Kazakh-British Technical University

Cloud Computing
Assingment3

Student: Khanfiyeva Elnara
Professor: Serek Azamat

2024

Table of Contents

Introduction

Today's cloud computing requires the development of efficient, scalable, and secure applications. By keeping developers at the heart of the process and avoiding the need for them to plan complex infrastructure extensively, Google Cloud Platform (GCP) provides a set of services that may be programmed to achieve these objectives. Key GCP components are examined in this document, including Google Kubernetes Engine (GKE), App Engine, and Google Cloud Functions. To ensure that users and services have the appropriate right of access to resources, these two leverage concepts like Identity and Access Management (IAM), role-based access control, encryption, and cloud login.

Using the well-known container orchestration platform Kubernetes, which gives developers control over a microservices environment, GKE facilitates the deployment and scaling of containerized applications. App Engine is a PaaS service that lets users launch web apps without having to worry about infrastructure management. It handles size adjustments and monitoring. However, Cloud Functions, which work best with small-scale, horizontal, and microservices, operate on a serverless principle in which a collection of functions are run in response to events such as file uploads, HTTP calls, and database updates. All of these elements work together to ensure scalability and flexibility through auto scaling, load balancing, and self-tension adjustments; security and governance with the latest tools; and better developer productivity due to the abstraction of much of the underlying infrastructure.

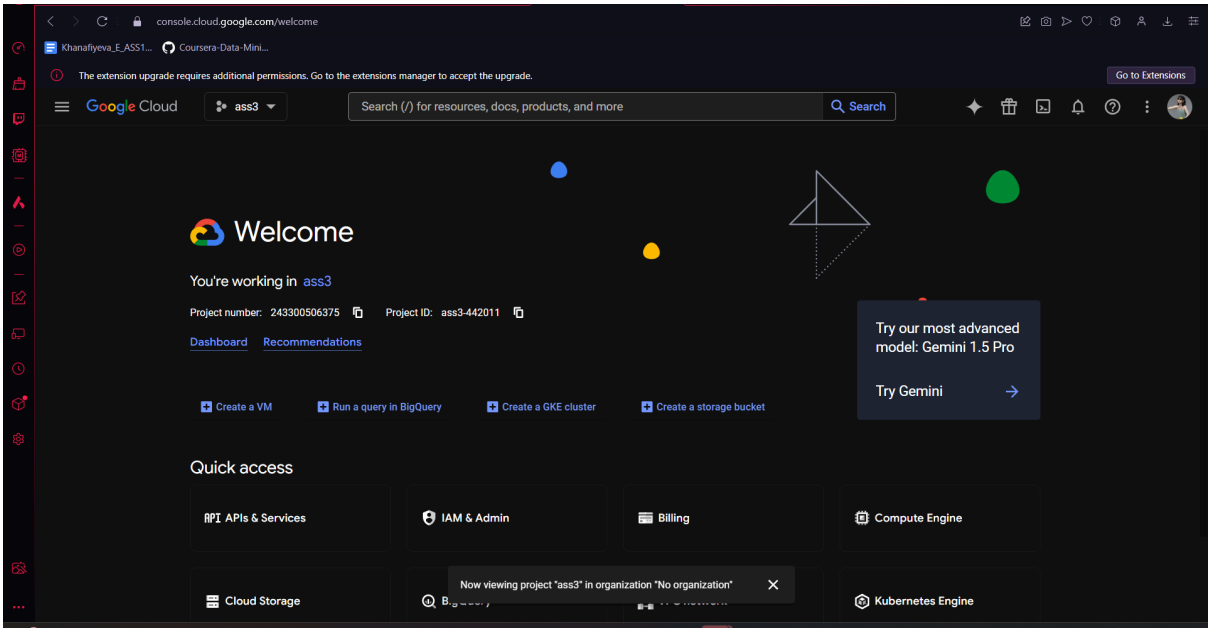
Additionally, managed services like App Engine and GKE, as well as serverless techniques like Cloud Functions, are cost-effective, pay only for what you use, consume model, and decrease operations. With the help of these GCP services, developers and businesses can create scalable, competitive, and secure cloud-based solutions that help them fulfill the ever-evolving economic demands of a technologically advanced market.

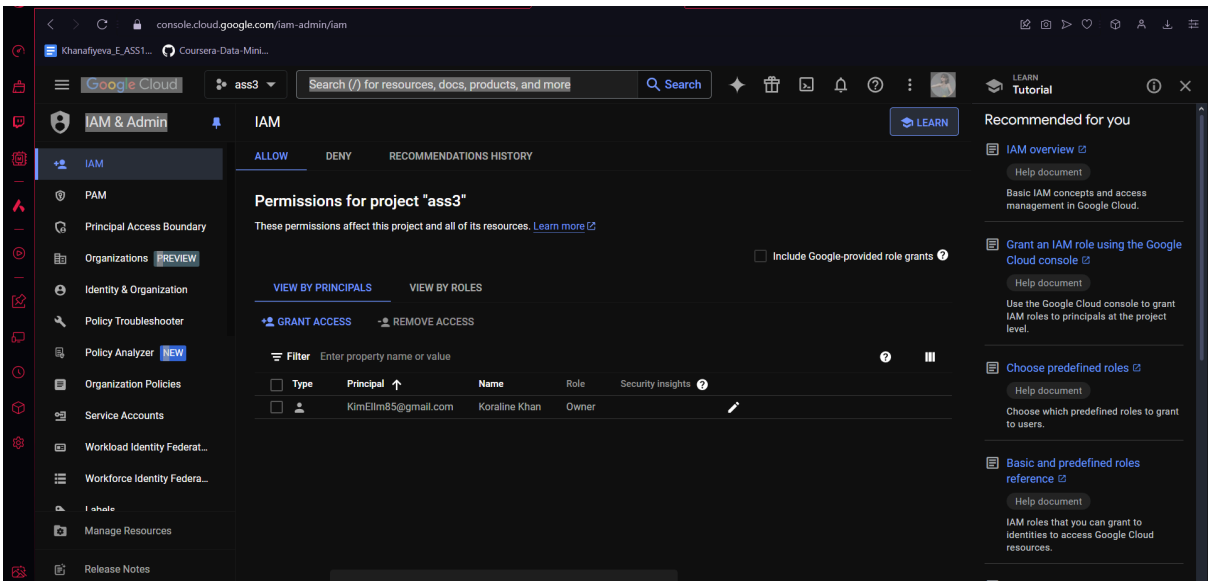
1. Identity and Security Management

Exercise 1: Setting Up IAM Roles

- Create a new Google Cloud project.
- Set up Identity and Access Management (IAM) roles for different team members (e.g., Viewer, Editor, Owner).
- Assign specific roles to users and document the permissions associated with each role.

First I go to the Google Cloud Console, and create the project fist.



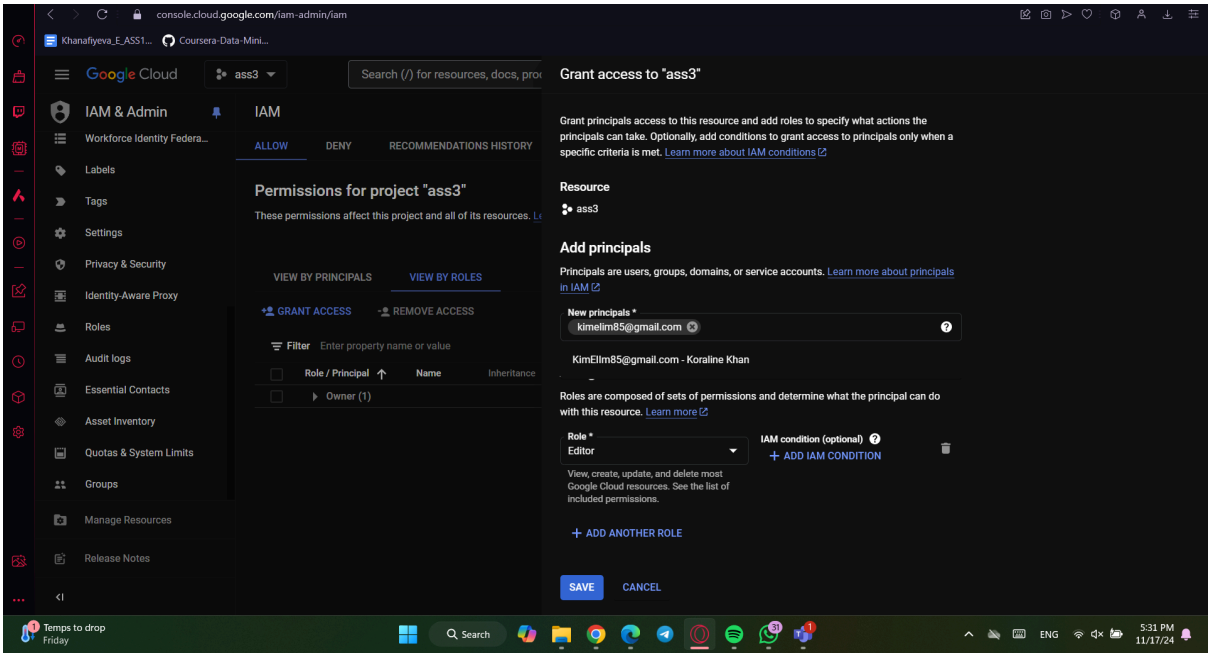


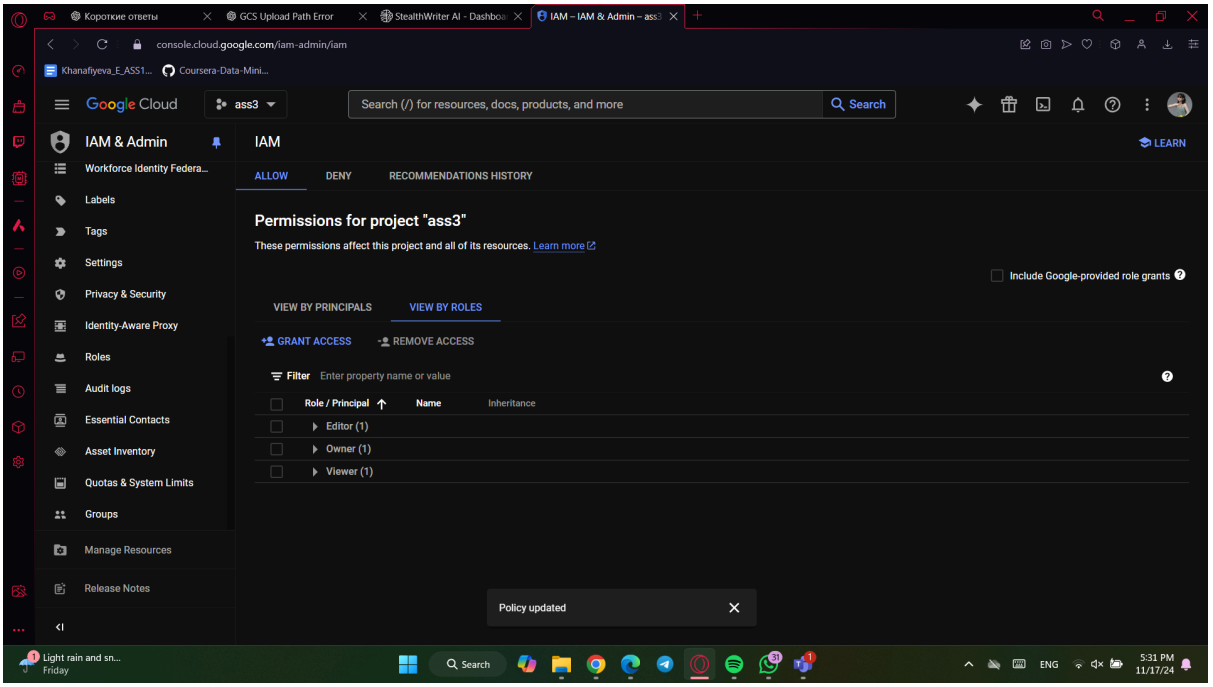
Now we create the IAM & Admin > IAM assign roles of Viewer, Editor, Owner to different users including my account.

