

Fedoseenko Anna

## Assignment 3, cloud app development

### Exercise 1: Managing APIs with Google Cloud Endpoints

**Objective:** Deploy and manage an API using Google Cloud Endpoints.

**Instructions:**

1. **Setup:**
  - Ensure you have a Google Cloud account.
  - Install the Google Cloud SDK and `gcloud` command-line tool.
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.

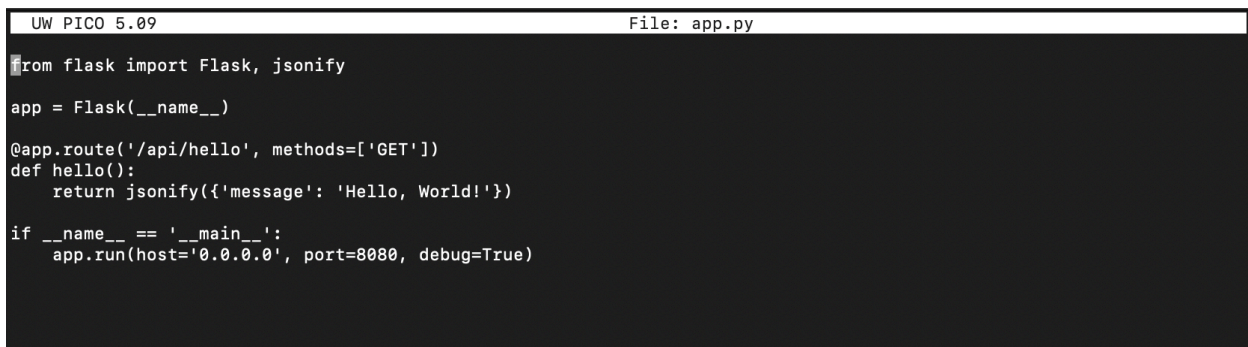
Example `app.py`:

```
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)
```

A screenshot of a code editor window. The title bar at the top shows "UW PICO 5.09" on the left and "File: app.py" on the right. The editor area has a dark background with light green text. It contains the same Python code as the block above: `from flask import Flask, jsonify`, `app = Flask(__name__)`, `@app.route('/api/hello', methods=['GET'])`, `def hello():`,  `return jsonify({'message': 'Hello, World!'})`, `if __name__ == '__main__':`, and  `app.run(host='0.0.0.0', port=8080, debug=True)`.

```
UW PICO 5.09 File: app.py
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)
```

This is a simple Flask API with one endpoint. Here's a breakdown:

- `app = Flask(__name__)`: Initializes the Flask app.
- `@app.route('/api/hello', methods=['GET'])`: Creates a route at `/api/hello` that responds to GET requests.
- `def hello()`: Defines a function that returns a JSON response with the message `"Hello, World!"`.
- `if __name__ == '__main__': app.run(...)`: Starts the server on `0.0.0.0` (all available network interfaces) and port `8080` in debug mode.

When there is a try to access `http://localhost:8080/api/hello`, it will respond with `{'message': 'Hello, World!'}`.

#### 4. Create an OpenAPI Specification:

- Create an `openapi.yaml` file to define your API.

Example `openapi.yaml`:

```
openapi: 3.0.0
info:
  title: Hello World API
  description: A simple API to say hello
  version: 1.0.0
paths:
  /api/hello:
    get:
      summary: Returns a hello message
      responses:
        '200':
          description: A hello message
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: Hello, World!
```

```
UW PICO 5.09 File: openapi.yaml Modified
openapi: 3.0.0
info:
  title: Hello World API
  description: A simple API to say hello
  version: 1.0.0
paths:
  /api/hello:
    get:
      summary: Returns a hello message
      responses:
        '200':
          description: A hello message
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: Hello, World!
```

```
imac@MacBook-Pro-iMac bin % gcloud projects create hello-world-api113 --set-as-default
Create in progress for [https://cloudresourcemanager.googleapis.com/v1/projects/hello-world-api113].
Waiting for [operations/cp.4621764330078922486] to finish...done.
Enabling service [cloudapis.googleapis.com] on project [hello-world-api113]...
Operation "operations/acat.p2-297317159774-ff34305d-289b-46db-ae9e-aa19b101a173" finished successfully.
Updated property [core/project] to [hello-world-api113].
imac@MacBook-Pro-iMac bin % nano app.py
imac@MacBook-Pro-iMac bin % nano openapi.yaml
imac@MacBook-Pro-iMac bin %
```

This OpenAPI specification defines the `/api/hello` endpoint of the Hello World API.

