID
```

2. Add an IAM Role to a User

```
sh
CopyEdit
gcloud projects add-iam-policy-binding PROJECT_ID \
    --member="user:example@gmail.com" \
    --role="roles/editor"
```

3. Remove an IAM Role

```
sh
CopyEdit
gcloud projects remove-iam-policy-binding PROJECT_ID \
    --member="user:example@gmail.com" \
    --role="roles/editor"
```

Step 7: Configuring gcloud Defaults

Set default region and zone:

```
sh
CopyEdit
gcloud config set compute/region us-central1
gcloud config set compute/zone us-central1-a
```

To check all configurations:

```
sh
CopyEdit
gcloud config list
```

Step 8: Exit Cloud Shell

Simply **close the browser tab** or type:

```
sh
CopyEdit
exit
```

Conclusion

Cloud Shell and `gcloud` CLI make managing Google Cloud resources easy and efficient. You can create and manage VMs, storage, IAM policies, and more—all from the terminal.

EXPERIMENT NUMBER 3

Deploying a Cloud Function to Automate a Task Based on a Cloud Storage Event

Google Cloud Functions allows you to run serverless code in response to Cloud Storage events (such as file uploads, deletions, or updates). In this guide, we will create and deploy a **Cloud Function** that triggers when a file is uploaded to a **Cloud Storage bucket**.

Step 1: Enable Required APIs

Before deploying a Cloud Function, enable the required APIs:

```
sh
CopyEdit
gcloud services enable cloudfunctions.googleapis.com storage.googleapis.com
```

Step 2: Create a Cloud Storage Bucket

If you don't already have a Cloud Storage bucket, create one:

```
sh
CopyEdit
gcloud storage buckets create my-cloud-function-bucket --location=us-central1
```

Replace `my-cloud-function-bucket` with your bucket name.

Step 3: Write the Cloud Function Code

Create a new directory and navigate into it:

```
sh
CopyEdit
mkdir cloud-function-storage && cd cloud-function-storage
```

Create a **Python script (`main.py`)** for the function:

```
python
```

```
CopyEdit
import functions_framework

@functions_framework.cloud_event
def process_file(event):
    """Triggered by a file upload to a Cloud Storage bucket."""
    bucket = event.data["bucket"]
    file_name = event.data["name"]
    print(f"File {file_name} uploaded to {bucket}")
```

Create a **requirements file (`requirements.txt`)** to specify dependencies:

```
pgsql
CopyEdit
functions-framework
```

Step 4: Deploy the Cloud Function

Deploy the function using the `gcloud` CLI:

```
sh
CopyEdit
gcloud functions deploy process_file \
    --runtime python310 \
    --trigger-event google.storage.object.finalize \
    --trigger-resource my-cloud-function-bucket \
    --entry-point process_file \
    --region us-central1 \
    --gen2
```

Explanation of Parameters:

- `process_file` → Name of the function.
- `--runtime python310` → Runtime environment (Python 3.10).
- `--trigger-event google.storage.object.finalize` → Trigger event (fires when a file is uploaded).
- `--trigger-resource my-cloud-function-bucket` → Storage bucket to monitor.
- `--entry-point process_file` → Function entry point.
- `--region us-central1` → Deployment region.
- `--gen2` → Deploys Cloud Function in **2nd generation (Gen 2)** for better performance.

Step 5: Test the Function

Upload a test file to the bucket:

```
sh
CopyEdit
gcloud storage cp test-file.txt gs://my-cloud-function-bucket/
```

Check the function logs to verify execution:

```
sh
CopyEdit
gcloud functions logs read process_file --region=us-central1
```

Step 6: Update or Delete the Function

Update the Function

If you modify `main.py`, redeploy with:

```
sh
CopyEdit
gcloud functions deploy process_file --region=us-central1
```

Delete the Function

