

SRI SAIRAM COLLEGE OF ENGINEERING, ANEKAL, BENGALURU
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



Subject Code: BCS601

Subject: Cloud Computing Laboratory

Prepared by
K.Karthika
Assistant professor/CSE

V.Yamuna
Assistant Professor /CSE

Laboratory Components

- 1. VM Deployment: Configure and deploy a virtual machine in Google Cloud.**
- 2. Cloud Shell & gcloud: Manage Google Cloud resources using gcloud commands in Cloud Shell.**
- 3. Cloud Functions: Create and deploy a Cloud Function for task automation on Cloud Storage events.**
- 4. App Engine Deployment: Deploy a web application with automatic scaling on Google App Engine.**
- 5. Cloud Storage Qwikstart: Manage data using Google Cloud Storage via Console or gsutil CLI.**
- 6. Cloud SQL for MySQL: Explore automated management and high availability in Google Cloud SQL for MySQL.**
- 7. Cloud Pub/Sub: Experiment with real-time messaging and communication using Google Cloud Pub/Sub.**
- 8. Multiple VPC Networks: Understand the benefits of multiple VPC networks for resource isolation.**
- 9. Cloud Monitoring: Track and analyze the performance and health of cloud resources using Cloud Monitoring.**
- 10. Kubernetes Engine Qwikstart: Deploy a containerized application to a Kubernetes Engine cluster.**

Experiment-01

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

Step 1: Sign in to Google Cloud Console

1. Go to **Google Cloud Console**: <https://console.cloud.google.com/>
2. Log in with your **Google Account**.
3. Select or create a **new project** from the top navigation bar.

Step 2: Open Compute Engine

1. In the left sidebar, **navigate to "Compute Engine"** → Click **"VM instances"**.
2. Click **"Create Instance"**.

Step 3: Configure the Virtual Machine

1. Name the VM

- Enter a **name** for your VM instance.

2. Select the Region and Zone

- Choose a **region** close to your target audience or users.
- Choose an **availability zone** (e.g., us-central1-a).

3. Choose the Machine Configuration

- Under **"Machine Configuration"**, select:
 - **Series** (E2, N1, N2, etc.)
 - **Machine type** (Select based on your CPU & RAM needs)

- Example:
 - **e2-medium** (2 vCPU, 4GB RAM)
 - **n1-standard-4** (4 vCPU, 16GB RAM)
 - Click "**Customize**" if you want specific CPU & RAM.

4. Boot Disk (Operating System)

- Click "**Change**" under Boot Disk.
- Choose an **Operating System** (e.g., Ubuntu, Windows, Debian).
- Select **disk size** (e.g., 20GB or more).

5. Networking and Firewall

- Enable "**Allow HTTP Traffic**" or "**Allow HTTPS Traffic**" if needed.
- Click "**Advanced options**" for networking configurations.

Step 4: Create and Deploy the VM

1. Review all the configurations.
2. Click "**Create**" to deploy the VM.
3. Wait for the instance to be provisioned.

Step 5: Connect to the VM

1. Using SSH (Web)

- Go to **Compute Engine** → **VM Instances**.
- Click "**SSH**" next to your VM instance.

2. Using SSH (Terminal)

- Open **Google Cloud SDK** (Cloud Shell) or your local terminal.
- Run:

```
gcloud compute ssh your-instance-name --zone=us-central1-a
```

Step 6: Verify and Use the VM

- Check CPU and Memory:

```
lscpu    # CPU details
```

```
free -h  # Memory details
```

- Install required software (example: Apache web server)

```
sudo apt update && sudo apt install apache2 -y
```

Step 7: Stop or Delete the VM (Optional)

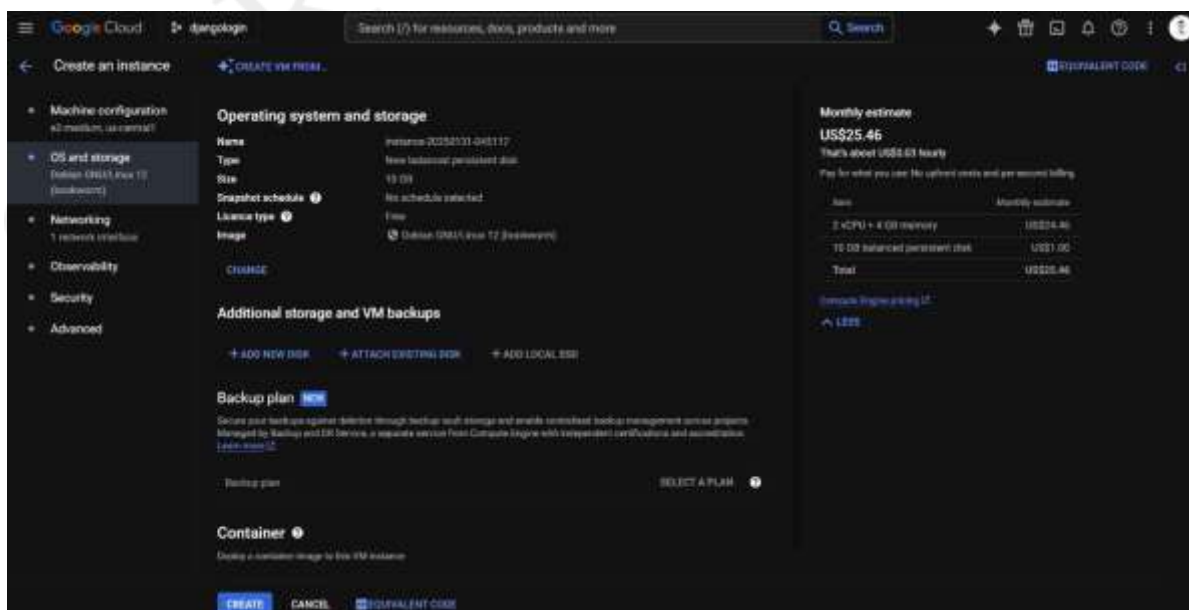
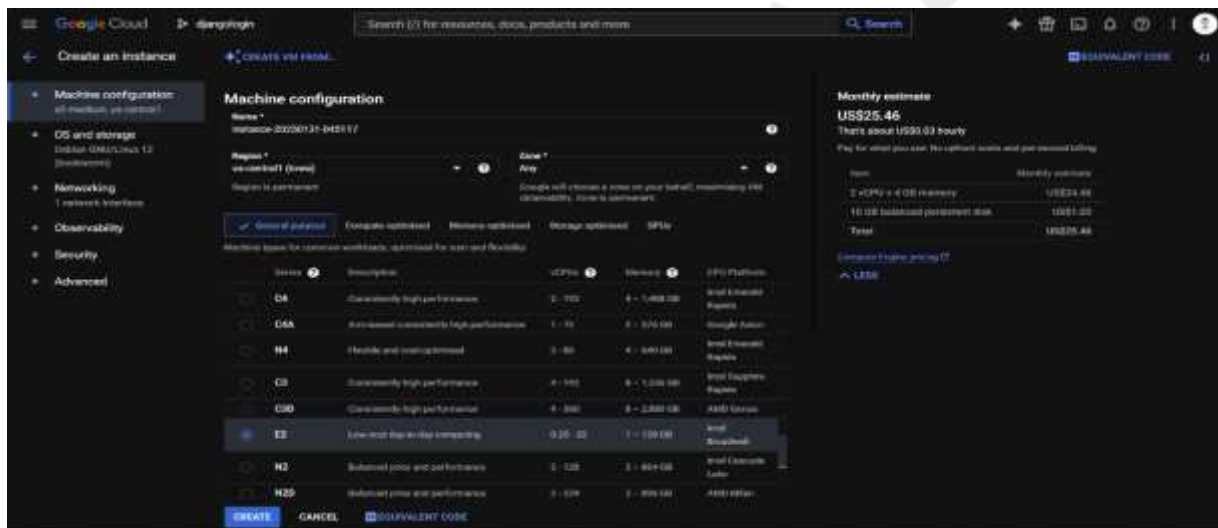
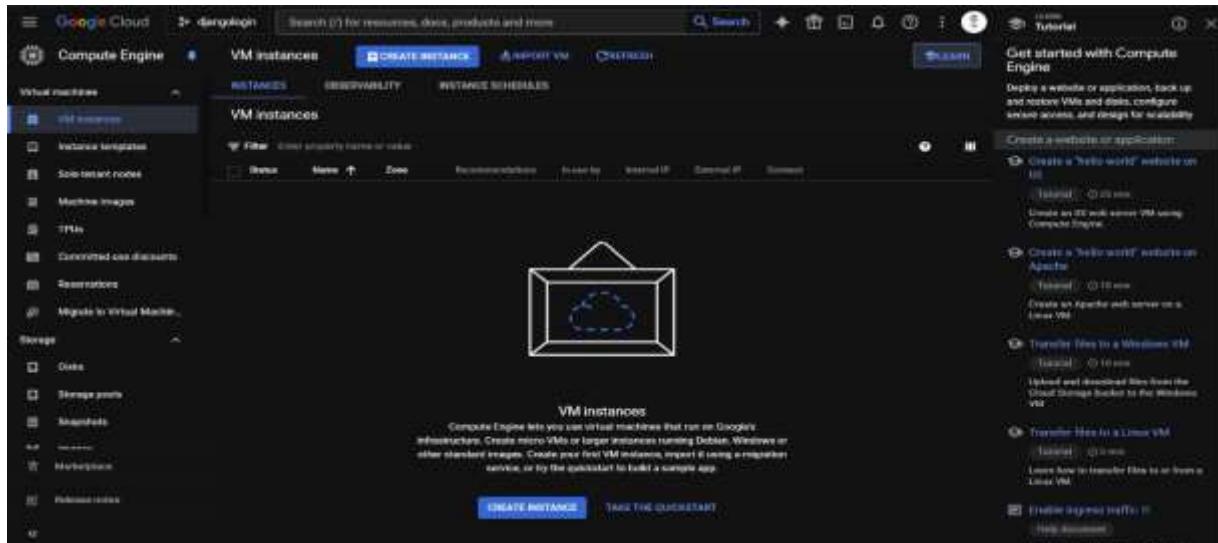
- Stop the VM:

