Alpha

This library is covered by the Pre-GA Offerings Terms of the Google Cloud Platform Terms of Service. Pre-GA libraries may have limited support, and changes to pre-GA libraries may not be compatible with other pre-GA versions. For more information, see the launch stage descriptions.

Setting Up Cloud Debugger for Python

Overview

This page describes how to configure your environment and your Python application to use Cloud Debugger. For some environments, you must explicitly specify the access scope to let the Cloud Debugger agent send data. We recommend setting the broadest possible access scope and then using Identity and Access Management (/iam/docs/overview) to restrict access. In keeping with this best practice, set the access scope to be all Cloud APIs with the option `cloud-platform`.

Language versions and compute environments

Cloud Debugger is available for Python 3 on the following compute environments:

App Engine Standard environment (/debugger/docs/setup/python#gae-standard)	App Engine Flexible environment (/debugger/docs/setup/python#gae-flex)	Compute Engine (/debugger/docs/setup/python
✓	✓	✓

Setting up Cloud Debugger

To set up Cloud Debugger, complete the following tasks:

1. Verify the Cloud Debugger API is enabled for your project.

2. Install and configure the Debugger on the compute environment you're using.
3. Select your source code.

Verifying the Cloud Debugger API is enabled

To begin using Cloud Debugger, ensure that the Cloud Debugger API is enabled. Cloud Debugger is enabled by default for most projects.

Enable Cloud Debugger API (https://console.cloud.google.com/apis/api/clouddebugger.googleapis.com/ov

Cloud Debugger is free to use. However, you need a Google Cloud project urce-manager/docs/creating-managing-projects) with billing enabled before you can use Cloud Debugger.

Canary snapshots and logpoints

The Debugger agent for Python can use canary snapshots and logpoints every time you set a snapshot or logpoint.

The Debugger agent canaries snapshots and logpoints to protect large jobs from any potential bug in the Debugger agent which can take the entire job down when a snapshot or a logpoint is applied.

To mitigate this, Debugger canaries snapshots and logpoints on a subset of your running instances each time they are set. After Debugger verifies the snapshot or logpoint does not adversely affect your running instances, Debugger then applies the snapshot or logpoint to all instances.

Canarying is available for the Python Debugger agent version 2.15 and later.

To learn how to use Debugger in canary mode, go to the the Debug snapshots (/debugger/docs/using/snapshots) and Debug logpoints (/debugger/docs/using/logpoints) pages.

Enabling canary snapshots and logpoints

When you install the latest version of the Debugger agent, you have the option to enable or disable canarying. Canarying is disabled by default.

When to enable canary snapshots and logpoints

To protect deployment and production-critical workloads, enable canarying when debugging these workloads.

If you have a single instance, you can still debug with canarying enabled, but your single instance runs without canarying the snapshot or logpoint.

When not to enable canary snapshots and logpoints

Don't enable canarying on workloads that have an execution time of less than 40 seconds, for instance, jobs using Cloud Functions.

Don't enable canarying if you want a faster snapshot-triggering cycle.

To configure the Debugger agent to not canary snapshots and logpoints, go to the installation instructions for the Google Cloud platform you're using.

App Engine standard environment

Python 3.7 or Python 3.8

If you are using Python 3.7 or Python 3.8, you must manually enable the Debugger agent by performing the following steps:

1. Make sure your `app.yaml` (/appengine/docs/standard/python3/configuring-your-app-with-app-yaml) file contains the following lines:

```
runtime: python37
or
runtime: python38
```

2. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using the Django web framework (version 1.* only).

   To debug with canarying enabled:

   ```
   try:
     import googleclouddebugger
     googleclouddebugger.enable(
       breakpoint_enable_canary=True
     )
   except ImportError:
     pass
   ```

   To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

   ```
   breakpoint_enable_canary=False
   ```

3. Add `google-python-cloud-debugger` to `requirements.txt`.
4. To have the **Debug** page in the Cloud Console automatically display source code matching the deployed app, go to Selecting source code automatically (/debugger/docs/source-context).