```
sh
CopyEdit
gcloud functions delete process_file --region=us-central1
```

Conclusion

You have successfully created and deployed a **Cloud Function** that triggers when a file is uploaded to **Cloud Storage**. You can extend this by processing images, sending notifications, or integrating with other Google Cloud services.

EXPERIMENT NUMBER 4

Deploying a Web Application on Google App Engine with Automatic Scaling

Google **App Engine** is a fully managed serverless platform that allows you to deploy and scale web applications automatically. In this guide, we will deploy a simple **Flask web application** on **App Engine (Standard Environment)** with automatic scaling enabled.

Step 1: Enable App Engine API

Before deploying, enable the App Engine API:

```
sh
CopyEdit
gcloud services enable appengine.googleapis.com
```

Then, initialize App Engine for your project:

```
sh
CopyEdit
gcloud app create --region=us-central
```

Replace `us-central` with your preferred region.

Step 2: Create a Simple Web Application

Create a new project directory and navigate into it:

```
sh
CopyEdit
mkdir app-engine-app && cd app-engine-app
```

Create a **Python web application** using Flask:

1. Install Flask

```
sh
CopyEdit
pip install flask
```

2. Create `main.py` (Web Application)

Create a Python file (`main.py`) with the following content:

```
python
CopyEdit
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, World! Welcome to App Engine with Auto Scaling."

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Step 3: Define App Engine Configuration

Create an `app.yaml` file to configure the App Engine environment:

```
yaml
CopyEdit
runtime: python310  # Define the runtime
entrypoint: gunicorn -b :$PORT main:app

automatic_scaling:
  min_instances: 1
  max_instances: 5
  target_cpu_utilization: 0.65
  target_throughput_utilization: 0.75

handlers:
- url: /.*
  script: auto
```

Explanation:

- **`runtime: python310`** → Specifies Python 3.10 as the runtime.
- **`entrypoint: gunicorn -b :$PORT main:app`** → Uses Gunicorn to run the app.
- **`automatic_scaling`**:
  - `min_instances: 1` → Keeps at least one instance running.
  - `max_instances: 5` → Scales up to 5 instances based on demand.
  - `target_cpu_utilization: 0.65` → Scales based on CPU load.
  - `target_throughput_utilization: 0.75` → Scales based on request load.
- **`handlers`** → Routes all incoming requests to the application.

Step 4: Install Dependencies

Create a `requirements.txt` file:

```
nginx
CopyEdit
Flask
gunicorn
```

Install the dependencies locally:

```
sh
CopyEdit
pip install -r requirements.txt
```

Step 5: Deploy the Web Application

Deploy your application using the following command:

```
sh
CopyEdit
gcloud app deploy
```

This will:

- Upload your application to **App Engine**.
- Set up automatic scaling.
- Deploy it with the configurations in `app.yaml`.

Step 6: Access the Application

Once deployed, open your app in a browser:

```
sh
CopyEdit
gcloud app browse
```

Alternatively, visit:
`https://<your-project-id>.appspot.com/`

Step 7: Monitor and Scale

1. Check App Engine Instances

```
sh
CopyEdit
gcloud app instances list
```

2. View Logs

```
sh
CopyEdit
gcloud app logs tail -s default
```

3. Update the App

Modify `main.py` or other files, then redeploy:

```
sh
CopyEdit
gcloud app deploy
```

4. Delete the App (If Needed)

```
sh
CopyEdit
gcloud app services delete default
```

Conclusion

You have successfully deployed a **Flask web application** on **Google App Engine** with **automatic scaling** enabled! 🚀
App Engine will now scale your app based on traffic demand automatically.

EXPERIMENT NUMBER 5

Qwikstart: Google Cloud Storage (GCS) 🚀

Google Cloud Storage (GCS) is a **scalable, secure, and durable object storage** solution that allows you to store and manage data efficiently. You can interact with GCS using the **Google Cloud Console**, **gsutil CLI**, or the **gcloud CLI**.

Step 1: Enable Cloud Storage API

Before using Cloud Storage, ensure the API is enabled:

```
sh
CopyEdit
gcloud services enable storage.googleapis.com
```

Step 2: Create a Cloud Storage Bucket

A **bucket** is a container for storing objects (files). Buckets have globally unique names.

