Connect an App to a Cloud SQL for PostgreSQL Instance

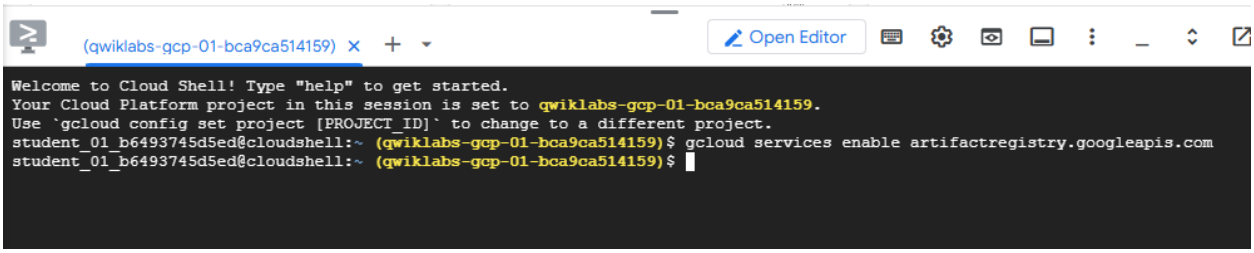
Task 1. Initialize APIs and create a Cloud IAM service account

To complete this task you must initialize the APIs and create an IAM service account that will be used to allow your application to connect to the Cloud SQL database.

Enable the APIs

You must enable the required APIs for this lab. You will build and push a container to the Artifact Registry in a later task, so you must enable the Artifact Registry API first.

1. In Cloud Shell, run the following command to enable the Artifact Registry API:



Create a Service Account for Cloud SQL

You need to configure IAM service account credentials for the application that you will deploy later. The service account must be bound to a role that allows it to create and access Cloud SQL databases.

1. In Cloud Shell, create a Service Account and bind it to the Cloud SQL admin role in the lab project:

```
(qwiklabs-gcp-01-bca9ca514159)                                  Open Editor
  role: roles/cloudbuild.serviceAgent
- members:
  - serviceAccount:cloudsql-service-account@qwiklabs-gcp-01-bca9ca514159.iam.gservice
  role: roles/cloudsql.admin
- members:
  - serviceAccount:service-240564450528@compute-system.iam.gserviceaccount.com
  role: roles/compute.serviceAgent
- members:
  - serviceAccount:service-240564450528@container-engine-robot.iam.gserviceaccount.co
  role: roles/container.serviceAgent
- members:
  - serviceAccount:240564450528-compute@developer.gserviceaccount.com
  - serviceAccount:240564450528@cloudservices.gserviceaccount.com
  - user:student-01-b6493745d5ed@qwiklabs.net
  role: roles/editor
- members:
  - serviceAccount:admiral@qwiklabs-services-prod.iam.gserviceaccount.com
  - serviceAccount:qwiklabs-gcp-01-bca9ca514159@qwiklabs-gcp-01-bca9ca514159.iam.gser
  - user:student-01-b6493745d5ed@qwiklabs.net
  role: roles/owner
- members:
  - serviceAccount:qwiklabs-gcp-01-bca9ca514159@qwiklabs-gcp-01-bca9ca514159.iam.gser
  role: roles/storage.admin
- members:
  - user:student-01-b6493745d5ed@qwiklabs.net
  role: roles/viewer
etag: BwY47jYaQqQ=
version: 1
```

2. In Cloud Shell, create and export keys to a local file:

```
student_01_b6493745d5ed@cloudshell:~ (qwiklabs-gcp-01-bca9ca514159)$ gcloud iam service-accounts keys create $CLOUDSQL_SERVICE_ACCO
UNT.json \
    --iam-account=$CLOUDSQL_SERVICE_ACCOUNT@$PROJECT_ID.iam.gserviceaccount.com \
    --project=$PROJECT_ID
created key [93f2b4039b75ac508f9dd89911f50b24689656e3] of type [json] as [cloudsql-service-account.json] for [cloudsql-service-acco
unt@qwiklabs-gcp-01-bca9ca514159.iam.gserviceaccount.com]
student_01_b6493745d5ed@cloudshell:~ (qwiklabs-gcp-01-bca9ca514159)$
```

Task 2. Deploy a lightweight GKE application

In this task you will create a Kubernetes cluster and deploy a lightweight Google Kubernetes Engine (GKE) application on that cluster. You will configure the application to have access to the supplied Cloud SQL instance.

