**Creation of MySQL database instance on GCP**

**1. Create a MySQL Database Instance**

**1.1 Log in to Google Cloud Console**

Navigate to Google Cloud Console.

Sign in with your Google account credentials.

**1.2 Navigate to the SQL Section**

In the Google Cloud Console, select the navigation menu (three horizontal lines in the upper left corner).

Scroll down and click on SQL to go to the Cloud SQL section.

**1.3 Create a New SQL Instance**

Click on Create Instance.

Choose MySQL as the database engine.

Click Next to configure the MySQL instance settings**.**

**1.4 Configure Instance Settings**

Instance ID: Enter a unique ID for your SQL instance.

Password: Set a password for the root user.

Region and Zone: Select the appropriate region and zone for your instance.

Click Create to deploy the instance.

Wait for the instance to be set up (this may take a few minutes).

Use the smallest micro instance

**2. Create a Database**

**2.1 Access the Instance**

Once the instance setup is complete, click on the instance name to open its overview page.

Go to the Databases tab.

**2.2 Add a New Database**

Click Create Database.

In the dialog, enter the name for your new database.

Optionally, adjust the character set and collation.

Click Create.

**3. Create Tables Using Cloud SQL Studio**

**3.1 Access Cloud SQL Studio**

Navigate to your SQL instance's overview page by selecting SQL from the Google Cloud Console sidebar and clicking on your instance.

Under the "SQL" section, click on Open in Cloud SQL Studio. This will launch a new tab with the Cloud SQL Studio interface.

**3.2 Connect to Your Database**

Once in Cloud SQL Studio, you will see an interface similar to a desktop SQL editor.

Select your instance from the sidebar if it is not already selected.

Enter the database credentials (if prompted) to establish a connection. Typically, you would use the root account and the password you set during the instance creation.

**3.3 Use the SQL Editor**

In the SQL Studio, find the SQL editor section. It usually features a large text area where you can type or paste your SQL commands.

Ensure you are connected to the correct database by selecting it from the drop-down menu near the top of the editor.

**3.4 Define Table Schema**

In the SQL editor, type your SQL commands to create new tables. For example, to create a students table, you might enter:

CREATE TABLE `student\_infos` ( `proposal\_id` varchar(255) DEFAULT NULL, `name` varchar(255) NOT NULL, `project\_name` varchar(255) NOT NULL, `mentor` varchar(255) DEFAULT NULL, `github\_link` varchar(255) DEFAULT NULL, `objective\_image` longblob, `objective` text, `rationale` text, `timeline` text, `contributors` text, `semester` varchar(100) DEFAULT NULL, `expected\_students` int DEFAULT NULL, `mentor\_email` varchar(255) DEFAULT NULL, `dataset\_image` longblob, `dataset` text, `approach` text, `possible\_issues` text, `possible\_issues\_image` longblob, `year` year DEFAULT NULL, `proposed\_by\_professor` tinyint(1) DEFAULT '0', `status` varchar(100) DEFAULT 'Pending Approval', `video\_link` varchar(255) DEFAULT NULL, `project\_website` text, `project\_document` blob, UNIQUE KEY `proposal\_id` (`proposal\_id`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3

Execute the SQL command by clicking the Run button or hitting the appropriate shortcut (often Ctrl+Enter).

**3.5 Verify Table Creation**

After running the SQL commands, you can verify the creation of your table directly in the SQL Studio. Run the following SQL query to list all tables:

Select \* from student\_infos

You should see students listed among the tables in your database.

**\*\*Make sure to create a cloud storage bucket called project-capstone in cloud storage\*\***

**Obtaining Information for Environment Variables from MySQL Instance on Google Cloud SQL**

To successfully connect your application to a MySQL database hosted on Google Cloud SQL, you need various pieces of information that should be specified in your environment variables. Here’s how you can gather each piece of data required for the environment variables from your Google Cloud SQL instance:

**Open the Google Cloud Console:** Navigate to the Google Cloud Console.

**Go to SQL:** In the navigation menu on the left-hand side, go to "SQL" to access the SQL databases dashboard.

**Select Your MySQL Instance:** Click on the MySQL instance for which you need the information.

**Instance Details:**

**Instance Connection Name:** Found under the "Overview" tab of your SQL instance. It usually follows the format project-id:region:instance-name.