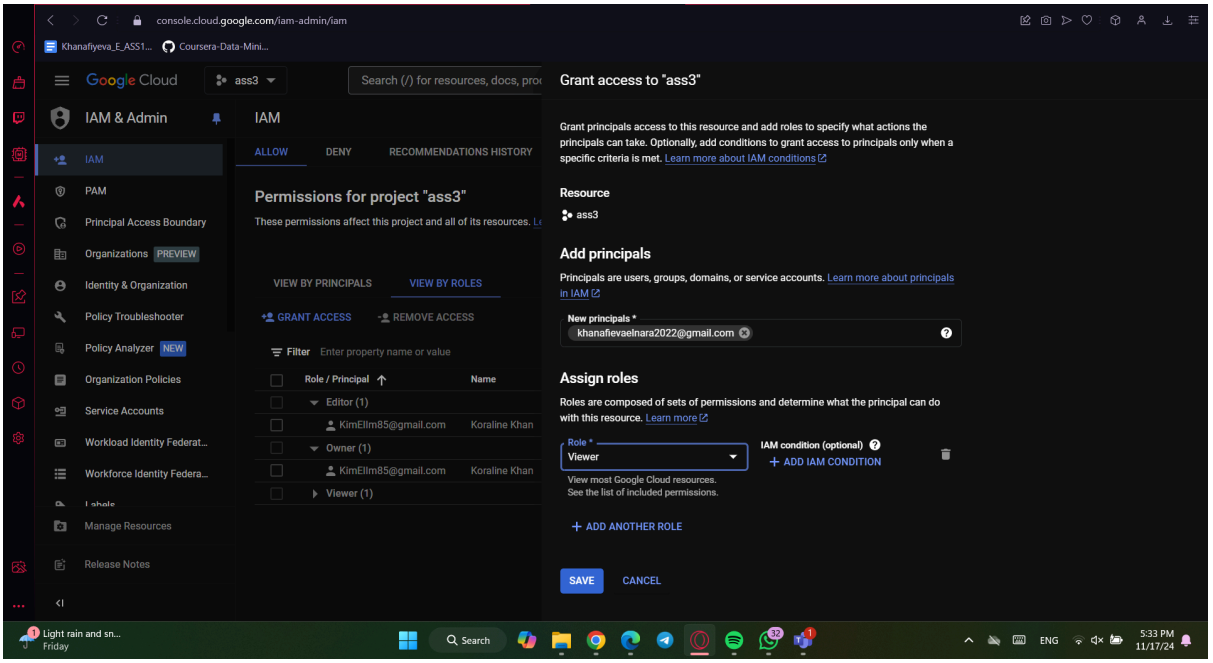
For the viewer the permissions usually are: resourcemanager.projects.get , resourcemanager.projects.list, compute.instances.get, storage.objects.get.

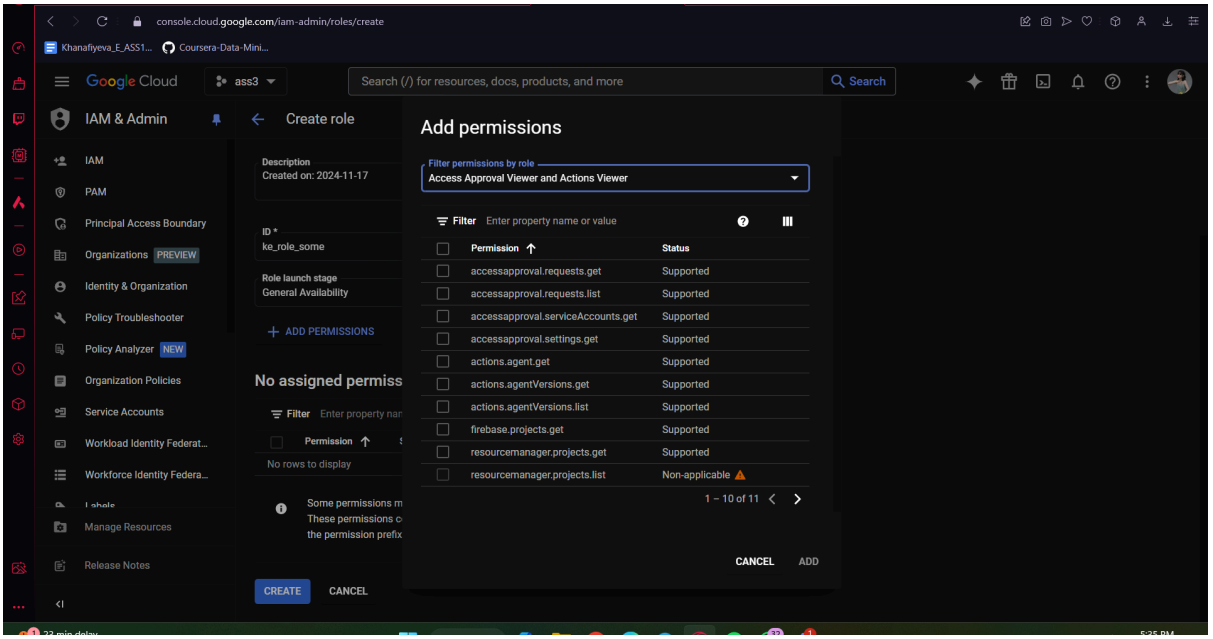
For editor is the create and delete permissions, and for the owner is the setting permissions policies, updating and deleting.

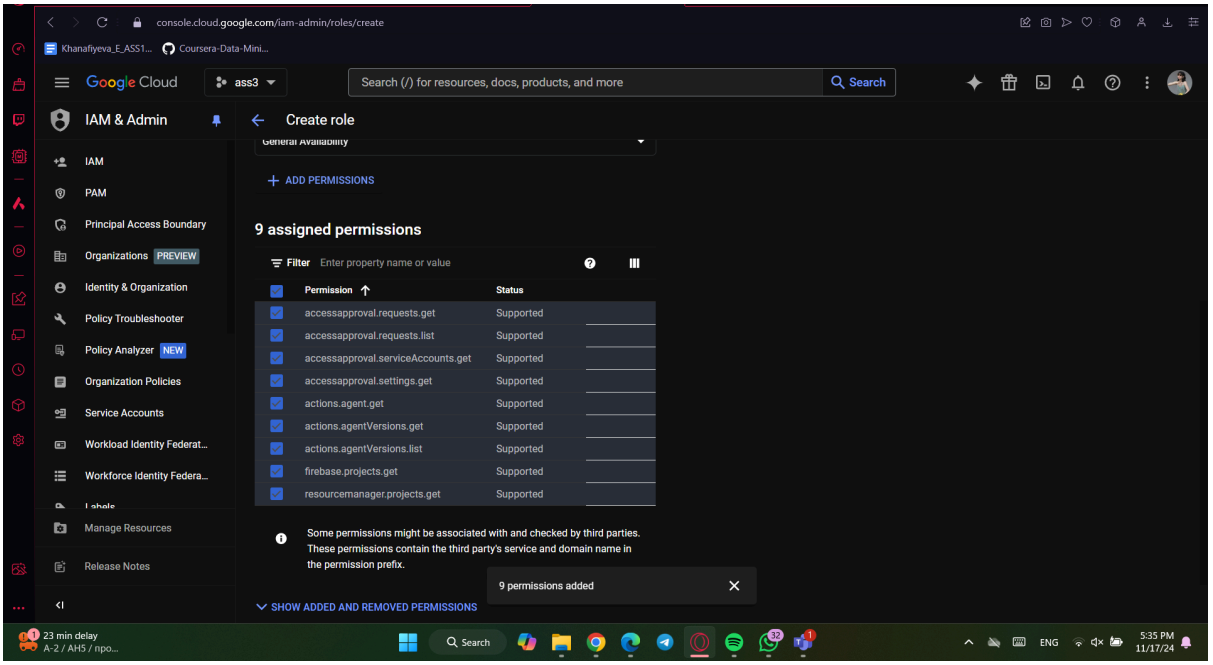
Expected Outcome: Users are assigned roles with specific permissions.

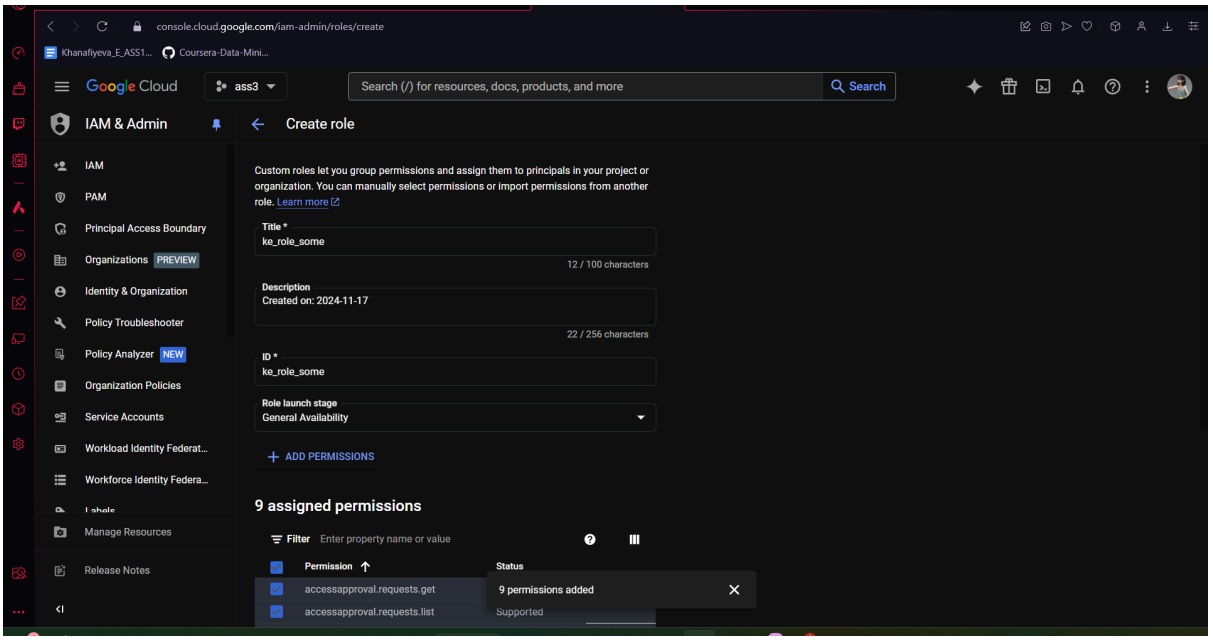





console.cloud.google.com/iam-admin/iam
Google Cloud
ass3
Search (/) for resources, docs, pro
IAM & Admin
IAM
ALLOW
DENY
RECOMMENDATIONS HISTORY
Permissions for project "ass3"
These permissions affect this project and all of its resources.
VIEW BY PRINCIPALS
VIEW BY ROLES
GRANT ACCESS
REMOVE ACCESS
Filter
Enter property name or value
Role / Principal
Name
Editor (1)
KimElIm85@gmail.com
Koraline Khan
Owner (1)
KimElIm85@gmail.com
Koraline Khan
Viewer (1)
IAM
PAM
Principal Access Boundary
Organizations
PREVIEW
Identity & Organization
Policy Troubleshooter
Policy Analyzer
NEW
Organization Policies
Service Accounts
Workload Identity Federat...
Workforce Identity Federa...
Manage Resources
Release Notes
Grant access to "ass3"
Grant principals access to this resource and add roles to specify what actions the principals can take. Optionally, add conditions to grant access to principals only when a specific criteria is met. Learn more about IAM conditions
Resource
ass3
Add principals
Principals are users, groups, domains, or service accounts. Learn more about principals in IAM
New principals *
khanafievaelnara2022@gmail.com
Assign roles
Roles are composed of sets of permissions and determine what the principal can do with this resource. Learn more
Role *
Viewer
IAM condition (optional)
ADD IAM CONDITION
View most Google Cloud resources.
See the list of included permissions.
ADD ANOTHER ROLE
SAVE
CANCEL
Light rain and sn...
Friday
Search
ENG
5:33 PM
11/17/24


