Python (https://cloud.google.com/python/) Guides

Running Django in the App Engine flexible environment

Django apps that run on the App Engine flexible environment (https://cloud.google.com/appengine/docs/flexible/) are running on the same infrastructure that powers all of Google's products, which generally improves scalability.

This tutorial assumes you're familiar with Django web development. If you're new to Django development, it's a good idea to work through writing your first Django app (https://docs.djangoproject.com/en/1.9/intro/tutorial01/) before continuing. In that tutorial, the app's models represent polls that contain questions, and you can interact with the models by using the Django admin console.

This tutorial requires Python 3.4 or later (https://www.python.org/).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Google Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).
4. Install and initialize the Cloud SDK (https://cloud.google.com/sdk/docs/).
5. Enable the Datastore, Pub/Sub, Cloud Storage JSON, Stackdriver Logging, and Google+ APIs.

   ENABLE THE APIS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=DATASTOR

Log in to gcloud

Acquire new credentials to use the Cloud SQL Admin API:

```
gcloud auth application-default login
```

Downloading and running the app

After you've completed the prerequisites, download and deploy the Django sample app. The following sections guide you through configuring, running, and deploying the app.

Cloning the Django app

The code for the Django sample app is in the `GoogleCloudPlatform/python-docs-samples` (https://github.com/GoogleCloudPlatform/python-docs-samples) repository on GitHub.

1. You can either download the sample (https://github.com/GoogleCloudPlatform/python-docs-samples/archive/master.zip) as a zip file and extract it or clone the repository to your local machine:

   ```
   git clone https://github.com/GoogleCloudPlatform/python-docs-samples.git
   ```

2. Go to the directory that contains the sample code:

   LINUX/MACOS | WINDOWS

   ```
   cd python-docs-samples/appengine/flexible/django_cloudsql
   ```

Setting up your local environment

When deployed, your app uses the Cloud SQL Proxy that is built in to the App Engine environment to communicate with your Cloud SQL instance. However, to test your app locally, you must install and use a local copy of the proxy in your development environment.

Learn more about the Cloud SQL Proxy (https://cloud.google.com/sql/docs/postgres/sql-proxy).

To perform basic admin tasks on your Cloud SQL instance, you can use the PostgreSQL client.

Note: You must authenticate `gcloud` (https://cloud.google.com/sql/docs/postgres/sql-proxy#gcloud) before you can use the Cloud SQL Proxy to connect from your local machine.

Enable the Cloud SQL Admin API

Before using Cloud SQL, you must enable the Cloud SQL Admin API:

```
gcloud services enable sqladmin
```

Installing the Cloud SQL Proxy

Download and install the Cloud SQL Proxy. The Cloud SQL Proxy connects to your Cloud SQL instance when running locally.

LINUX 64-BIT | LINUX 32-BIT | MORE

1. Download the proxy:

```
wget https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64 -O cloud_sql_
```

2. Make the proxy executable:

```
chmod +x cloud_sql_proxy
```

If your operating system isn't included here, you can also compile the proxy from source (http://github.com/GoogleCloudPlatform/cloudsql-proxy).

Creating a Cloud SQL instance

1. Create a Cloud SQL for PostgreSQL instance. (https://cloud.google.com/sql/docs/postgres/create-instance)

   Name the instance **`polls-instance`** or similar. It can take a few minutes for the instance to be ready. When the instance is ready, it's visible in the instances list.

2. Use the Cloud SDK to run the following command where **`[YOUR_INSTANCE_NAME]`** represents the name of your Cloud SQL instance:

   ```
   gcloud sql instances describe [YOUR_INSTANCE_NAME]
   ```

   In the output, note the value shown for **`[CONNECTION_NAME]`**.

   The **`[CONNECTION_NAME]`** value is in the format **`[PROJECT_NAME]:[REGION_NAME]:[INSTANCE_NAME]`**.

Initializing your Cloud SQL instance

1. Start the Cloud SQL Proxy by using the **`[CONNECTION_NAME]`** value from the previous step:

   LINUX/MACOS | WINDOWS

   ```
   ./cloud_sql_proxy -instances="[YOUR_INSTANCE_CONNECTION_NAME]"=tcp:5432
   ```

   Replace **`[YOUR_INSTANCE_CONNECTION_NAME]`** with the **`[CONNECTION_NAME]`** value that you recorded in the previous step.