The Debugger is now ready for use with your app.

App Engine flexible environment

You can use the Debugger with the App Engine Python runtime or a custom runtime.

1. Make sure your App Engine Flexible VM instances are running:
   - A 64-bit Debian Linux image
   - Python 3
2. Make sure your `app.yaml` (/appengine/docs/flexible/python/configuring-your-app-with-app-yaml) file contains the following lines:

```
runtime: python
env: flex
```

If you are using a Custom Runtime, use `runtime: custom`.

3. Add `google-python-cloud-debugger` to `requirements.txt`.
4. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using the Django web framework (version 1.* only).

   To debug with canarying enabled:

```
try:
  import googleclouddebugger
  googleclouddebugger.enable(
    breakpoint_enable_canary=True
  )
except ImportError:
  pass
```

   To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

5. To have the **Debug** page in the Cloud Console automatically display source code matching the deployed app, see Selecting source code automatically (/debugger/docs/source-context).

The Debugger is now ready for use with your app.

Google Kubernetes Engine

GCLOUDCONSOLE (#console)

To enable Debugger using `gcloud`, complete the following steps:

1. Create your cluster with one of the following access scopes:

   - `https://www.googleapis.com/auth/cloud-platform` grants your cluster access to all Google Cloud APIs.
   - `https://www.googleapis.com/auth/cloud_debugger` grants your cluster access to only the Debugger API. Use this access scope to harden your cluster's security (/kubernetes-engine/docs/how-to/hardening-your-cluster).

```
gcloud container clusters create example-cluster-name \
      --scopes=https://www.googleapis.com/auth/cloud_debugger
```

★ **Note:** You cannot change the access scopes of a cluster after creation.

2. Add the Debugger package to your app:

   If you use a `requirements.txt` file, add the following line:

```
google-python-cloud-debugger
```

   If you use a `Dockerfile`, add the following line:

```
RUN pip install google-python-cloud-debugger
```

3. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using the Django web framework:

   To debug with canarying enabled:

```
try:
  import googleclouddebugger
  googleclouddebugger.enable(
    breakpoint_enable_canary=True
  )
except ImportError:
  pass
```

To debug with canarying NOT enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

On the **Debug** page, select the location of the source code. To have the **Debug** page in the Cloud Console automatically display source code matching the deployed app, see Selecting source code automatically (/debugger/docs/source-context).

The Debugger is now ready to use.

Compute Engine

1. Make sure your Compute Engine VM instances are running:
   - A 64-bit Debian Linux image
   - Python 3
2. Make sure your Compute Engine VM instances are created with the access scope option **Allow full access to all Cloud APIs**, or have one of the following access scopes:
   - `https://www.googleapis.com/auth/cloud-platform`
   - `https://www.googleapis.com/auth/cloud_debugger`
3. Download the Debugger agent.

   The easiest way to install the Python Debugger is with [`pip`][pip]:

   ```
   pip install google-python-cloud-debugger
   ```

4. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using the Django web framework.

   To debug with canarying enabled:

```
try:
  import googleclouddebugger
  googleclouddebugger.enable(
    module='[MODULE]',
    version='[VERSION]'
    breakpoint_enable_canary=True
  )
except ImportError:
  pass
```

To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

If you can't change the code, run the Debugger agent as a module.

To debug with canarying enabled:

```
python -m googleclouddebugger \
      --module=[MODULE] \
      --version=[VERSION] \
      --breakpoint_enable_canary=True
      -- \
      myapp.py
```

To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

★ **Note:** When running Debugger as a module, the value for **`--breakpoint_enable_canary`** is a string. Any value other than **`True`** is treated as **`False`**.

Replace the placeholders in the command as follows:

- `[MODULE]` is the name of your app.
  This, along with the version, is used to identify the debug target in the Cloud Console **Debug** page.
  Examples: `MyApp`, `Backend`, or `Frontend`.
- `[VERSION]` is the app version (for example, the build ID).
  The Cloud Console **Debug** page displays the running version as `[MODULE] - [VERSION]`.
  Example values: `v1.0`, `build_147`, or `v20170714`.