Using the Google Cloud Console

1. Navigate to **Cloud Storage**
2. Click **Create bucket**
3. Enter a **globally unique name** (e.g., `my-unique-bucket-123`)
4. Select a **storage class** (e.g., **Standard** for frequent access)
5. Choose a **location** (e.g., `us-central1`)
6. Click **Create**

Using the gcloud CLI

```
sh
CopyEdit
gcloud storage buckets create my-bucket --location=us-central1
```

Replace `my-bucket` with your unique bucket name.

Step 3: Upload a File to Cloud Storage

Using Google Cloud Console

1. Open **Cloud Storage** in the **Google Cloud Console**
2. Click your **bucket**
3. Click **Upload files**
4. Select a file and upload

Using the gcloud CLI

```
sh
CopyEdit
gcloud storage cp myfile.txt gs://my-bucket/
```

Replace `myfile.txt` with your file and `my-bucket` with your bucket name.

Using gsutil (Cloud Storage CLI)

```
sh
CopyEdit
gsutil cp myfile.txt gs://my-bucket/
```

Step 4: List Files in a Bucket

Using the gcloud CLI

```
sh
CopyEdit
gcloud storage ls gs://my-bucket/
```

Using gsutil

```
sh
CopyEdit
gsutil ls gs://my-bucket/
```

Step 5: Download a File from Cloud Storage

Using the gcloud CLI

```
sh
CopyEdit
gcloud storage cp gs://my-bucket/myfile.txt .
```

Using gsutil

```
sh
CopyEdit
gsutil cp gs://my-bucket/myfile.txt .
```

Step 6: Delete a File or Bucket

Delete a File

```
sh
CopyEdit
gcloud storage rm gs://my-bucket/myfile.txt
```

Delete a Bucket (Ensure it's empty first)

```
sh
CopyEdit
gcloud storage buckets delete my-bucket
```

Step 7: Make a File Public (Optional)

To share a file publicly, set its permissions:

```
sh
CopyEdit
gcloud storage objects add-iam-policy-binding gs://my-bucket/myfile.txt \
    --role=roles/storage.objectViewer \
    --member=allUsers
```

Now, anyone can access it via:

```
perl
CopyEdit
https://storage.googleapis.com/my-bucket/myfile.txt
```

Conclusion

You have successfully **created a Cloud Storage bucket, uploaded files, listed objects, and managed storage using Cloud Console, gcloud, and gsutil!** 🎉

EXPERIMENT NUMBER 6

Cloud SQL for MySQL: Managed MySQL with High Availability in Google Cloud

Google **Cloud SQL for MySQL** is a **fully managed relational database** service that automates tasks such as **setup, maintenance, backups, and scaling**. It ensures **high availability** and integrates seamlessly with Google Cloud services.

Key Features of Cloud SQL for MySQL

✅ **Automated Backups & Patching** – No need to manage database updates manually.
✅ **High Availability (HA)** – Supports automatic failover in case of outages.
✅ **Automatic Scaling** – Adjusts resources based on workload demands.
✅ **Security** – Provides encryption, IAM-based access, and private IP support.
✅ **Integration** – Works with App Engine, Compute Engine, Kubernetes (GKE), and BigQuery.

Step 1: Enable Cloud SQL API

Before creating a database instance, enable the Cloud SQL API:

```
sh
CopyEdit
gcloud services enable sqladmin.googleapis.com
```

Step 2: Create a Cloud SQL for MySQL Instance

Using the Google Cloud Console

1. Go to **Cloud SQL**
2. Click **Create Instance**
3. Select **MySQL**
4. Choose a **MySQL version** (e.g., **MySQL 8.0**)
5. Enter an **instance ID** (e.g., `my-mysql-instance`)
6. Select a **region** and **zone**
7. Choose the **Machine Type** (e.g., 2 vCPUs, 8GB RAM)
8. Enable **High Availability (HA)** if required
9. Set a **root password**
10. Click **Create**

Using the gcloud CLI

```
sh
CopyEdit
gcloud sql instances create my-mysql-instance \
    --database-version=MYSQL_8_0 \
    --tier=db-n1-standard-2 \
    --region=us-central1 \
    --root-password=mysecurepassword
```

- 🔹 Replace `my-mysql-instance` with your instance name.
- 🔹 Change `--tier=db-n1-standard-2` for different CPU/RAM configurations.
- 🔹 Replace `mysecurepassword` with a strong password.

