Cloud Application Development. Assignment №3 Report
Amen Azat, 21B030774

## Exercise 1: Managing APIs with Google Cloud Endpoints

1. **Objective**: Deploy and manage an API using Google Cloud Endpoints.
2. **Instructions**:
    1. **Setup**:
        - Ensure you have a Google Cloud account.
        - Install the Google Cloud SDK and `gcloud` command-line tool.

```
C:\Users\azikkw>gcloud version
Google Cloud SDK 495.0.0
bq 2.1.8
core 2024.09.27
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.9
gsutil 5.30
Updates are available for some Google Cloud CLI components.  To install them,
please run:
  $ gcloud components update
```

    2. **Create a Project**:
        - Create a new project in the Google Cloud Console.
    3. **Prepare the API**:
        - Create a simple REST API using Python Flask.

          Example `main.py`:

Assignment 3 > Exercise1 > main.py > ...

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/api/hello', methods=['GET'])
def hello():
    return jsonify({'message': 'Hello, World!'})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080, debug=True)
```

    4. **Create an OpenAPI Specification**:
        - Create an `openapi.yaml` file to define your API.

Example `openapi.yaml`:

Assignment 3 > Exercise1 > openapi.yaml

```yaml
swagger: '2.0'
info:
  title: Hello World API
  description: A simple API to say hello
  version: "1.0.0"
host: western-avatar-435512-h0.appspot.com
schemes:
  - https
paths:
  /api/hello:
    get:
      operationId: getHelloMessage
      summary: Returns a hello message
      responses:
        200:
          description: A hello message
          schema:
            type: object
            properties:
              message:
                type: string
                example: "Hello, World!"
```

5. **Deploy the API to Google Cloud Endpoints**:
   - Create a new service and deploy your API. Use the following commands to do that:
     1. To deploy the API configuration:

        `gcloud endpoints services deploy openapi.yaml`

```
E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise1>gcloud endpoints services deploy openapi.yaml
Waiting for async operation operations/serviceConfigs.western-avatar-435512-h0.appspot.com:368fb72a-0a74-466d-b219-18f52d369cb8 to complete...
Operation finished successfully. The following command can describe the Operation details:
 gcloud endpoints operations describe operations/serviceConfigs.western-avatar-435512-h0.appspot.com:368fb72a-0a74-466d-b219-18f52d369cb8

Waiting for async operation operations/rollouts.western-avatar-435512-h0.appspot.com:476f4231-40e4-44c4-9795-f74cfb374cd6 to complete...
Operation finished successfully. The following command can describe the Operation details:
 gcloud endpoints operations describe operations/rollouts.western-avatar-435512-h0.appspot.com:476f4231-40e4-44c4-9795-f74cfb374cd6

Service Configuration [2024-10-08r0] uploaded for service [western-avatar-435512-h0.appspot.com]

To manage your API, go to: https://console.cloud.google.com/endpoints/api/western-avatar-435512-h0.appspot.com/overview?project=western-avatar-435512-h0

E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise1>pip freeze > requirements.txt

E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise1>
```

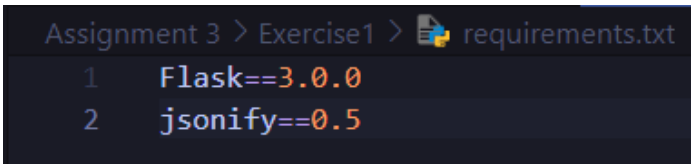
Активация Windows
Чтобы активировать Windows, перейдите в раздел

     2. To deploy service, firstly create `app.yaml`:

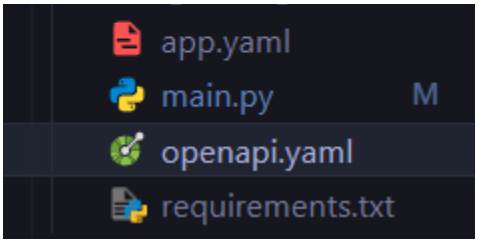
Assignment 3 > Exercise1 > app.yaml