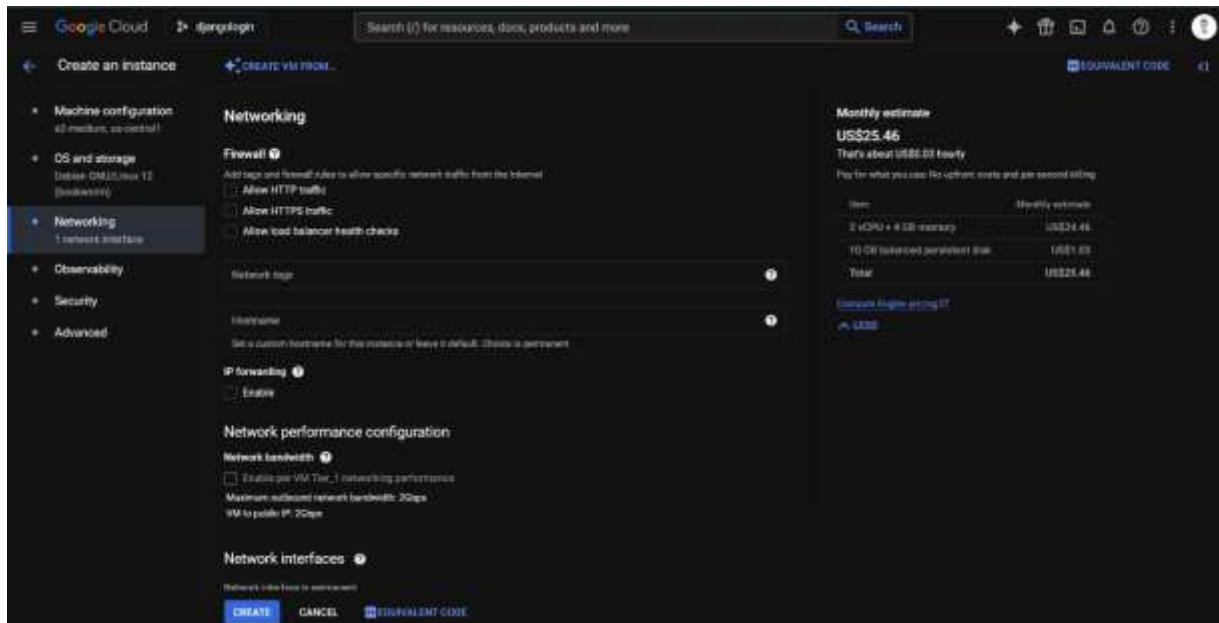
```
gcloud compute instances stop your-instance-name --zone=us-central1-a
```

- Delete the VM:

```
gcloud compute instances delete your-instance-name --zone=us-central1-a
```

Output

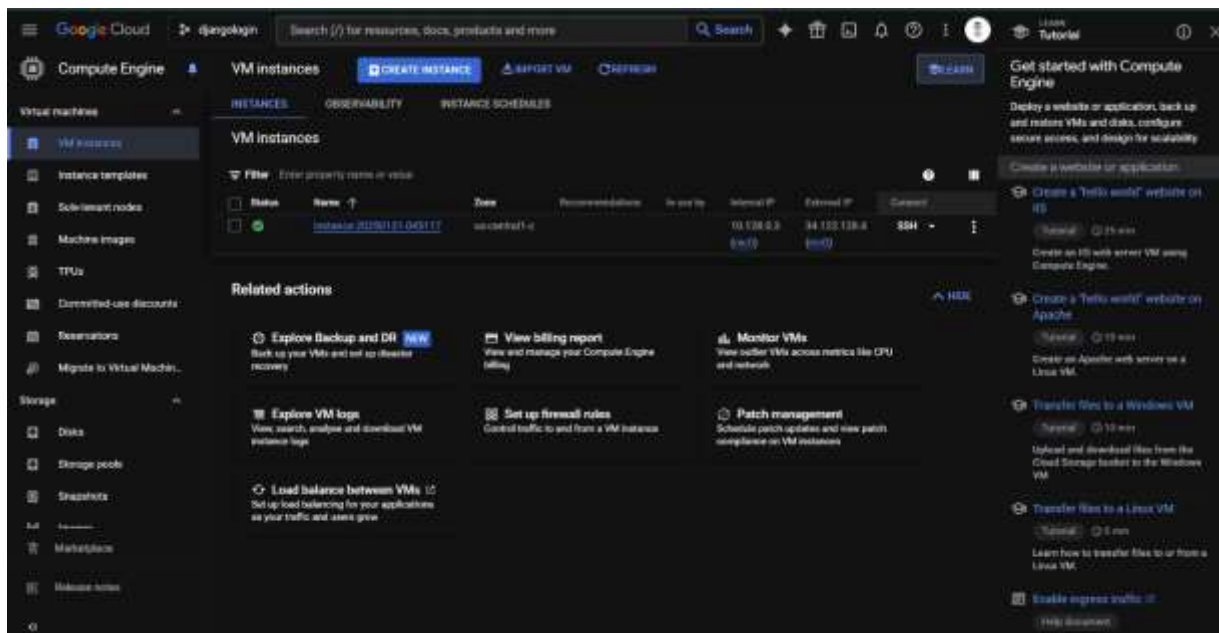




This screenshot shows the 'Create an instance' wizard in the Google Cloud Platform console, specifically the 'Networking' tab. The left sidebar lists various configuration sections: Machine configuration, OS and storage, Networking (selected), Observability, Security, and Advanced. The main content area is divided into several sections:

- Firewall:** Includes checkboxes for 'Allow HTTP traffic', 'Allow HTTPS traffic', and 'Allow load balancer health checks'.
- Network tags:** A field to specify network tags.
- Hostname:** A field to set a custom hostname.
- IP forwarding:** A checkbox to 'Enable' IP forwarding.
- Network performance configuration:** Includes a checkbox for 'Enable per-VM Tier_1 networking performance' and notes on maximum egress bandwidth (20Gbps) and VM-to-public-IP 20Gbps.
- Network interfaces:** A section to manage network interfaces.

At the bottom right, a 'Monthly estimate' box shows a total cost of US\$25.46, broken down by item: 2 vCPU + 4 GB memory (US\$3.46), 10 GB boot disk (US\$1.00), and Total (US\$25.46). The 'CREATE' button is visible at the bottom left.



This screenshot shows the 'VM instances' page in the Google Cloud Platform console. The left sidebar lists various resources: VM instances (selected), Instance templates, Scheduled nodes, Machine images, TPUs, Committed-use discounts, Reservations, Migrate to Virtual Machine, Storage (Disks, Storage pools, Snapshots, Marketplace, Release notes), and VM instances. The main content area displays a table of VM instances:

Status	Name	Zone	Recommendations	IP pool	Internal IP	External IP	Connect
Running	instance-20201111-045117	us-central1-a			10.128.0.3	34.132.138.4	SSH

Below the table, there are several 'Related actions' cards:

- Explore backup and DR:** Back up your VMs and set up disaster recovery.
- View billing report:** View and manage your Compute Engine billing.
- Monitor VMs:** View outlier VMs across metrics like CPU and network.
- Explore VM logs:** View, search, analyze and download VM instance logs.
- Set up firewall rules:** Control traffic to and from a VM instance.
- Patch management:** Schedule patch updates and view patch compliance on VM instances.
- Load balance between VMs:** Set up load balancing for your applications as your traffic and users grow.

On the right side, there is a 'Get started with Compute Engine' tutorial section with various links and instructions.

Experiment-02

Getting Started with Cloud Shell and gcloud: Discover the use of gcloud commands to manage Google Cloud resources from Cloud Shell.

Step 1: Open Cloud Shell

1. **Sign in** to the **Google Cloud Console**:

— <https://console.cloud.google.com/>

2. Click the **Cloud Shell** icon (● Terminal icon) in the top-right corner.
3. A terminal will open at the bottom of the page.

Step 2: Initialize gcloud CLI

1. Run the following command in Cloud Shell:

```
gcloud init
```

2. Follow the prompts to:
 - **Authenticate your Google account**
 - **Select a Google Cloud project**

Step 3: Verify gcloud Setup

To check if gcloud is properly configured, run:

```
gcloud config list
```

This displays your **current project, account, and region settings**.

Step 4: List Available Projects

Run the following command to **view all projects** associated with your Google account:

```
gcloud projects list
```

Step 5: Set Active Project

To set a specific project as the active one, run:

```
gcloud config set project PROJECT_ID
```

Replace PROJECT_ID with your actual project ID.

Step 6: Check Authentication Status

Run this command to verify that you're authenticated:

```
gcloud auth list
```

This will show the currently logged-in Google account.

Step 7: Create a Virtual Machine (VM) Instance

Launch a new Compute Engine VM instance:

```
gcloud compute instances create my-vm --zone=us-central1-a
```

- my-vm → Name of the instance
- --zone=us-central1-a → Choose a different zone if needed

Step 8: List Running VM Instances

To check all running VM instances, run:

```
gcloud compute instances list
```

Step 9: Delete a VM Instance

If you no longer need a VM, delete it using:

```
gcloud compute instances delete my-vm
```

Confirm the deletion when prompted.

Step 10: Enable an API (Example: Compute Engine API)

To enable an API, such as the Compute Engine API, run:

```
gcloud services enable compute.googleapis.com
```

Step 11: Deploy an Application to App Engine

If you have an application ready, deploy it using:

```
gcloud app deploy
```

Follow the instructions to **deploy and access your app**.

Experiment-03

Cloud Functions: Create and deploy a Cloud Function to automate a specific task based on a Cloud Storage event.

Step 1: Enable Required APIs

Before deploying the Cloud Function, enable the necessary APIs:

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

Step 2: Create a Cloud Storage Bucket

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

```
gcloud storage buckets create BUCKET_NAME --location=us-central1
```

Replace BUCKET_NAME with a unique name for your bucket.

Step 3: Write the Cloud Function Code

1. Open **Cloud Shell** and create a working directory:

```
mkdir gcs-function && cd gcs-function
```

2. Create and open a new Python file (main.py):

```
nano main.py
```

3. Add the following code inside main.py:

```
import functions_framework
```

```
@functions_framework.cloud_event
```

```
def gcs_trigger(cloud_event):
```

```
    """Triggered when a file is uploaded to Cloud Storage."""
```

```
    data = cloud_event.data
```

```
    bucket = data["bucket"]
```

```
    file_name = data["name"]
```

```
    print(f"👉 File {file_name} uploaded to {bucket}")
```

4. Save and close the file (CTRL + X, Y, Enter).

Step 4: Create a requirements.txt File

Create and open a requirements.txt file:

```
nano requirements.txt
```

Add the required dependency:

```
functions-framework
```

Save and close (CTRL + X, Y, Enter).