**IP Address (HOST):** Found under the "Overview" tab, labeled as the "Public IP address" if you have set up public IP connectivity.

**Database Details:**

**Database Name (DB\_NAME):** Go to the "Databases" tab inside your SQL instance where you can see a list of databases. You need the name of the database you are connecting to.

**Credentials:**

**Username (DB\_USER) and Password (DB\_PASS):** These are set when you create the instance or can be managed under the "Users" tab. If you haven't set a user and password, you will need to create a new user account here.

**Port:**

The default port for MySQL is 3306, which is what Google Cloud SQL uses unless configured otherwise.

**Project Documentation: Deployment and Management of Dockerized Applications on Google Cloud**

**Overview**

This document outlines the steps I followed to develop, containerize, and deploy a Python-based application using Docker and Google Cloud services. The focus was on maintaining secure management of sensitive data through environment variables and deploying the containerized application to Google Cloud Platform (GCP).

**Step 1: Local Development and Environment Setup**

I developed a Python application which required several environment variables for configuration, including database connections and API keys. These variables were stored locally in an .env file to keep sensitive information out of the public codebase.

**Key Tools Used:**

Python 3.8

Docker

.env file for environment variables

**Step 2: Dockerization of the Application**

I created a Dockerfile to containerize the application. The Dockerfile instructions included copying the application code into the Docker image, installing dependencies from a requirements.txt file, and setting the command to run the application.

**Dockerfile:**

# Use the official Python image from Docker Hub

FROM python:3.8-slim

# Update system and install necessary packages

RUN apt-get update && apt-get install -y \

    gcc \

    libc-dev \

    libssl-dev \

    default-libmysqlclient-dev \

    pkg-config \

    git \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/\*

# Set the working directory in the container

WORKDIR /app

# Install any dependencies

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

# Make port 8501 available to the world outside this container

EXPOSE 8501

# Copy the content of the local src directory to the working directory

COPY src/ ./src/

# Copy other necessary files or directories

COPY data/ ./data/

# Command to run the application

CMD ["streamlit", "run", "src/app.py",  "--server.port=8501"]

.dockerignore File:

To prevent the .env file from being included in the Docker image, I included it in the .dockerignore file:

**Creating a secret key for gcloud authentication:**

To download the Google Cloud service account key file, which you'll use for authentication in various applications and services, follow these step-by-step instructions:

Step 1: Log into Google Cloud Console

Open your web browser and navigate to the Google Cloud Console.

Log in with your Google account credentials that have access to your Google Cloud project.

Step 2: Select or Create a Project

Once logged in, make sure you are working within the correct Google Cloud project. You can select your project from the project dropdown menu at the top of the console dashboard.

Step 3: Navigate to IAM & Admin

In the navigation menu on the left-hand side of the console, find and click on "IAM & Admin". This will expand to show more options.

Click on "Service Accounts" within the expanded IAM & Admin menu.

Step 4: Create a Service Account (if necessary)

If you haven't already created a service account for your application, click on "CREATE SERVICE ACCOUNT" at the top of the service accounts list.

Fill in the service account details:

Name: Provide a name that describes the service account's purpose.

ID: Automatically filled based on the name, but can be customized.

Description: Optional, but helpful for understanding the account's purpose.

Click "Create".

Assign roles to the service account according to the permissions it requires (e.g., Project > Viewer).

Roles needed : Cloud SQL Client, Storage Object Admin

Click "Continue" and then "Done" to finish creating the service account.

Step 5: Manage Keys for the Service Account

In the list of service accounts, click on the newly created service account to manage it.

Navigate to the "Keys" tab.

Step 6: Add a New Key

Click on "ADD KEY" and select "Create new key" from the dropdown menu.

Choose the "JSON" key type, which will generate a new JSON key file.

Click "Create". The JSON key file will automatically download to your computer.

Step 7: Secure the Key File

Store the downloaded JSON key file securely. It contains sensitive information that can provide access to your Google Cloud resources. And name it to “creds.json”

Ensure you never commit this file to your source code repositories or expose it in public or insecure environments.