The application provided is a simple Flask-SQLAlchemy web application called gMemegen. It creates memes by supplying a set of photographs and capturing header and footer text, storing them in the database and rendering the meme to a local folder. It runs on a single pod with two containers; one for the application and one for the Cloud SQL Auth Proxy deployed in the side-car pattern.

A load balancer will marshal requests between the app and the database through the side-car. This load balancer will expose an external Ingress IP address through which you will access the app in your browser.

Create a Kubernetes cluster

In this step, you will create a minimal Kubernetes cluster. The cluster will take a couple of minutes to be deployed.

1. In Cloud Shell, create a minimal Kubernetes cluster as follows:

```
student_01_b6493745d5ed@cloudshell:~ (qwiklabs-gcp-01-bca9ca514159)$ ZONE=us-west1-a
gcloud container clusters create postgres-cluster \
--zone=$ZONE --num-nodes=2
Note: The Kubelet readonly port (10255) is now deprecated. Please update your workloads to use the recommended alternatives. See ht
tps://cloud.google.com/kubernetes-engine/docs/how-to/disable-kubelet-readonly-port for ways to check usage and for migration instru
ctions.
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster postgres-cluster in us-west1-a... Cluster is being configured...working...
```

```
Creating cluster postgres-cluster in us-east4-c... Cluster is being health-checked (Kubernetes Control Plane is healthy)...done.
Created [https://container.googleapis.com/v1/projects/qwiklabs-gcp-01-8515b3d6b4f6/zones/us-east4-c/clusters/postgres-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-east4-c/postgres-cl
uster?project=qwiklabs-gcp-01-8515b3d6b4f6
kubeconfig entry generated for postgres-cluster.
NAME: postgres-cluster
LOCATION: us-east4-c
MASTER_VERSION: 1.32.4-gke.1415000
MASTER_IP: 35.245.76.148
MACHINE_TYPE: e2-medium
NODE_VERSION: 1.32.4-gke.1415000
NUM_NODES: 2
STATUS: RUNNING
```

Create Kubernetes secrets for database access

In this step you will create a pair of Kubernetes secrets containing the credentials that are needed to connect to the Cloud SQL instance and database.

1. In Cloud Shell, run the following commands to create the secrets:

```
STATUS: RUNNING
student_02_a6409d95f054@cloudshell:~ (qwiklabs-gcp-01-8515b3d6b4f6)$ kubectl create secr
--from-file=credentials.json=$CLOUDSQL_SERVICE_ACCOUNT.json

kubectl create secret generic cloudsql-db-credentials \
--from-literal=username=postgres \
--from-literal=password=supersecret! \
--from-literal=dbname=gmemegen_db
secret/cloudsql-instance-credentials created
secret/cloudsql-db-credentials created
student_02_a6409d95f054@cloudshell:~ (qwiklabs-gcp-01-8515b3d6b4f6)$
```

Download and build the GKE application container

Before you can deploy the gMemegen application to your GKE cluster you need to build the container and push it to a repository.

1. In Cloud Shell, download the provided application code and change to the application directory:

```
Copying gs://spls/gsp919/gmemegen/app/static/images/Kubernetes_name.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/blb.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/cloudsql.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/images/gmemegen.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/gmemegen2.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/kubernetes.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/logo_gcp_hexagon_rgb.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/logo_gcp_horizontal_rgb.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/random.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/recent.png...
Copying gs://spls/gsp919/gmemegen/app/static/images/submit.png...
Copying gs://spls/gsp919/gmemegen/app/static/templates/aliens.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/templates/awkwardpenguin.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/templates/bill-lumbergh.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/templates/confessionbear.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/templates/fry.jpg...
Copying gs://spls/gsp919/gmemegen/app/static/templates/successkid.jpg...
Copying gs://spls/gsp919/gmemegen/app/templates/create_meme.html...
Copying gs://spls/gsp919/gmemegen/app/templates/footer.html...
Copying gs://spls/gsp919/gmemegen/app/templates/header.html...
Copying gs://spls/gsp919/gmemegen/app/templates/recent.html...
Copying gs://spls/gsp919/gmemegen/app/templates/view.html...
Copying gs://spls/gsp919/gmemegen/app/uwsgi.ini...
Copying gs://spls/gsp919/gmemegen/cloud_sql_proxy...
Copying gs://spls/gsp919/gmemegen/cloudbuild.yaml0...
Copying gs://spls/gsp919/gmemegen/gmemegen_deployment.yaml...
Copying gs://spls/gsp919/gmemegen/setup.sh...e
\ [84/84 files][ 21.5 MiB/ 21.5 MiB] 100% Done
Operation completed over 84 objects/21.5 MiB.
```