The Debugger is now ready for use with your app.

To have the **Debug** page in the Cloud Console automatically display source code matching the deployed app see Selecting source code automatically (/debugger/docs/source-context).

Cloud Run and Cloud Run for Anthos on Google Cloud

1. Python package.

   If you use a `requirements.txt` file, add the following line:

   ```
   google-python-cloud-debugger
   ```

   If you do not, add the following line to your `Dockerfile`:

   ```
   RUN pip install google-python-cloud-debugger
   ```

2. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using the Django web framework:

   To debug with canarying enabled:

   ```
   try:
     import googleclouddebugger
   ```

```
  googleclouddebugger.enable(
    breakpoint_enable_canary=True
  )

except ImportError:
  pass
```

To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

On the **Debug** page, select the location of the source code. To have the **Debug** page in the Cloud Console automatically display source code matching the deployed app, see Selecting source code automatically (/debugger/docs/source-context).

The Debugger is now ready to use.

Local and elsewhere

1. Make sure your workstation is running:
   - A 64-bit Debian Linux image
   - Python 3
2. Download the Debugger agent.

   The easiest way to install the Python Debugger is with [`pip`][pip]{: .external}:

   ```
   pip install google-python-cloud-debugger
   ```

3. Download service account credentials.

   To use the Cloud Debugger agent for Python on machines *not* hosted by Google Cloud, the agent must use Google Cloud service-account credentials to authenticate with the Cloud Debugger Service.

Use the Cloud Console **Service Accounts** page (https://console.cloud.google.com/iam-admin/serviceaccounts/project) to create a credentials file for an existing or new service-account. The service-account must have at least the `Cloud Debugger Agent` role.

Place the service-account JSON file alongside the Cloud Debugger agent for Python.

4. Add the following lines as early as possible in your initialization code, such as in your main function, or in `manage.py` when using Django Web Framework.

   To debug with canarying enabled:

```
try:
  import googleclouddebugger
  googleclouddebugger.enable(
      module='[MODULE]',
      version='[VERSION]',
      breakpoint_enable_canary=True
      service_account_json_file='/opt/cdbg/gcp-svc.json')
except ImportError:
  pass
```

   To debug with canarying NOT enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

   If you can't change the code, run the Debugger agent as a module.

   To debug with canarying enabled:

```
python \
    -m googleclouddebugger \
    --module=[MODULE] \
    --version=[VERSION] \
    --breakpoint_enable_canary=True
    --service_account_json_file=/opt/cdbg/gcp-svc.json \
    -- \
    myapp.py
```

To debug with canarying *not* enabled, set the `breakpoint_enable_canary` parameter to `False`:

```
breakpoint_enable_canary=False
```

★ **Note:** When running Debugger as a module, the value for `--breakpoint_enable_canary` is a string. Any value other than `True` is treated as `False`.

Replace the placeholders in the command as follows:

- `[MODULE]` is the name of your app.
  This, along with the version, is used to identify the debug target in the Cloud Console **Debug** page.
  Examples: `MyApp`, `Backend`, or `Frontend`.
- `[VERSION]` is the app version (for example, the build ID).
  The Cloud Console **Debug** page displays the running version as `[MODULE] - [VERSION]`.
  Example values: `v1.0`, `build_147`, or `v20170714`.
- The `GOOGLE_APPLICATION_CREDENTIALS` environment variable can be used instead of specifying `service_account_json_file`.

The Debugger is now ready for use with your app.

The **Debug** page in the Cloud Console can display local source files, without upload, for local development. See Selecting source code manually (/debugger/docs/source-options).

You can find the Debugger agent's code and documentation on GitHub
://github.com/GoogleCloudPlatform/cloud-debug-python).

Last updated 2020-07-21 UTC.