**Step 3: Managing Environment Variables Securely**

For local testing with Docker, I used the --env-file option when running the Docker container to pass environment variables:

**To build the docker file**

docker build -f "D:\Capstone Website - streamlit\_dup\Data-Science-Capstone-Website-updated\DockerFile" -t my-streamlit-app .

**to run the docker file**

docker run -p 8501:8501 --env-file .env -v "D:/creds.json:/secrets/key-file.json" -e "GOOGLE\_APPLICATION\_CREDENTIALS=/secrets/key-file.json" my-streamlit-app

**Step 4: Pushing the Docker Image to google private container registry**

To deploy your Docker image to Google Container Registry using Google Cloud SDK, follow these steps:

**Step 1: Initialize Google Cloud SDK**

Ensure that the Google Cloud SDK is installed on your machine. If not, you can download and install it from the Google Cloud SDK Documentation. Once installed, open your command line interface and run:

gcloud init

Follow the prompts to log in with your Google account and set up your cloud project.

**Step 2: Configure Docker to Authenticate with Google Cloud**

Run the following command to configure Docker to authenticate directly with Google Container Registry. This step ensures that Docker can push images to your Google Cloud account.

gcloud auth configure-docker

**Step 3: Tag Your Docker Image**

Before pushing your Docker image to the registry, you need to tag it with the registry's name. Replace my-streamlit-app with the name of your Docker image, and use your Google Cloud project ID in place of astral-name-421619.

docker tag my-streamlit-app gcr.io/astral-name-421619/my-streamlit-app:latest

“astral-name-421619” this is the name of my project

**Step 4: Push the Image to Google Container Registry**

Now, push the tagged image to the Google Container Registry using the following command. This uploads your Docker image to your Google Cloud project, allowing it to be used on Google Cloud services like Google Kubernetes Engine or Google Cloud Run.

docker push gcr.io/astral-name-421619/my-streamlit-app:latest

This will upload your Docker image to the Google Container Registry under your specified project. Ensure you have the necessary permissions to push images to the Container Registry and that the project ID and image name are correctly specified.

**CloudRun to deploy the application:**

To deploy a Docker container from your private Google Container Registry to Google Cloud Run, including setting environment variables, follow these step-by-step instructions:

Step 1: Open Google Cloud Console

Start by navigating to the Google Cloud Console in your web browser.

Step 2: Go to Cloud Run

In the Google Cloud Console, go to the navigation menu (three horizontal lines in the top left corner), scroll down and click on "Cloud Run" under the "Compute" section.

Step 3: Create a New Service

Click on the "Create Service" button to start deploying a new Cloud Run service.

Step 4: Configure the Service

Select Deployment Platform: Choose "Cloud Run (fully managed)".

Service Name: Enter a name for your service.

Region: Select a region where you want your service to be deployed. It’s recommended to choose the region closest to your users to reduce latency.

Step 5: Set up the Container Image

Click on "Select" next to "Container image URL".

In the "Select container image" dialog, go to the "Private Images" tab.

You should see your Docker images listed here. Select the image you want to deploy. The image should be tagged appropriately as you pushed it to the registry (e.g., gcr.io/astral-name-421619/my-streamlit-app:latest).

Step 6: Add Environment Variables

Below the container image URL, find the "Variables & Secrets" section.

Click on "Add Variable".

Enter the environment variable name and value. Repeat this step for each variable you need for your application. For example:

Name: API\_KEY

Value: your\_api\_key\_value

This can be obtained from the .env file sent!!

Step 7: Configure Advanced Settings (Optional)

Memory allocated: Depending on the needs of your application, you may adjust the memory allocation.

CPU allocated: Assign CPUs based on the expected workload.

Request timeout: Set this according to how long a function might take to respond.

Concurrency: Define the number of requests that can be handled simultaneously by a single container instance.

Step 8: Authentication

Set the "Allow unauthenticated invocations" to allow public access, or leave it unchecked to require authentication, depending on your security needs.

Step 10: Changing container port settings

In the container section change the container port to 8501

Step 11: Deploy

Click on the "Create" button to deploy your container. Google Cloud Run will pull the image from your Container Registry and deploy it. The deployment might take a few minutes.

Step 12: Access Your Deployed Service

Once deployed, Cloud Run will provide you with a URL to access your service. You can click this URL directly from the Cloud Run dashboard to view your deployed application.