2. Create environment variables for the region, Project ID and Artifact Registry repository:

```
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$ export REGION=us-east4
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
export REPO=gmemegen
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$
```

3. Configure Docker authentication for the Artifact Registry:

```
    "docker.southamerica-east1.rep.pkg.dev": "gcloud",
    "southamerica-west1-docker.pkg.dev": "gcloud",
    "docker.southamerica-west1.rep.pkg.dev": "gcloud",
    "us-docker.pkg.dev": "gcloud",
    "us-central1-docker.pkg.dev": "gcloud",
    "docker.us-central1.rep.pkg.dev": "gcloud",
    "us-central2-docker.pkg.dev": "gcloud",
    "docker.us-central2.rep.pkg.dev": "gcloud",
    "us-east1-docker.pkg.dev": "gcloud",
    "docker.us-east1.rep.pkg.dev": "gcloud",
    "us-east4-docker.pkg.dev": "gcloud",
    "docker.us-east4.rep.pkg.dev": "gcloud",
    "us-east5-docker.pkg.dev": "gcloud",
    "docker.us-east5.rep.pkg.dev": "gcloud",
    "us-east7-docker.pkg.dev": "gcloud",
    "docker.us-east7.rep.pkg.dev": "gcloud",
    "us-south1-docker.pkg.dev": "gcloud",
    "docker.us-south1.rep.pkg.dev": "gcloud",
    "us-west1-docker.pkg.dev": "gcloud",
    "docker.us-west1.rep.pkg.dev": "gcloud",
    "us-west2-docker.pkg.dev": "gcloud",
    "docker.us-west2.rep.pkg.dev": "gcloud",
    "us-west3-docker.pkg.dev": "gcloud",
    "docker.us-west3.rep.pkg.dev": "gcloud",
    "us-west4-docker.pkg.dev": "gcloud",
    "docker.us-west4.rep.pkg.dev": "gcloud",
    "us-west8-docker.pkg.dev": "gcloud"
  }
}
Adding credentials for: us-east4-docker.pkg.dev
```

4. Create the Artifact Registry repository:

```
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$ gcloud artifacts repositories create $REPO \
    --repository-format=docker --location=$REGION
Create request issued for: [gmemegen]
Waiting for operation [projects/qwiklabs-gcp-01-8515b3d6b4f6/locations/us-east4/operations/80917d02-b2b0-44e8-a1c0-1d29d5aedeba] t
o complete...working...
```