Step 3: Connect to Cloud SQL for MySQL

Once your instance is ready, connect using one of the following methods.

Option 1: Cloud SQL Proxy (Recommended for Security)

1. Install the **Cloud SQL Proxy**:

```
sh
CopyEdit
curl -o cloud_sql_proxy
https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64
chmod +x cloud_sql_proxy
```

2. Start the proxy:

```
sh
CopyEdit
./cloud_sql_proxy
-instances=<PROJECT-ID>:us-central1:my-mysql-instance=tcp:3306
```

3. Connect using MySQL CLI:

```
sh
CopyEdit
mysql -h 127.0.0.1 -u root -p
```

Option 2: Direct Connection via Public IP

1. Get the **public IP address** of your instance:

```
sh
CopyEdit
gcloud sql instances describe my-mysql-instance
--format="value(ipAddresses.ipAddress)"
```

2. Use MySQL CLI to connect:

```
sh
CopyEdit
mysql -h <INSTANCE_IP> -u root -p
```

3. Enter your password when prompted.

Step 4: Create a Database & Table

After connecting, create a database:

```
sql
CopyEdit
CREATE DATABASE mydatabase;
USE mydatabase;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE
);
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
SELECT * FROM users;
```

Step 5: Enable High Availability (Optional)

To enable **High Availability (HA)** for automatic failover:

```
sh
```

```
CopyEdit
gcloud sql instances patch my-mysql-instance --availability-type=REGIONAL
```

This ensures **automatic failover** to a standby instance in case of failure.

Step 6: Backup and Restore

Enable Automated Backups

```
sh
CopyEdit
gcloud sql instances patch my-mysql-instance --backup-start-time=02:00
```

Manually Create a Backup

```
sh
CopyEdit
gcloud sql backups create --instance=my-mysql-instance
```

Restore a Backup

```
sh
CopyEdit
gcloud sql backups restore BACKUP_ID --instance=my-mysql-instance
```

Replace `BACKUP_ID` with the ID of the backup.

Step 7: Monitor and Scale Cloud SQL

Check Instance Status

```
sh
CopyEdit
gcloud sql instances list
```

View Database Metrics

Use **Cloud Monitoring** to track **CPU, memory, and storage usage**.

Upgrade the Instance (Increase CPU/RAM)

```
sh
CopyEdit
gcloud sql instances patch my-mysql-instance --tier=db-n1-standard-4
```

Step 8: Delete Cloud SQL Instance (If Needed)

```
sh
CopyEdit
gcloud sql instances delete my-mysql-instance
```

Conclusion

✅ **Cloud SQL for MySQL** simplifies database management with **automated scaling, high availability, and security**.
✅ Supports **manual and automatic backups**, **private networking**, and **IAM-based access control**.
✅ Easily integrates with **App Engine, Compute Engine, Kubernetes, and BigQuery**.

EXPERIMENT NUMBER 7

Cloud Pub/Sub: Real-Time Messaging and Communication 🚀

Google **Cloud Pub/Sub** is a **fully managed messaging service** that enables **real-time event-driven communication** between distributed applications. It follows a **publisher-subscriber** model, making it ideal for **asynchronous message processing, event streaming, and data ingestion pipelines**.

How Cloud Pub/Sub Works?

✅ **Publishers** → Send messages to a **topic**.
✅ **Subscribers** → Receive messages from a **subscription**.
✅ **Message Delivery** → Ensures **at-least-once delivery** with automatic retries.
✅ **Event-Driven** → Triggers Cloud Functions, Dataflow, or other services.

Step 1: Enable Cloud Pub/Sub API

Before using Pub/Sub, enable the API:

```
sh
CopyEdit
gcloud services enable pubsub.googleapis.com
```

Step 2: Create a Pub/Sub Topic

A **topic** is where messages are sent.