console.cloud.google.com/iam-admin/roles/create
Google Cloud
ass3
Search (/) for resources, docs, products, and more
Search
IAM & Admin
Create role
Description
Created on: 2024-11-17
ID *
ke_role_some
Role launch stage
General Availability
ADD PERMISSIONS
No assigned permiss
Filter
Enter property nar
Permission
No rows to display
Some permissions m
These permissions c
the permission prefix
CREATE
CANCEL
Add permissions
Filter permissions by role
Access Approval Viewer and Actions Viewer
Filter
Enter property name or value
Permission
Status
accessapproval.requests.get
Supported
accessapproval.requests.list
Supported
accessapproval.serviceAccounts.get
Supported
accessapproval.settings.get
Supported
actions.agent.get
Supported
actions.agentVersions.get
Supported
actions.agentVersions.list
Supported
firebase.projects.get
Supported
resourcemanager.projects.get
Supported
resourcemanager.projects.list
Non-applicable
1 – 10 of 11
CANCEL
ADD



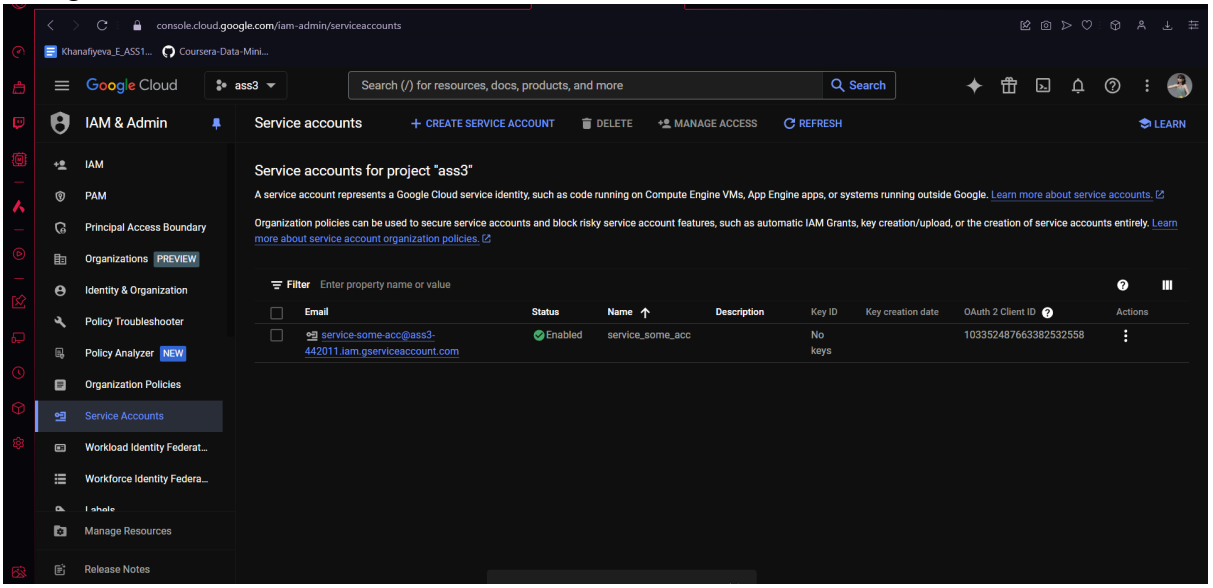


Exercise 2: Service Accounts

- Create a service account that can access Google Cloud Storage.
- Generate and download a key for this service account, and use it to authenticate a Python script that uploads a file to a Cloud Storage bucket.

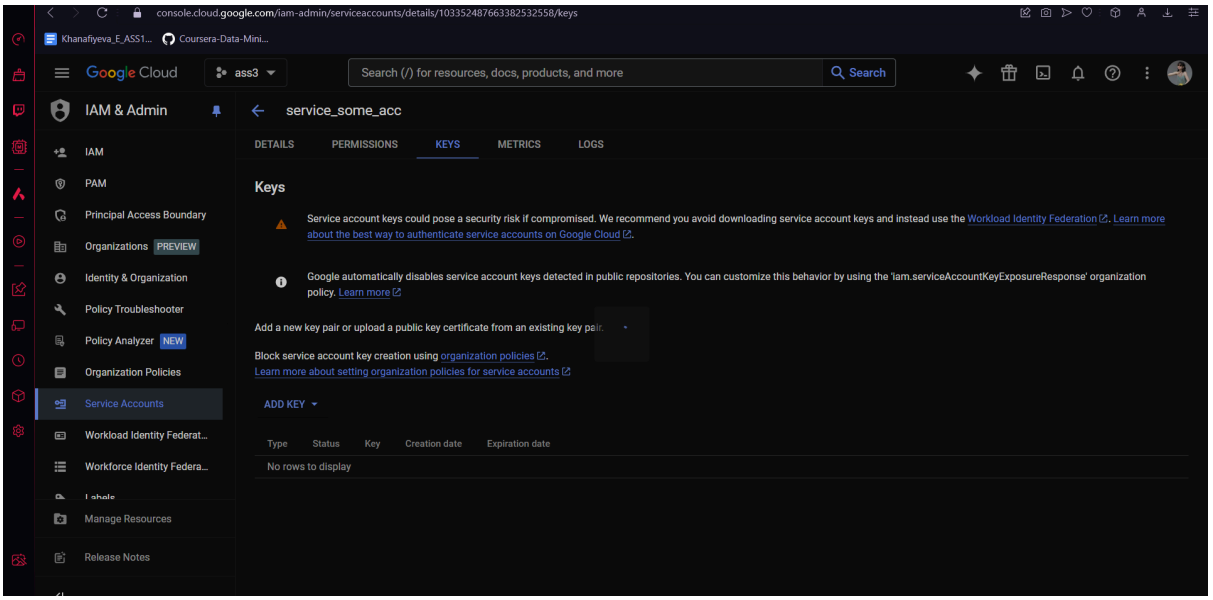
The service account is a special kind of account used by an application or compute workload, rather than a person, we will use it to authenticate a script for accessing Google Cloud

Storage.



Create a service account in IAM console. I will assign the role of **Storage Object Admin** to the service account, to have an access from my console to a storage.

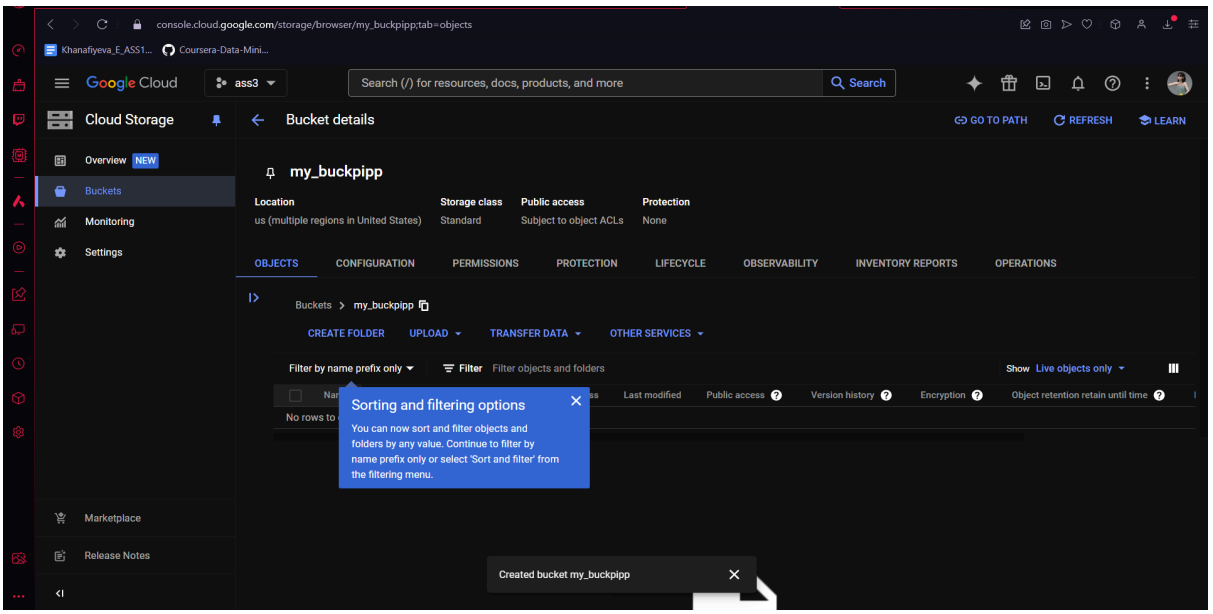
I will generate JSON key of service account in the console. and download it.



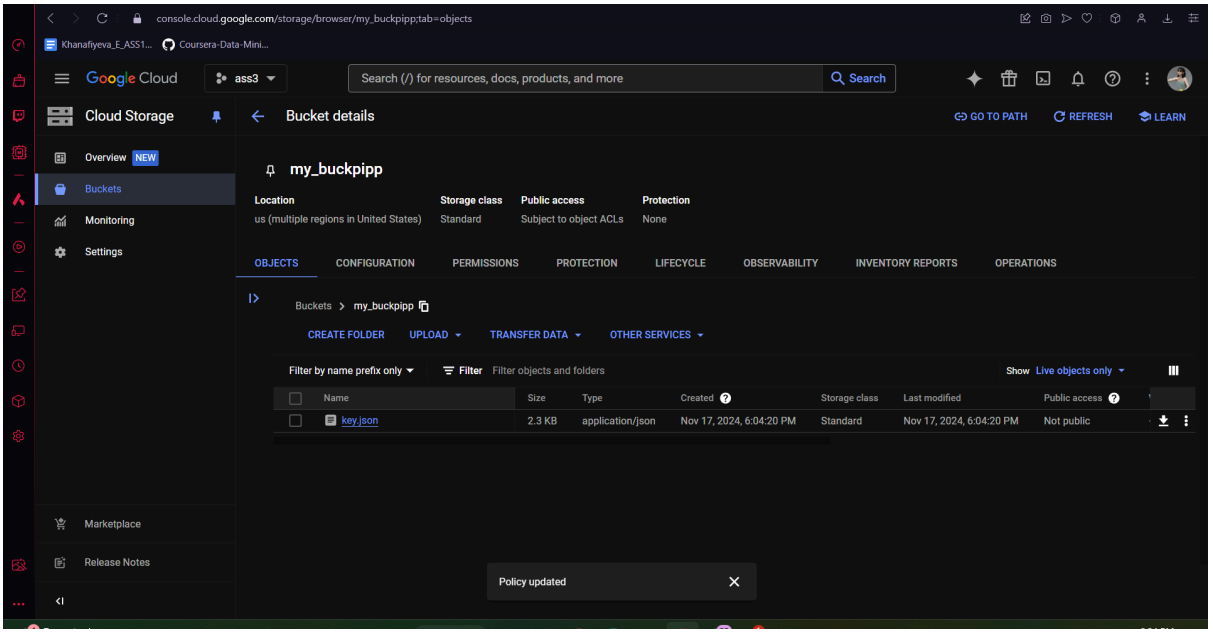
```
from google.cloud import storage

def upload_to_bucket(bucket_name, source_file_name,
destination_blob_name):
    storage_client =
storage.Client.from_service_account_json('C:/Users/elnus/Downloads/as
s3-442011-a63ede4088c4.json')
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)
    blob.upload_from_filename(source_file_name)
    print(f"File {source_file_name} uploaded to
{destination_blob_name}.")
upload_to_bucket('my_buckpipp',
'C:/Users/elnus/Downloads/ass3-442011-a63ede4088c4.json', 'key.json')
```

This script will upload the file to the bucket.