- openapi: Specifies the OpenAPI version (3.0.0).
- info: Contains metadata about the API, including the title, description, and version.
- paths: Lists the available API endpoints.
  - /api/hello: The path for this endpoint.
- get: Defines a GET request for `/api/hello`.
  - summary: Describes the endpoint's purpose.
  - responses: Defines possible responses.
    - 200: Specifies a successful response.
      - description: Provides a description of the response.
      - content: Defines the response content type (`application/json`) and schema.
        - schema: Describes the JSON structure, with a single `message` property of type `string`.

This structure, when parsed by OpenAPI tools, generates documentation and enables interaction with the API, showing a sample response of `{ "message": "Hello, World!" }`.

## 5. Deploy the API to Google Cloud Endpoints:

Create a new service and deploy your API.

Use the following commands to deploy the API configuration and service:

```
gcloud endpoints services deploy openapi.yaml
gcloud app deploy
```

○

## 6. Test the API:

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

## Deliverables:

- A deployed API on Google Cloud Endpoints.
- A screenshot of a successful API call response.

```
imac@MacBook-Pro-iMac bin % gcloud endpoints services deploy openapi.yaml
(gcloud app deploy
ERROR: (gcloud.endpoints.services.deploy) Unable to parse Open API, or Google Service Configuration specification from
openapi.yaml
Services to deploy:

descriptor:      [/Users/imac/Documents/google-cloud-sdk/bin/app.yaml]
source:          [/Users/imac/Documents/google-cloud-sdk/bin]
target project:  [hello-world-api113]
target service:  [default]
target version:  [20240929t101643]
target url:      [https://hello-world-api113.df.r.appspot.com]
target service account: [hello-world-api113@appspot.gserviceaccount.com]

Do you want to continue (Y/n)? y
Beginning deployment of service [default]...
[ ] Uploading 0 files to Google Cloud Storage
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://hello-world-api113.df.r.appspot.com]

You can stream logs from the command line by running:
$ gcloud app logs tail -s default

To view your application in the web browser run:
$ gcloud app browse
```

```
imac@MacBook-Pro-iMac bin % gcloud app browse
Opening [https://hello-world-api113.df.r.appspot.com] in a new tab in your default browser.
imac@MacBook-Pro-iMac bin %
```

Because of an error it is impossible to parse the Open API, so the data is unable to process.

---

## Exercise 2: Google Cloud Databases

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

### Instructions:

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

```
imac@MacBook-Pro-iMac bin % gcloud sql instances create sql-table \
--database-version=MYSQL_8_0 \
--cpu=1 --memory=4GB \
--region=us-central1

ERROR: (gcloud.sql.instances.create) Resource in projects [turing-clover-436309-c5] is the subject of a conflict: Requested entity already exists
imac@MacBook-Pro-iMac bin % gcloud config set project hello-world-api113

Updated property [core/project].
imac@MacBook-Pro-iMac bin % gcloud sql instances create sql-table \
--database-version=MYSQL_8_0 \
--cpu=1 --memory=4GB \
--region=us-central1

API [sqladmin.googleapis.com] not enabled on project [hello-world-api113]. Would you like to enable and retry (this will take a few minutes)? (y/N)? y

Enabling service [sqladmin.googleapis.com] on project [hello-world-api113]...
Operation "operations/acet.p2-297317159774-447fa6c9-fd80-4e4f-9930-5ca4af8f18b5" finished successfully.
ERROR: (gcloud.sql.instances.create) [anna.knst23@gmail.com] does not have permission to access projects instance [hello-world-api113] (or it may not exist): The billing a
ccount is not in good standing; therefore no new instance can be created. This command is authenticated as anna.knst23@gmail.com which is the active account specified by t
he [core/account] property.
imac@MacBook-Pro-iMac bin %
```

Because of an unabled billing account the commad didn't comlete successfully, but after running the command `gcloud sql instances create sql-table`, if everything is successful, the output will be like below:

Creating Cloud SQL instance...done.