Using Google Cloud Console

1. Open **Cloud Pub/Sub**
2. Click **Create Topic**
3. Enter a topic name (e.g., `my-topic`)
4. Click **Create**

Using gcloud CLI

```
sh
CopyEdit
gcloud pubsub topics create my-topic
```

Step 3: Create a Subscription

Subscribers need a **subscription** to receive messages.

Using Google Cloud Console

1. Open **Cloud Pub/Sub**
2. Click **Subscriptions** → **Create Subscription**
3. Choose the topic (`my-topic`)
4. Set **Delivery Type**:
   - **Pull** → Manual message retrieval
   - **Push** → Automatic HTTP delivery to a service (e.g., Cloud Function)
5. Click **Create**

Using gcloud CLI

```
sh
CopyEdit
gcloud pubsub subscriptions create my-subscription --topic=my-topic
```

Step 4: Publish a Message

Using gcloud CLI

```
sh
CopyEdit
gcloud pubsub topics publish my-topic --message="Hello, Pub/Sub!"
```

Step 5: Receive Messages

Pull Subscription (Manual Retrieval)

```
sh
CopyEdit
gcloud pubsub subscriptions pull my-subscription --auto-ack
```

☑ **`--auto-ack`** ensures the message is acknowledged and removed from the queue.

Push Subscription (Automatic Delivery)

If using a push subscription, Pub/Sub will **send messages to a configured endpoint** (e.g., Cloud Function, HTTP API).

Step 6: Integrate with Cloud Functions (Event-Driven Processing)

Deploy a Cloud Function to **process messages automatically**:

1 Create a Python function (`main.py`):

```
python
CopyEdit
import functions_framework
```

```
@functions_framework.cloud_event
def process_pubsub_event(event):
    """Triggered by a Pub/Sub message."""
    message = event.data["message"]["data"]
    decoded_message = base64.b64decode(message).decode("utf-8")
    print(f"Received message: {decoded_message}")
```

2️⃣ **Deploy the function**:

```
sh
CopyEdit
gcloud functions deploy process_pubsub_event \
    --runtime python310 \
    --trigger-topic my-topic \
    --region us-central1
```

📌 **Now, whenever a message is published, this function will execute automatically.**

Step 7: Monitor & Manage Pub/Sub

List Topics

```
sh
CopyEdit
gcloud pubsub topics list
```

List Subscriptions

```
sh
CopyEdit
gcloud pubsub subscriptions list
```

Delete a Topic

```
sh
CopyEdit
gcloud pubsub topics delete my-topic
```

Delete a Subscription

```
sh
CopyEdit
gcloud pubsub subscriptions delete my-subscription
```

Use Cases for Cloud Pub/Sub

🚀 **Real-time event processing** → Process streaming events in real time.
📩 **Asynchronous microservices communication** → Decouple services using Pub/Sub.
📊 **Data ingestion for BigQuery & Dataflow** → Efficiently move data across services.
🔔 **Trigger serverless functions** → Automate tasks using Cloud Functions.

Conclusion

You've successfully experimented with **Cloud Pub/Sub** for **real-time messaging** and **event-driven processing**! 🎉

EXPERIMENT NUMBER 8

Multiple VPC Networks in Google Cloud: Benefits & Use Cases

Google Cloud **Virtual Private Cloud (VPC)** allows you to create and manage **multiple VPC networks** to better organize and isolate resources. Using multiple VPCs improves **security, scalability, and flexibility** in cloud architectures.

🔹 Why Use Multiple VPC Networks?

✅ **Isolation & Security** – Separate resources for better access control (e.g., production vs. development).
✅ **Multi-Tenancy** – Assign different VPCs to different teams, customers, or business units.
✅ **Hybrid & Multi-Cloud Networking** – Connect on-premises infrastructure or other cloud providers securely.
✅ **Compliance & Governance** – Keep sensitive workloads isolated for regulatory requirements.
✅ **Traffic Segmentation** – Control and monitor internal communication between different workloads.