Build a local Docker image

```
Created repository [gmemegen].
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$ docker build -t ${REGION}-docker.pkg.dev/${PROJECT_ID
}/gmemegen/gmemegen-app:v1 .
[+] Building 5.7s (4/7)                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 379B                                                                                  0.0s
 => [internal] load metadata for docker.io/tiangolo/uwsgi-nginx-flask:python3.6                                      3.9s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 2B                                                                                       0.0s
 => [1/3] FROM docker.io/tiangolo/uwsgi-nginx-flask:python3.6@sha256:07b3a61dc2f6d6b294065f929d33cdf6416133320cf61cc66b7cdb1  1.7s
 => => resolve docker.io/tiangolo/uwsgi-nginx-flask:python3.6@sha256:07b3a61dc2f6d6b294065f929d33cdf6416133320cf61cc66b7cdb1  0.0s
 => => sha256:893854f32f35504c63b9c26920224a14a9386f3970ee16d52d5752fda54ebb75 17.02kB / 17.02kB                      0.0s
 => => sha256:07b3a61dc2f6d6b294065f929d33cdf6416133320cf61cc66b7cdb1428460e75 6.58kB / 6.58kB                        0.0s
 => => sha256:9b99af5931b39ce167150ad668cfa57ddf7664697be9996cb7e0e6aebbf05843 50.44MB / 50.44MB                      0.8s
 => => sha256:b6013b3e77fe6fd3dcf46a05f8e5b3afa9fbca7ba0161c62e56beb4058334dec 7.83MB / 7.83MB                        0.8s
 => => sha256:bbced17b6899896c8e4016d62c885d737fe667acace2733e17c64bb974232887 10.00MB / 10.00MB                      0.9s
 => => extracting sha256:9b99af5931b39ce167150ad668cfa57ddf7664697be9996cb7e0e6aebbf05843                              0.9s
 => => sha256:8b609dabefa83fae157bcd42123a8ed45199bb6c301e09a11260c1cad9babfef 51.84MB / 51.84MB                      1.7s
 => => sha256:964244a9290251f119f00e920dbd27a25aaddb861833e0971b19278a003be884 6.15MB / 6.15MB                        1.5s
 => => sha256:50544bfef33d1d653b7bc10316e20bd84889fd56dd2b7e2f742db8364f6aeb70 51.38MB / 192.43MB                     1.7s
 => => sha256:ef5d3eb9cc29dbe223b94578946c3e6e2564beeb6a832c3254fc558a68491b5f 0B / 15.56MB                           1.7s
 => [internal] load build context                                                                                     0.1s
 => => transferring context: 4.21MB                                                                                   0.1s
```

6. Push the image to the Artifact Registry:

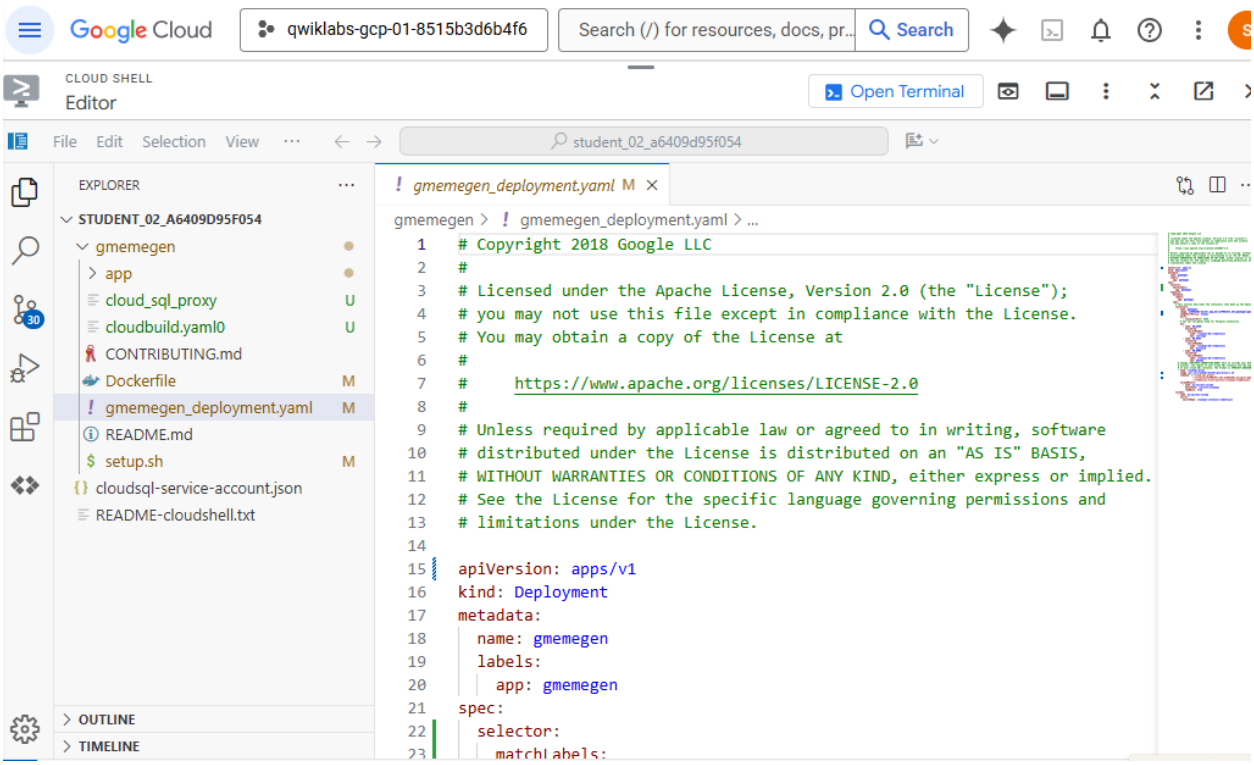
docker push ${REGION}-docker.pkg.dev/${PROJECT_ID}/gmemegen/gmemegen-app:v1