Step 5: Deploy the Cloud Function

Run the following command to **deploy the function**:

```
gcloud functions deploy gcs_trigger \  
  
  --gen2 \  
  
  --runtime=python311 \  
  
  --region=us-central1 \  
  
  --source=. \  
  
  --entry-point=gcs_trigger \  
  
  --trigger-event-filters="type=google.cloud.storage.object.v1.finalized" \  
  
  --trigger-event-filters="bucket=BUCKET_NAME" \  
  
  --allow-unauthenticated
```

Replace BUCKET_NAME with your actual Cloud Storage bucket name.

Step 6: Test the Cloud Function


1. Upload a file to the Cloud Storage bucket:

```
gcloud storage cp test-file.txt gs://BUCKET_NAME
```

2. Check logs to verify function execution:

```
gcloud functions logs read gcs_trigger --region=us-central1
```

Output



```

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Shell project in this session is set to djanglings-400104.
The gcloud CLI in this session is set to djanglings-400104.
The gcloud CLI in this session is set to djanglings-400104.
Operation "operations/terraform-202207151515-5103641-7420-400104-7420-400104" finished successfully.
djanglings-400104:~$ gcloud storage buckets create MY_BUCKET --location=us-central1
Created bucket my-bucket.
djanglings-400104:~$ gcloud storage buckets rm MY_BUCKET --location=us-central1
Deleted bucket my-bucket.
djanglings-400104:~$

```

Experiment-04

App Engine: Deploy a web application on App Engine with automatic scaling enabled.

Step 1: Enable Required APIs

Before deploying your application, enable the **App Engine API**:

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

Step 2: Create an App Engine Application

Run the following command to create an **App Engine** application in your project:

```
gcloud app create --region=us-central1
```

You can replace us-central1 with another region if needed.

Step 3: Create a Simple Web Application

1. Open **Cloud Shell** and create a project directory:

```
mkdir app-engine-demo && cd app-engine-demo
```

2. Create a Python file (main.py):

```
nano main.py
```

3. Add the following simple Flask application code:

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
```

```
def home():
```

```
    return 🌈Welcome to Google App Engine with Auto Scaling!"
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=8080)
```

4. Save and close (CTRL + X, Y, Enter).

Step 4: Create a requirements.txt File

Create and open a requirements.txt file:

```
nano requirements.txt
```

Add the following dependencies:

```
Flask
```

```
gunicorn
```

Save and close (CTRL + X, Y, Enter).

Step 5: Create an app.yaml File for App Engine

Create an **app.yaml** file to configure App Engine:

```
nano app.yaml
```

Add the following content:

```
runtime: python311
```

```
entrypoint: gunicorn -b :$PORT main:app
```

```
automatic_scaling:
```

```
  min_instances: 1
```

```
  max_instances: 5
```

```
  target_cpu_utilization: 0.65
```

```
  target_throughput_utilization: 0.75
```

Save and close (CTRL + X, Y, Enter).

Step 6: Deploy the Web Application

Run the following command to deploy the application:

```
gcloud app deploy
```

- Confirm the deployment when prompted (Y).

Step 7: Access the Deployed Application

Once deployed, open your web app in a browser:

```
gcloud app browse
```

This will return a **URL** (e.g., <https://your-project-id.appspot.com>), where you can view your running app.

Step 8: View Logs and Monitor Scaling

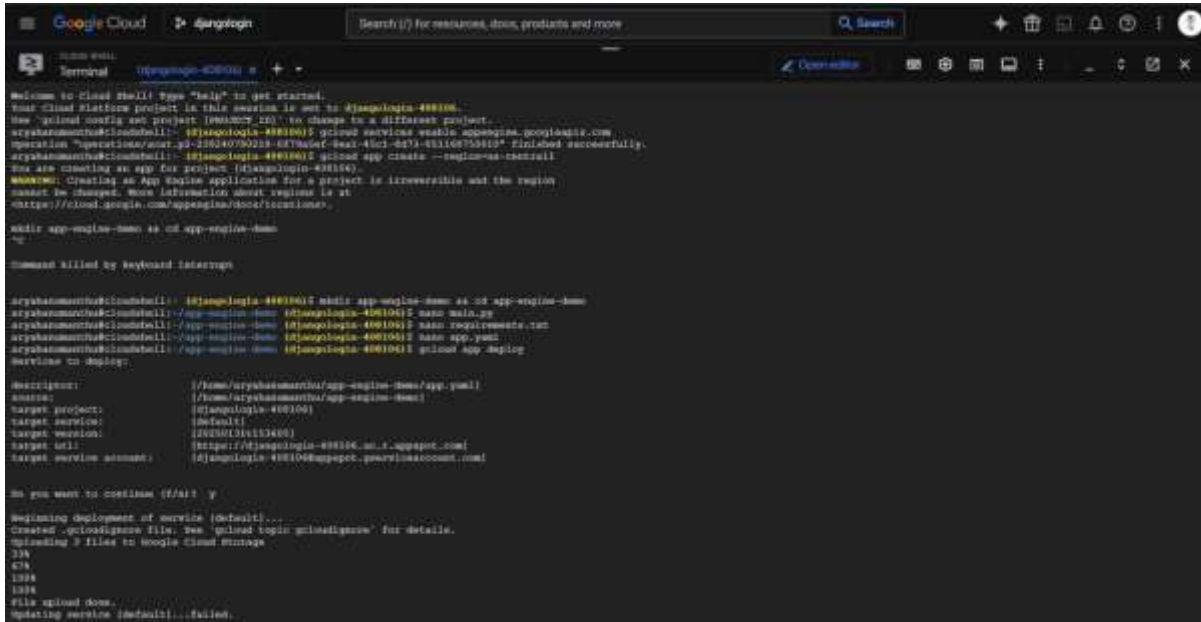
Check the logs of your application:

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

Monitor the deployed services:

```
gcloud app services list
```

Output



```
Google Cloud | djanglogia-408104 | Search [f] for resources, docs, products and more | Search
Terminal | djanglogia-408104 | + - | Open editor
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Shell project in this session is set to djanglogia-408104.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
arykhanamurthi@cloudshell: ~$ gcloud config set project djanglogia-408104
Operation "gcloud config set project djanglogia-408104" finished successfully.
arykhanamurthi@cloudshell: ~$ gcloud app create --engine=app-engine
You are creating an app for project [djanglogia-408104].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/versions>.
shdir app-engine-demo as cd app-engine-demo
~$
Command killed by keyboard interrupt

arykhanamurthi@cloudshell: ~$ cd app-engine-demo as cd app-engine-demo
arykhanamurthi@cloudshell: /app-engine-demo $ nano main.py
arykhanamurthi@cloudshell: /app-engine-demo $ nano requirements.txt
arykhanamurthi@cloudshell: /app-engine-demo $ nano app.yaml
arykhanamurthi@cloudshell: /app-engine-demo $ gcloud app deploy
Service to deploy:
Description:      [/home/arykhanamurthi/app-engine-demo/app.yaml]
Source:           [/home/arykhanamurthi/app-engine-demo]
Target project:   [djanglogia-408104]
Target service:   [default]
Target version:   [20230131v153400]
Target url:        [https://djanglogia-408104.appspot.com/]
Target service account: [djanglogia-408104@appspot.gcp-svc.com]