🛠️ Step 1: Create Multiple VPC Networks

Using Google Cloud Console

1. Navigate to **VPC Networks**
2. Click **Create VPC Network**
3. Provide a **name** (e.g., `vpc-prod`, `vpc-dev`)
4. Choose **Custom Subnet Mode** to define subnets manually
5. Click **Create**

Using gcloud CLI

```
sh
CopyEdit
gcloud compute networks create vpc-prod --subnet-mode=custom
gcloud compute networks create vpc-dev --subnet-mode=custom
```

🔹 `--subnet-mode=custom` allows you to define subnets explicitly instead of auto-created ones.

🛠️ Step 2: Create Subnets in Each VPC

Each VPC network requires **subnets** in different regions.

```
sh
CopyEdit
gcloud compute networks subnets create subnet-prod \
    --network=vpc-prod \
    --region=us-central1 \
    --range=10.0.1.0/24

gcloud compute networks subnets create subnet-dev \
    --network=vpc-dev \
    --region=us-central1 \
    --range=10.0.2.0/24
```

🔹 `--range=10.0.1.0/24` defines the IP address range for the subnet.

🔹 Step 3: Connecting Multiple VPCs

Since VPCs are isolated by default, you need **VPC Peering or VPN** to enable communication.

Option 1: VPC Peering (For Low-Latency Private Communication)

```
sh
CopyEdit
gcloud compute networks peerings create vpc-prod-to-dev \
    --network=vpc-prod \
    --peer-network=vpc-dev

gcloud compute networks peerings create vpc-dev-to-prod \
    --network=vpc-dev \
    --peer-network=vpc-prod
```

🔹 This allows **private communication** between `vpc-prod` and `vpc-dev`.

Option 2: Cloud VPN (For Secure Communication Across Regions or Cloud Providers)

1️⃣ Create VPN Gateway in `vpc-prod`

```
sh
CopyEdit
gcloud compute vpn-gateways create vpn-gateway-prod \
    --network=vpc-prod \
    --region=us-central1
```

2️⃣ Create a VPN Tunnel to `vpc-dev`

```
sh
```

```
CopyEdit
gcloud compute vpn-tunnels create vpn-tunnel-prod-to-dev \
    --region=us-central1 \
    --peer-gcp-gateway=vpn-gateway-dev \
    --ike-version=2 \
    --shared-secret=my-vpn-secret
```

🔹 Replace `my-vpn-secret` with a strong **pre-shared key**.

🔹 Step 4: Set Up Firewall Rules for Communication

By default, VPC networks have **strict firewall rules**. You need to explicitly **allow traffic**.

```
sh
CopyEdit
gcloud compute firewall-rules create allow-internal \
    --network=vpc-prod \
    --allow tcp,udp,icmp \
    --source-ranges=10.0.0.0/16

gcloud compute firewall-rules create allow-internal \
    --network=vpc-dev \
    --allow tcp,udp,icmp \
    --source-ranges=10.0.0.0/16
```

🔹 This allows communication **inside each VPC**.

🔹 Step 5: Verify Connectivity

To check if multiple VPCs can communicate:

1️⃣ **Create two VM instances in separate VPCs**

```
sh
CopyEdit
gcloud compute instances create vm-prod \
    --network=vpc-prod \
    --zone=us-central1-a \
    --subnet=subnet-prod

gcloud compute instances create vm-dev \
    --network=vpc-dev \
    --zone=us-central1-a \
    --subnet=subnet-dev
```

2️⃣ **SSH into `vm-prod` and ping `vm-dev`**

```
sh
CopyEdit
ping 10.0.2.2
```

(Replace with the actual **private IP** of `vm-dev`.)

If successful, your **VPC peering or VPN is correctly configured**! 🎉

🔹 Best Practices for Multiple VPC Networks

✅ Use **VPC Peering** for **low-latency private networking** between VPCs.
✅ Use **Cloud VPN or Interconnect** for **cross-region or hybrid cloud setups**.
✅ **Segment workloads** based on environments (e.g., **prod, dev, test**).

✅ **Define IAM policies** to control access to VPCs based on team roles.
✅ **Monitor network traffic** using **VPC Flow Logs & Cloud Logging**.