   This step establishes a connection from your local computer to your Cloud SQL instance for local testing purposes. Keep the Cloud SQL Proxy running the entire time you test your app locally.

2. Create a Cloud SQL user and database:

   CLOUD CONSOLE | POSTGRES CLIENT

   a. Create a new database by using the Cloud Console (https://cloud.google.com/sql/docs/postgres/create-manage-databases#create) for your Cloud SQL instance **`polls-instance`**. For example, you can use the name **`polls`**.

b. Create a new user by using the Cloud Console (https://cloud.google.com/sql/docs/postgres/create-manage-users#creating) for your Cloud SQL instance `polls-instance`.

Configuring the database settings

1. Open `mysite/settings.py` for editing.
    a. To use MySQL instead of PostgreSQL:
        i. Follow the instructions in the file comments to enable the MySQL database driver.
        ii. Open `requirements.txt` for editing and follow the instructions to add the MySQL database driver to your dependencies.
    b. To help set up the connection to the database for both App Engine deployment and local testing, set `<your-database-user>` and `<your-database-password>` to the username and password you created previously in the step Creating a Cloud SQL instance (#creating_a_cloud_sql_instance)
    c. Get the values for your instance:

```
gcloud sql instances describe [YOUR_INSTANCE_NAME]
```

    d. From the output, copy the `connectionName` value for use in the next step.
    e. Set `<your-cloudsql-connection-string>` to the `connectionName` from the previous step.
2. Close and save `settings.py`.

Running the app on your local computer

1. To run the Django app on your local computer, set up a Python development environment (https://cloud.google.com/python/setup), including Python, `pip`, and `virtualenv`.
2. Create an isolated Python environment and install dependencies:

LINUX/MACOS WINDOWS

```
virtualenv env
source env/bin/activate
pip install -r requirements.txt
```

3. Run the Django migrations to set up your models:

```
python manage.py makemigrations
python manage.py makemigrations polls
python manage.py migrate
```

4. Start a local web server:

```
python manage.py runserver
```

5. In your browser, go to http://localhost:8000 (http://localhost:8000).

```
http://localhost:8000
```

The page displays the following text: "Hello, world. You're at the polls index." The Django web server running on your computer delivers the sample app pages.

6. Press `Control+C` to stop the local web server.

Using the Django admin console

1. Create a superuser. You need to define a username and password.

```
python manage.py createsuperuser
```

2. Start a local web server:

```
python manage.py runserver
```

3. In your browser, go to http://localhost:8000/admin (http://localhost:8000/admin).

```
http://localhost:8000/admin
```

4. Log in to the admin site using the username and password you used when you ran `createsuperuser`.

Deploying the app to the App Engine flexible environment

When the app is deployed to Google Cloud, it uses the Gunicorn server. Gunicorn doesn't serve static content, so the app uses Cloud Storage to serve static content.

1. Create a Cloud Storage bucket:
    a. Create a Cloud Storage bucket and make it publicly readable. Replace `<your-gcs-bucket>` with a bucket name of your choice. For example, you could use your project ID as a bucket name.

    ```
    gsutil mb gs://<your-gcs-bucket>
    gsutil defacl set public-read gs://<your-gcs-bucket>
    ```

    b. Gather all the static content locally into one folder:

    ```
    python manage.py collectstatic
    ```

    c. Upload the static content to Cloud Storage:

    ```
    gsutil rsync -R static/ gs://<your-gcs-bucket>/static
    ```

2. Edit settings.py:
    a. Open `mysite/settings.py` for editing.
    b. Set the value of `STATIC_URL` to the following URL. Replace `<your-gcs-bucket>` with the bucket name you created earlier.

    ```
    https://storage.googleapis.com/<your-gcs-bucket>/static/
    ```

    c. Close and save `settings.py`.
3. Edit app.yaml:
    a. Open `app.yaml` for editing.
    b. Run the following command from the command line:

```
gcloud sql instances describe [YOUR_INSTANCE_NAME]
```

c. From the output, copy the `connectionName` value for use in the next step.

d. Replace `<your-cloudsql-connection-string>` with `connectionName` from the previous step.

e. Close and save `app.yaml`.

4. Deploy the sample:

```
gcloud app deploy
```

Wait for the message that notifies you that the update has completed.

