Kazakh-British Technical University

Assignment 3, Cloud Computing

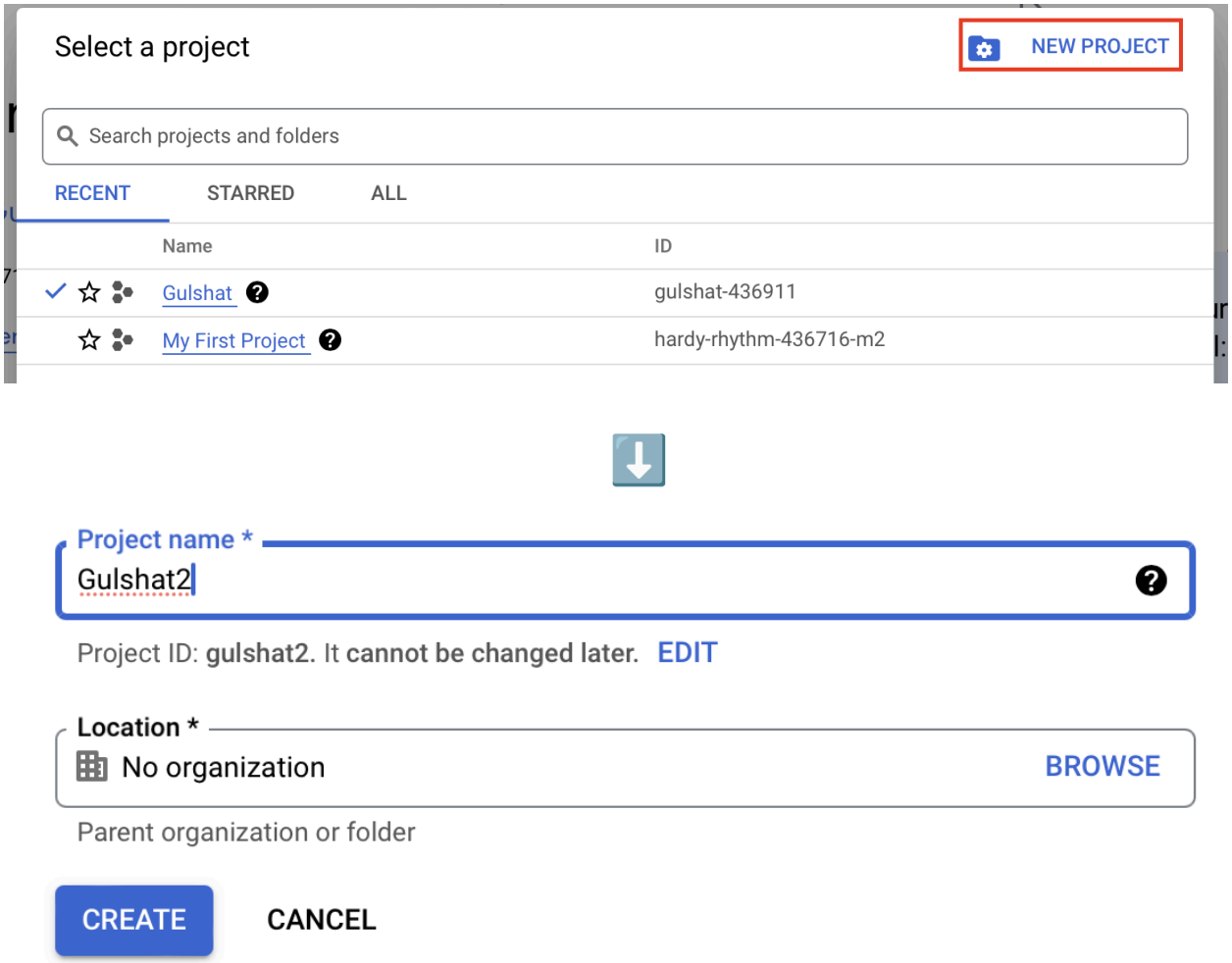
Student: Khamidulla Gulshat
Date: 17.11.2024
Course: MSITM 5103