Do you want to continue (Y/n)? y
Beginning deployment of service [default]...
Created .gcloudignore file. See "gcloud topic gcloudignore" for details.
Uploading 3 files to Google Cloud Storage
33%
67%
100%
File upload done.
Updating service [default]...failed.
```

Experiment-05

Cloud Storage: Qwikstart: Google Cloud Storage provides scalable and secure object storage for managing data, accessible via the Cloud Console or gsutil CLI.

Step 1: Open Google Cloud Console

1. Go to Google Cloud Console.
2. If not already logged in, sign in with your **Google account**.
3. Ensure that you have an active **Google Cloud Project**.
 - If not, click on the project dropdown (top bar) and select an existing project or **create a new project**.

Step 2: Enable Cloud Storage API (If Not Enabled)

1. In the Google Cloud Console, click the **Navigation Menu** (≡) on the top left.
2. Go to **APIs & Services** → **Library**.
3. Search for **Cloud Storage API**.
4. Click **Enable** if it is not already enabled.

Step 3: Create a Cloud Storage Bucket

1. In the **Navigation Menu** (☰), go to **Storage** → **Buckets**.
2. Click **Create**.
3. Enter a **globally unique bucket name** (e.g., your-unique-bucket-name).
4. Choose a **Location** (e.g., us-central1 for the USA).
5. Select a **Storage Class** (Choose based on your needs):
 - **Standard** (Frequent access, low latency)
 - **Nearline** (Access once a month)
 - **Coldline** (Rare access, backup storage)
 - **Archive** (Long-term storage)
6. Choose **Access Control**:
 - **Fine-grained** (More detailed control)
 - **Uniform** (Simpler access control)
7. Click **Create**.

 **Your bucket is now ready!**

Step 4: Upload a File to the Bucket

1. Open your bucket from **Storage** → **Buckets**.
2. Click the **Upload Files** button.
3. Select a file from your computer and click **Open**.
4. Wait for the file to upload.

 **Your file is now stored in Cloud Storage!**

Step 5: Download a File from the Bucket

1. Open your bucket in **Storage** → **Buckets**.
2. Click on the file you want to download.
3. Click **Download** to save the file to your computer.

Step 6: Make a File Public (Optional)

1. Open your bucket and click on the file.
2. Click the **Permissions** tab.
3. Click **Add Principal**.
4. In the **New Principals** field, enter:

allUsers
5. Select the **Role**:
 - **Storage Object Viewer** (roles/storage.objectViewer)

6. Click **Save**.

 **Now your file is publicly accessible!**

You will see a **public URL** like:

`https://storage.googleapis.com/your-unique-bucket-name/your-file-name`

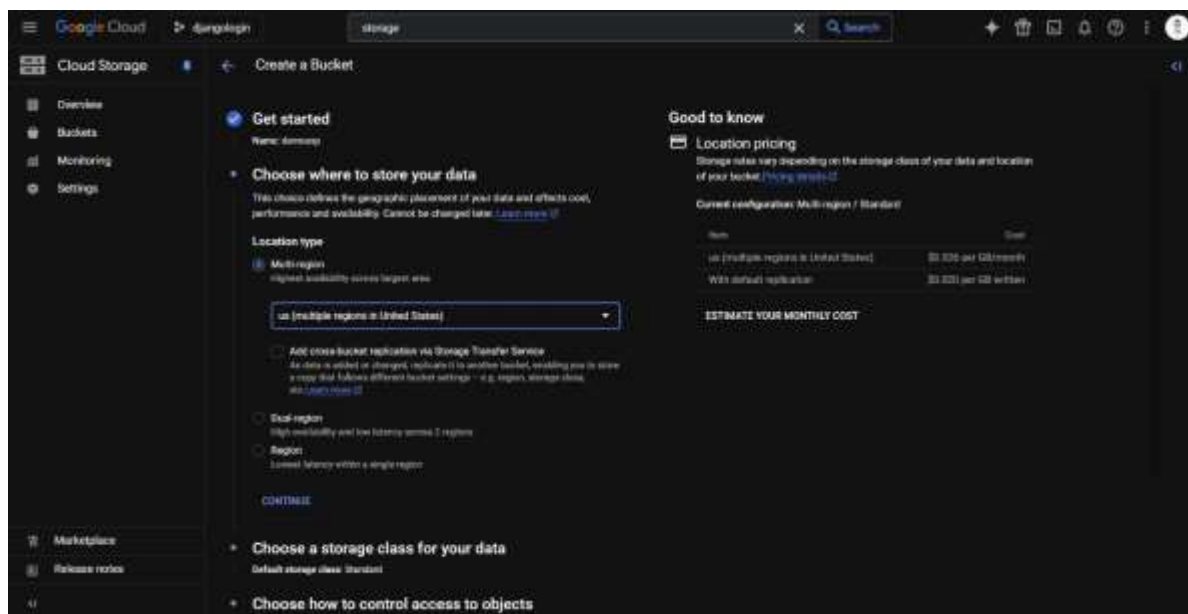
Anyone can access the file using this link.

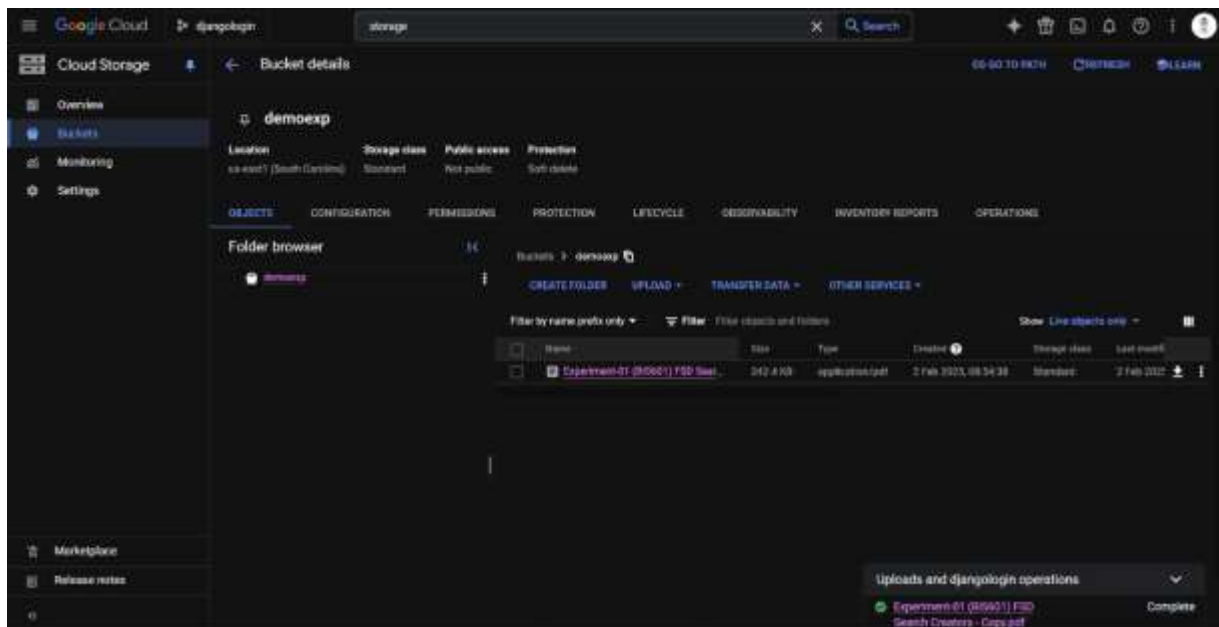
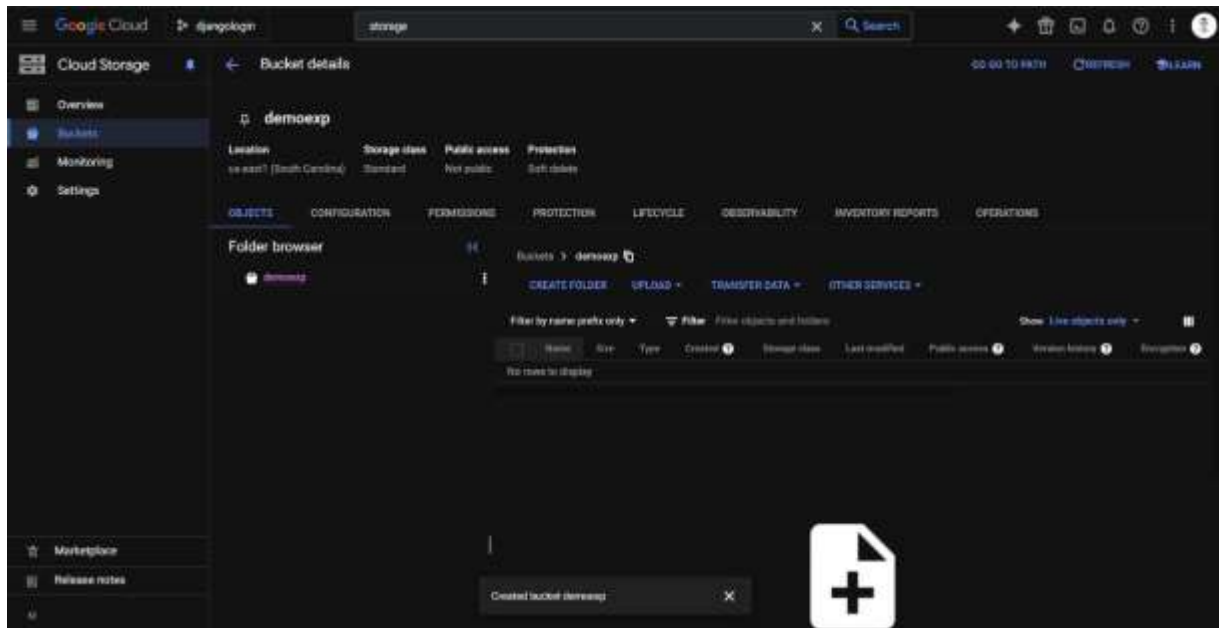
Step 7: Delete a File or Bucket (Optional)

- To **delete a file**, click on the file and select **Delete**.
- To **delete a bucket**, go to **Storage** → **Buckets**, select the bucket, and click **Delete**.

(You must first delete all files inside before deleting the bucket.)

Output





Experiment-06

Cloud SQL for MySQL: Discover how Google Cloud SQL for MySQL provide automated management and high availability for MySQL databases?

◆ Key Features of Cloud SQL for MySQL

Automated Management

■ **Automatic Backups** – Cloud SQL provides daily automated backups and point-in-time recovery.

■ **Automatic Updates & Patching** – Google automatically applies security patches.

■ **Automatic Failover** – High-availability instances automatically switch to a standby node if the primary fails.

High Availability (HA)

■ **Regional Replication** – Cloud SQL offers **multi-zone high availability**.

■ **Failover Support** – If a zone fails, the system automatically switches to a standby instance.

■ **Read Replicas** – You can create read replicas for load balancing and performance improvement.

Security & Compliance

■ **IAM-Based Access Control** – Secure access via **Identity and Access Management (IAM)**.

■ **Encryption** – Data is encrypted at rest and in transit.

■ **VPC Peering & Private IPs** – Secure database connections using **private networking**.

Scalability & Performance

■ **Automatic Storage Increase** – If storage runs out, Cloud SQL expands automatically.

■ **Vertical Scaling** – You can increase CPU and memory as needed.

■ **Read Replicas** – Scale reads by distributing queries across replicas.

🔴 Set Up Cloud SQL for MySQL

Step 1: Enable Cloud SQL API

1. Open Google Cloud Console.
2. Go to **APIs & Services** → **Library**.
3. Search for **Cloud SQL Admin API** and click **Enable**.

Step 2: Create a Cloud SQL for MySQL Instance

1. Go to **Navigation Menu** (☰) → **SQL**.
2. Click **Create Instance** → **Choose MySQL**.
3. Set:
 - **Instance ID** (e.g., my-mysql-instance)
 - **Password** (for root user)
 - **Region & Zone** (choose near your app)
 - **Machine Type** (choose appropriate CPU & RAM)
 - **Storage Capacity** (set auto-increase if needed)
4. Click **Create** and wait for the instance to initialize.

Step 3: Connect to Cloud SQL

Using Cloud Console

1. Open **SQL** → Click on your instance.
2. Under **Connections**, find **Public IP** or **Private IP**.
3. Use the **Cloud SQL Auth Proxy** or MySQL client to connect.

Using MySQL Client

```
gcloud sql connect my-mysql-instance --user=root
```

or

```
mysql -u root -p -h [INSTANCE_IP]
```

Replace [INSTANCE_IP] with the actual instance IP.

Using Django/Flask