🎯 Conclusion

You have successfully explored **multiple VPC networks**, configured **subnets**, set up **VPC Peering**, and ensured **secure communication** between VPCs! 🚀

EXPERIMENT NUMBER 9

Cloud Monitoring in Google Cloud: Tracking & Analyzing Performance

Google **Cloud Monitoring** helps you track, analyze, and optimize the **performance, availability, and health** of your cloud resources. It provides **real-time metrics, dashboards, alerts, and logs** to detect and resolve issues quickly.

🔹 Key Features of Cloud Monitoring

✅ **Real-time Metrics** – Collects performance data from Google Cloud services, VMs, and applications.
✅ **Dashboards** – Customizable dashboards for visualizing resource health.
✅ **Alerts & Notifications** – Set up alerts for CPU, memory, network usage, etc.
✅ **Logs Integration** – Works with **Cloud Logging** to analyze errors and troubleshoot issues.
✅ **Uptime Monitoring** – Ensures your websites and applications are available globally.
✅ **SLO & SLA Tracking** – Define and monitor **Service Level Objectives (SLOs)** to meet business goals.

🛠️ Step 1: Enable Cloud Monitoring

Before using Cloud Monitoring, enable the API:

```
sh
CopyEdit
gcloud services enable monitoring.googleapis.com logging.googleapis.com
```

Alternatively, enable it in the **Google Cloud Console**:

1. Open **Cloud Monitoring**
2. Click **Enable Cloud Monitoring**

🛠️ Step 2: Create a Monitoring Dashboard

Using Google Cloud Console

1. Navigate to **Cloud Monitoring** → **Dashboards**
2. Click **Create Dashboard**
3. Add **widgets** (CPU, memory, network, etc.)
4. Click **Save**

Using gcloud CLI

To create a dashboard with CPU usage:

```
sh
CopyEdit
gcloud monitoring dashboards create --config-from-file=my-dashboard.json
```

(Requires a JSON configuration file defining metrics.)

🛠️ Step 3: Set Up Alerts for Performance Issues

Alerts notify you of potential issues (e.g., high CPU, low memory).

Create an Alert in Cloud Console

1. Go to **Cloud Monitoring** → **Alerting**
2. Click **Create Policy**
3. Select **Condition** (e.g., CPU utilization > 80%)
4. Set a **Notification Channel** (Email, Slack, PagerDuty, etc.)

5. Click **Create**

Using gcloud CLI

To create an alert for **high CPU usage**:

```
sh
CopyEdit
gcloud alpha monitoring policies create \
    --display-name="High CPU Usage Alert" \

--conditions="metric.type=compute.googleapis.com/instance/cpu/utilization,
threshold_value=0.8" \
    --notification-channels=projects/my-project/notificationChannels/12345
```

🔹 This sends an alert when CPU usage **exceeds 80%**.

🛠️ Step 4: Monitor Logs for Troubleshooting

Cloud Monitoring integrates with **Cloud Logging** to analyze system errors and application logs.

View Logs in Cloud Console

1. Go to **Cloud Logging** → **Logs Explorer**
2. Select a **resource type** (e.g., VM instance, Cloud Function)
3. Apply **filters** (e.g., `severity=ERROR`)
4. Analyze logs to troubleshoot issues

Using gcloud CLI

To view logs for a specific VM:

```
sh
CopyEdit
gcloud logging read "resource.type=gce_instance AND severity>=ERROR"
--limit=10
```

🔹 Fetches the **last 10 error logs** from a Compute Engine instance.

🛠️ Step 5: Set Up Uptime Checks

Uptime checks monitor website and service availability.

Create an Uptime Check in Cloud Console

1. Go to **Cloud Monitoring** → **Uptime Checks**
2. Click **Create Uptime Check**
3. Enter the **URL or IP address** of your service
4. Set **Frequency & Locations**
5. Click **Create**

Using gcloud CLI

```
sh
CopyEdit
```

```
gcloud monitoring uptime-checks create http \
    --display-name="My Website Uptime" \
    --host="example.com" \
    --path="/" \
    --check-interval=60s
```

🔹 Checks **example.com** every 60 seconds.