Here is bucket created :




```
C:\Users\elnus\Downloads>python script.py
File C:/Users/elnus/Downloads/ass3-442011-a63ede4088c4.json uploaded to key.json.

C:\Users\elnus\Downloads>
```

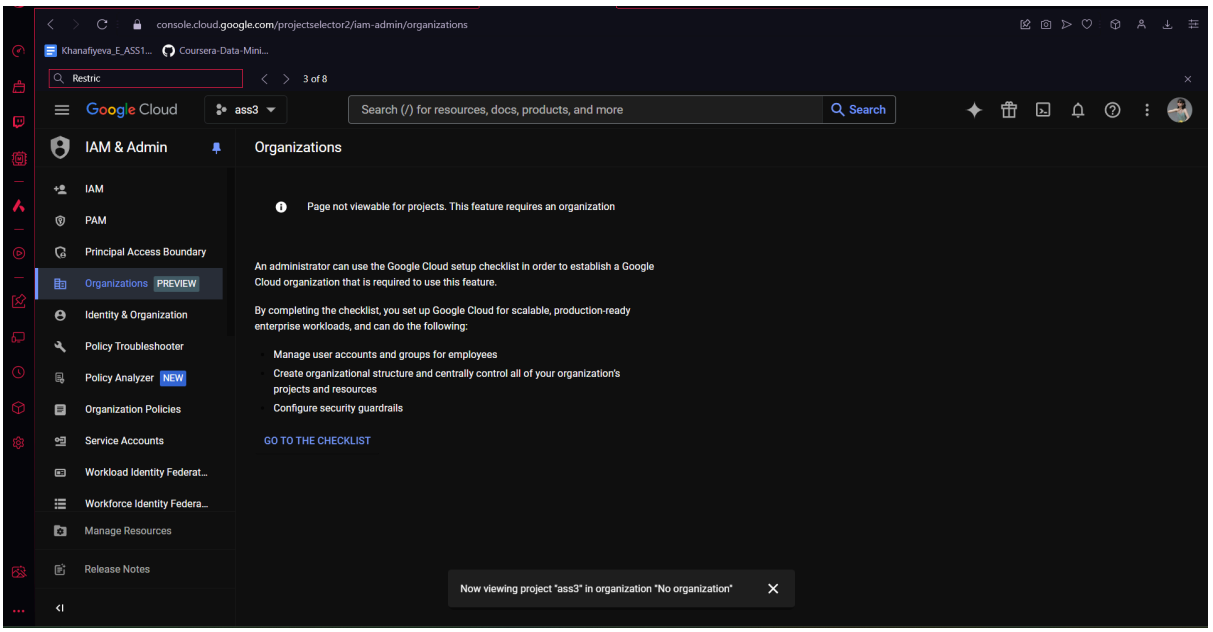


I ran the script and the key got uploaded to the bucket.

Exercise 3: Organization Policies

- Explore organization policies by applying a restriction (e.g., disabling the creation of public IP addresses) at the organization or project level.
- Document the steps to enforce and view the organization policy settings.

I go to IAM & Admin > Organization Policies and then apply a policy disabling the public IP, and there for I can't access it from the outside now.



2. Google Kubernetes Engine (GKE)

Exercise 4: Deploying a Simple Application

- Set up a GKE cluster using the Google Cloud Console or gcloud command line.
- Deploy a simple containerized application (e.g., a Hello World app) to the cluster and expose it via a LoadBalancer service.

I go to the Google Console and Kubernetes Engine > Clusters > Create Cluster.

After creating a cluster I create the yaml file for an image for K8s.

deply.yaml | service.yaml | multi-container-deployment.yaml

C: > Users > elnus > Downloads > deply.yaml

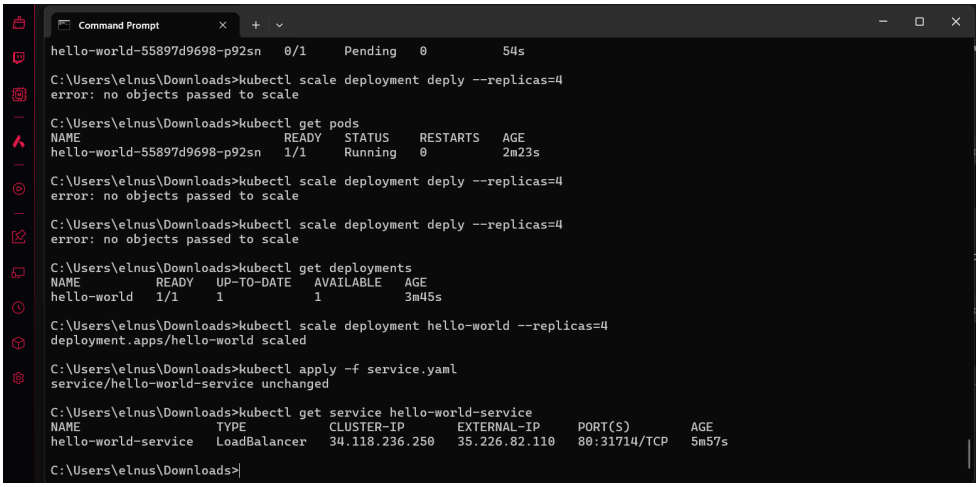
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-world
        image: gcr.io/google-samples/hello-app:1.0
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world-service
spec:
  type: LoadBalancer
  selector:
    app: hello-world
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

I apply the file to k8s by kubectl apply -f deply.yaml

gcloud init

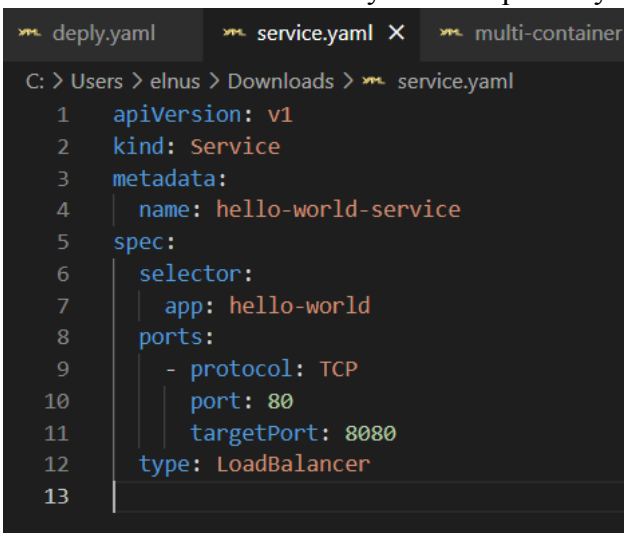
gcloud container clusters create smth --zone us-central1-a

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



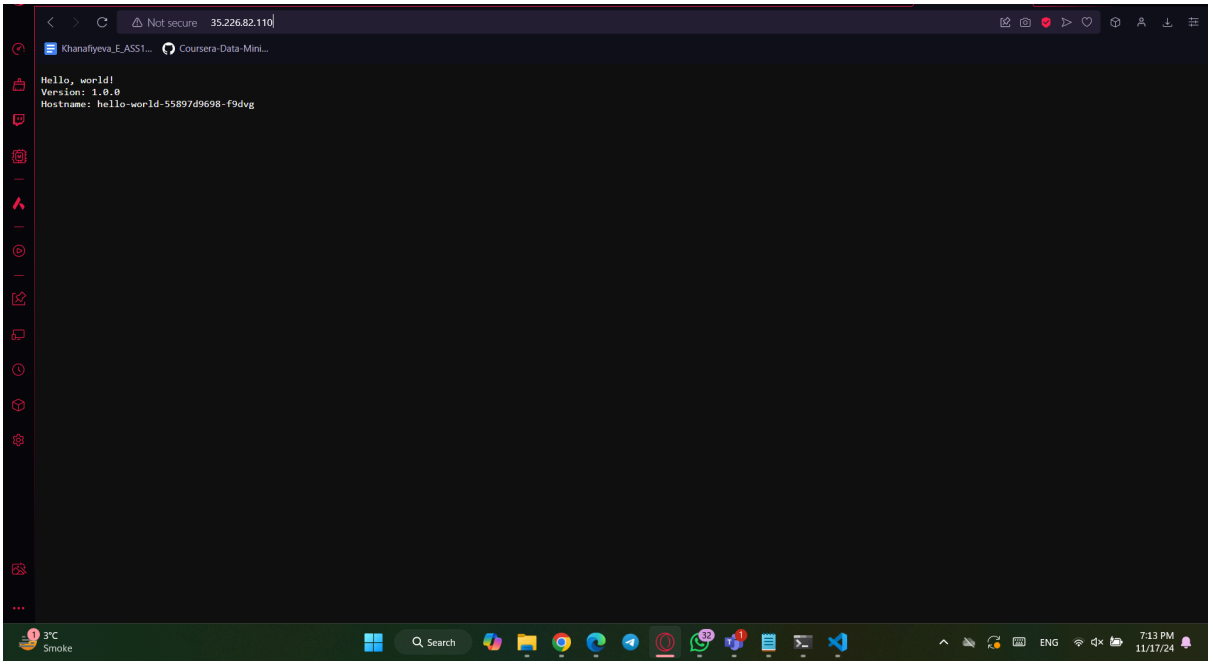
kubectl get pods

I create a file named service.yaml to expose my application using a LoadBalancer:



kubectl apply -f service.yaml

kubectl get service service



As expected it’s available.

Exercise 5: Managing Pods and Deployments

- Create a Deployment for a multi-container application using Kubernetes YAML files.
- Scale the Deployment to manage the number of replicas and update the application with a new container image.

We scale the deployment that we have by kubectl scale deployment hello-world --replicas=3.