```
DATABASES = {
```

```
    'default': {
```

```
        'ENGINE': 'django.db.backends.mysql',
```

```
        'NAME': 'your-db-name',
```

```
        'USER': 'root',
```

```
        'PASSWORD': 'your-password',
```

```
        'HOST': '/cloudsql/your-project-id:your-region:your-instance',
```

```
        'PORT': '3306',
```

```
    }
```

```
}
```

Step 4: Enable High Availability (HA) (Optional)

1. Open your instance → Click **Edit**.
2. Enable **High Availability** and select a standby zone.
3. Save changes.

Step 5: Create a Read Replica (Optional)

1. Open your instance → Click **Create Read Replica**.
2. Select the region and name.
3. Click **Create**.

Step 6: Backup & Restore

Enable Automated Backups

1. Open your instance → Click **Backups**.
2. Click **Edit** → Enable automatic backups.

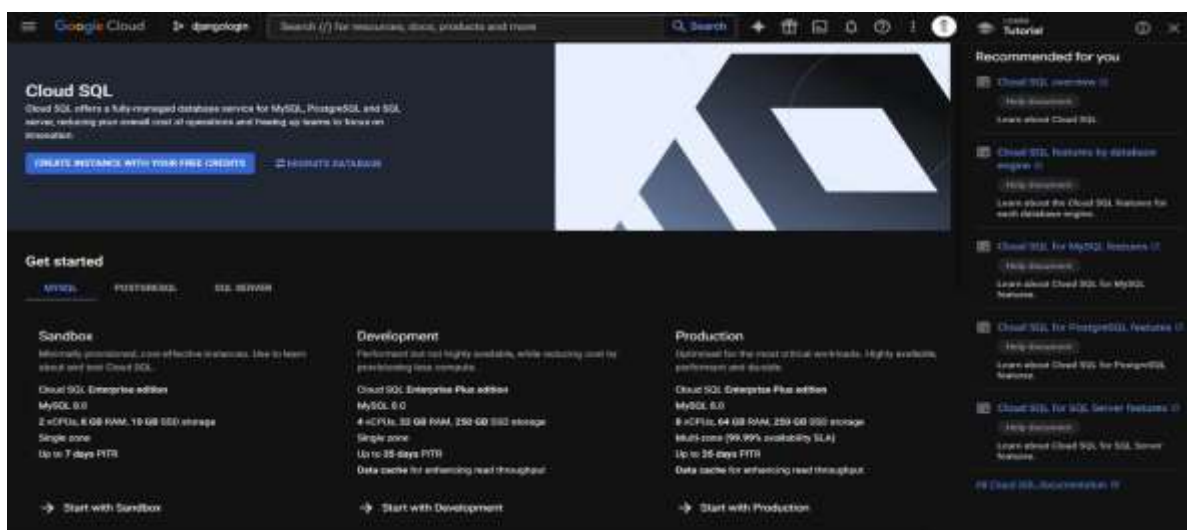
Manually Create a Backup

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

Restore from Backup

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

Output








Experiment-07

Cloud Pub/Sub: Experiment how Google Cloud Pub/Sub facilitate real-time messaging and communication between distributed applications

Google Cloud Pub/Sub is a fully managed **messaging service** that enables **asynchronous, real-time communication** between distributed applications. It follows a **publish-subscribe** model where **publishers send messages to topics** and **subscribers receive them** via push or pull delivery.

Why Use Cloud Pub/Sub?

-  **Real-time messaging** – Delivers messages instantly across services.
-  **Decouples components** – Microservices can communicate asynchronously.
-  **High availability & scalability** – Handles millions of messages per second.
-  **Guaranteed delivery** – Retries messages until they are acknowledged.
-  **Security** – Integrated with **IAM** for access control.

Cloud Pub/Sub Architecture

Publisher – Sends messages to a **Topic**.

Topic – A named channel where messages are published.

Subscription – Defines how messages are delivered to subscribers.

Subscriber – Reads messages from a subscription.

Delivery Mechanism – **Pull** (manual retrieval) or **Push** (automatic HTTP delivery).

◆ Step-by-Step: Experimenting with Cloud Pub/Sub

Step 1: Enable Cloud Pub/Sub API

1. Open **Google Cloud Console** → **Navigation Menu** (☰) → **APIs & Services** → **Library**.
2. Search for **Cloud Pub/Sub API** and click **Enable**.

Step 2: Create a Pub/Sub Topic

1. Go to **Navigation Menu** (☰) → **Pub/Sub** → **Topics**.
2. Click **Create Topic**.
3. Enter a **Topic ID** (e.g., my-topic).
4. Click **Create**.

 **Your topic is now ready!**

Step 3: Create a Subscription

1. Click on your **Topic** → **Create Subscription**.
2. Enter a **Subscription ID** (e.g., my-subscription).
3. Choose a **Delivery Type**:
 - **Pull** – Messages are manually fetched by the subscriber.
 - **Push** – Messages are automatically sent to an HTTP endpoint.

4. Click **Create**.

 **Your subscription is now linked to the topic!**

Step 4: Publish a Message (Using gcloud CLI)

Run the following command in **Cloud Shell**:

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

 **Message published successfully!**

Step 5: Pull Messages from Subscription (Using gcloud CLI)

Run the following command:

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

This will **retrieve and acknowledge messages** from my-subscription.

 **You will see the message: Hello, Pub/Sub!**

Step 6: Publish & Subscribe Using Python (Optional)

Install Google Cloud Pub/Sub SDK

```
pip install google-cloud-pubsub
```

Publisher Code (Python)

```
from google.cloud import pubsub_v1
```

```
project_id = "your-project-id"

topic_id = "my-topic"


publisher = pubsub_v1.PublisherClient()

topic_path = publisher.topic_path(project_id, topic_id)


message = "Hello, Pub/Sub from Python!"

future = publisher.publish(topic_path, message.encode("utf-8"))

print(f"Published message ID: {future.result()}")
```

◆ **Subscriber Code (Python)**

```
from google.cloud import pubsub_v1


project_id = "your-project-id"

subscription_id = "my-subscription"


subscriber = pubsub_v1.SubscriberClient()

subscription_path = subscriber.subscription_path(project_id, subscription_id)


def callback(message):

    print(f"Received: {message.data.decode('utf-8')}")
```

```
message.ack()
```

```
subscriber.subscribe(subscription_path, callback=callback)
```

```
print("Listening for messages...")
```

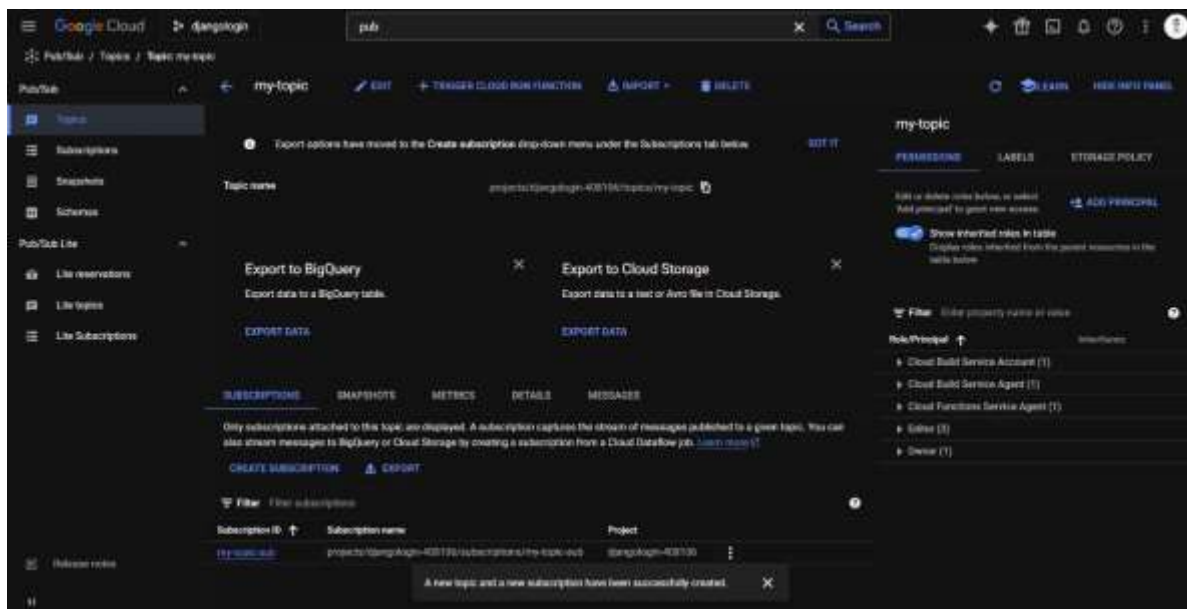
```
import time
```

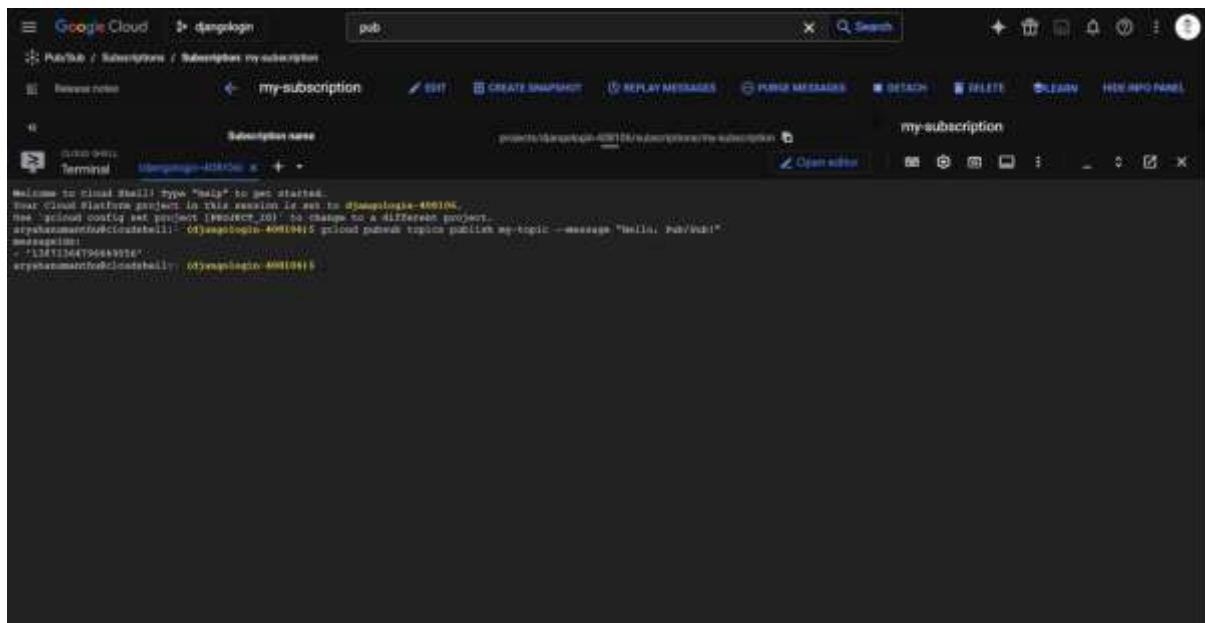
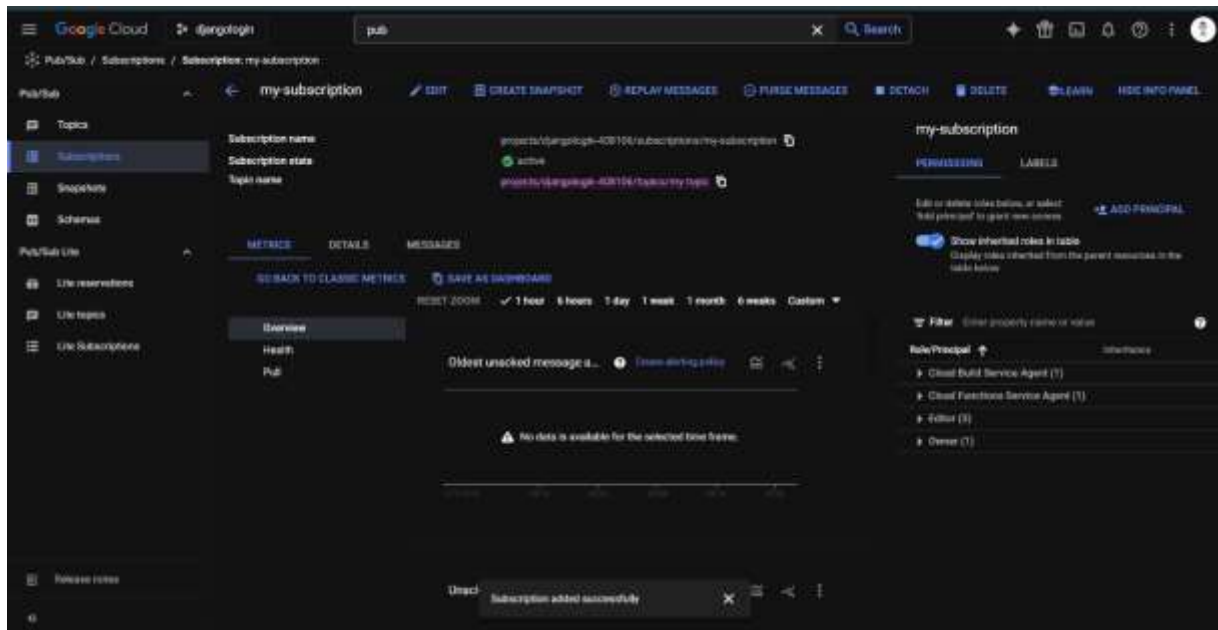
```
while True:
```