🔹 Best Practices for Cloud Monitoring

✅ **Use Dashboards** – Visualize key performance metrics in real time.
✅ **Set Alerts** – Get notified when performance thresholds are exceeded.
✅ **Enable Logging** – Capture and analyze logs for troubleshooting.
✅ **Monitor Uptime** – Ensure websites and APIs are accessible globally.
✅ **Define SLOs** – Track **Service Level Objectives (SLOs)** for reliability.

🎯 Conclusion

Google **Cloud Monitoring** helps you track, analyze, and maintain **optimal performance** for cloud resources. 🚀

EXPERIMENT NUMBER 10

Kubernetes Engine: Quick Start - Deploying a Containerized Application

Google **Kubernetes Engine (GKE)** allows you to deploy, manage, and scale containerized applications using **Kubernetes**. This guide will help you deploy a **containerized application** on a **GKE cluster** quickly.

🔹 Step 1: Enable Required APIs

Before using GKE, enable the necessary APIs:

```
sh
CopyEdit
gcloud services enable container.googleapis.com
```

Or enable it via **Google Cloud Console**:

1. Navigate to **APIs & Services**
2. Search for **Kubernetes Engine API**
3. Click **Enable**

🛠️ Step 2: Create a GKE Cluster

A **Kubernetes cluster** is required to deploy applications.

Using Google Cloud Console

1. Go to **Kubernetes Engine → Clusters**
2. Click **Create Cluster**
3. Choose **Standard Cluster**
4. Set the **name** (e.g., `my-cluster`)
5. Choose a **region** (e.g., `us-central1`)
6. Click **Create**

Using gcloud CLI

```
sh
CopyEdit
gcloud container clusters create my-cluster \
    --num-nodes=3 \
    --zone=us-central1-a
```

🔹 Creates a **3-node cluster** in `us-central1-a`.

Verify the cluster:

```
sh
CopyEdit
gcloud container clusters list
```

🛠️ Step 3: Connect to the Cluster

Once the cluster is created, configure `kubectl` to connect:

```
sh
CopyEdit
gcloud container clusters get-credentials my-cluster --zone us-central1-a
```

Check cluster nodes:

```
sh
CopyEdit
kubectl get nodes
```

🔹 This ensures **Kubernetes is running** correctly.

🛠️ Step 4: Deploy a Containerized Application

For this quick start, we will deploy an **Nginx web server**.

Create a Deployment

```
sh
CopyEdit
kubectl create deployment my-app --image=nginx
```

🔹 This deploys **Nginx** in the cluster.

Verify the deployment:

```
sh
CopyEdit
kubectl get deployments
```

Expose the Deployment

To access the app, expose it with a **LoadBalancer** service:

```
sh
CopyEdit
kubectl expose deployment my-app --type=LoadBalancer --port=80
```

Check the service:

```
sh
CopyEdit
kubectl get services
```

Look for the **EXTERNAL-IP** of your app. If pending, wait a few minutes:

```
sh
```

```
CopyEdit
kubectl get services my-app --watch
```

Once available, **open in a browser**:

```
cpp
CopyEdit
http://EXTERNAL-IP
```

You should see the **Nginx Welcome Page**! 🎉

🔹 Step 5: Clean Up Resources

To avoid unnecessary costs, **delete the cluster** when done:

```
sh
CopyEdit
gcloud container clusters delete my-cluster --zone=us-central1-a
```

🔹 Best Practices for Deploying on GKE

✅ **Use Managed GKE** – Reduces operational overhead.
✅ **Deploy via YAML** – Define deployments in `deployment.yaml` for better control.
✅ **Use Ingress** – For advanced traffic management instead of `LoadBalancer`.
✅ **Enable Auto-scaling** – Optimize resource usage with **HPA (Horizontal Pod Autoscaler)**.
✅ **Monitor & Secure** – Use **Cloud Monitoring & IAM roles** to enhance security.

🎯 Conclusion

You have successfully **created a GKE cluster, deployed a containerized app, and exposed it to the internet**! 🚀