```yaml
runtime: python39
handlers:
  - url: /.*
    script: auto
```

After that, since the application is in python, you need to add `requirements.txt` with all the necessary libraries to run the project:

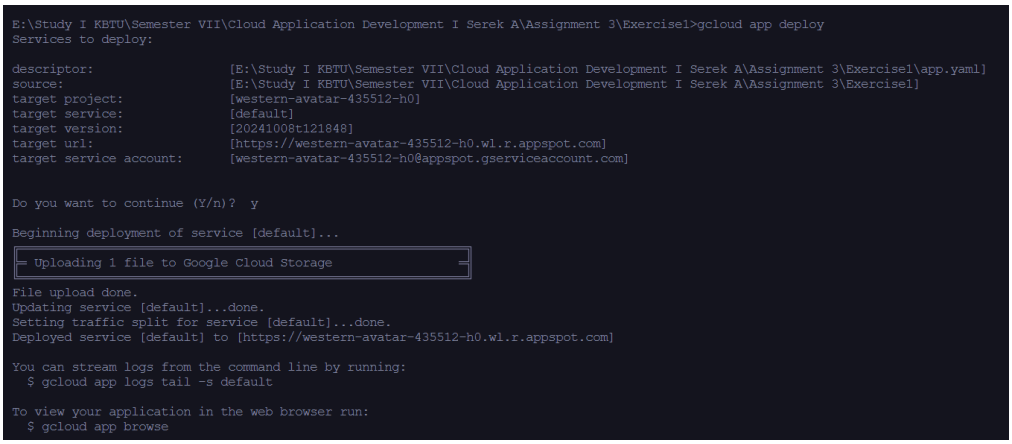



Then your folder should contain following files:

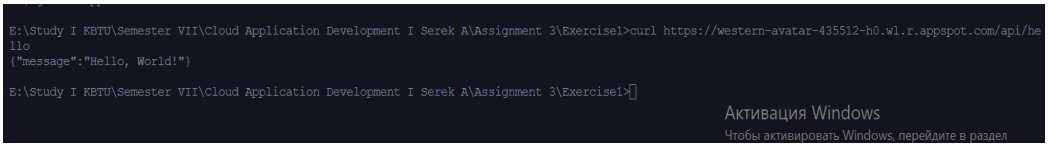



Finally, use following command to deploy your service to App Engine:

```
gcloud app deploy
```



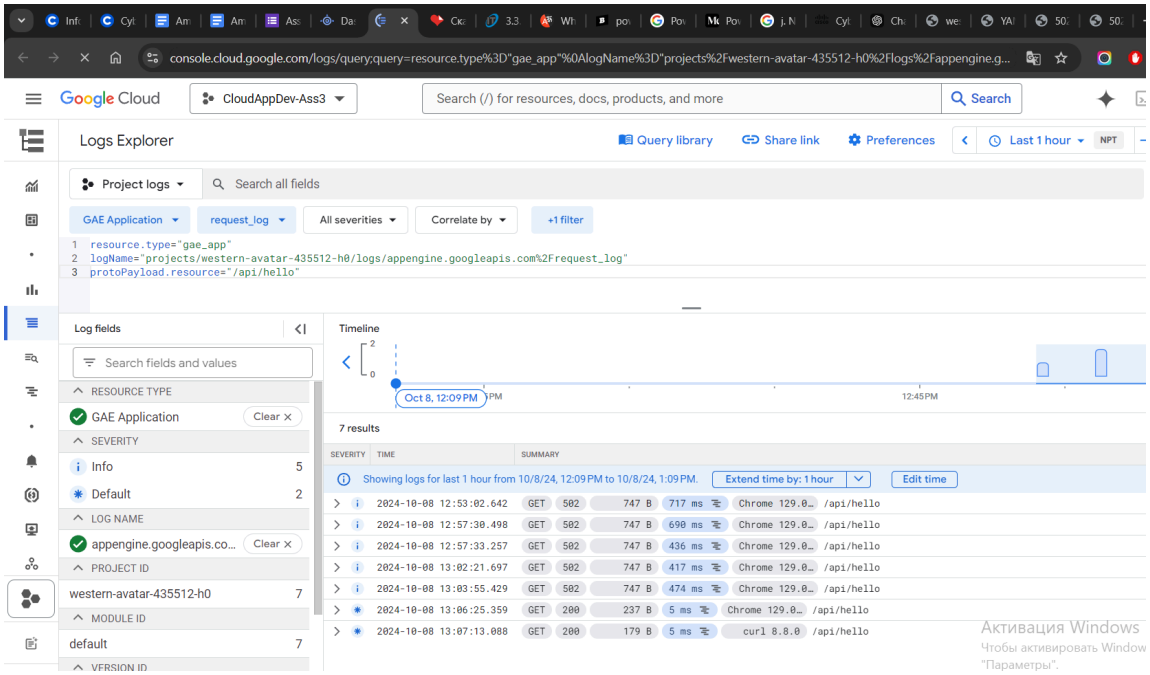

6. **Test the API**:
   - Once deployed, use the provided URL to test the API endpoint via a web browser or `curl`. Test result:




As you can see by the returned response. My API successfully created and deployed to the Cloud Endpoints, and my Flask application also created and deployed to the App Engine successfully.

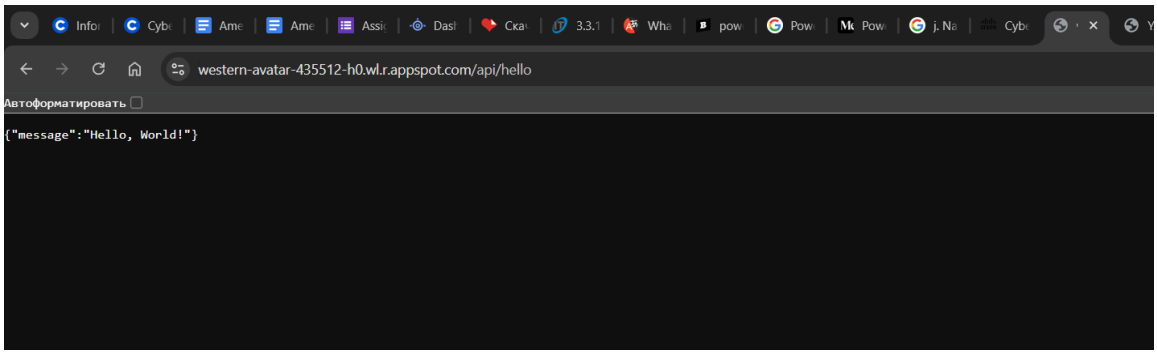
3. **Deliverables**:

- A deployed API on Google Cloud Endpoints.

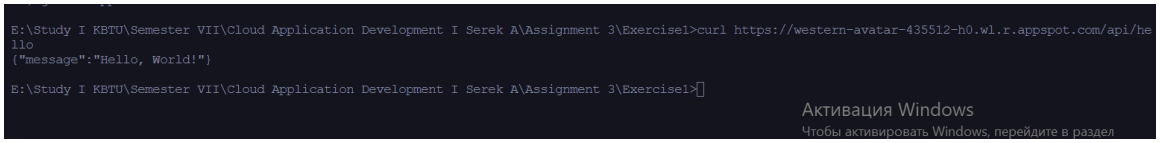



- A screenshot of a successful API call response.

  1. Response in browser:




  2. And response by curl:




## Exercise 2: Google Cloud Databases

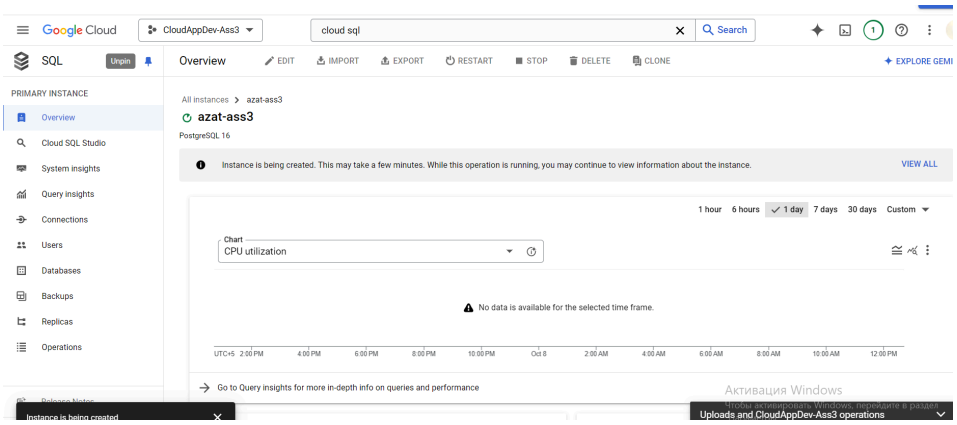
1. **Objective**: Set up and interact with a Google Cloud SQL database.
2. **Instructions**:

1. **Setup**:
    - Ensure you have a Google Cloud account.
    - Install the Google Cloud SDK.