Table of contents

1. Identity and Security Management

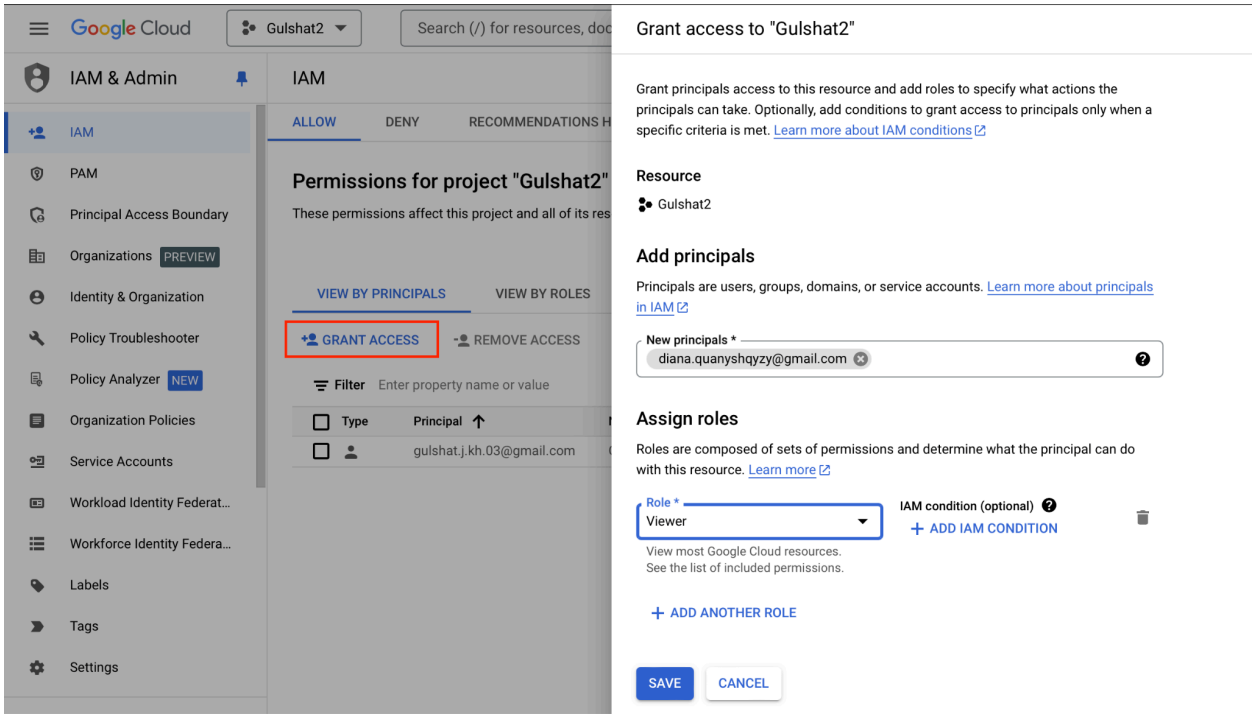
Exercise 1: Setting Up IAM Roles

I began by logging into the **Google Cloud Console** and chose **New Project** and gave it a name, "*Gulshat2*". After clicking Create, the new project was successfully set up.