We make another yaml file:

deply.yaml | service.yaml | multi-container-deployment.yaml

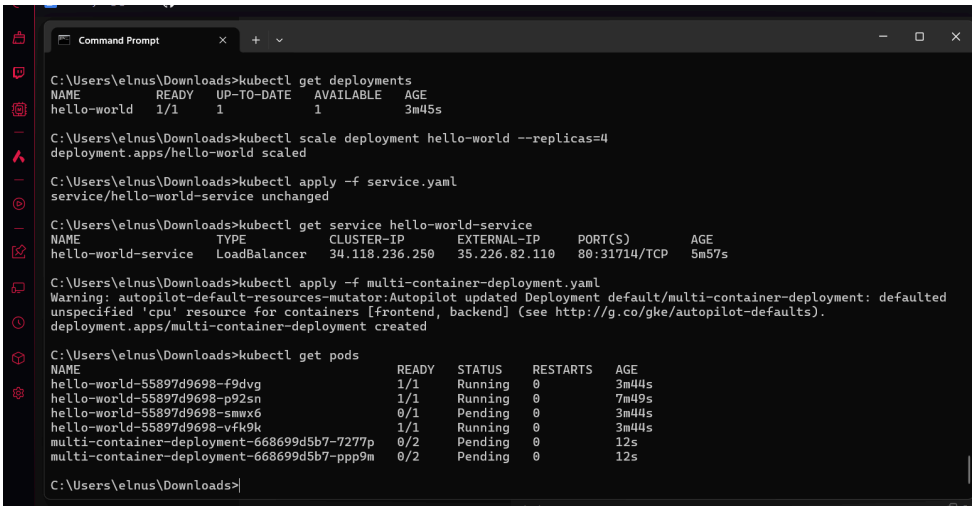
C: > Users > elnus > Downloads > multi-container-deployment.yaml

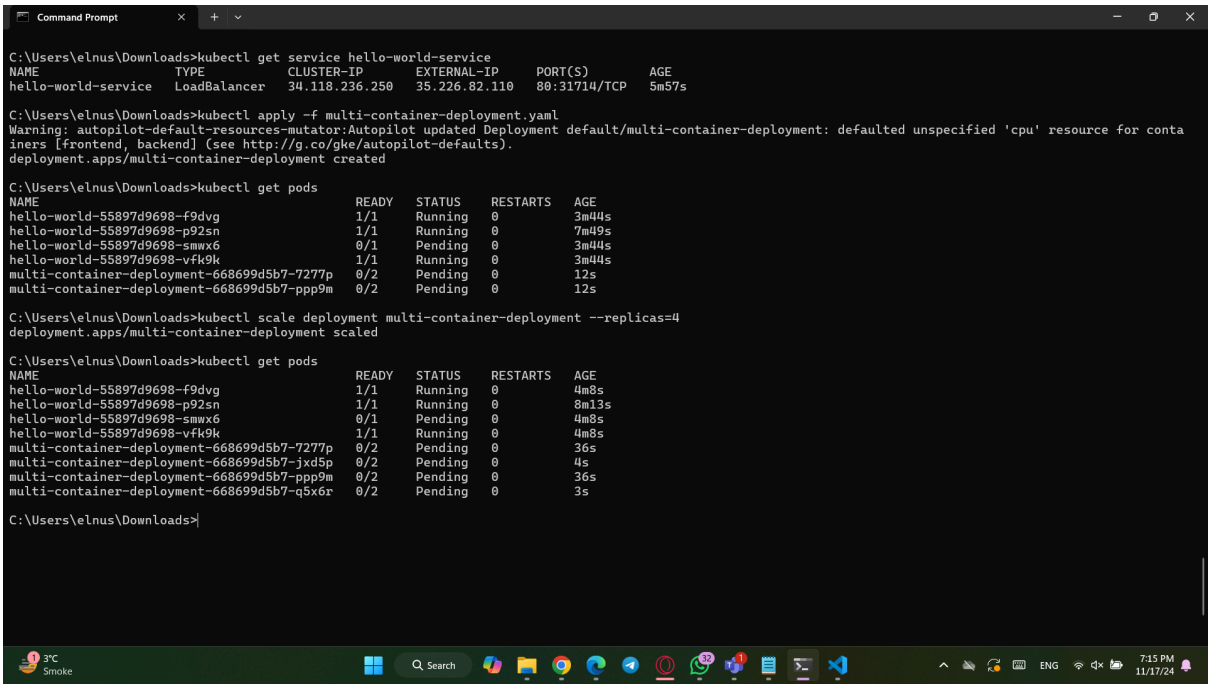
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: multi-container-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: multi-container
  template:
    metadata:
      labels:
        app: multi-container
    spec:
      containers:
      - name: frontend
        image: gcr.io/google-samples/hello-app:1.0
        ports:
        - containerPort: 8080
      - name: backend
        image: gcr.io/google-samples/hello-backend:1.0
        ports:
        - containerPort: 9090

```

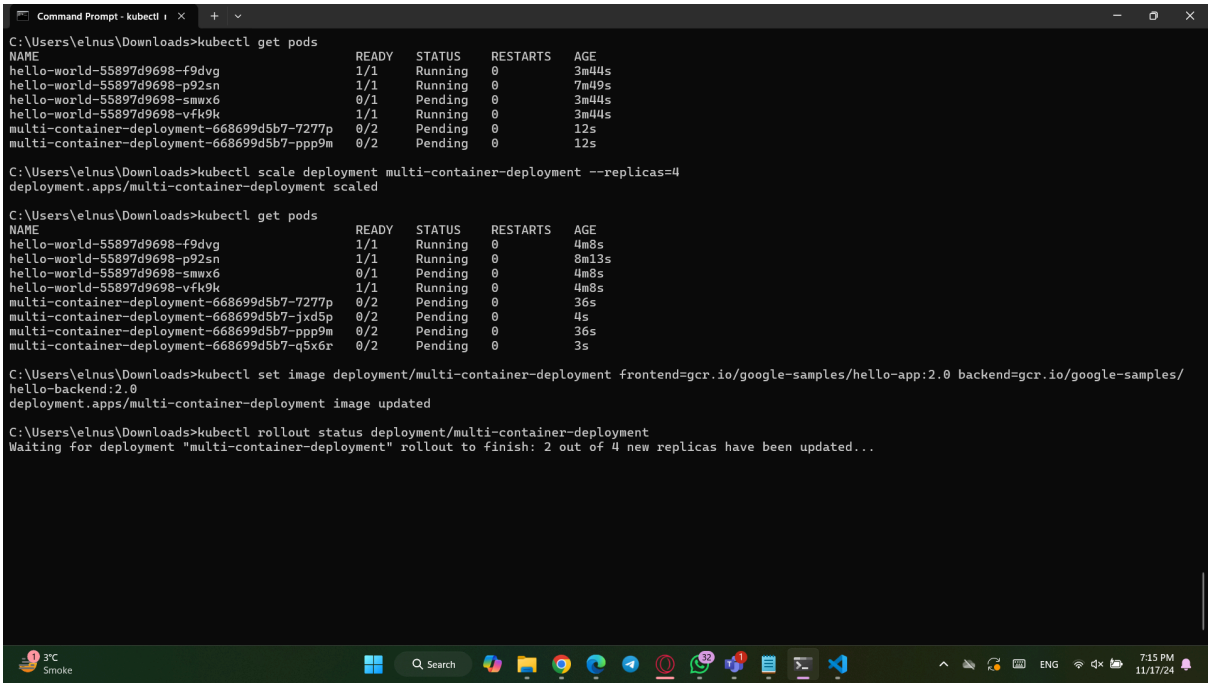
This defines the 2 containers here, so I apply it to deploy: **kubectl apply -f multi-container-deployment.yaml** and **kubectl get pods.**

Then we scale the app: **kubectl scale deployment multi-container-deployment --replicas=4**

kubectl get pods

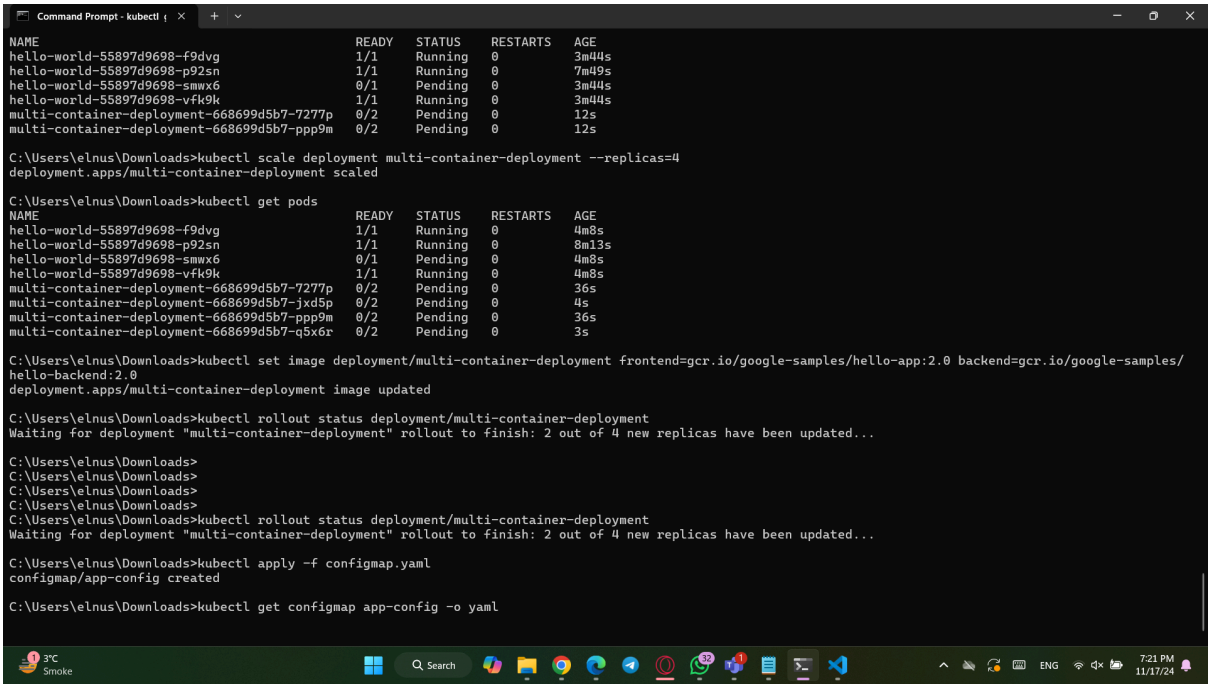




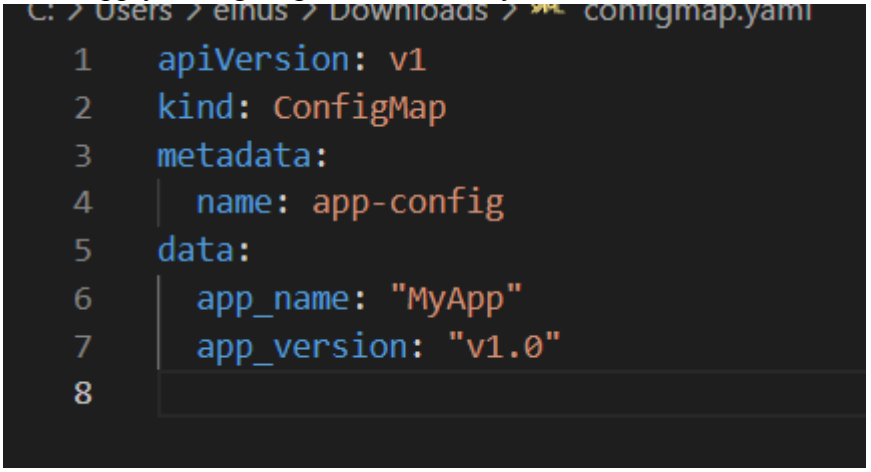


Exercise 6: ConfigMaps and Secrets

- Implement ConfigMaps and Secrets in your GKE application.
- Use a ConfigMap to pass configuration data and a Secret to manage sensitive information (e.g., API keys).

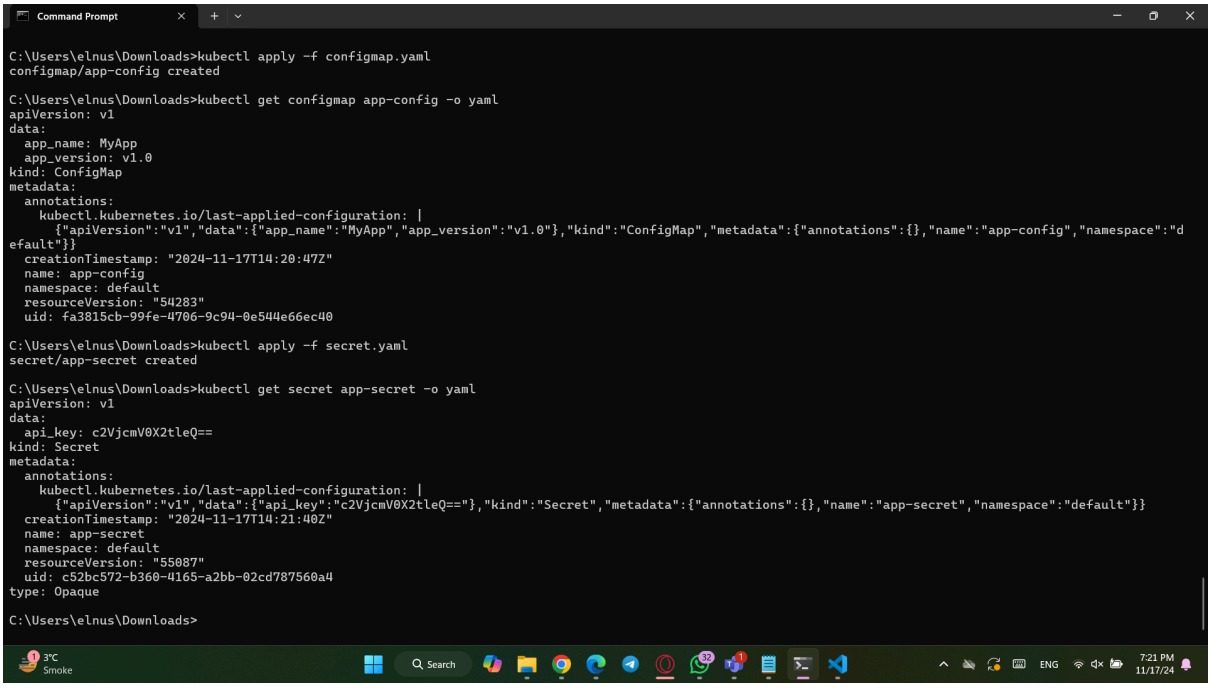


Here I apply config maps, first i create yaml file:



kubectl apply -f configmap.yaml

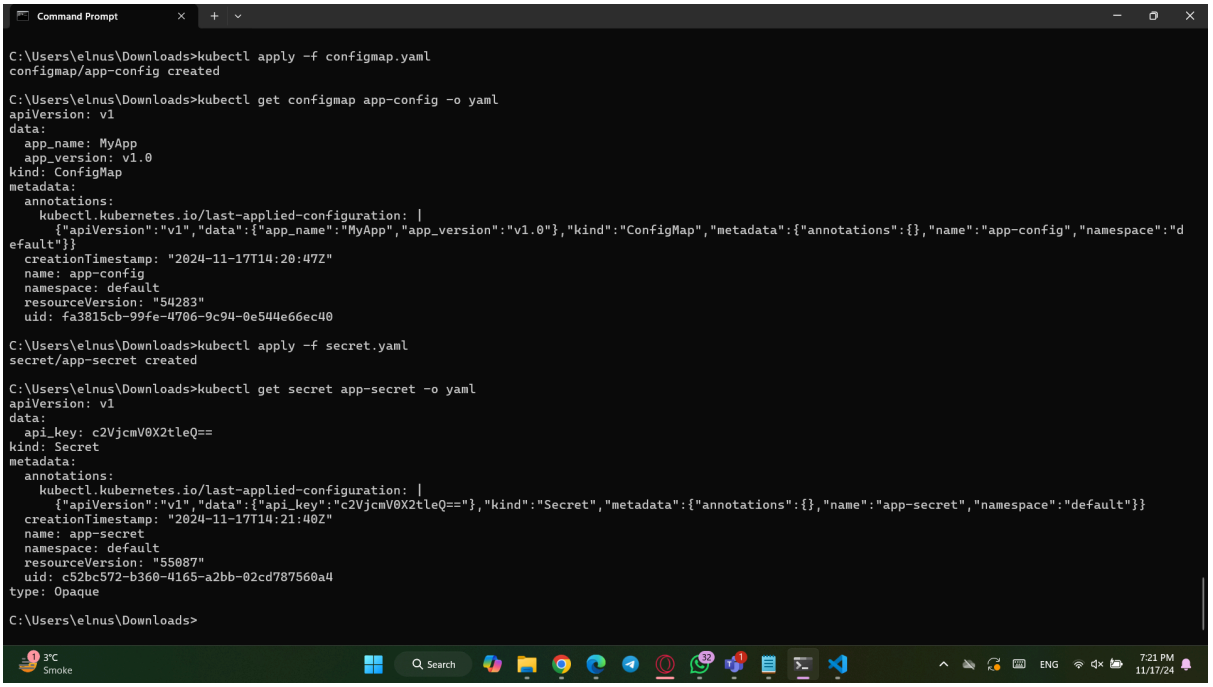
kubectl get configmap app-config -o yaml



Then i also create the secret yaml file:

apiVersion: v1 kind: Secret metadata: name: app-secret data: api_key: c2VjcmV0X2tleQ==

Then we apply it again to the k8s: **kubectl apply -f secret.yaml**


Command Prompt
C:\Users\elnus\Downloads>kubectl apply -f configmap.yaml
configmap/app-config created
C:\Users\elnus\Downloads>kubectl get configmap app-config -o yaml
apiVersion: v1
data:
app_name: MyApp
app_version: v1.0
kind: ConfigMap
metadata:
annotations:
kubectl.kubernetes.io/last-applied-configuration: |
{"apiVersion":"v1","data":{"app_name":"MyApp","app_version":"v1.0"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"app-config","namespace":"d
efault"}}
creationTimestamp: "2024-11-17T14:20:47Z"
name: app-config
namespace: default
resourceVersion: "54283"
uid: fa3815cb-99fe-4706-9c94-0e544e66ec40
C:\Users\elnus\Downloads>kubectl apply -f secret.yaml
secret/app-secret created
C:\Users\elnus\Downloads>kubectl get secret app-secret -o yaml
apiVersion: v1
data:
api_key: c2VjcmV0X2tleQ==
kind: Secret
metadata:
annotations:
kubectl.kubernetes.io/last-applied-configuration: |
{"apiVersion":"v1","data":{"api_key":"c2VjcmV0X2tleQ=="},"kind":"Secret","metadata":{"annotations":{},"name":"app-secret","namespace":"default"}}
creationTimestamp: "2024-11-17T14:21:40Z"
name: app-secret
namespace: default
resourceVersion: "55087"
uid: c52bc572-b360-4165-a2bb-02cd787560a4
type: Opaque
C:\Users\elnus\Downloads>
3°C
Smoke
Search
ENG
7:21 PM
11/17/24

3. App Engine and Cloud Functions

Exercise 7: Deploying an App on App Engine

- Create a simple web application (e.g., a Flask or Node.js app) and deploy it to Google App Engine.
- Configure app.yaml for the deployment, specifying runtime and service settings.

I create the simple web app here with as before flask) :

mkdir my-flask-app

cd my-flask-app

Requirements.txt with flask version, of course.

In the app.py I write dome basic stuff:

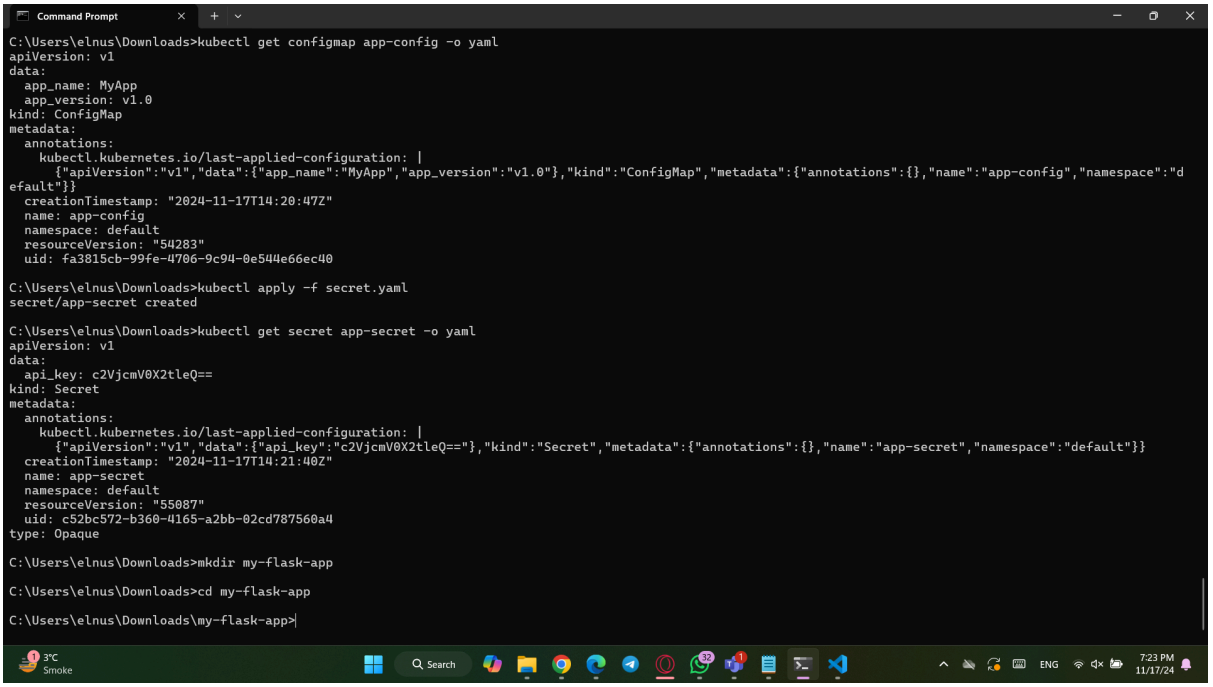
```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