```
    time.sleep(10)
```

Now, whenever you publish a message, the subscriber will receive it in real-time!

Output





Experiment-08

Multiple VPC Networks: Explore benefits of using multiple VPC networks in Google Cloud for organizing and isolating resources.

Google Cloud, Virtual Private Cloud (VPC) allows you to define and control networking environments for your resources. You can have multiple VPC networks, each isolated from one another or interconnected for specific use cases. Managing multiple VPCs helps you scale, secure, and organize resources efficiently.

~~B~~enefits of Using Multiple VPC Networks

Resource Isolation & Security

- **Network Isolation** – You can isolate resources within different VPC networks to improve security and control traffic between services.
- **Private Connectivity** – Each VPC can have private IPs that do not communicate with other VPCs unless explicitly allowed, keeping sensitive data isolated.
- **Granular Firewall Rules** – Define specific firewall rules for each VPC, limiting access to resources within a VPC or between multiple VPCs.

Organizational Structure & Management

- **Separation by Department or Service** – Different teams or services (e.g., dev, test, production) can operate within their own VPCs, helping to organize and manage resources based on logical groupings.

- **Custom Subnetting** – Each VPC can have its own subnet structure tailored to the needs of specific projects or services.

Traffic Control

- **VPC Peering** – You can allow traffic between two or more VPCs by creating VPC peering connections. This gives you flexibility in managing traffic flow while maintaining network isolation.
- **Shared VPC** – A **Shared VPC** allows multiple projects to connect to a common VPC network, enabling central management of network resources.
- **Private Google Access** – For certain services, you can configure access to Google Cloud services without using public IPs, enhancing security.

Scaling Flexibility

- **Scalability for Different Environments** – As projects or environments grow, you can add more VPCs, allowing the architecture to scale without impacting other parts of the system.
- **Cross-Region Connectivity** – Create VPCs in different regions for disaster recovery and global distribution of your resources. Google Cloud provides the ability to set up global VPCs and establish secure connections across regions.

Enhanced Network Performance

- **Low Latency Communication** – By grouping resources that need high throughput and low latency within a specific VPC, you can optimize performance for specific workloads.

- **Dedicated Resources** – Certain VPCs can be dedicated to specific high-performance workloads (e.g., compute-intensive tasks), while others may be used for general workloads, ensuring efficient resource use.

' **Z Use Cases for Multiple VPCs**

Multi-Tier Applications

You can deploy **multi-tier architectures** where each tier (e.g., web, app, database) resides in separate VPC networks, enabling better isolation and security between tiers.

Cross-Region Architecture

You can deploy resources in multiple regions for **disaster recovery** or to meet **local compliance requirements** while maintaining network isolation between regions. For instance, a production VPC in one region and a disaster recovery VPC in another.

Hybrid Cloud or Multi-Cloud

If you're integrating **on-premises infrastructure** or other **cloud platforms** with Google Cloud, using separate VPCs for each environment allows secure and controlled network communication across different systems.

Managed Service Integration

You might have **managed services** (like **Cloud SQL** or **BigQuery**) in one VPC while using compute instances or other resources in another, optimizing resource placement.

◆ Set Up Multiple VPC Networks in Google Cloud

Step 1: Create a VPC Network

1. Go to **Google Cloud Console** → **Navigation Menu (☰)** → **VPC Network** → **Create VPC Network**.
2. Specify the **name**, **region**, and **subnet configuration** for your VPC.
3. Click **Create**.

Step 2: Create Additional VPC Networks

1. You can repeat the process to create as many VPCs as needed.
2. Choose **Custom** subnet mode to define your own subnets or **Auto mode** for auto-assigned subnets.

Step 3: Set Up VPC Peering (Optional)

1. Go to **VPC Network Peering** → **Create Peering Connection**.
2. Select the **Source VPC** and **Destination VPC**.
3. Define the **network and routes** that can be shared across the VPCs.
4. Click **Create**.

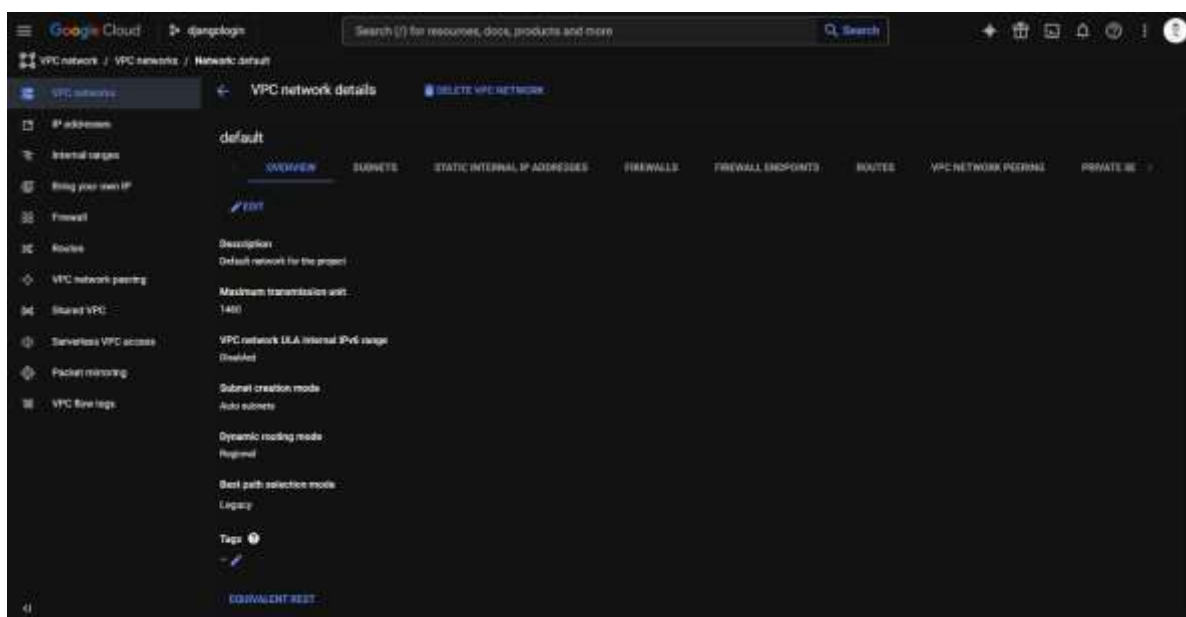
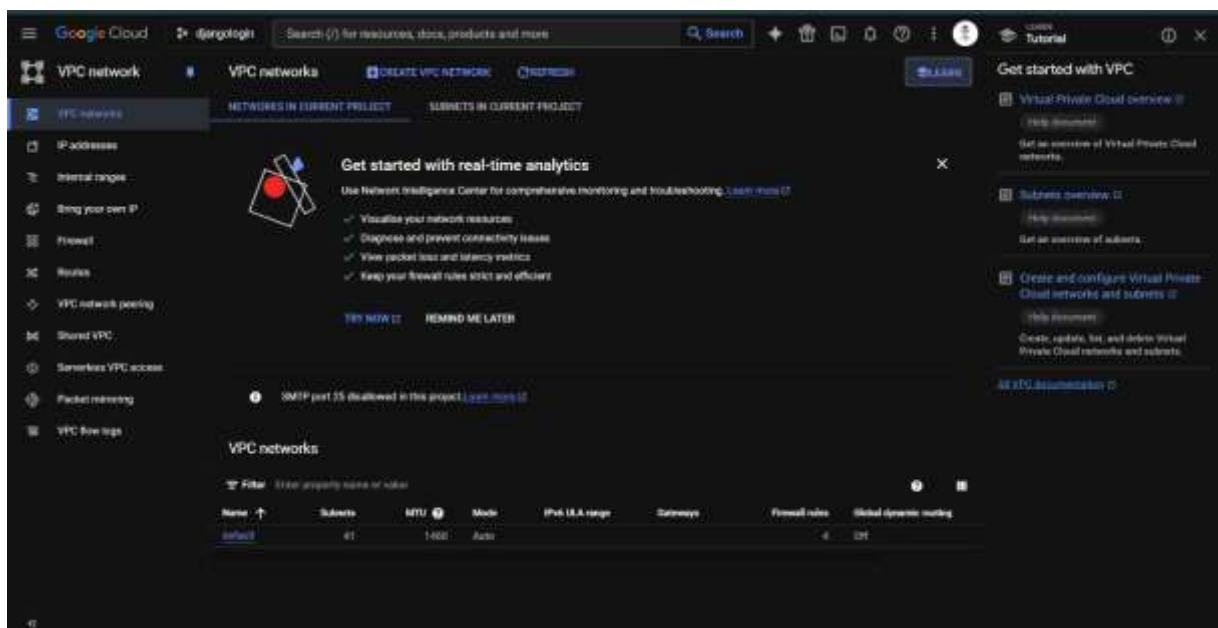
Step 4: Create Firewall Rules (Optional)

1. Go to **Firewall Rules** → **Create Firewall Rule**.
2. Define the **source and destination** VPCs, and configure the firewall to allow or deny traffic between the VPCs.

Step 5: Set Up Shared VPC (Optional)

1. Go to **VPC Networks** → **Shared VPC** → **Set up a Shared VPC**.
2. Select the **host project** and **service projects**.
3. Share the VPC resources with other projects.

Output



Experiment-09

Cloud Monitoring: Discover how Cloud Monitoring help in tracking and analyzing the performance and health of cloud resources?

Google Cloud Monitoring (formerly **Stackdriver Monitoring**) provides robust monitoring, alerting, and dashboard capabilities to track and analyze the performance and health of resources in your Google Cloud environment. It helps you ensure that your cloud infrastructure and applications are functioning efficiently and can scale as needed.

Key Features of Google Cloud Monitoring

Performance Tracking