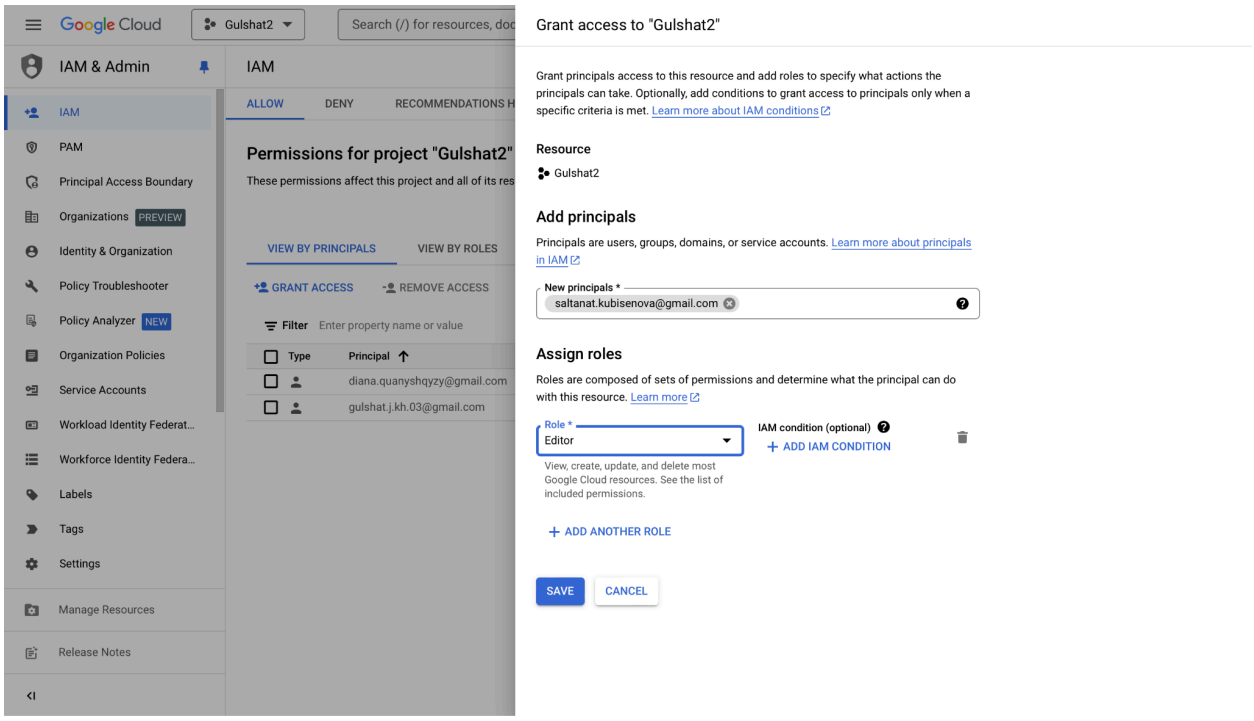


Then inside the new project, I navigated to **IAM & Admin > IAM** to assign roles. At the top of the IAM page, I clicked **Grant Access** to add principals for each member's email address and assign specific roles. I assigned the following roles: **Viewer** was assigned to diana.quanyshkyzy@gmail.com only to view resources without making any changes, **Editor** was assigned to saltanat.kubissenova@gmail.com to modify resources within the project.

1. Viewer



2. Editor

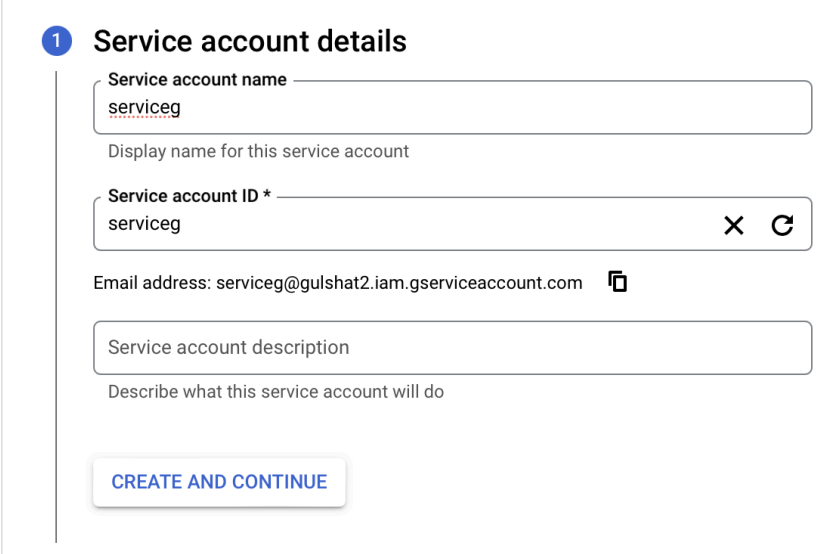


After assigning the roles, I reviewed the IAM settings in the Google Cloud Console to confirm that each user had the correct permissions. Each user's role appeared under IAM, matching the access level I assigned to them:

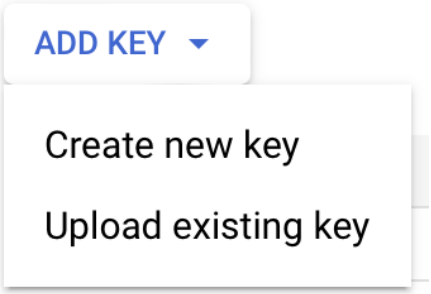
Type	Principal	Name	Role	Security insights	
	diana.quanyshqyzy@gmail.com		Viewer		
	gulshat.j.kh.03@gmail.com	Gulshat Khamidulla	Owner		
	saltanat.kubisenova@gmail.com		Editor		

