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

Logging app events with Python

This part of the Python Bookshelf app tutorial shows how an app can incorporate detailed logging to help with detecting, debugging, and monitoring potential issues. Logging app events can help you identify issues and solve problems, both during development and after your app is in production.

This page is part of a multipage tutorial. To start from the beginning and read the setup instructions, go to Python Bookshelf app (https://cloud.google.com/python/getting-started/tutorial-app).

You can use the Python standard logging handlers introduced in Understanding the code (#understanding_the_code) on the App Engine flexible environment, Compute Engine, or Google Kubernetes Engine (GKE).

Configuring settings

This section uses code in the `5-logging` directory. Edit the files and run commands in this directory.

1. Open the `config.py` file for editing and replace the following values:
    - Set the value of `[PROJECT_ID]` to your project ID, which is visible in the Cloud Console.
    - Set the value of `[DATA_BACKEND]` to the same value you used during the Using structured data (https://cloud.google.com/python/getting-started/using-structured-data) tutorial.
    - If you are using Cloud SQL or MongoDB, set the values under the `Cloud SQL` or `Mongo` section to the same values you used during the *Using structured data* step.

- Set the value of `[CLOUD_STORAGE_BUCKET]` to your Cloud Storage bucket name.
- Under the `OAuth2 configuration` section, set the values of `[GOOGLE_OAUTH2_CLIENT_ID]` and `[GOOGLE_OAUTH2_CLIENT_SECRET]` to the application client ID and secret that you created previously.

2. Save and close the `config.py` file.

If you are using Cloud SQL:

1. Open the `app.yaml` file for editing.
2. Set the value of `cloudsql-instance` to the same value used for `[CLOUDSQL_CONNECTION_NAME]` in the `config.py` file. Use the format `project:region:cloudsql-instance`. Uncomment this entire line.
3. Save and close the `app.yaml` file.

Installing dependencies

To create a virtual environment and install dependencies, use the following commands:

LINUX/MACOS WINDOWS

```
virtualenv -p python3 env
source env/bin/activate
pip install -r requirements.txt
```

Running the app on your local machine

1. Start a local web server:

```
python main.py
```

2. In your browser, enter the following address:

```
http://localhost:8080
```

Press `Control+C` to exit the worker and then the local web server.

Deploying the app to the App Engine flexible environment

1. Deploy the sample app:

```
gcloud app deploy
```

2. In your browser, enter the following address. Replace `[YOUR_PROJECT_ID]` with your Google Cloud project ID:

```
https://[YOUR_PROJECT_ID].appspot.com
```

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 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.

To delete an app version:

1. In the Cloud Console, go to the **Versions** page for App Engine.

   GO TO THE VERSIONS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS)

2. Select the checkbox for the non-default app version you want to delete.

   Note: The only way you can delete the default version of your App Engine app is by deleting your project. However, you can stop the default version in the Cloud Console (https://console.cloud.google.com/appengine/versions). This action shuts down all instances associated with the version. You can restart these instances later if needed.

   In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.

3. Click **Delete** to delete the app version.

For more information about cleaning up billable resources, see the Cleaning up (https://cloud.google.com/python/getting-started/using-pub-sub#clean-up) section in the final step of this tutorial.

Understanding the code

After you deploy an app, it's important to understand how well the app is working. Google Cloud provides logging and monitoring tools in the Google Cloud Console that show detailed activity within your app, helping you quickly identify critical issues or trends.

The Bookshelf sample app uses Python's Logging facility for Python (https://docs.python.org/2/library/logging.html) to manage logging.

The app can also log important events from anywhere by using the following code.

```
logging.info("Something happened");
logging.error("Something bad happened");
```

When the sample app is running in the App Engine flexible environment, anything logged to `stderr` and `stdout` is automatically collected by Stackdriver Logging. Use the logs viewer in the Cloud Console to view, search, and export log content.

