Experiment – 6 in Cloud Computing (PVP23)

A basic Flask application for Google App Engine (Standard environment) requires a root directory containing at least three files: **`main.py`**, **`app.yaml`**, and **`requirements.txt`**. For serving web pages and static assets, the standard Flask structure with `templates/` and `static/` folders is used.

Recommended Project Structure

```
your-project-folder/
├── main.py
├── app.yaml
├── requirements.txt
├── static/
│   ├── css/
│   ├── js/
│   └── img/
└── templates/
    ├── index.html
    └── base.html
```

Essential Files and Directories

main.py

- The application's entry point. App Engine expects to find a WSGI-compatible `app` object in this file by default. This file contains your core Flask application logic and routing.

- **Code Snippet (`main.py`)**:

```python
from flask import Flask, render_template
app = Flask(__name__)
@app.route("/")
def home():
    return render_template("index.html")

if __name__ == "__main__":
    # Used for local development
    # app.run(host="0.0.0.0", port=8080, debug=True)
    app.run(host="127.0.0.1", port=8080, debug=True)
```

app.yaml

The configuration file for Google App Engine. It tells App Engine which runtime to use and how to handle requests.

- **Code Snippet (`app.yaml`)**:

```yaml
runtime: python312 # Use a supported Python version, e.g., python312
instance_class: F1 # Smallest instance class, good for simple apps
automatic_scaling:
  min_idle_instances: 0 # Scales to zero when idle to save costs
```

- For serving static files efficiently, you can add handlers:

```yaml
handlers:
- url: /static
  static_dir: static/
- url: /.*
  script: auto # Routes everything else to the Flask app
```

requirements.txt

- This file lists all the Python dependencies your application needs. App Engine reads this file during deployment and installs the listed packages.

- **Code Snippet (`requirements.txt`)**:

```
Flask
# Add other dependencies like requests, PyMySQL, etc..
```

- **`templates/`**: A standard Flask directory where HTML templates (using Jinja2) are stored. The `render_template` function looks for files in this folder.

- **`static/`**: A standard Flask directory for static assets like CSS files, JavaScript files, and images.

Deployment

Once the structure is in place and the Google Cloud SDK is installed and configured, you can deploy your application by navigating to your project's root directory in the terminal and running the following command:

bash

```
gcloud app deploy
```

Installing the Google Cloud CLI

To install the Google Cloud CLI (formerly Cloud SDK) on Windows, the recommended method is to use the interactive installer, which simplifies the process of setting up prerequisites like Python.

Step-by-Step Installation Guide

1. **Download the installer:**

- Navigate to the official Google Cloud CLI download page.
- Download the installer file (`GoogleCloudCliInstaller.exe`) for your version of Windows (64-bit is most common).

2. **Run the installer:**

- Locate the downloaded file and double-click to run it.
- If prompted by User Account Control, click **Yes** to allow the application to make changes.

3. **Follow the installation wizard:**

- **Welcome Screen:** Click **Next** to begin the installation.
- **License Agreement:** Read and **accept** the license agreement.
- **Installation Location:** Choose an installation directory, or leave the default path as is.
- **Python Prerequisite:** The installer can bundle Python 3 automatically. It is recommended to leave the option to install the bundled Python checked, as the gcloud CLI requires a supported Python version (3.10-3.14).
- **Components and Shortcuts (Optional):** You can optionally choose to install beta commands or create desktop/start menu shortcuts.

4. **Complete the installation:**

- Click **Install** and wait for the process to complete. This may take a few minutes.
- Once the installation is complete, click **Next**, then **Finish**. By default, the installer will automatically open a terminal window and run `gcloud init` to configure your setup.

Configuration and Initialization

After the physical installation is complete, you need to configure the SDK to link it with your Google Cloud account and project.

1. **Run `gcloud init`:**

- If the installer didn't automatically run it, open a new command prompt or the **Google Cloud CLI Shell** from your Start menu and type `gcloud init`.

2. **Authorize your account:**

- The command prompt will ask you to log in. It will open a web browser to a Google sign-in page.
- Sign in with your Google account and grant the necessary permissions to access your Google Cloud resources.
- A message in the browser will confirm successful authentication, and you can close the window.

3. **Configure project and region:**

- Back in the command prompt, you will be prompted to select a Google Cloud project from a list of your available projects.
- You may also be asked to configure a default Compute Engine region and zone (this is optional but recommended).

Verification

To verify the installation, open a new terminal window and run:

```
gcloud version
```

This command should display the installed gcloud CLI version and other component details, confirming a successful installation. You can now use the `gcloud`, `gsutil`, and `bq` command-line tools to interact with Google Cloud services.

Execution

To execute a basic Flask application on Google App Engine (GAE), you need to set up your project files, test the application locally, configure a Google Cloud project, and then deploy using the `gcloud` CLI.

Prerequisites

- A Google Cloud project with billing enabled.
- The Google Cloud SDK (gcloud CLI) installed and initialized on your local machine (`gcloud init`).
- Python 3.8 or later installed locally.

Step-by-Step Guide

1. Create Your Flask Application Files

Create a project directory and add the following three files in the root folder:

- **`main.py`**: The Python application file.
- **`requirements.txt`**: A list of Python packages the application needs.
- **`app.yaml`**: The configuration file for Google App Engine.

`main.py`

This file contains your Flask application code. Google App Engine's standard environment expects an `app` variable in `main.py` by default:

python

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello() -> str:
    """Return a friendly HTTP greeting."""
    return "Hello World! This is a Flask app on Google App Engine."

if __name__ == "__main__":
    # This is used when running locally only. When deploying to Google App
    # Engine, a webserver process such as
```

python

```
Gunicorn
```

python

```python
 will serve the app.
    app.run(host="127.0.0.1", port=8080, debug=True)
```

`requirements.txt`

This file lists the Python packages required for your app to run:

```
Flask
```

You can create this file automatically by running `pip freeze > requirements.txt` in your activated virtual environment.

`app.yaml`

This file tells App Engine how to run your application. For a basic Python 3 app in the standard environment, you only need to specify the runtime:

yaml

```yaml
runtime: python3
```

2. Test the Application Locally

Run your application locally to ensure it works before deploying:

1. **Install dependencies** in a virtual environment:

   bash

   ```bash
   python3 -m venv env
   source env/bin/activate  # On Windows use `.\env\Scripts\activate`
   pip install -r requirements.txt
   ```

2. **Run the Flask app**:

   bash

   ```bash
   python main.py
   ```

3. **View the app** by opening `http://127.0.0.1:8080` in your web browser.

3. Deploy to Google App Engine

Once the app works locally, you can deploy it to GAE:

1. **Ensure you are authenticated** with the Google Cloud CLI:

   bash

   ```bash
   gcloud auth login
   ```

2. **Set the project ID** if not already set:

   bash

   ```bash
   gcloud config set project YOUR_PROJECT_ID
   ```

3. **Deploy your application** from your project's root directory:

   bash

   ```bash
   gcloud app deploy
   ```

4. **Confirm the deployment** when prompted. The process will upload your files, build a container, and provision the necessary infrastructure.
5. **View your live application** in a web browser using the command:

   bash

   ```
   gcloud app browse
   ```

   This command will output the URL of your deployed service (e.g., `https://PROJECT_ID.REGION_ID.r.appspot.com`).