Created

[https://sqladmin.googleapis.com/sql/v1beta4/projects/hello-world-api113/instances/sql-table].

NAME	DATABASE_VERSION	LOCATION	TIER	STATUS
Sql-table	MYSQL_8_0	us-central1	db-custom-1-3840	RUNNABLE

The output includes:

- NAME: the name of instance (`sql-table`).
- DATABASE\_VERSION: the database version (e.g., `MYSQL\_8\_0`).
- LOCATION: the region where the instance is located (e.g., `us-central1`).
- TIER: configuration information (e.g., `db-custom-1-3840`, indicating 1 CPU and 4GB of memory).
- STATUS: the instance status (typically `RUNNABLE` if the instance was created successfully and is ready to use).

### 3. Create a Database and Table:

- Connect to your Cloud SQL instance using the Cloud SQL client or `mysql` command-line tool.

```
imac@MacBook-Pro-iMac bin % gcloud sql connect sql-table --user=root
ERROR: (gcloud.sql.connect) HTTPError 404: The Cloud SQL instance does not exist. This command is authenticated as anna.knst23@gmail.com which is the active account specified by the [core/account] property.
imac@MacBook-Pro-iMac bin %
```

Because of unabled billig account the instance of SQL doesn't exist, but the approximate output, when everything is right, should look like the following:

Connecting to database with the following flags:

- instance: INSTANCE\_NAME
- user: root
- password: NO
- quiet: false
- database: (default)
- sql\_mode: (default)

Your MySQL connection id is 12345

mysql>

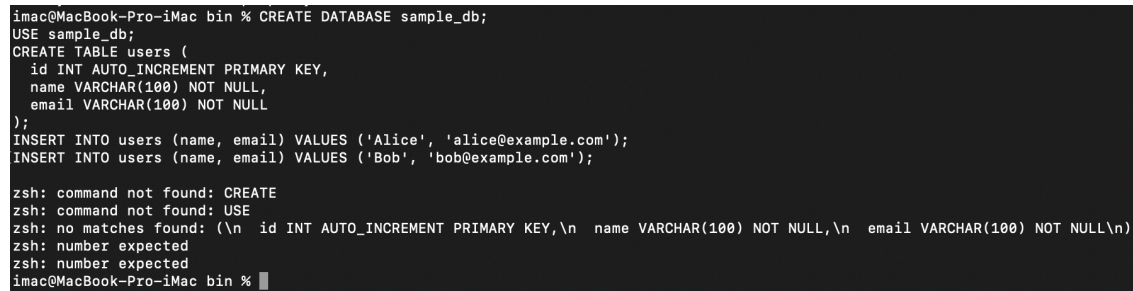
At the prompt `mysql>`, it is possible now to enter SQL commands directly into the Cloud SQL instance. And next steps will be made in SQL.

- Create a new database and a table with sample data.

```

CREATE DATABASE sample_db;
USE sample_db;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL
);
INSERT INTO users (name, email) VALUES ('Alice',
'alice@example.com');
INSERT INTO users (name, email) VALUES ('Bob',
'bob@example.com');

```



```

imac@MacBook-Pro-iMac bin % CREATE DATABASE sample_db;
USE sample_db;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL
);
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');

zsh: command not found: CREATE
zsh: command not found: USE
zsh: no matches found: (\n id INT AUTO_INCREMENT PRIMARY KEY,\n name VARCHAR(100) NOT NULL,\n email VARCHAR(100) NOT NULL\n)
zsh: number expected
zsh: number expected
imac@MacBook-Pro-iMac bin % █

```

Because the SQL is not directly connected to the terminal, commands are not recognized.