Exercise 2: Service Accounts

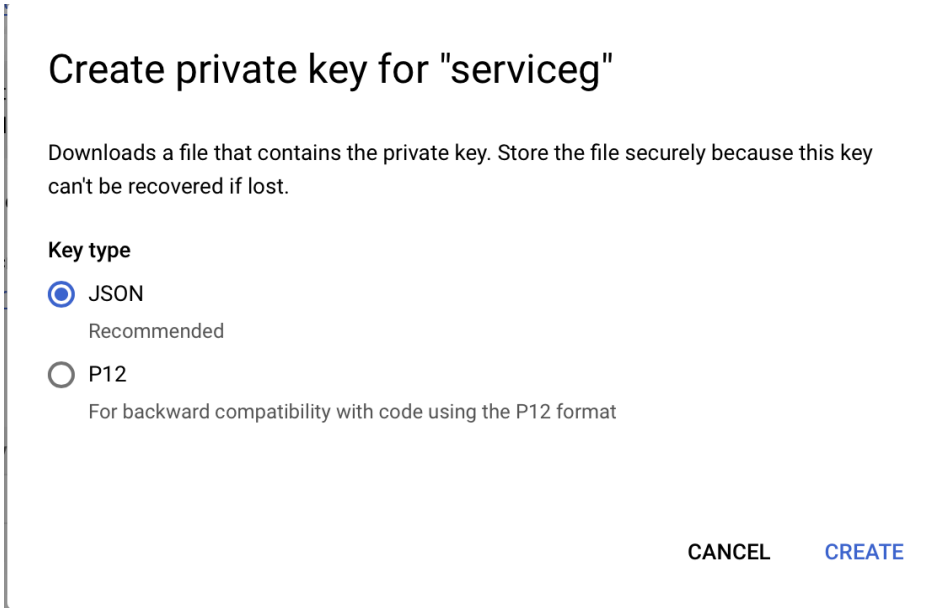
I went to **IAM & Admin > Service** Accounts in the Google Cloud Console to set up a service account specifically for accessing Google Cloud Storage. I clicked Create Service Account and gave it the name “serviceg”.

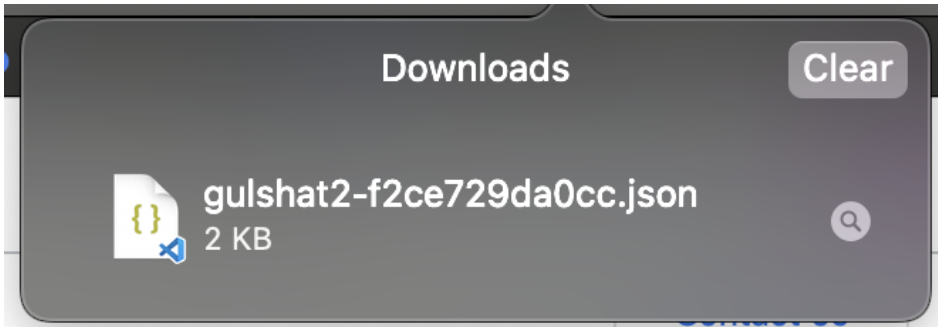


With the service account created, I needed a key to authenticate it. In the service account details page, I navigated to the **Keys** section and clicked **Add Key > Create new key**.

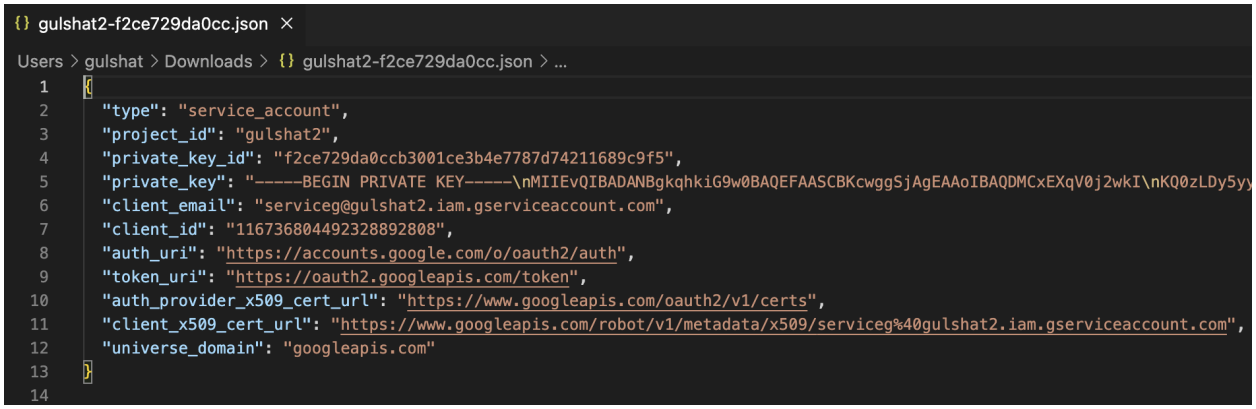


I selected **JSON** as the key type, which generated a key file and downloaded it to my laptop.





