Assignment 3

Cloud Computing

Zhanat Seitkuzhin

November 17, 2024

Table of Contents

1. Introduction

Assignment 3 focuses on three main areas: Identity and Security Management, Google Kubernetes Engine (GKE), and App Engine with Cloud Functions, which are essential tools for building, securing, and managing cloud-based applications.

In Identity and Security Management, the tasks include setting up IAM roles, service accounts, and organization policies to control and secure resource access. For GKE, the exercises involve deploying containerized applications, scaling deployments, and managing configurations with ConfigMaps and Secrets. Finally, the App Engine and Cloud Functions section covers deploying a web app, creating automated workflows, and setting up monitoring and logging for better visibility.

2. Exercise Description

Exercise 1: Setting Up IAM Roles

- **Objective:**
  - To configure Identity and Access Management (IAM) roles and assign permissions for different users in a Google Cloud project.
- **Implementation Steps:**
  - Create a new Google Cloud project.
  - Navigate to the IAM & Admin section in the GCC.
  - Add new members and assign roles (Viewer, Editor, and Owner).
- **Expected Outcome:**
  - A Google Cloud project with properly assigned IAM roles with ensured security and appropriate controls for all users.

Exercise 2: Service Accounts

- **Objective:**
  - To create a service account for accessing Google Cloud Storage and use it to authenticate a Python script for uploading files.
- **Implementation Steps:**
  - Create a new service account in the GCC.
  - Assign the Storage Admin role to the service account.
  - Generate and download the service account key in JSON format.
  - Use existing project on GitHub, replace the global environmental variable GOOGLE_CREDENTIALS_JSON on my generated key.
  - Import in the project.
  - Test the script to verify the upload process works as expected.
- **Expected Outcome:**
  - A service account and working script that securely uploads files to a Cloud Storage bucket.

Exercise 3: Organization Policies

- **Objective:**
  - To enforce organization-level restrictions to improve resource security and control.
- **Implementation Steps:**
  - Open Organization Policies section in the GCC.
  - Apply a restriction. For example, disable public IP address creation (at the project or organization level).

    - Verify the policy by attempting actions that should be restricted.
- **Expected Outcome:**
    - Organization policies successfully applied to ensure adherence to security and operational guidelines.

Exercise 4: Deploying a Simple Application

- **Objective:**
    - To set up a GKE cluster and deploy a simple containerized application accessible via a LoadBalancer service.
- **Implementation Steps:**
    - Create a GKE cluster using the Google Cloud Console or gcloud CLI.
    - Build and push a "Hello World" container image to Google Container Registry.
    - Deploy the application using Kubernetes.
    - Expose the "Hello World" via a LoadBalancer service so that it is accessible to the public.
- **Expected Outcome:**
    - A running "Hello World" application accessible through a public IP address.

Exercise 5: Managing Pods and Deployments

- **Objective:**
    - To manage a multi-container application and scale it using Kubernetes YAML configurations.
- **Implementation Steps:**
    - Write a Kubernetes YAML file to define a Deployment for a multi-container application.
    - Use kubectl to apply YAML and deploy.
    - Scale the Deployment by modifying the number of replicas in YAML and applying changes.
    - Update the application to use a new container image version and verify the deployment.
- **Expected Outcome:**
    - A scalable multi-container application with updated configurations running on GKE.

Exercise 6: ConfigMaps and Secrets

- **Objective:**
    - To implement ConfigMaps and Secrets in a Kubernetes application for managing configuration and sensitive data.
- **Implementation Steps:**
    - Create a ConfigMap to store non-sensitive app configuration.
    - Create a Secret to store sensitive data.
    - Modify the Deployment YAML to use the ConfigMap and Secret in the application pods.
    - Verify that the application reads data correctly from the ConfigMap and Secret.
- **Expected Outcome:**
    - A GKE application utilizing ConfigMaps and Secrets for dynamic configuration and sensitive data management.

Exercise 7: Deploying an App on App Engine

- **Objective:**
    - To deploy a web application to Google App Engine using an app.yaml configuration file.
- **Implementation Steps:**
    - Develop a simple web application.
    - Write an app.yaml file specifying the runtime and configuration details.
    - Deploy the application to App Engine using gcloud app deploy.
    - Test the deployment to confirm the app is running.
- **Expected Outcome:**
    - A live web application deployed and accessible via App Engine.