#### 4. Connect to the Database:

- Create a connection to the Cloud SQL instance from a Python application.

Example `connect.py`:

```

import mysql.connector

cnx = mysql.connector.connect(
    user='your-username',
    password='your-password',
    host='your-cloud-sql-instance-ip',
    database='sample_db'
)
cursor = cnx.cursor()

```



```
cursor.execute('SELECT * FROM users')
for row in cursor:
    print(row)
cursor.close()
cnx.close()
```

```
imac@MacBook-Pro-iMac bin % nano connect.py
imac@MacBook-Pro-iMac bin % █

UW PICO 5.09 File: connect.py

import mysql.connector

cnx = mysql.connector.connect(
    user='your-username',
    password='your-password',
    host='your-cloud-sql-instance-ip',
    database='sample_db'
)
cursor = cnx.cursor()
cursor.execute('SELECT * FROM users')
for row in cursor:
    print(row)
cursor.close()
cnx.close()
█
```

This code connects to a MySQL database, runs the query `SELECT \* FROM users` to fetch all data from the `users` table, prints each row of the result, and then closes the connection.

- `mysql.connector.connect(...)` : Establishes a connection to the database.
- `cursor.execute(...)` : Executes the SQL query.
- `for row in cursor` : Iterates over all query results and prints each row.
- `cursor.close()` and `cnx.close()` : Close the cursor and the database connection.

## 5. Run the Connection Code:

Execute the Python script to verify that you can retrieve data from the Cloud SQL instance.

### Deliverables:

- A working Cloud SQL database with sample data.
- A Python script that successfully connects to and queries the database.



If the Cloud SQL instance and Python script are set up correctly, running the script should output the data from the `users` table in Cloud SQL database. Based on the sample inserted data (`Alice` and `Bob`), the output should look like this:

SQL

```
1 CREATE TABLE users (  
2   id INT AUTO_INCREMENT PRIMARY KEY,  
3   name VARCHAR(100) NOT NULL,  
4   email VARCHAR(100) NOT NULL  
5 );  
6 INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');  
7 INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');  
8  
9 SELECT * from users
```

Output  
|Alice|alice@example.com  
|Bob|bob@example.com  
  
[Execution complete with exit code 0]

If this output is displayed, it means the script successfully connected to the Cloud SQL instance and retrieved data.