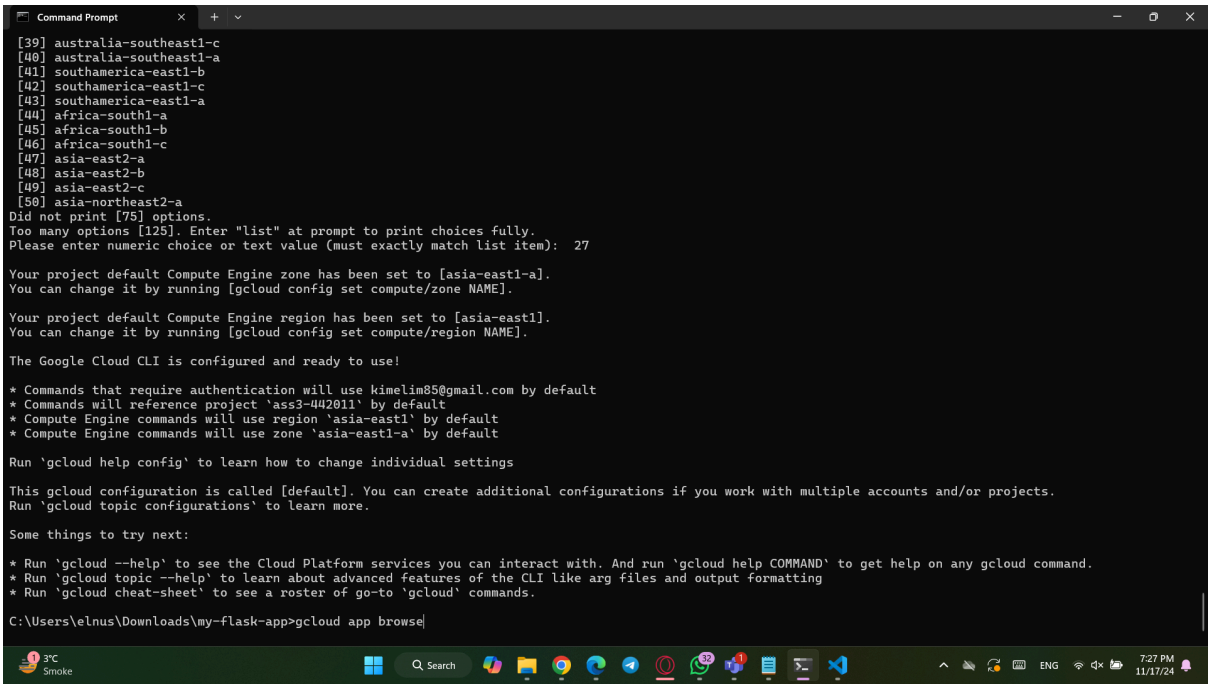
ThenI configure the app.yaml file with:

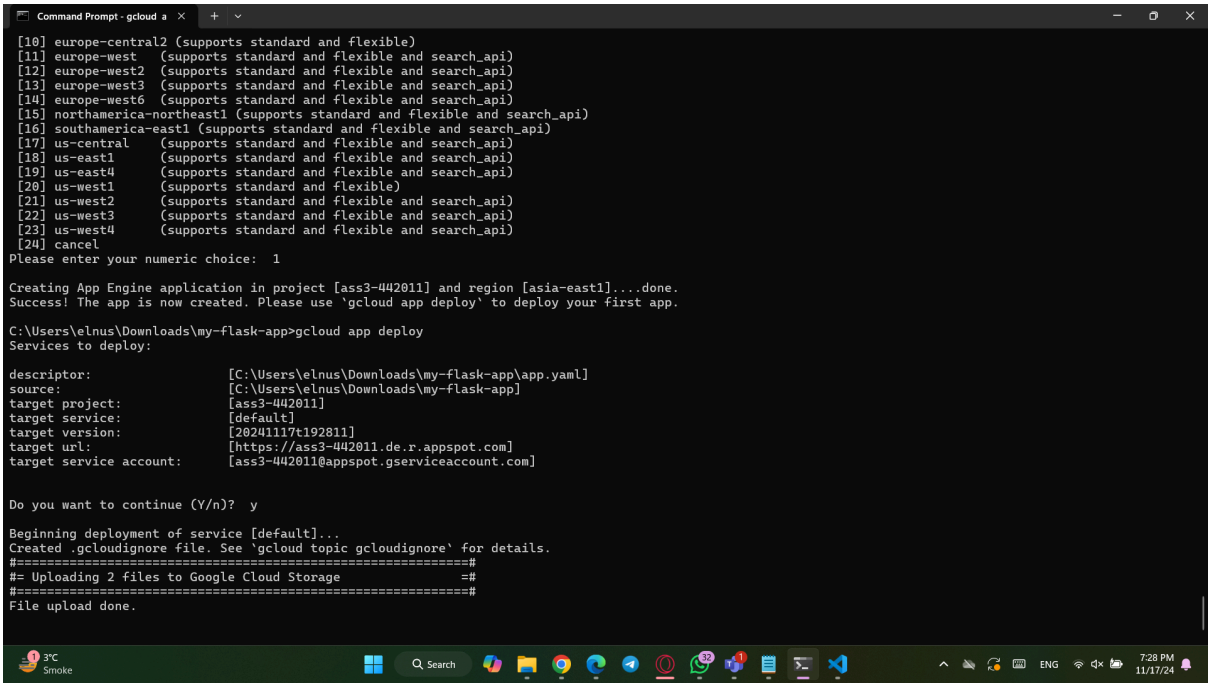
C: > Users > elnus > Downloads > my-flask-app > app.yaml

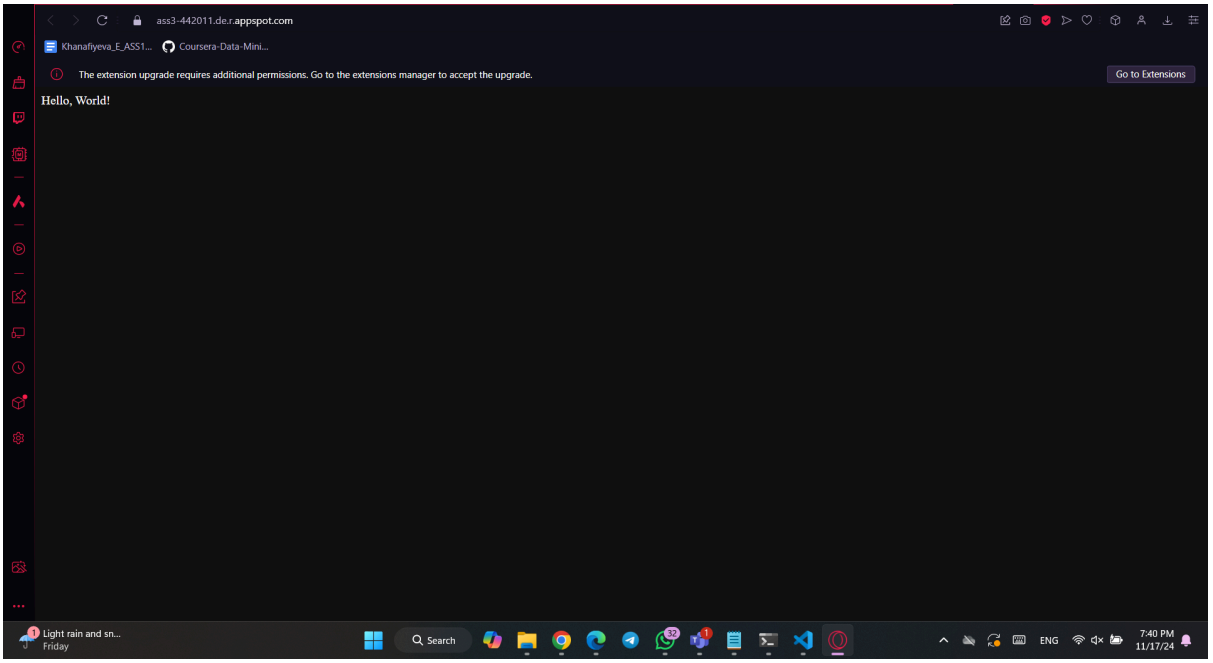
```yaml
runtime: python39
entrypoint: gunicorn -b :$PORT app:app

env_variables:
  FLASK_ENV: 'production'
```

Then let’s deploy it to the GAE, so open gcloud cli and run : **gcloud init**, then **gcloud app deploy , gcloud app browse**







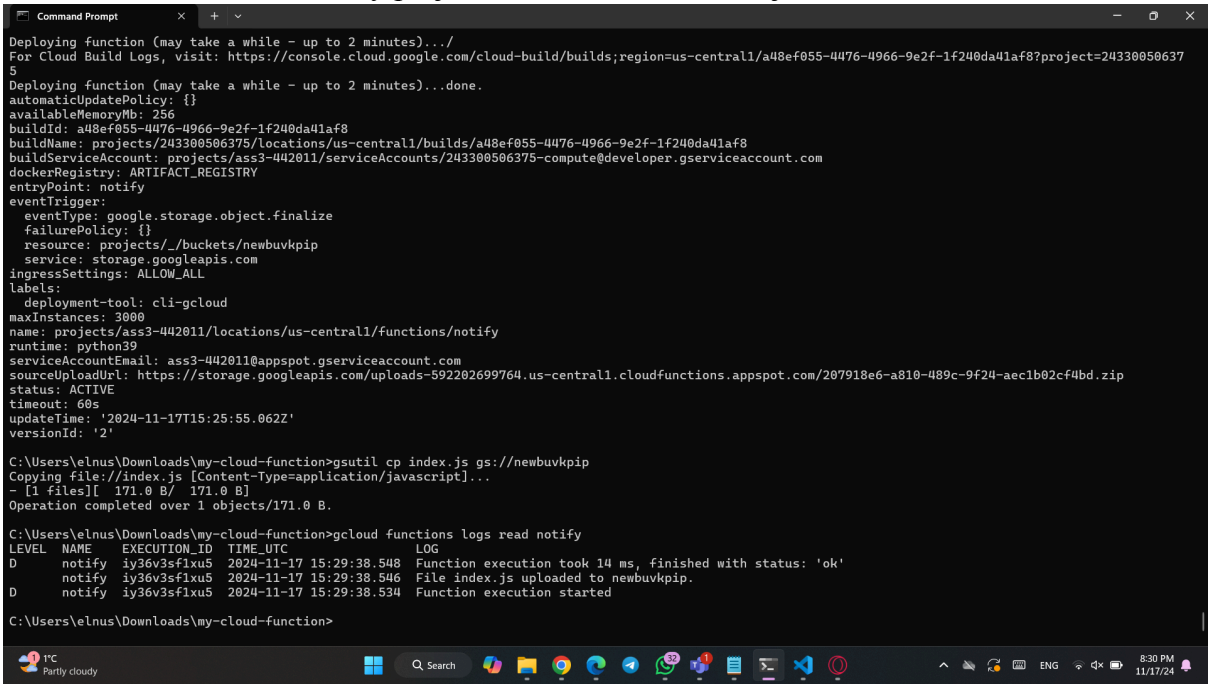


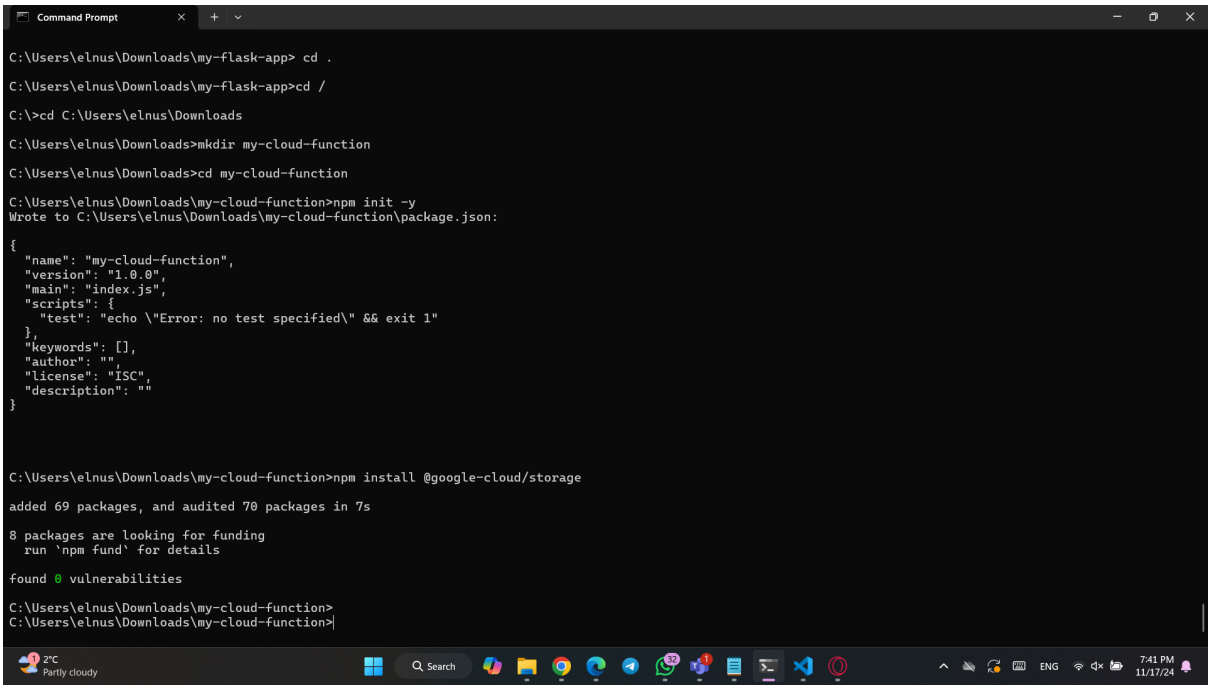
Then I can see it from the browser.