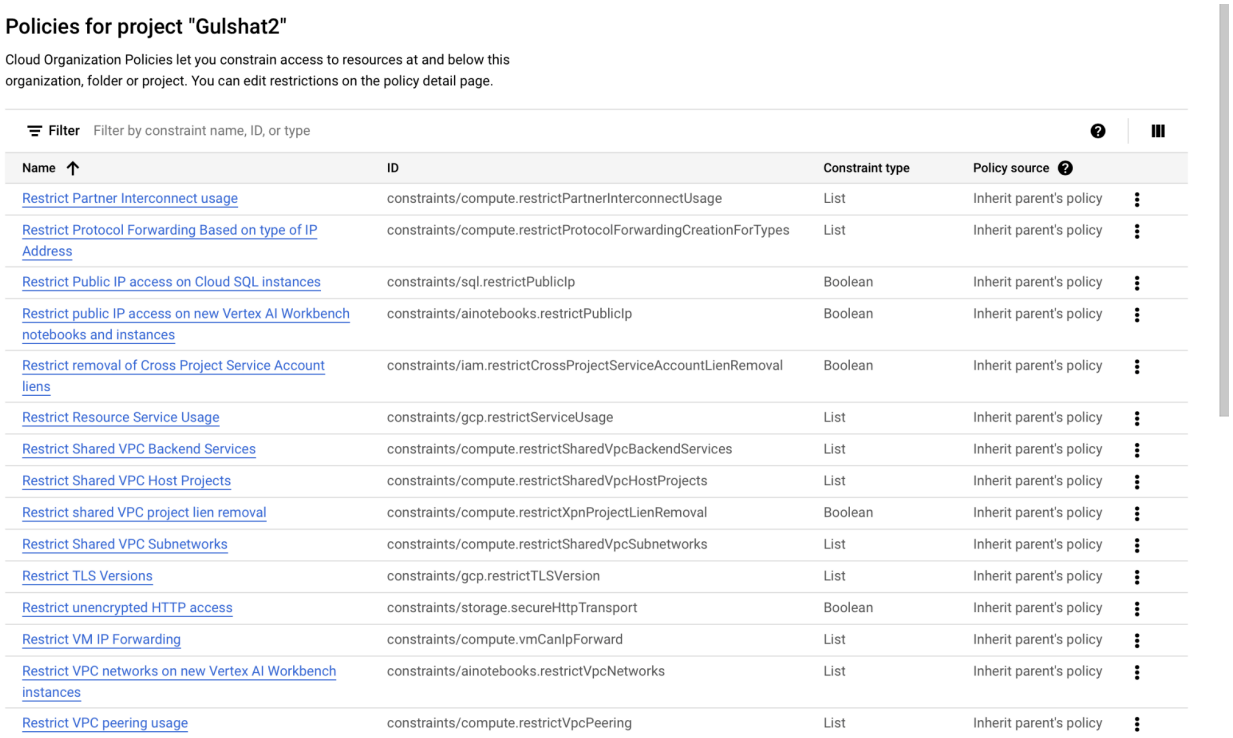
The downloaded key has the following format, where *PRIVATE_KEY* is the private portion of the public/private key pair:



Although I won't be completing the Python part, here's how I would use the downloaded key file to authenticate a Python script: in a Python script, I would use the downloaded JSON key to authenticate and upload a file to a Cloud Storage bucket by specifying path of the .JSON file and bucket details.

Exercise 3: Organization Policies

First, I navigated to **IAM & Admin > Organization Policies** in the Google Cloud Console. This section lets view and apply security and governance policies across projects or the organization. In my case, it shows policies of my project. I looked through the list of available policies and selected a policy to restrict public IP addresses on Cloud SQL instances.



Policies for project "Gulshat2"

Cloud Organization Policies let you constrain access to resources at and below this organization, folder or project. You can edit restrictions on the policy detail page.

Filter | Filter by constraint name, ID, or type

Name	ID	Constraint type	Policy source
Restrict Partner Interconnect usage	constraints/compute.restrictPartnerInterconnectUsage	List	Inherit parent's policy
Restrict Protocol Forwarding Based on type of IP Address	constraints/compute.restrictProtocolForwardingCreationForTypes	List	Inherit parent's policy
Restrict Public IP access on Cloud SQL instances	constraints/sql.restrictPublicIp	Boolean	Inherit parent's policy
Restrict public IP access on new Vertex AI Workbench notebooks and instances	constraints/ainotebooks.restrictPublicIp	Boolean	Inherit parent's policy
Restrict removal of Cross Project Service Account liens	constraints/iam.restrictCrossProjectServiceAccountLienRemoval	Boolean	Inherit parent's policy
Restrict Resource Service Usage	constraints/gcp.restrictServiceUsage	List	Inherit parent's policy
Restrict Shared VPC Backend Services	constraints/compute.restrictSharedVpcBackendServices	List	Inherit parent's policy
Restrict Shared VPC Host Projects	constraints/compute.restrictSharedVpcHostProjects	List	Inherit parent's policy
Restrict shared VPC project lien removal	constraints/compute.restrictXpnProjectLienRemoval	Boolean	Inherit parent's policy
Restrict Shared VPC Subnetworks	constraints/compute.restrictSharedVpcSubnetworks	List	Inherit parent's policy
Restrict TLS Versions	constraints/gcp.restrictTLSVersion	List	Inherit parent's policy
Restrict unencrypted HTTP access	constraints/storage.secureHttpTransport	Boolean	Inherit parent's policy
Restrict VM IP Forwarding	constraints/compute.vmCanIpForward	List	Inherit parent's policy
Restrict VPC networks on new Vertex AI Workbench instances	constraints/ainotebooks.restrictVpcNetworks	List	Inherit parent's policy
Restrict VPC peering usage	constraints/compute.restrictVpcPeering	List	Inherit parent's policy