Seeing the app run in Google Cloud

The following command deploys the app as described in `app.yaml` and sets the newly deployed version as the default version, causing it to serve all new traffic.

- In your browser, enter the following address. Replace `<your-project-id>` with your Google Cloud project ID.

```
https://<your-project-id>.appspot.com
```

Your request is served by a web server running in the App Engine flexible environment.

As the app deploys, you might see several repeated messages while the platform checks whether the app is serving. This is normal. Wait for the message that notifies you that the update of the app is complete.

If you update your app, you deploy the updated version by entering the same command that you used to deploy the app. The deployment creates a new version (https://console.cloud.google.com/project/_/appengine/versions) of your app and promotes it to the default version. The earlier versions of your app remain, as do their associated virtual machine (VM) instances. All of these app versions and VM instances are billable resources. To reduce costs, delete the non-default versions of your app.

For information about deleting the non-default versions of your app or stopping your VM instances, see Cleaning up (https://cloud.google.com/python/getting-started/delete-tutorial-resources).

Production

When you are ready to serve your content in production, in `mysite/settings.py`, change the `DEBUG` variable to `False`.

Understanding the code

The Django sample app was created by using standard Django tooling.

- The following commands create the project and the polls app:

```
django-admin startproject mysite

python manage.py startapp polls
```

- The `settings.py` file contains the configuration for your SQL database:

[appengine/flexible/django_cloudsql/mysite/settings.py](https://github.com/GoogleCloudPlatform/python-docs-samples/blob/a07a09a2cdbcde938e58b9764708931d0fa6798e/appengine/flexible/django_cloudsql/mysite/settings.py) (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/a07a09a2cdbcde938e58b9764708931d0fa6798e/appengine/flexible/django_cloudsq

DE938E58B9764708931D0FA6798E/APPENGINE/FLEXIBLE/DJANGO_CLOUDSQL/MYSITE/SETTINGS.PY)

```
DATABASES = {
    'default': {
        # If you are using Cloud SQL for MySQL rather than PostgreSQL, set
        # 'ENGINE': 'django.db.backends.mysql' instead of the following.
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'polls',
        'USER': '<your-database-user>',
        'PASSWORD': '<your-database-password>',
        # For MySQL, set 'PORT': '3306' instead of the following. Any Cloud
        # SQL Proxy instances running locally must also be set to tcp:3306.
        'PORT': '5432',
    }
}
# In the flexible environment, you connect to CloudSQL using a unix socket.
# Locally, you can use the CloudSQL proxy to proxy a localhost connection
# to the instance
DATABASES['default']['HOST'] = '/cloudsql/<your-cloudsql-connection-string>'
if os.getenv('GAE_INSTANCE'):
    pass
```

```
else:
    DATABASES['default']['HOST'] = '127.0.0.1'
```

- To specify how the app serves static content, in the `settings.py` file, set the value of `STATIC_URL`:

appengine/flexible/django_cloudsql/mysite/settings.py (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/a07a09a2cdbcde938e58b9764708931d0fa6798e/appengine/flexible/django_cloudsq

DE938E58B9764708931D0FA6798E/APPENGINE/FLEXIBLE/DJANGO_CLOUDSQL/MYSITE/SETTINGS.PY)

```
# Fill in your cloud bucket and switch which one of the following 2 lines
# is commented to serve static content from GCS
# STATIC_URL = 'https://storage.googleapis.com/<your-gcs-bucket>/static/'
STATIC_URL = '/static/'
```

- The `app.yaml` (https://cloud.google.com/appengine/docs/python/config/appconfig?hl=en) file contains configuration information for deployment to the flexible environment:

appengine/flexible/django_cloudsql/app.yaml (https://github.com/GoogleCloudPlatform/python-docs-samples/blob/a07a09a2cdbcde938e58b9764708931d0fa6798e/appengine/flexible/django_cloudsq

A09A2CDBCDE938E58B9764708931D0FA6798E/APPENGINE/FLEXIBLE/DJANGO_CLOUDSQL/APP.YAML)

```
runtime: python
env: flex
entrypoint: gunicorn -b :$PORT mysite.wsgi

beta_settings:
    cloud_sql_instances: <your-cloudsql-connection-string>

runtime_config:
  python_version: 3
```


Last updated December 5, 2019.