Exercise 8: Using Cloud Functions

- Write a Cloud Function that triggers on a specific event (e.g., an object creation in Cloud Storage) and performs a task (e.g., sending a notification).
- Deploy the function and test it by uploading a file to the designated Cloud Storage bucket.

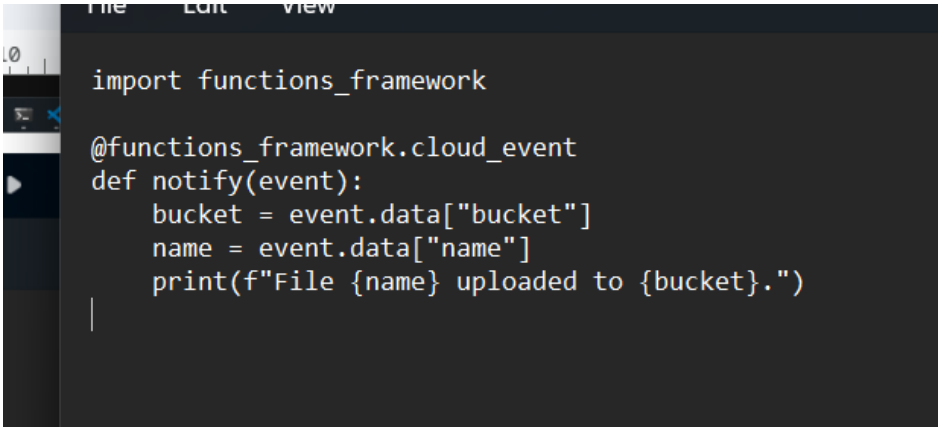
I install the firebase sdk to my project, thne i create the idex.js :





```
exports.notify = (event, context) => {
    const file = event.name;
    const bucket = event.bucket;
    console.log(`File ${file} uploaded to bucket ${bucket}`);
};
```

And main py :



This is made for notifying.

Then I deploy it to the cloud function:

gcloud functions deploy notify --runtime python39--trigger-resource buck --trigger-event google.storage.object.finalize

Then we upload the file here by gsutil cp file.txt gs://buck

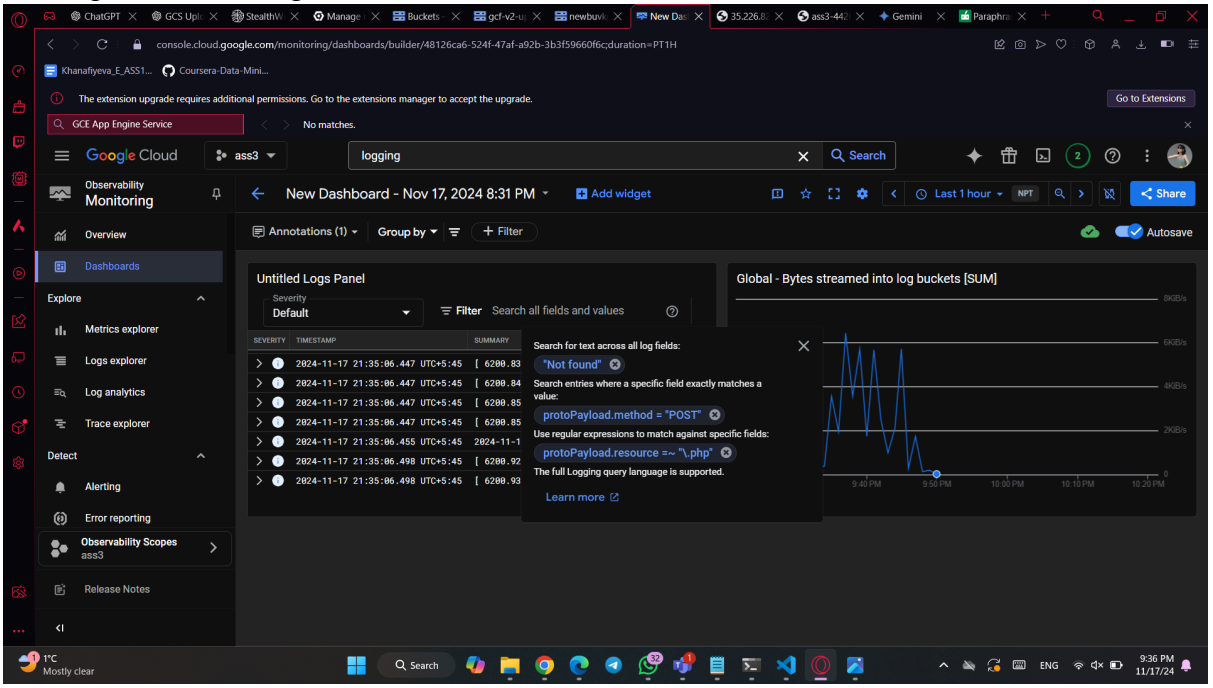
gcloud functions logs read notify

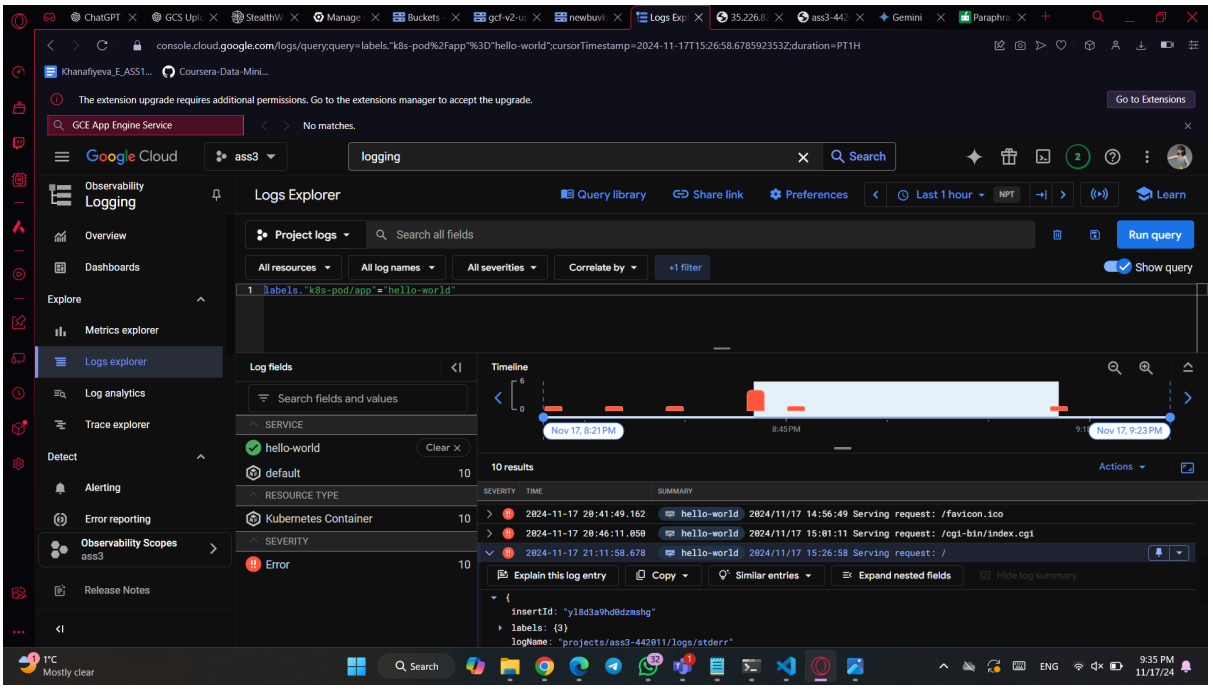
```
C:\Users\elnus\Downloads\my-cloud-function>gsutil cp index.js gs://newbuvkpip
Copying file://index.js [Content-Type=application/javascript]...
- [1 files][  171.0 B/  171.0 B]
Operation completed over 1 objects/171.0 B.

C:\Users\elnus\Downloads\my-cloud-function>gcloud functions logs read notify
LEVEL  NAME    EXECUTION_ID  TIME_UTC                LOG
D      notify  iy36v3sf1xu5  2024-11-17 15:29:38.548  Function execution took 14 ms, finished with status: 'ok'
       notify  iy36v3sf1xu5  2024-11-17 15:29:38.546  File index.js uploaded to newbuvkpip.
D      notify  iy36v3sf1xu5  2024-11-17 15:29:38.534  Function execution started

C:\Users\elnus\Downloads\my-cloud-function>
```

Exercise 9: Monitoring and Logging

- Set up monitoring and logging for your App Engine application and Cloud Functions.
- Use Google Cloud's monitoring tools to analyze the performance and logs of your deployed services.

To setup the monitoring i make custom dashboard here:





Results

Key Google Cloud services, such as App Engine, Cloud Functions, and monitoring tools, were made practical through the exercises.

App.yaml was used to establish runtime and service configurations for a basic Flask application that was uploaded to Google App Engine for Exercise 7. A misconfigured app.yaml was the cause of the initial deployment issues; this was fixed by fixing the syntax and defining the runtime and entry point. The application was successfully tested after deployment, showcasing how App Engine makes web application hosting easier.

Creating a Cloud Function that was triggered by a Cloud Storage event was the task for Exercise 8. The purpose of the function was to notify the user when a file upload was detected. A bucket location issue that was managed by making sure both resources were in the same place as the function's trigger region and a permissions issue that was overcome by giving the Cloud Storage service account the necessary roles/pubsub.publisher IAM role were among the challenges. These challenges demonstrated how crucial exact IAM role configuration and regional alignment are for event-driven systems.

Both App Engine and Cloud Functions monitoring and logging were set up in Exercise 9. Cloud Monitoring was used to track metrics like error rates and response times, and alerting rules were set up. Alert levels had to be changed during testing in order to represent actual application performance. These tools shown their worth in preserving the performance visibility and health of applications.

Conclusion

The adaptability and integration of Google Cloud services in contemporary cloud architecture were highlighted by this set of exercises. The need for appropriately allocated IAM roles was demonstrated by the importance of identity and security management in maintaining seamless operations and preventing deployment problems. Cloud Functions enabled serverless, event-driven operations with great flexibility, while App Engine worked well for developing straightforward, scalable applications with no overhead. Proactive troubleshooting was made possible by monitoring and logging technologies, which also offered insights into application performance. When combined, these services demonstrated Google Cloud's resilience and versatility for creating cloud-native apps that are scalable, safe, and effective.

References

Google Cloud Documentation https://cloud.google.com/docs/?gad_source=1&gclid=CjwKCAiAxea5BhBeEiwAh4t5K13qhD9zANEVnnnujlcqXosy7PHAVeQk4fJOhOJ3oSmUwkOQzVErHRoCtf0QAvD_BwE&gclsrc=aw.ds

Cloud Functions Overview https://cloud.google.com/functions/docs