Exercise 8: Using Cloud Functions

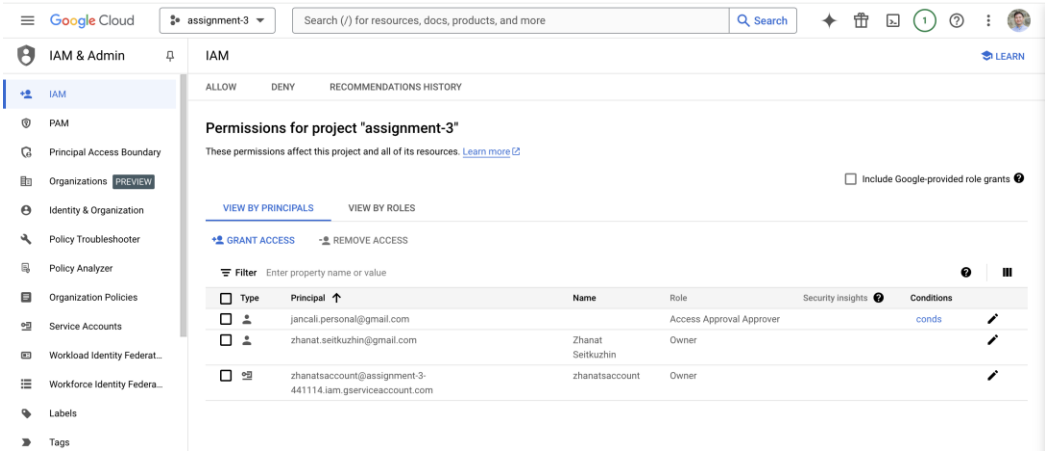
- **Objective:**
    - Create a Cloud Function triggered by an event in Cloud Storage.
- **Implementation Steps:**
    - Write a Cloud Function script to execute a task (e.g., send a notification upon file upload).
    - Deploy the function and set the trigger to a Cloud Storage bucket object creation event.
    - Test the function by uploading a file to the bucket.
    - Verify the function executes as expected.
- **Expected Outcome:**
- Functional Cloud Function triggered by file uploads to a designated Cloud Storage bucket.

Exercise 9: Monitoring and Logging

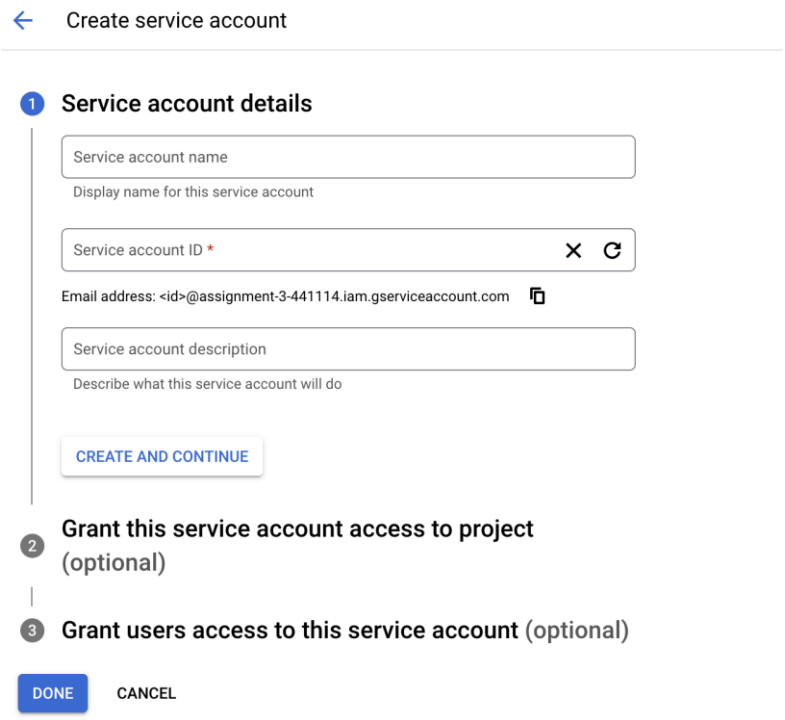
- **Objective:**
  - To set up monitoring and logging for applications on App Engine and Cloud Functions.
- **Implementation Steps:**
  - Enable Google Cloud Monitoring.
  - Use monitoring tools to track application performance and detect issues.
  - Access logs to analyze application activity and troubleshoot problems.
  - Create alerts and dashboards for metrics.
- **Expected Outcome:**
  - Comprehensive monitoring and logging for deployed applications that provides insights into performance and system health.