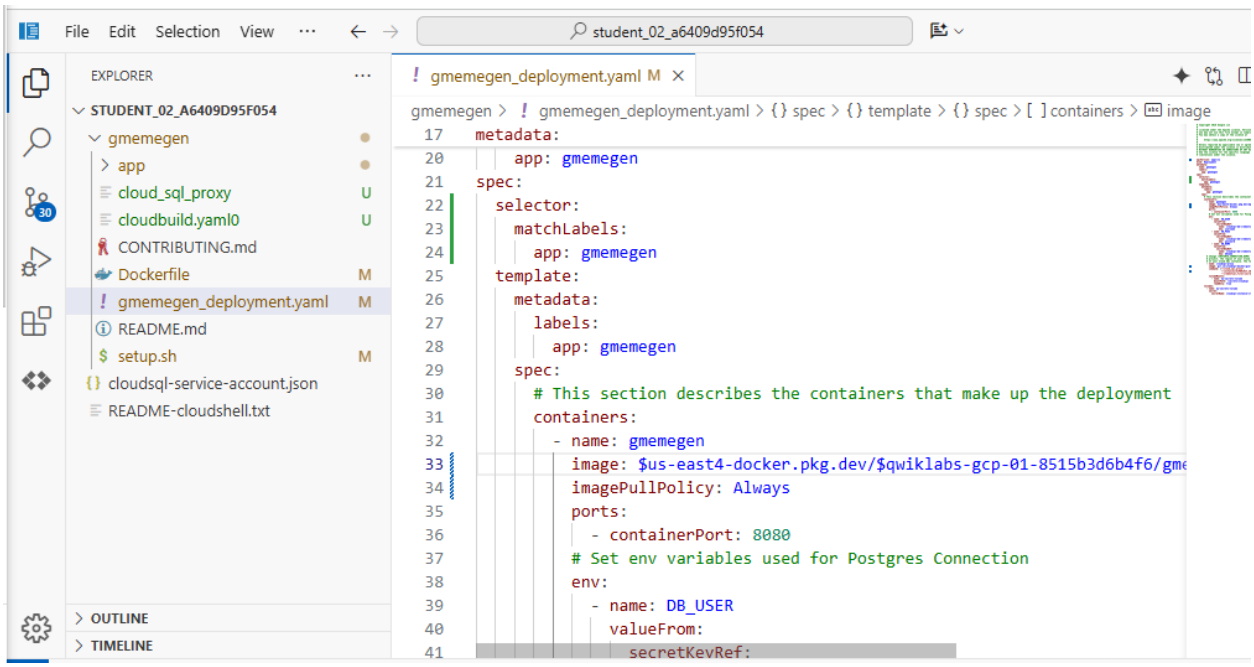
Configure and deploy the GKE application