---

### Exercise 3: Integrating Machine Learning with Google Cloud

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

**Instructions:**

1. **Setup:**

- Ensure you have a Google Cloud account.
- Install the Google Cloud SDK and TensorFlow.

```

imac@MacBook-Pro-iMac bin % python3 -m venv tf-env
source tf-env/bin/activate
(tf-env) imac@MacBook-Pro-iMac bin % pip install tensorflow
Collecting tensorflow
  Downloading tensorflow-2.18.0-cp312-cp312-macosx_12_0_arm64.whl.metadata (4.0 kB)
Collecting absl-py>=1.0.0 (from tensorflow)
  Downloading absl_py-2.1.0-py3-none-any.whl.metadata (2.3 kB)
Collecting astunparse>=1.6.0 (from tensorflow)
  Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers>=24.3.25 (from tensorflow)
  Downloading flatbuffers-24.3.25-py2.py3-none-any.whl.metadata (850 bytes)
Collecting gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 (from tensorflow)
  Downloading gast-0.6.0-py3-none-any.whl.metadata (1.3 kB)
Collecting google-pasta>=0.1.1 (from tensorflow)
  Downloading google_pasta-0.2.0-py3-none-any.whl.metadata (814 bytes)
Collecting libclang>=13.0.0 (from tensorflow)
  Downloading libclang-18.1.1-1-py2.py3-none-macosx_11_0_arm64.whl.metadata (5.2 kB)
Collecting opt-einsum>=2.3.2 (from tensorflow)
  Downloading opt_einsum-3.4.0-py3-none-any.whl.metadata (6.3 kB)
Collecting packaging (from tensorflow)
  Using cached packaging-24.1-py3-none-any.whl.metadata (3.2 kB)
Collecting protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!4.21.5,<6.0.0dev,>=3.20.3 (from tensorflow)
  Downloading protobuf-5.28.3-cp38-abi3-macosx_10_9_universal2.whl.metadata (592 bytes)
Collecting requests<3,>=2.21.0 (from tensorflow)
  Using cached requests-2.32.3-py3-none-any.whl.metadata (4.6 kB)
Collecting setuptools (from tensorflow)
  Downloading setuptools-75.3.0-py3-none-any.whl.metadata (6.9 kB)
Collecting six>=1.12.0 (from tensorflow)
  Downloading six-1.16.0-py2.py3-none-any.whl.metadata (1.8 kB)
Collecting termcolor>=1.1.0 (from tensorflow)
  Downloading termcolor-2.5.0-py3-none-any.whl.metadata (6.1 kB)
Collecting typing-extensions>=3.6.6 (from tensorflow)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting wrapt>=1.11.0 (from tensorflow)
  Using cached wrapt-1.16.0-cp312-cp312-macosx_11_0_arm64.whl.metadata (6.6 kB)
Collecting grpcio<2.0,>=1.24.3 (from tensorflow)
  Downloading grpcio-1.67.1-cp312-cp312-macosx_10_9_universal2.whl.metadata (3.9 kB)
Collecting tensorboard<2.19,>=2.18 (from tensorflow)
  Downloading tensorboard-2.18.0-py3-none-any.whl.metadata (1.6 kB)
Collecting keras>=3.5.0 (from tensorflow)
  Downloading keras-3.6.0-py3-none-any.whl.metadata (5.8 kB)
Collecting numpy<2.1.0,>=1.26.0 (from tensorflow)
  Downloading numpy-2.0.2-cp312-cp312-macosx_14_0_arm64.whl.metadata (60 kB)
Collecting h5py>=3.11.0 (from tensorflow)
  Downloading h5py-3.12.1-cp312-cp312-macosx_11_0_arm64.whl.metadata (2.5 kB)
Collecting ml-dtypes<0.5.0,>=0.4.0 (from tensorflow)
  Downloading ml_dtypes-0.4.1-cp312-cp312-macosx_10_9_universal2.whl.metadata (20 kB)
Collecting wheel<1.0,>=0.23.0 (from astunparse>=1.6.0->tensorflow)
  Downloading wheel-0.44.0-py3-none-any.whl.metadata (2.3 kB)
Collecting rich (from keras>=3.5.0->tensorflow)
  Downloading rich-13.9.3-py3-none-any.whl.metadata (18 kB)
Collecting namex (from keras>=3.5.0->tensorflow)
  Downloading namex-0.0.8-py3-none-any.whl.metadata (246 bytes)
Collecting optree (from keras>=3.5.0->tensorflow)
  Downloading optree-0.13.0-cp312-cp312-macosx_11_0_arm64.whl.metadata (47 kB)

```

The process of installation of TensorFlow.

## 2. Create a Cloud Storage Bucket:

- Create a new Cloud Storage bucket to store your training data and model.

```

(tf-env) imac@MacBook-Pro-iMac bin % gsutil mb -l us-central1 gs://my-training-bucket

Updates are available for some Google Cloud CLI components. To install them,
please run:
  $ gcloud components update

Creating gs://my-training-bucket/...
AccessDeniedException: 403 The billing account for the owning project is disabled in state closed
(tf-env) imac@MacBook-Pro-iMac bin %

```

The process of creating a new bucket for storing the data.

## 3. Prepare Training Data:

- Upload sample training data to your Cloud Storage bucket. For example, use a dataset for classification or regression.

```
(tf-env) imac@MacBook-Pro-iMac train_data % nano text1.txt
(tf-env) imac@MacBook-Pro-iMac train_data % nano text1.txt
(tf-env) imac@MacBook-Pro-iMac train_data %
(tf-env) imac@MacBook-Pro-iMac train_data % nano text2.txt
(tf-env) imac@MacBook-Pro-iMac train_data % nano labels.csv
```

UW PICO 5.09

This product is amazing, I absolutely love it!