```
C:\Users\azikkw>gcloud version
Google Cloud SDK 495.0.0
bq 2.1.8
core 2024.09.27
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.9
gsutil 5.30
Updates are available for some Google Cloud CLI components.  To install them,
please run:
  $ gcloud components update
```

2. **Create a Cloud SQL Instance**:
    - Navigate to the Google Cloud Console and create a new Cloud SQL instance.
    - Choose MySQL, PostgreSQL, or SQL Server as the database type.
    - Configure the instance settings (region, machine type, etc.).

      I do all this steps and finally created PostgreSQL Server called azat-ass3:




3. **Create a Database and Table**:
    - Connect to your Cloud SQL instance using the Cloud SQL client or `mysql` command-line tool.
    - Create a new database and a table with sample data.

      To do this we need to do the following steps:

      1. Firstly we need to connect to created Cloud SQL Server using following command:

```
gcloud sql connect SQL_SERVER_NAME
--user=USER_NAME
```

      I connected and listed current databases in my SQL Server.

```
E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise2>gcloud sql connect azat-ass3 --user=postgres
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [postgres].Password:
psql (16.4)
WARNING: Console code page (866) differs from Windows code page (1251)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

postgres=> \l
                                                                List of databases
     Name      |       Owner       | Encoding | Locale Provider |  Collate   |   Ctype    | ICU Locale | ICU Rules |           Access pr
s
---------------+-------------------+----------+-----------------+------------+------------+------------+-----------+---------------------
------------
 cloudsqladmin | cloudsqladmin     | UTF8     | libc            | en_US.UTF8 | en_US.UTF8 |            |           |
 postgres      | cloudsqlsuperuser | UTF8     | libc            | en_US.UTF8 | en_US.UTF8 |            |           |
 template0     | cloudsqladmin     | UTF8     | libc            | en_US.UTF8 | en_US.UTF8 |            |           | =c/cloudsqladmin
           +
               |                   |          |                 |            |            |            |           | cloudsqladmin=CTc/cl
dmin
 template1     | cloudsqlsuperuser | UTF8     | libc            | en_US.UTF8 | en_US.UTF8 |            |           | =c/cloudsqlsuperuser
           +
               |                   |          |                 |            |            |            |           | cloudsqlsuperuser=CT
sqlsuperuser
(4 rows)
```

2. After connection to Server you need to create database using:

```
CREATE DATABASE database_name;
```

```
postgres=> CREATE DATABASE sample_db;
CREATE DATABASE
postgres=> \l

     Name      |       Owner       | Encoding | Locale Provider |  Co
s
---------------+-------------------+----------+-----------------+----
-------------
 cloudsqladmin | cloudsqladmin     | UTF8     | libc            | en_
 postgres      | cloudsqlsuperuser | UTF8     | libc            | en_
 sample_db     | postgres          | UTF8     | libc            | en_
 template0     | cloudsqladmin     | UTF8     | libc            | en_
           +
               |                   |          |                 |
dmin
 template1     | cloudsqlsuperuser | UTF8     | libc            | en_
           +
               |                   |          |                 |
sqlsuperuser
(5 rows)
```

3. In the next step, connect to created database using:

```
\c database_name;
```

```
postgres=> \c sample_db;
Password:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384,
You are now connected to database "sample_db" as user "postgres".
```

4. Then create table using following command:

```
CREATE TABLE table_name(
  id SERIAL PRIMARY_KEY,
  name VARCHAR(100) NOT_NULL,
  email VARCHAR(100) NOT_NULL);
```

```
sample_db=> CREATE TABLE users (
sample_db(> id SERIAL PRIMARY KEY,
sample_db(> name VARCHAR(100) NOT NULL,
sample_db(> email VARCHAR(100) NOT NULL
sample_db(> );
CREATE TABLE
sample_db=> \dt;
         List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | users | table | postgres
(1 row)
```

5. And finally insert some data to your table like in screenshot under:

```
sample_db=> INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
INSERT 0 1
sample_db=> INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
INSERT 0 1
sample_db=> SELECT * FROM users;
 id | name  |       email
----+-------+-------------------
  1 | Alice | alice@example.com
  2 | Bob   | bob@example.com
(2 rows)


sample_db=>
```

4. **Connect to the Database**:
   - Create a connection to the Cloud SQL instance from a Python application. Example `connect.py`:


Here, I write code using the `psycopg2` library. My code connects to the `sample_db` database in SQL Server and selects all users from the `users` table.

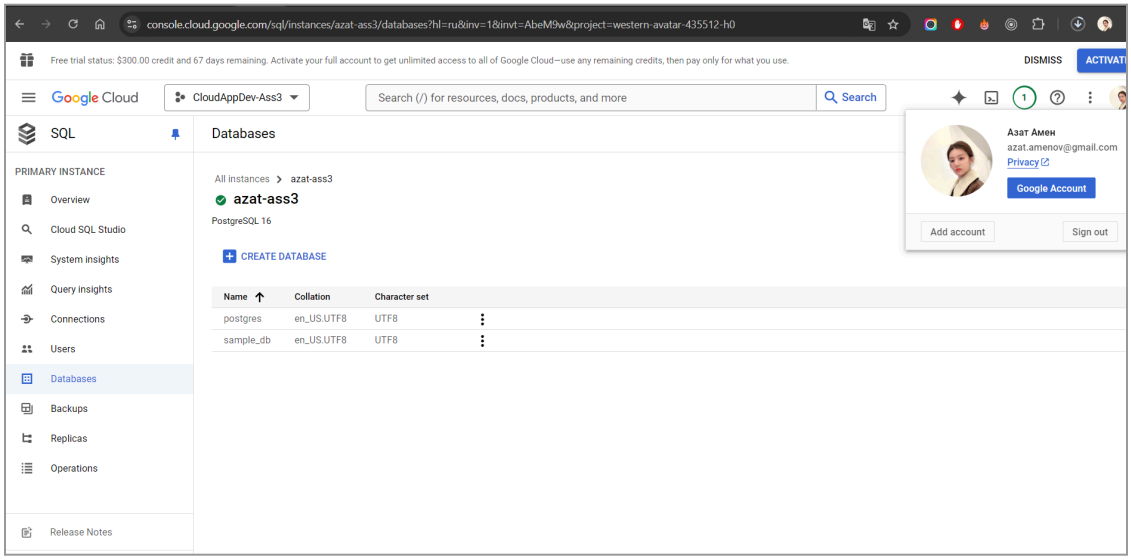
5. **Run the Connection Code**:
    - Execute the Python script to verify that you can retrieve data from the Cloud SQL instance.

```
PS E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise2> python .\connect.py
(1, 'Alice', 'alice@example.com')
(2, 'Bob', 'bob@example.com')
PS E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise2>
```

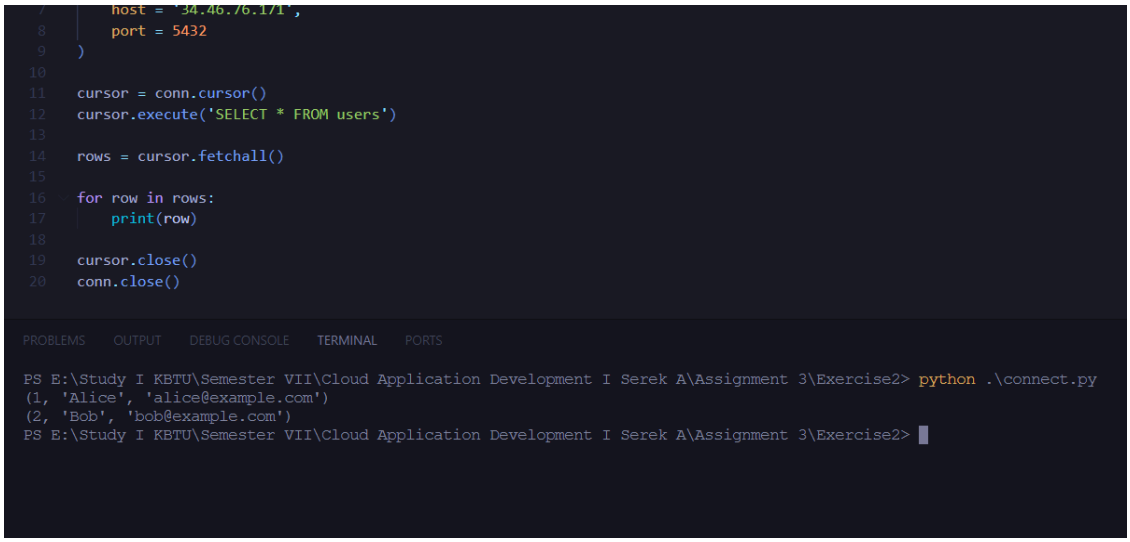
I executed my `connect.py` and it returned me the right output with my data in the users table.

3. **Deliverables**:

- A working Cloud SQL database with sample data.




- A Python script that successfully connects to and queries the database.