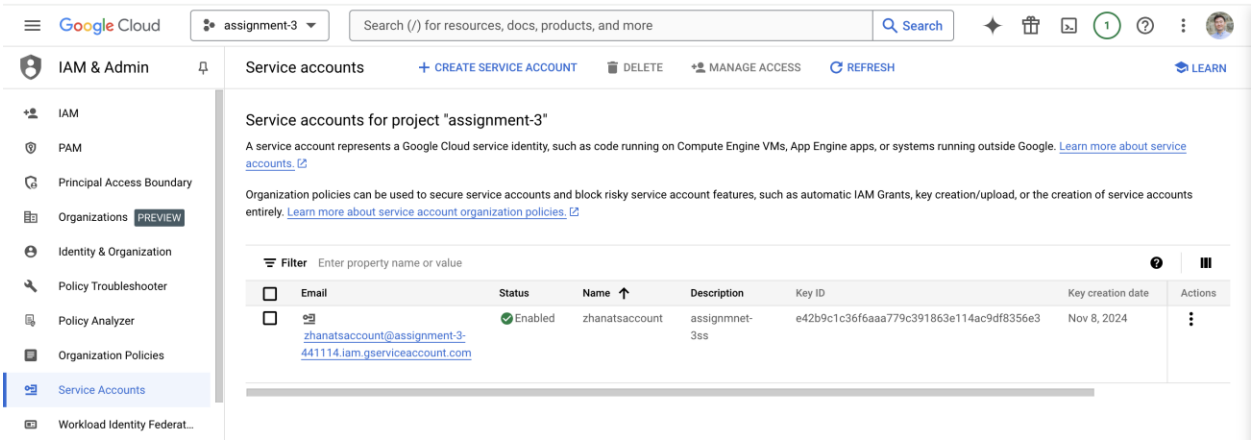
3. Code Snippets

1. **Exercise 1:** Enable IAM API, then Navigate to IAM & Admin in Google Console search, and Select roles & add conditions.

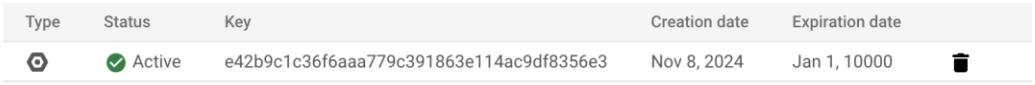


2. **Exercise 2:** In IAM & Admin Navigate to Service Account, then click on Create Service Account and fill in the following information:



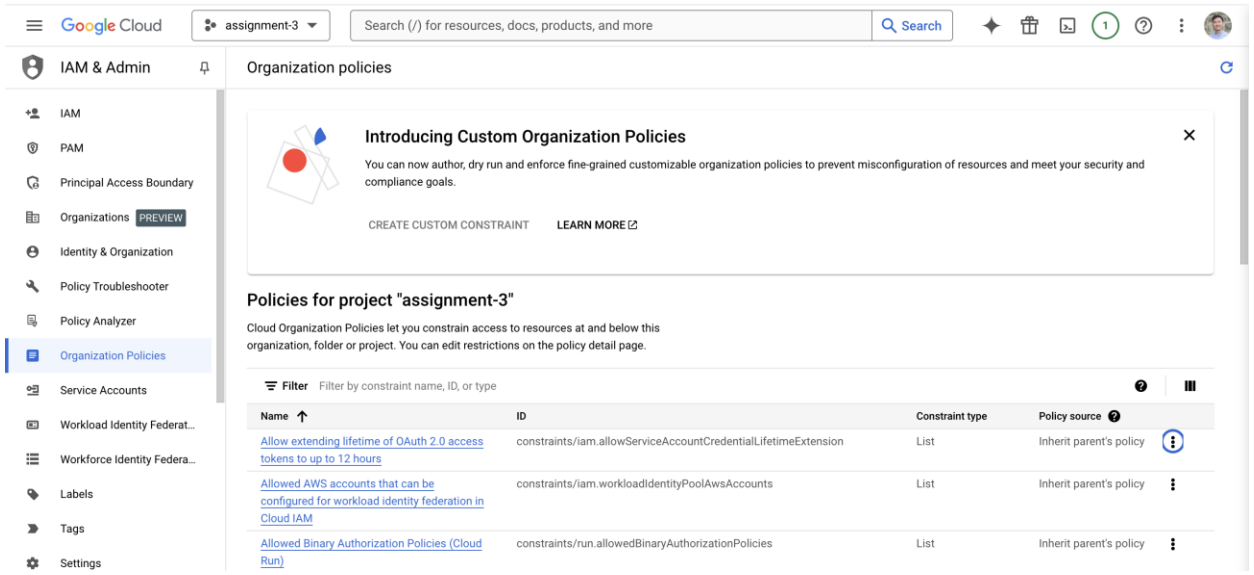


- After we finish setting up the service account, we generate and download a key for this service account and use this key to authenticate a Python script that uploads a file to a Cloud Storage bucket.



- In console to authorize gcloud to access Google Cloud using an existing service account while also specifying a project, run:
    - gcloud auth activate-service-account SERVICE_ACCOUNT@DOMAIN.COM --key-file=/path/key.json --project=PROJECT_ID

3. Exercise 3:

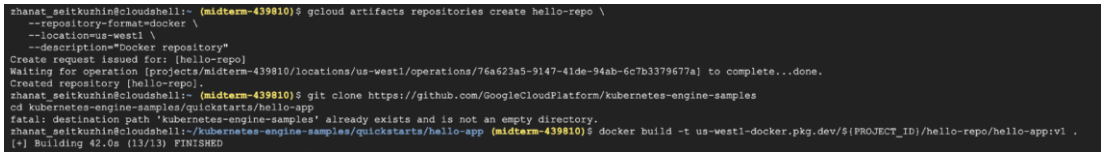


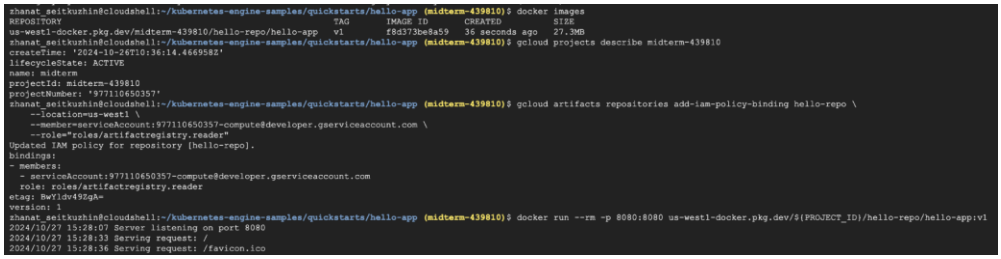
4. Exercise 4:

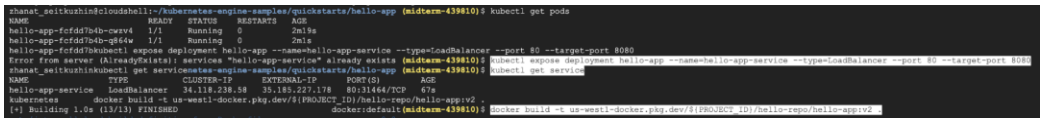
We can deploy our application using Docker and Google Kubernetes Engine (GKE). We start by packaging our web application into a Docker container image, which involves creating a Dockerfile that specifies the environment and dependencies required for our application. Then, we build the Docker image (*docker build -t gcr.io/$PROJECT_ID/hello-app:v1*) and push it to Google Container Registry to become accessible for deployment (*docker push gcr.io/$PROJECT_ID/hello-app:v1*).

First, we enable the Kubernetes Engine API (*gcloud services enable container.googleapis.com*) to allow us to create and manage our cluster. Next, we create a cluster tailored to our project specifications (*gcloud container clusters create hello-cluster --num-nodes=3*). Then, we configure kubectl to connect to it, enabling us to manage our Kubernetes resources directly from the terminal (*gcloud container clusters get-credentials hello-cluster*). We then set deployment configurations by selecting the number of replicas to run to ensure that our application is available and can handle user traffic (*kubectl create deployment hello-web --image=gcr.io/$PROJECT_ID/hello-app:v1*).

Next, Kubernetes automatically distributes our application replicas across the nodes in the cluster. To expose our application to users, we created a LoadBalancer service to direct incoming traffic to our application replicas (*kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-port 8080*). After running *kubectl get service*, terminal outputs the IP address that we use to check our application.