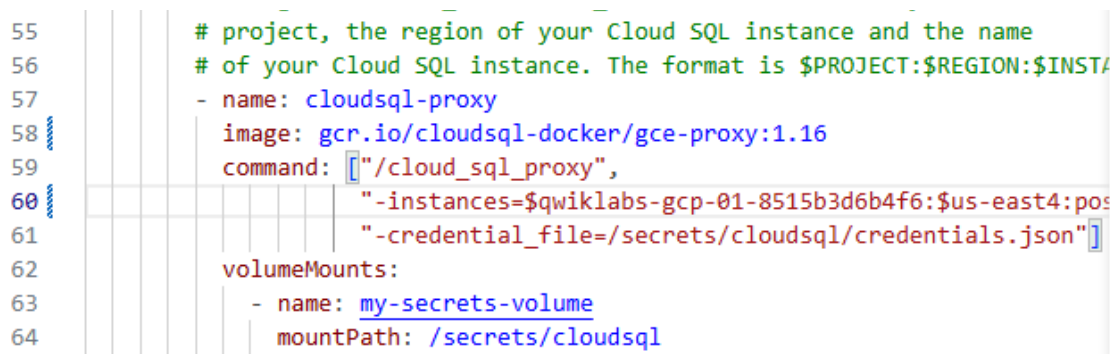
You must modify the Kubernetes deployment manifest for the gMemegen application to point at the correct container and configure the Cloud SQL Auth Proxy side-car with the connection string for the Cloud SQL PostgreSQL instance.

The instructions explain how to edit the file using the Cloud Shell Editor, but if you prefer you can use another editor, such as vi or nano, from Cloud Shell for these steps.

1. On the Cloud Shell menu bar, click **Open Editor** to open the Cloud Shell Editor.
2. Navigate the **Explorer** panel on the left hand side, expanding the gmemegen folder and then selecting gmemegen_deployment.yaml to edit the file.
3. On **line 33**, in the image attribute, replace ${REGION} with and ${PROJECT_ID} with . The line should now read:







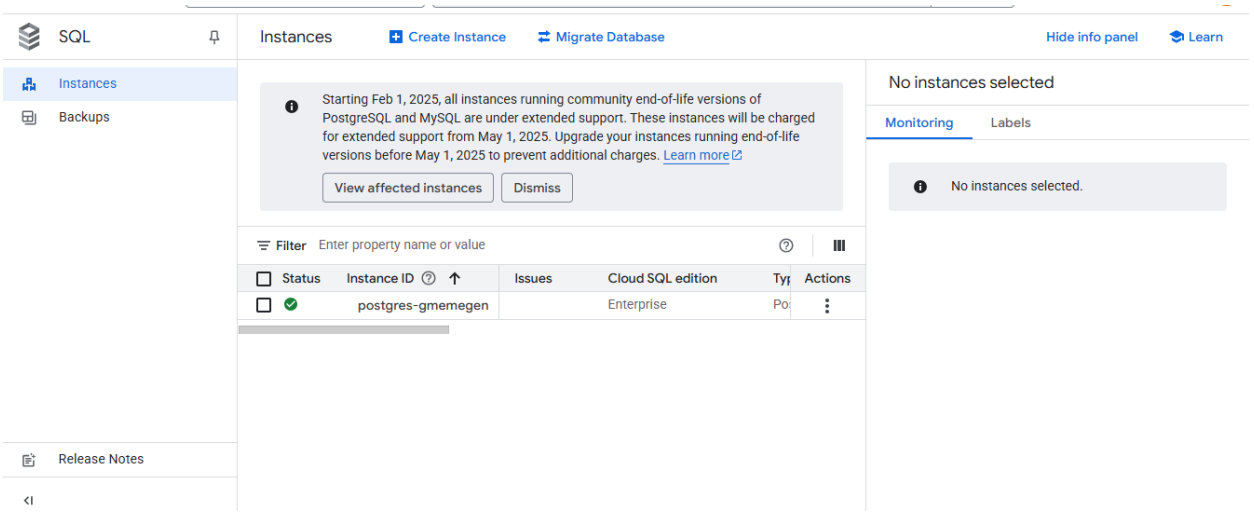
4. On **line 60**, replace ${REGION} with and ${PROJECT_ID} with . The line should now read:

To confirm that the connection name is correct, in the Cloud Console, navigate to **Databases** > **SQL**, select the postgres-gmemegen instance and compare with the **Connection name** in the **Overview** pane. A valid connection name is of the format PROJECT_ID:REGION:CLOUD_SQL_INSTANCE_ID.

5. Save your changes by selecting **File** > **Save** from the Cloud Shell Editor menu.

6. In the Cloud console click the **Open Terminal** to re-open Cloud Shell. You may need to resize the Terminal window by dragging down the handle at the centre top of the menu bar, in order to see your Cloud Console window above.