```
    host = '34.46.76.171',
    port = 5432
)

cursor = conn.cursor()
cursor.execute('SELECT * FROM users')

rows = cursor.fetchall()

for row in rows:
    print(row)

cursor.close()
conn.close()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise2> python .\connect.py
(1, 'Alice', 'alice@example.com')
(2, 'Bob', 'bob@example.com')
PS E:\Study I KBTU\Semester VII\Cloud Application Development I Serek A\Assignment 3\Exercise2>
```

## Exercise 3: Integrating Machine Learning with Google Cloud

1. **Objective**: Train and deploy a machine learning model using Google Cloud AI Platform.

- Unfortunately, my free trial period has ended after the Midterm Project, and now I can't use the full functionality of Google Cloud to create a storage bucket, ai model and deploy ai model to Google Cloud AI Platform *(But I already completed the first and second exercises, but I didn't have time to do 3rd exercise)*. Nevertheless, I will try to explain in detail the process of creating a storage bucket, adding data there, as well as deploying the model on the Google Cloud AI Platform, training the model and testing it. *I will also train my model locally*.

2. **Instructions**:

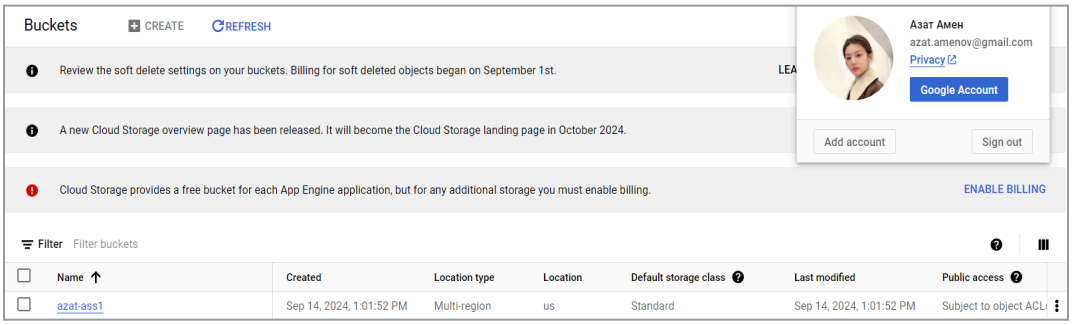
1. **Setup**:
   - Ensure you have a Google Cloud account.
   - Install the Google Cloud SDK and TensorFlow.

```
C:\Users\azikkw>gcloud version
Google Cloud SDK 495.0.0
bq 2.1.8
core 2024.09.27
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.9
gsutil 5.30
Updates are available for some Google Cloud CLI components.  To install them,
please run:
  $ gcloud components update
```

```
C:\Users\azikkw>pip show tensorflow
Name: tensorflow
Version: 2.17.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\users\azikkw\appdata\local\programs\python\python310\lib\site-packages
Requires: tensorflow-intel
Required-by:
```

2. **Create a Cloud Storage Bucket**:
   - Create a new Cloud Storage bucket to store your training data and model. To create storage bucket we need to use following command:

```
gsutil mb gs://bucket_name
```



As a result, in the Google Cloud Console, you will see the created storage basket (like on screenshot: azat-ass1) in the Cloud Storage service

3. **Prepare Training Data**:
   - Upload sample training data to your Cloud Storage bucket. For example, use a dataset for classification or regression.
     To do this, you need to use the next command:

     ```
     gsutil cp your_file_path gs://bucket_name
     ```

     When you did that, you can use following command to see that files uploaded to storage bucket:

     ```
     gsutil ls gs://bucket_name
     ```

     Or just open your storage bucket in Cloud Storage to check.
4. **Create a Training Script**:
   - Write a simple TensorFlow training script.
     Example `train.py`:


5. **Train the Model**:
   - Submit a training job to Google Cloud AI Platform.
     Use the following command to start training:

     ```
     gcloud ai custom-jobs create \
       --region=your-region \
     ```

```
--display-name=ml-job \
--python-package-uris=gs://your-bucket/train.py \
--python-module=train \
--container-image-uri=gcr.io/cloud-aiplatform/training/tf-cpu.2-4:latest
```

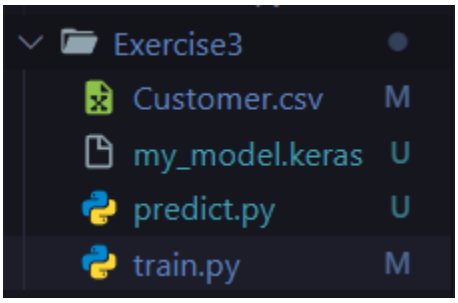
So, using this command, we train our model `train.py` based on the container image specified in the `container-image-uri` in the Google Cloud AI Platform. And finally it saves trained model in created storage bucket from step 2:

```
model.save('gs://created-bucket-name/models/my_model.keras')
```

As I said, my free trial has expired. So i will save my trained model locally:

```
model.save('my_model.keras')
```

As a result it creates trained model `my_model.ceras`:




6. **Deploy the Model**:
   - Deploy the trained model to an AI Platform endpoint.
     Use the following commands:

```
gcloud ai models create your-model \
  --region=your-region
```

This command creates a model in a selected region on Google Cloud AI Platform. Next command:

```
gcloud ai versions create v1 \
  --model=your-model \
  --origin=gs://your-bucket/model \
  --runtime-version=2.7 \
```

```
        --python-version=3.8
```

This command adds a new version to the created model. Here you add a path to your trained model in the storage bucket and specify runtime versions of tensorflow and python. And finally, when all steps are done you can test your created model.

7. **Test the Model**:
    - Use the deployed model endpoint to make predictions.

      Example `predict.py`:

```
from google.cloud import aiplatform

def predict():
    client=aiplatform.gapic.PredictionServiceClient()
endpoint =
client.endpoint_path(project='your-project',
location='your-region', endpoint='your-endpoint-id')
     instance = {'input': [/* your data */]}
     response = client.predict(endpoint=endpoint,
instances=[instance])
    print(response.predictions)

if __name__ == '__main__':
    predict()
```

      This python code makes predictions for the specified dataset using your trained model that you have deployed in the Google Cloud AI Platform. You just need to specify project id, your region and created endpoint id. And in instance specify your dataset for which you will make predictions.

      As I said, my free trial period has expired. Therefore, I will test my trained model locally. My `predict.py`:

```
Assignment 3 > Exercise3 > predict.py > ...
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

model = tf.keras.models.load_model('my_model.keras')
data = pd.read_csv('Customer.csv')

label_encoder = LabelEncoder()
data['Segment'] = label_encoder.fit_transform(data['Segment'])

input_data = pd.DataFrame({'Segment': ['Consumer']})

input_data['Segment'] = label_encoder.transform(input_data['Segment'])
scaler = StandardScaler()
input_scaled = scaler.fit_transform(input_data)

predicted_age = model.predict(input_scaled)

print(f'Predicted age: {predicted_age[0][0]}')
```

This code will predict customer age using my trained model.

3. **Deliverables**:
   - A trained machine learning model deployed on Google Cloud AI Platform.
     My free trial has expired therefore I did it locally, but I fully explained all the steps.
   - A script that makes predictions using the deployed model.


So as you can see, my `predict.py` really makes predictions based on `my_model.keras` trained model.