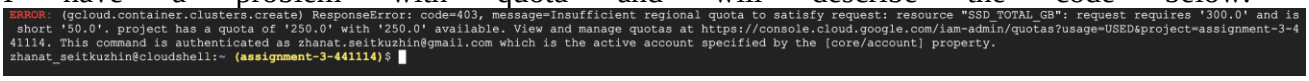






5. Exercise 5:

I have a problem with quota and will describe the code below.



- Open the Kubernetes Engine clusters page in the Google Cloud console. Then, on the Kubernetes Engine clusters page, click Deploy. Select your container and click Continue. You will see the Configuration section. Then, Under Configuration YAML, click View YAML to get a sample Kubernetes resource file.
- Connect to our cluster via code in console: gcloud container clusters get-credentials [CLUSTER_NAME] --zone [CLUSTER_ZONE].
- We create deployment with two containers.
- Use the Cloud Shell file upload option to upload multi-container-deployment.yaml to the Cloud Shell environment.
- Then we apply the deployment: kubectl apply -f multi-container-deployment.yaml
- Verify deployment: kubectl get deployments and kubectl get pods.
- We can also scale replicas here: kubectl scale deployment multi-container-app --replicas=5.

6. **Exercise 6:**

- In cloud shell open our YAML filee: nano multi-container-deployment.yaml.
- Create the ConfigMap using the following command: kubectl create configmap app-config --from-literal=db_url="http://example-db.com"
- Create a secret: kubectl create secret generic app-secrets --from-literal=api_key="12345-secret-api-key"
- Now we apply the deployment: kubectl apply -f multi-container-deployment.yaml.
- To inspect the deployment: kubectl get pods
- To Inspect the ConfigMap and Secret in the Containers: kubectl exec -it <pod-name> -c container-1 -- env
- There we would look for:
    - db_url environment variable, which should be loaded from the app-config ConfigMap
    - api_key environment variable, which should be loaded from the app-secrets Secret

7. Exercise 7:

- We install Flask: pip install flask
- Create main.py for the Flask app: nano main.py
- We also create YAML: nano app.yaml
- After both files are ready we deploy: gcloud app deploy
- To verify we: gcloud app browse.
- We should see that the app's URL will open in our browser.

8. **Exercise 8:**

- First, we need to enable the Cloud Functions API and Cloud Storage API
    - **g**cloud services enable cloudfunctions.googleapis.com storage.googleapis.com

- Create bucket: gsutil mb gs://YOUR_BUCKET_NAME/
- Create index.js file for the Cloud Function code: nano index.js and add code.
- To define the dependencies for your Cloud Function, create a package.json file: nano package.json and add content.
- Then we can deploy cloud function:
    - gcloud functions deploy fileUploaded \ --runtime nodejs16 \ --trigger-resource gs://YOUR_BUCKET_NAME \ --trigger-event google.storage.object.finalize
- Upload the fiile to bucket: gsutil cp your-file.txt gs://YOUR_BUCKET_NAME/
- Check functional log: gcloud functions logs read fileUploaded

9. Exercise 9:

- Enable APIs for Monitoring and Logging:
    - gcloud services enable monitoring.googleapis.com logging.googleapis.com
- Create alert policy:
    - gcloud alpha monitoring policies create \ --notification-channels=YOUR_NOTIFICATION_CHANNEL_ID \ --conditions='metric.type="compute.googleapis.com/instance/disk/write_bytes_count" comparison=COMPARISON_GREATERTHAN threshold=VALUE'
- View log for App Engine:
    - gcloud logging read "resource.type=\"gae_app\""
- View log for Cloud function:
    - gcloud logging read "resource.type=\"cloud_function\""

4. Conclusion

In conclusion, this project has been a rewarding experience that provided the opportunity to translate theoretical knowledge learned in the classroom into practical application. This assignment provided a great opportunity to dive into Google Cloud services and get hands-on experience with key tools like Identity and Security Management, Google Kubernetes Engine (GKE), App Engine, Cloud Functions, and Cloud Monitoring and Logging. Overall, this assignment helped us gain practical experience with cloud technologies and reinforced the importance of being able to deploy, manage, and monitor cloud applications. These skills are essential for building scalable, secure, and reliable applications in today's cloud-first world.

5. References

- Use VPC firewall rules
- Running a basic Apache web server
- Create and manage VPC networks
- Gloud Docker

6. Appendices