After selecting the **Restrict Public IP Addresses on SQL instances** policy, I had to click **Edit** to modify its settings. In the editing options, choose **Enforced**, after setting this restriction, I had to click **Save** to apply the policy.
Once I saved the policy, I would return to the **Organization Policies** page to confirm that the restriction was active.

2. Google Kubernetes Engine (GKE)

Exercise 4: Deploying a Simple Application

A **GKE** cluster includes a cluster that runs Kubernetes processes. I created an Autopilot cluster called **gulshat-cluster** using the following command that sets up a fully managed cluster in the us-centrall region. The process took several minutes to complete.

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ gcloud container clusters create-auto gulshat-cluster \
    --location=us-central1
```

After creating the cluster, I retrieved the authentication credentials to interact with it. This step is essential for configuring *kubectl* to recognize and access the last created cluster. I ran:

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ gcloud container clusters get-credentials gulshat-cluster \
    --location us-central1
```

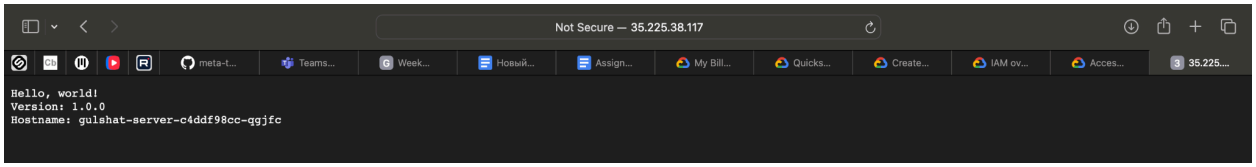
I proceeded to deploy a containerized application, specifically a sample web application called **hello-app** after the cluster was ready and *kubectl* configured. To deploy **hello-app**, I created a Deployment named **gulshat-server** and specified the container image *hello-app:1.0* to run in the cluster.

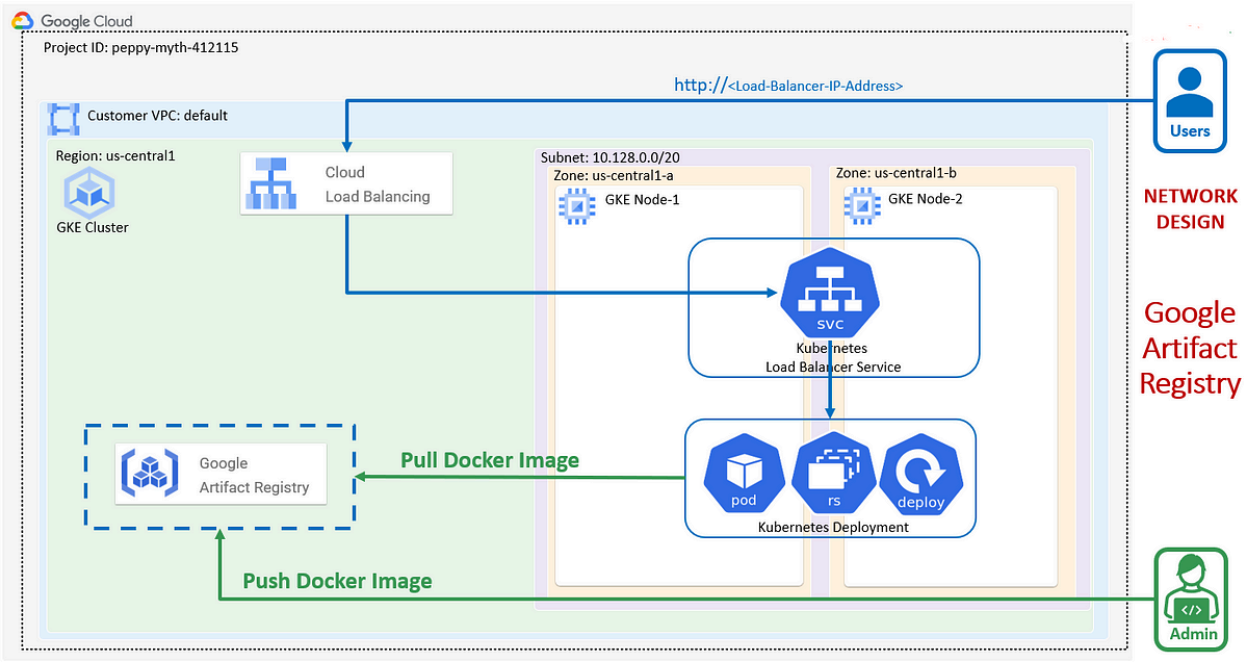
```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl create deployment gulshat-server \
    --image=us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0
```