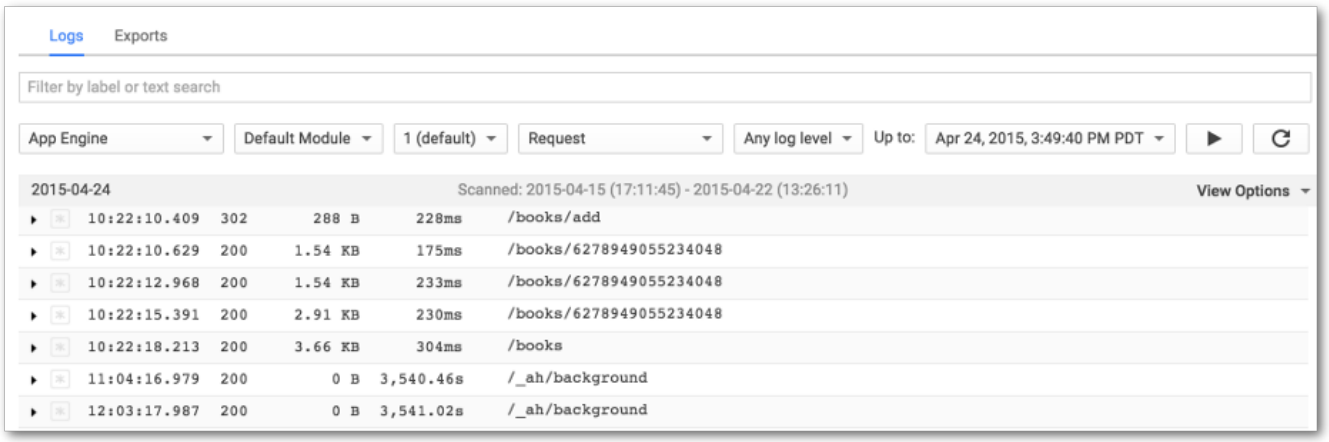
GO TO THE LOGS VIEWER IN THE CLOUD CONSOLE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECT/_

Viewing logs

While the sample app runs, it collects logging data. You can analyze this data by using the log monitoring tools in the Cloud Console.

GO TO THE LOG MONITORING TOOLS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECT/_/LOGS)

The following image shows logged events in the Cloud Console.



For a more detailed analysis, you can stream or import the app's logs into BigQuery (https://cloud.google.com/logging/docs/install/logs_export) or export them to a Cloud Storage bucket (https://cloud.google.com/logging/docs/install/logs_export). You can use the Cloud Console to do both.

Using Python logging handlers

By default, the log messages captured in `stdout` and `stderr` aren't captured with severity metadata. If you query for log statements that contain `WARN`, the results set includes statements that contain the string `WARN`, regardless of the severity of the logging call. However, if metadata is correctly reported, you can use a `metadata.severity >= WARN` logging filter to return only log statements that are logged at `WARN` or higher.

The Cloud Client Libraries for Python (https://github.com/GoogleCloudPlatform/google-cloud-python) include Python standard logging handlers (https://docs.python.org/2/library/logging.handlers.html), that ensure that all `logging` statements are reported to Logging with the correct metadata. The following code attaches the default Logging handlers, which vary depending on your deployment environment, to the root logger. In the App Engine flexible environment, Logging reports logs by using the `fluentd` agent that accompanies the runtime. For more information, see the Python Stackdriver Logging API client library (https://pypi.python.org/pypi/google-cloud-logging).

5-logging/bookshelf/__init__.py (https://github.com/GoogleCloudPlatform/getting-started-python/blob/504b3d550b551502cfe96f32542c31b232135eff/5-logging/bookshelf/__init__.py)

YTHON/BLOB/504B3D550B551502CFE96F32542C31B232135EFF/5-LOGGING/BOOKSHELF/__INIT__.PY)

```
if not app.testing:
    client = google.cloud.logging.Client(app.config['PROJECT_ID'])
    # Attaches a Google Stackdriver logging handler to the root logger
    client.setup_logging(logging.INFO)
```

< PREV (HTTPS://CLOUD.GOOGLE.COM/PYTHON/MONITOR-AND-DEBUG/INDEX)

NEXT > (HTTPS://CLOUD.GOOGLE.COM/PYTHON/MONITOR-AND-DEBUG/UPTIME-ALERT)

Last updated December 5, 2019.