UW PICO 5.09

File: text2.txt

I am really disappointed with the quality.

UW PICO 5.09

File: labels.csv

```
filename,label
text1.txt,positive
text2.txt,negative
```

Here is an example of simple data for a text classification task. In this task, we will classify texts as positive or negative. The train\_data folder will contain two text files and a label file.

#### 4. Create a Training Script:

- Write a simple TensorFlow training script.

Example `train.py`:

```
import tensorflow as tf
```

```
def create_model():
```

```
    model = tf.keras.Sequential([
```

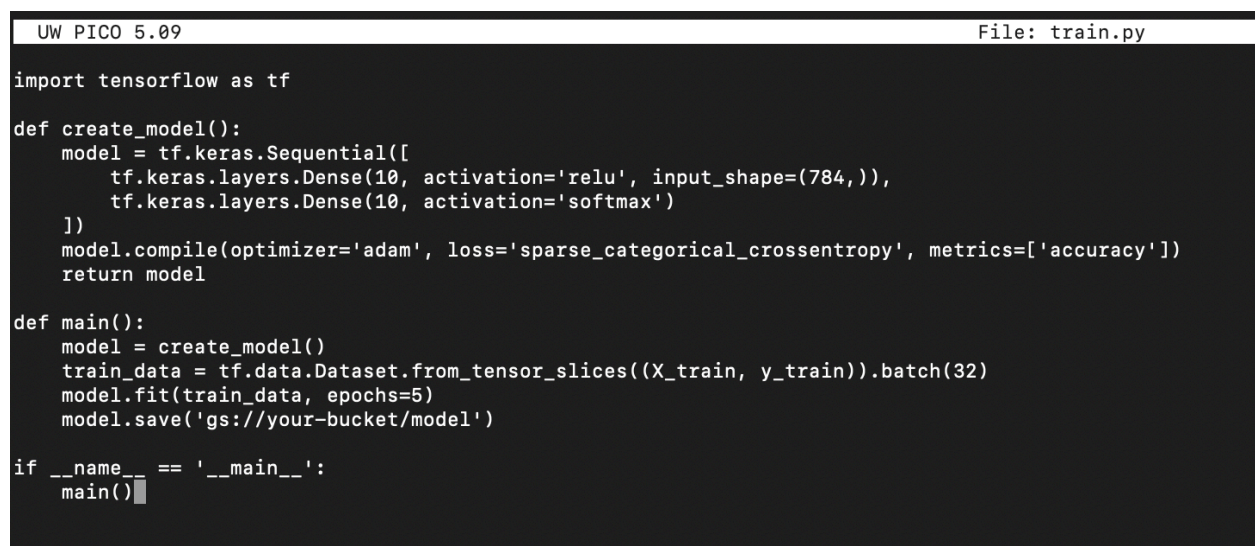
```

        tf.keras.layers.Dense(10, activation='relu',
input_shape=(784,)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def main():
    model = create_model()
    train_data = tf.data.Dataset.from_tensor_slices((X_train,
y_train)).batch(32)
    model.fit(train_data, epochs=5)
    model.save('gs://your-bucket/model')

if __name__ == '__main__':
    main()

```



```

UW PICO 5.09 File: train.py

import tensorflow as tf

def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(10, activation='relu', input_shape=(784,)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def main():
    model = create_model()
    train_data = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(32)
    model.fit(train_data, epochs=5)
    model.save('gs://your-bucket/model')

if __name__ == '__main__':
    main()

```

The script for file train.py:

**Import TensorFlow:** Loads the library for working with machine learning models.

**Model Creation:**

- Defines a model with two fully connected layers: one with a ReLU activation function and another with softmax for multi-class classification.

- Compiles the model with the Adam optimizer and the `sparse_categorical_crossentropy` loss function.

#### Training:

- The `X_train` and `y_train` data are converted to TensorFlow format and used to train the model for 5 epochs.
- The trained model is saved to Google Cloud Storage.