To make **gulshat-server** accessible, I used the *kubectl expose* command to create a LoadBalancer service, which automatically provisions a load balancer in Google Cloud:

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl expose deployment gulshat-server \
    --type LoadBalancer \
    --port 80 \
    --target-port 8080
service/gulshat-server exposed
```

We can view the application from web browser by using the external IP address:

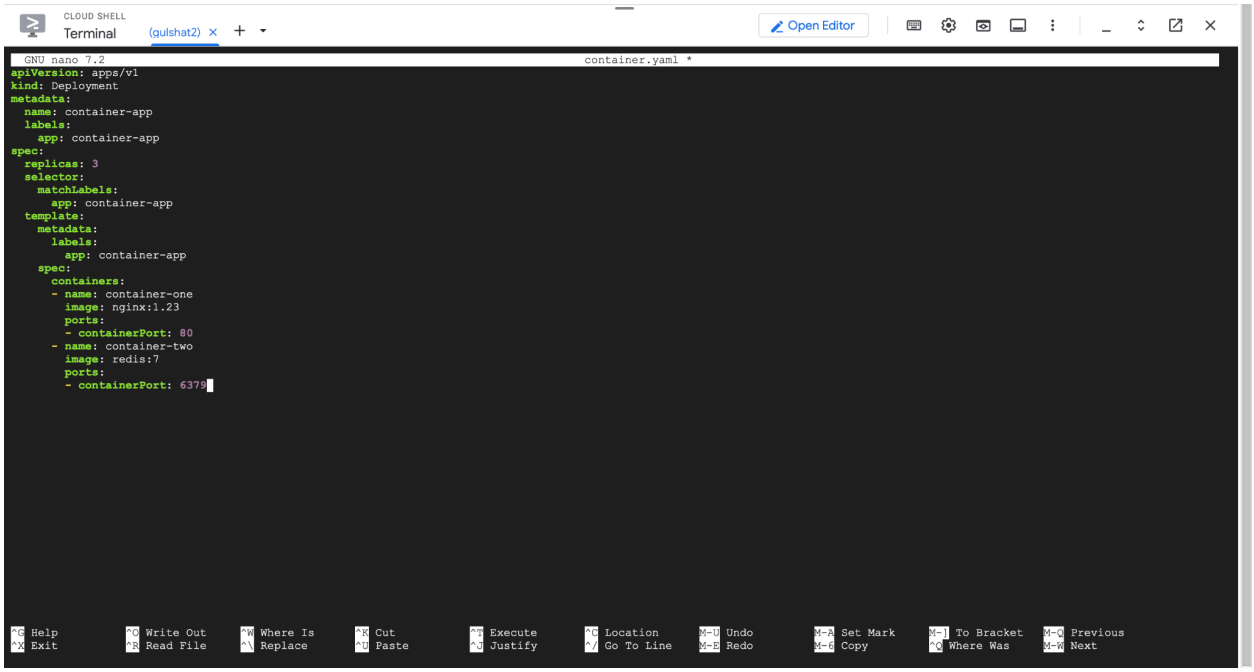




Picture above illustrates **Implementing GKE with Artifact Registry**.

Exercise 5: Managing Pods and Deployments

I created the YAML file for the multi-container application in Google Cloud Shell using the nano editor.



Then I saved by pressing Ctrl+O, Enter after that I applied an app to the cluster using `kubectl apply -f container.yaml`. Next, I scaled the deployment to increase the number of replicas. I used:

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl scale deployment container-app --replicas=5
deployment.apps/container-app scaled
```

Before:

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl get deployments
kubectl get pods
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
container-app   3/3     3            3           132m
NAME                                READY   STATUS    RESTARTS   AGE
container-app-b955fd49c-gcqtf   2/2     Running   0          132m
container-app-b955fd49c-m9hvx   2/2     Running   0          132m
container-app-b955fd49c-xrr42   2/2     Running   0          132m
```

After:

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl get deployments
kubectl get pods
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
container-app     3/5     5            3           134m
NAME                                  READY   STATUS    RESTARTS   AGE
container-app-b955fd49c-cmdt8         0/2     Pending   0          37s
container-app-b955fd49c-gcqtf         2/2     Running   0          134m
container-app-b955fd49c-m9hvx         2/2     Running   0          134m
container-app-b955fd49c-vkhtx         0/2     Pending   0          37s
container-app-b955fd49c-xrr42         2/2     Running   0          134m
```

Finally, I updated the application to use a newer version of the *nginx* container image. I ran and the result showed that the image was updated.

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl set image deployment/container-app container-one=nginx:1.24
deployment.apps/container-app image updated
```

Description of deployment showed that I have 7 total replicas.

```
gulshat_j_kh_03@cloudshell:~ (gulshat2)$ kubectl describe deployment container-app
Name:                   container-app
Namespace:              default
CreationTimestamp:      Sun, 17 Nov 2024 09:30:14 +0000
Labels:                 app=container-app
Annotations:            autopilot.gke.io/resource-adjustment:
                          {"input":{"containers":[{"name":"container-one"},{"name":"container-two
                        autopilot.gke.io/warden-version: 3.0.41
                        deployment.kubernetes.io/revision: 2
Selector:               app=container-app
Replicas:               5 desired | 3 updated | 7 total | 3 available | 4 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
```

This exercise provides essential Kubernetes skills for managing, scaling, and updating cloud-based applications.

Exercise 6: ConfigMaps and Secrets

A **ConfigMap** is a Kubernetes object used to store non-sensitive configuration data like application settings, environment variables, configuration files as key-value pairs. And we can create it with command `kubectl create configmap`.

A **Secret** is similar to a ConfigMap, but it is specifically designed for managing sensitive data like passwords, API keys, tokens, certificates, or any other information that should not be exposed in plain text. And we can create it with command `kubectl create secret`.

3. App Engine and Cloud Functions

Exercise 7: Deploying an App on App Engine

This exercise helps us understand how to deploy a web application to Google App Engine. First, we have to create a basic web app using Flask. Flask is a Python framework that makes it easy to build web applications. We start by creating a new folder for our project and go inside it. Then, we write a simple Python script. This file has to contain the code to create a web page. To deploy the app, we need to tell App Engine what dependencies our app uses and how it should run.

1. **`requirements.txt` File**:
   We list the libraries our app uses (like Flask) in this file so that App Engine knows to install them during deployment.
2. **`app.yaml` File**:
   This file tells App Engine which programming language and version the app uses and any special instructions for handling requests.

In Google Cloud Shell deploy an app with command `gcloud app deploy`. This uploads the app to Google App Engine and prepares it for hosting. After deployment, App Engine provides a link to the deployed app.

Exercise 8: Using Cloud Functions

First, we have to write the function using a supported programming language (e.g., Python, Node.js) and specify what the function should do when triggered. Then connect the function to a Cloud Storage bucket. This means the function will automatically run when the event occurs (e.g., a file upload). To deploy use the following command in Google Cloud Shell: `gcloud functions deploy`. Last step is testing by performing the triggering action and verifying that the function executes as expected.

Exercise 9: Monitoring and Logging

Google Cloud Monitoring provides insights into application performance. We can view metrics like response times, error rates, and traffic volume. Cloud Logging collects logs from our services. So we can view all the events and errors that occur during application runtime. Together, they ensure that applications are reliable and efficient.

Conclusion

Through these exercises, I learned how to use the core features of Google Cloud Platform (GCP) for managing cloud resources, deploying applications, and monitoring performance. These tasks showed me how GCP supports modern cloud computing by focusing on things like security, scalability, automation, and monitoring. By working on setting up permissions, deploying apps,

and creating automated functions, now I have a clearer understanding of how to build and manage systems in the cloud more effectively.

References

https://cloud.google.com/kubernetes-engine/docs/how-to/node-pools
https://cloud.google.com/iam/docs/keys-create-delete#iam-service-account-keys-create-console
https://cloud.google.com/iam/docs/overview
https://cloud.google.com/resource-manager/docs/access-control-org
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
https://www.youtube.com/watch?v=PCN4j0pRaiE