- **Resource Metrics** – Track metrics for **Compute Engine**, **Kubernetes Engine**, **Cloud Functions**, **Cloud SQL**, and other services to assess resource usage (e.g., CPU, memory, disk, network usage).
- **Custom Metrics** – You can define your own custom metrics for specific applications to monitor application-level health or performance.
- **Real-Time Metrics** – Get near real-time data updates on how your infrastructure is performing.

Health Monitoring

- **Health Checks** – Cloud Monitoring can check the health of your **Compute Engine** instances, **App Engine**, or any other services you configure, ensuring that everything is running smoothly.

- **Uptime Checks** – Automate the monitoring of service availability across regions and ensure that downtime is minimized.

Alerting and Notifications

- **Smart Alerts** – Create alerts based on specific thresholds for metrics such as CPU utilization, memory usage, or response time.
- **Notification Channels** – Alerts can be routed through email, SMS, Slack, or other communication platforms, ensuring that you are immediately notified about any issues.
- **Escalation Policies** – Implement escalation policies so that alerts are sent to the right teams if an issue persists.

Dashboards and Visualization

- **Custom Dashboards** – Create visual dashboards that provide a snapshot of the health and performance of your Google Cloud services, apps, and infrastructure.
- **Predefined Dashboards** – Use built-in dashboards for common Google Cloud resources like **Compute Engine**, **Kubernetes Engine**, and **Cloud Pub/Sub**.

Distributed Tracing and Logging

- **Distributed Tracing** – Track the performance of services that interact with one another, pinpointing latency or bottlenecks in distributed systems, microservices, and serverless architectures.
- **Log Analysis** – Cloud Monitoring integrates with **Cloud Logging** to collect logs from your applications and infrastructure. It helps you quickly diagnose issues by correlating logs with performance metrics.

Integration with Cloud Services

- **Cloud Monitoring for Kubernetes** – Keep track of the health and performance of your Kubernetes clusters, containers, and pods.
- **Integration with Google Cloud Services** – Seamlessly integrates with **Compute Engine**, **Google Kubernetes Engine**, **Cloud Functions**, and **Cloud Run** to give you full visibility into your infrastructure.

Use Cloud Monitoring

Step 1: Enable Cloud Monitoring API

1. Open the **Google Cloud Console** → **Navigation Menu (☰)** → **APIs & Services** → **Library**.
2. Search for **Cloud Monitoring API** and click **Enable**.

Step 2: Set Up Monitoring for Your Resources

1. Go to **Navigation Menu (☰)** → **Monitoring** → **Dashboards**.
2. Click on **Create Dashboard** to start building your custom dashboard.
3. Select **Metrics** from the dropdown and choose the service you want to monitor (e.g., **Compute Engine**, **Cloud Storage**, **Cloud Functions**).
4. Add the required metrics to your dashboard and adjust visualizations like **line charts**, **heat maps**, or **bar charts**.

Step 3: Set Up Alerts

1. Go to **Navigation Menu (☰) → Monitoring → Alerting**.
2. Click **Create Policy**.
3. Choose the **condition** (e.g., CPU usage > 80%).
4. Select the **notification channels** (e.g., email, Slack, SMS) where the alert should be sent.
5. Set up **escalation policies** to ensure the right team is notified.
6. Click **Create** to finish the alert policy.

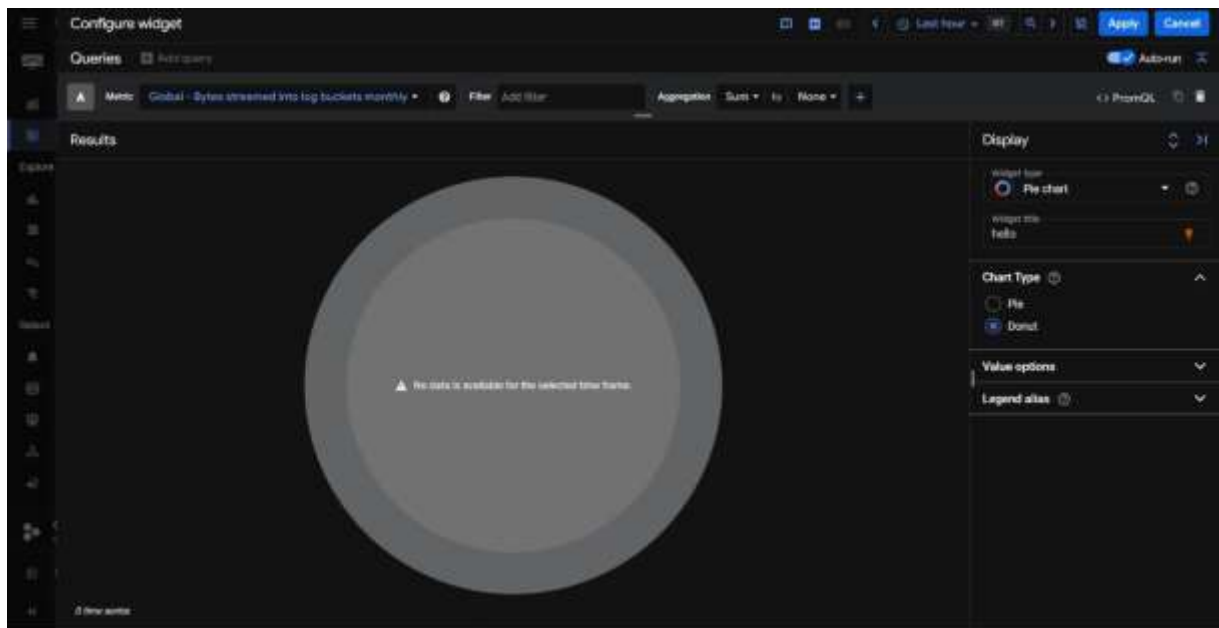
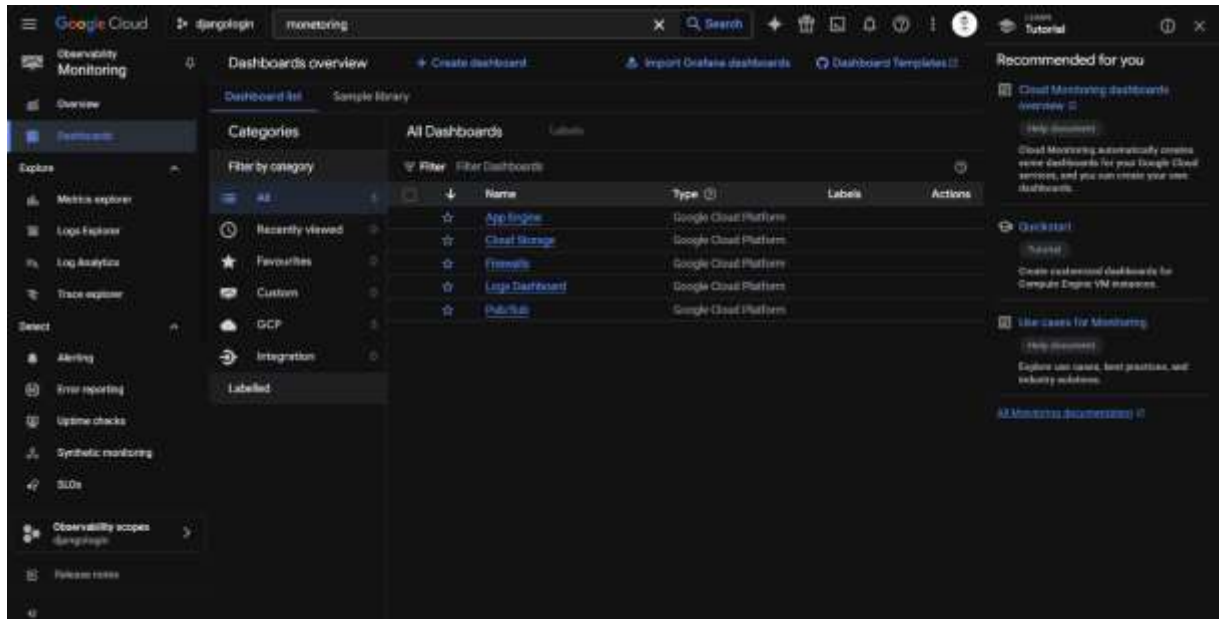
Step 4: Review Logs

1. Go to **Navigation Menu (☰) → Logging**.
2. Choose the **resource type** (e.g., VM instances, Kubernetes Engine) and define your log filter.
3. Use **Log Explorer** to search for specific logs, such as **error messages** or **performance warnings**, and correlate them with metrics in Cloud Monitoring.

Step 5: Set Up Distributed Tracing (Optional)

1. Open **Cloud Trace** from the **Navigation Menu**.
2. Enable **trace collection** in your services (e.g., by using the **Cloud Trace SDK** for your app).
3. View trace data to identify latency or bottlenecks in your system.

Output



Experiment-10

Kubernetes Engine: Qwik Start: Deploy a containerized application to a Kubernetes Engine cluster.

Steps to Deploy a Containerized Application to GKE

Set Up Google Cloud SDK and Kubernetes Tools

1. Install Google Cloud SDK

If you haven't already installed the Google Cloud SDK, follow the instructions here:

Google Cloud SDK Installation

2. Install kubectl

kubectl is the Kubernetes command-line tool used to manage Kubernetes clusters. The Cloud SDK includes kubectl, so if you have the SDK installed, you already have kubectl.

Create a Google Cloud Project

1. Create a new Google Cloud project (if you don't already have one):

- Go to the Google Cloud Console.
- Click on **Select a Project > New Project**.
- Name your project and click **Create**.

2. Set your project in the gcloud CLI:

`gcloud config set project PROJECT_ID`

Enable Required APIs

1. Enable Kubernetes Engine API:

`gcloud services enable container.googleapis.com`

2. Enable Compute Engine API (if not already enabled):

`gcloud services enable compute.googleapis.com`

Create a Kubernetes Cluster

1. Create the Kubernetes Engine cluster:

`gcloud container clusters create my-cluster --zone us-central1-a`

Replace my-cluster with your desired cluster name and adjust the zone if necessary.

2. Get the credentials for your cluster: This command configures kubectl to use the cluster you just created.

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

Create a Containerized Application

1. Create a Dockerfile for your application. Below is an example for a simple web application using Node.js:

Dockerfile:


```
dockerfile
```

```
CopyEdit
```

```
FROM node:14
```

```
WORKDIR /usr/src/app
```

```
COPY . .
```

```
RUN npm install
```

```
EXPOSE 8080
```

```
CMD [ "npm", "start" ]
```

2. Build the Docker image:

```
docker build -t gcr.io/PROJECT_ID/my-app:v1 .
```

Replace PROJECT_ID with your Google Cloud project ID.

3. Push the image to Google Container Registry:

```
docker push gcr.io/PROJECT_ID/my-app:v1
```

Create a Kubernetes Deployment

1. Create a Kubernetes Deployment configuration file

(deployment.yaml) for your containerized application.

deployment.yaml:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

name: my-app

spec:

replicas: 3

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: my-app

image: gcr.io/PROJECT_ID/my-app:v1

ports:

- containerPort: 8080

Replace PROJECT_ID with your project ID.

2. Apply the Deployment to the Kubernetes cluster:

kubectl apply -f deployment.yaml

Expose the Application via a Service

1. **Create a Service to expose the application** (either LoadBalancer or ClusterIP for internal access).

Example **Service.yaml** for external exposure (LoadBalancer type):

```
apiVersion: v1
```

```
kind: Service
```

```
metadata:
```

```
  name: my-app-service
```

```
spec:
```

```
  selector:
```

```
    app: my-app
```

```
  ports:
```

```
    - protocol: TCP
```

```
      port: 80
```

```
      targetPort: 8080
```

```
  type: LoadBalancer
```

2. **Apply the Service configuration:**

```
kubectl apply -f service.yaml
```

3. **Get the external IP address:** It may take a few moments for the LoadBalancer to be provisioned.

```
kubectl get svc
```

The **EXTERNAL-IP** column will show the public IP once the LoadBalancer is provisioned.

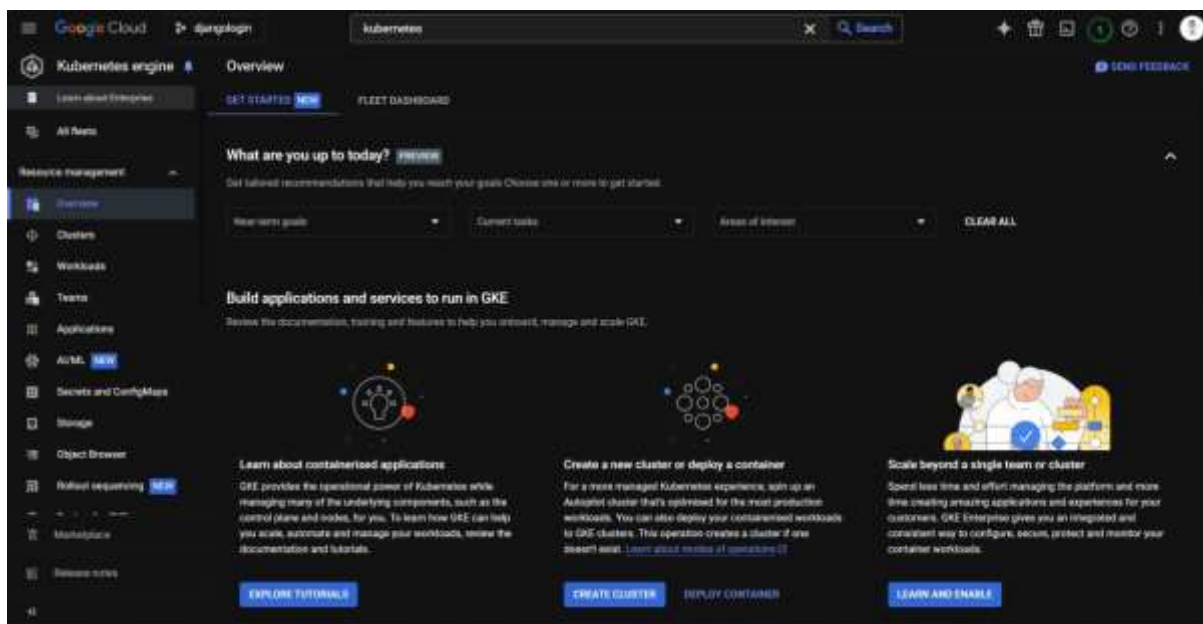
Verify the Application

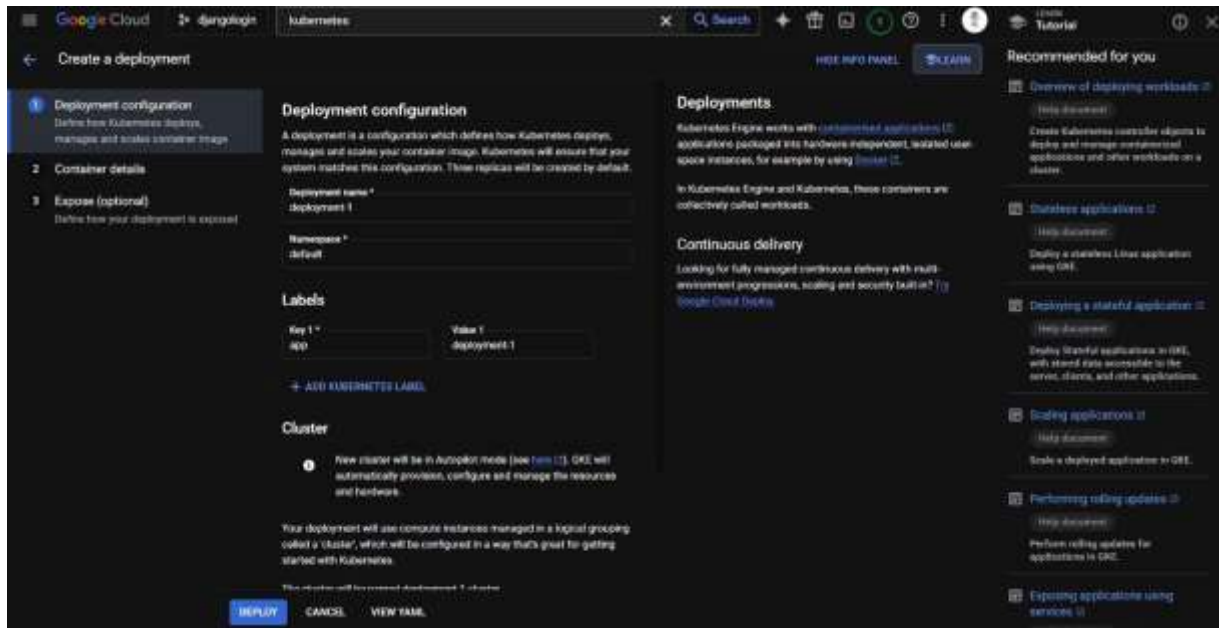
1. Open a web browser and navigate to the external IP address to verify your application is running.
2. You can also use kubectl to get the status of your pods and services:

```
kubectl get pods
```

```
kubectl get svc
```

Output





The screenshot shows the 'Create a deployment' wizard in the Google Cloud console for Kubernetes Engine. The interface is in a dark theme.

- Left Sidebar:** Contains navigation links for 'Deployment configuration', 'Container details', and 'Expose (optional)'.
- Deployment configuration:**
 - Deployment name:** deployment-1
 - Namespace:** default
 - Labels:** A table with one entry: Key 'id' and Value 'deployment-1'.
 - Cluster:** A note stating 'New cluster will be in Autopilot mode (see here)'. It mentions that GKE will automatically provision, configure, and manage resources and hardware.
- Deployments:** A section explaining that Kubernetes Engine works with containerized applications and that containers are collectively called workloads.
- Continuous delivery:** A section mentioning 'Looking for fully managed continuous delivery with multi-environment pipelines, scaling and security built in?'.
- Recommended for you:** A list of suggested actions:
 - Overview of deploying workloads
 - Deploying applications to GKE
 - Deploying a stateful application
 - Scaling applications in GKE
 - Performing rolling updates
 - Exposing applications using services
- Bottom:** Buttons for 'DEPLOY', 'CANCEL', and 'VIEW YAML'.