7. In Cloud Shell, deploy the application by running the following command:




```
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$ kubectl create -f gmemegen_deployment.yaml
error: error validating "gmemegen_deployment.yaml": error validating data: failed to download openapi: the server has asked for the client to provide credentials; if you
 choose to ignore these errors, turn validation off with --validate=false
student_02_a6409d95f054@cloudshell:~/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$
```

8. In Cloud Shell, check that the deployment was successful by running the following command:

```
-bash: 200~kubectl: command not found
student_02_a6409d95f054@cloudshell:~/gmemegen/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$ kubectl get pods
E0702 09:48:11.748307    3285 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credenti
als"
E0702 09:48:12.406011    3285 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credenti
als"
E0702 09:48:13.069538    3285 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credenti
als"
E0702 09:48:13.724342    3285 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credenti
als"
E0702 09:48:14.375339    3285 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: the server has asked for the client to provide credenti
als"
error: You must be logged in to the server (the server has asked for the client to provide credentials)
student_02_a6409d95f054@cloudshell:~/gmemegen/gmemegen (qwiklabs-gcp-01-8515b3d6b4f6)$
```

Task 3. Connect the GKE application to an external load balancer

In this task you will create a load balancer to marshal requests between the containers in your GKE pods and access the application using its external IP address from your browser.

Create a load balancer to make your GKE application accessible from the web

In this step you will create a Kubernetes load balancer service that will provide your application with a public IP address.

1. In Cloud Shell, run the following command to create a load balancer for the application:

```
kubectl expose deployment gmemegen \
  --type "LoadBalancer" \
  --port 80 --target-port 8080
```

Test the application to generate some data

In this step you will access the gMemegen application from your web browser.

The application has a very simple interface. It launches to the application home page, which displays 6 candidate images for making memes. You can select an image by clicking on it.

The **Create Meme** page is displayed, where you enter two items of text, to be displayed at the top and bottom of the image. Clicking **Submit** renders the meme and displays it. The interface provides no navigation from the completed meme page. You will have to use the browser's back button to return to the home page.

There are two other pages, **Recent** and **Random**, which display a set of recently generated memes and a random meme, respectively. Generating memes and navigating the UI will generate database activity which you can view in the logs as described below.

Wait for the load balancer to expose an external IP, which you can retrieve as follows:

1. In Cloud Shell, copy the external IP address attribute of the LoadBalancer Ingress from the output of:

kubectl describe service gmemegen

2. In a browser, navigate to the load balancer's Ingress IP address.

You can create a clickable link to the external IP address of the load balancer in Cloud Shell using the following commands:

export LOAD_BALANCER_IP=$(kubectl get svc gmemegen

-o=jsonpath='{.status.loadBalancer.ingress[0].ip}' -n default) echo gMemegen Load Balancer Ingress IP: http://$LOAD_BALANCER_IP

4. Click the link in Cloud Shell and you will see the gMemegen application running in a new tab in your browser.
5. Create a meme as follows:
    a. On the **Home** page, click on one of the presented images.
    b. Enter text in the **Top** and **Bottom** text boxes.
    c. Click the **Submit** button.

6. To create more memes, use the browser's back button to navigate to the home page.
7. To view existing memes, click **Recent** or **Random** in the application menu. (Note that **Random** opens a new browser tab)
8. In Cloud Shell, view the application's activity by running the following:

```
POD_NAME=$(kubectl get pods --output=json | jq -r ".items[0].metadata.name")

kubectl logs $POD_NAME gmemegen | grep "INFO"
```

Task 4. Verify full read/write capabilities of application to database

In this task you will verify that the application is able to write to and read from the database.

Connect to the database and query an application table

In this step you will connect to the Cloud SQL instance by running **PL/SQL** in Cloud Shell.

1. In Google Cloud Console, navigate to **Databases** > **SQL** and select the postgres-gmemegen instance.
2. In the **Overview** pane , scroll down to **Connect to this instance** and click the **Open Cloud Shell** button.
3. Run the auto-populated command in Cloud Shell.
4. When prompted, enter the password: supersecret!
5. At the postgres=> prompt enter the following command to select the gmemegen_db database:

\c gmemegen_db

6. When prompted, enter the password: supersecret!
7. At the gmemegen_db=> prompt enter:

select * from meme;