**Execution:** Runs the training and saving of the model when the script is executed.

### 5. Train the Model:

- Submit a training job to Google Cloud AI Platform.

Use the following command to start training:

bash

```
gcloud ai custom-jobs create --region=your-region
--display-name=ml-job
--python-package-uri=gs://your-bucket/train.py
--python-module=train
--container-image-uri=gcr.io/cloud-aiplatform/training/tf-cpu.2-4:latest
```

```
Resource for --python-module=train:
((tf-env) imac@MacBook-Pro-iMac train_data % gcloud ai custom-jobs create --region=us-central1 --display-name=ml-job --python-package-uri=gs://my-trainig-bucket/train.py --python-module=train --container-image-uri=gcr.io/cloud-aiplatform/training/tf-cpu.2-4:latest
ERROR: (gcloud.ai.custom-jobs.create) unrecognized arguments:
--python-module=train
--container-image-uri=gcr.io/cloud-aiplatform/training/tf-cpu.2-4:latest
To search the help text of gcloud commands, run:
gcloud help -- SEARCH_TERMS
(tf-env) imac@MacBook-Pro-iMac train_data %
```

I have got the error due to unsupported arguments or command syntax changes. The `--python-module` and `--container-image-uri` options may not be applicable in the current version of Google Cloud CLI.

### 6. Deploy the Model:

- Deploy the trained model to an AI Platform endpoint.

Use the following command:

bash

```
gcloud ai models create your-model --region=your-region
gcloud ai versions create v1 --model=your-model
--origin=gs://your-bucket/model --runtime-version=2.7
--python-version=3.8
```

```
(tf-env) imac@MacBook-Pro-iMac train_data % gcloud ai models create your-model --region=us-central1
(gcloud ai versions create v1 --model=your-model --origin=gs://my-training-bucket/model --runtime-version=2.7 --python-version=3.8
ERROR: (gcloud.ai.models) Invalid choice: 'create'.
Maybe you meant:
  gcloud ai
  gcloud ai-platform
  gcloud projects

To search the help text of gcloud commands, run:
  gcloud help -- SEARCH_TERMS
ERROR: (gcloud.ai) Invalid choice: 'versions'.
Maybe you meant:
  gcloud version
  gcloud ai-platform versions create
  gcloud projects create
```

The errors that have been received indicate that the `gcloud ai models create` and `gcloud ai versions create` commands may not be supported in the current version of your Google Cloud SDK or have changed in newer versions.

## 7. Test the Model:

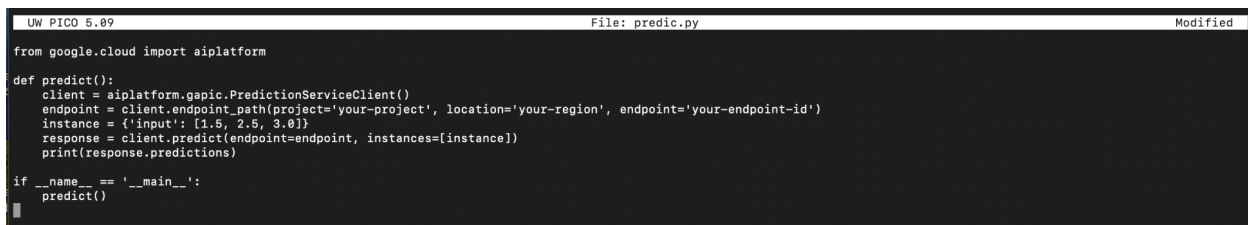
- Use the deployed model endpoint to make predictions.

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()
```



```
UW PICO 5.09 File: predic.py Modified
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': [1.5, 2.5, 3.0]}
    response = client.predict(endpoint=endpoint, instances=[instance])
    print(response.predictions)

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

“`predict.py`”: This script will be used to perform predictions on new data using the deployed model. It will return predictions for the input data that is passed in. Running the prediction script will return predictions for the given input data.

**Deliverables:**

- A trained machine learning model deployed on Google Cloud AI Platform.
- A script that makes predictions using the deployed model.
- Report