oduct or feature is in a pre-release state and might change or have limited support. For more information, see the pr
stages (/products/#product-launch-stages).

This page describes how to modify your Python application to capture profiling data and have that data sent to your Google Cloud project. For general information about profiling, see Profiling concepts (/profiler/docs/concepts-profiling).

Profile types for Python:

- CPU time
- Wall time (main thread, requires Python 3.6 or higher)

Supported Python language versions:

- Python 3.2 or higher.

Supported operating systems:

- Linux versions whose standard C library is implemented with `glibc`.

Supported environments:

- Compute Engine
- Google Kubernetes Engine (GKE)
- App Engine flexible environment
- App Engine standard environment (requires Python 3 runtime environment (/appengine/docs/standard/python3/runtime))
- Outside of Google Cloud (For information on the additional configuration requirements, see Profiling applications running outside of Google Cloud (/profiler/docs/profiling-external).)

Before you use the profiling agent, ensure that the underlying Profiler API is enabled. You can check the status of the API and enable it if necessary by using either the Cloud SDK `gcloud` command-line tool or the Cloud Console:

For best practices using Python, go to Setting up a Python development environment (/python/setup).

The `googlecloudprofiler.start` function creates a daemon thread which continuously collects and uploads profiles. You should call `start` one time, and as early as possible in your application.

Parameter	Description
`service`[1]	(Required) The name for the service being profiled. For restrictions on the service name, see Service name and version arguments (#svc-name-and-version).
`service_version`[1]	(Optional) The version of the service being profiled. For restrictions on the service version, see Service name and version arguments (#svc-name-and-version).
`verbose`	(Optional) The logging level. For details on the logging levels, see Agent logging (#agent_logging). Default value is **0 (Error)**.
`project_id`[2]	(Optional) Your Google Cloud project ID.
`disable_cpu_profiling`	(Optional) To disable CPU time profiling, set `disable_cpu_profiling=True`. This parameter is supported only for Python version 3.2 and higher. For all other Python versions, CPU time profiling isn't supported and this parameter is ignored. Default value is **False**.
`disable_wall_profiling`	(Optional) To disable Wall profiling, set `disable_wall_profiling=True`. This parameter is supported for Python 2, and for Python 3.6 and higher. For all other Python versions, Wall profiling isn't supported and this parameter is ignored. For restrictions on the `start` function when Wall profiling is enabled, see Limitations (#limitations). Default value is **False**.

[1] For only Compute Engine and GKE. For App Engine, the value is derived from the environment.

[2] For Google Cloud, the value is derived from the environment. For non-Google Cloud environments, you must provide a value. For information, see Profiling applications running outside Google Cloud (/profiler/docs/profiling-external).

After Profiler has collected data, you can view and analyze this data using the Profiler interface. To get started using this interface, see Opening the Profiler interface (/profiler/docs/using-profiler).

When you load the Profiler agent, you specify a service-name argument and an optional service-version argument to configure it.

In the App Engine flexible environment, you do not have to specify these arguments; they are derived from the nment.

The **service name** lets Profiler collect profiling data for all replicas of that service. The profiler service ensures a collection rate of one profile per minute, on average, for each service name across each combination service versions and zones.

For example, if you have a service with two versions running across replicas in three zones, the profiler will create an average of 6 profiles per minute for that service.

If you use different service names for your replicas, then your service will be profiled more often than necessary, with a correspondingly higher overhead.

When selecting a service name:

- Choose a name that clearly represents the service in your application architecture. The choice of service name is less important if you only run a single service or application. It is more important if your application runs as a set of micro-services, for example.
- Make sure to not use any process-specific values, like a process ID, in the service-name string.
- The service-name string must match this regular expression:

  `^[a-z]([-a-z0-9_.]{0,253}[a-z0-9])?$`

A good guideline is to use a static string like `imageproc-service` as the service name.

The **service version** is optional. If you specify the service version, Profiler can aggregate profiling information from multiple instances and display it correctly. It can be used to mark different versions of your services as they get deployed. The Profiler UI lets you filter the data by service version; this way, you can compare the performance of older and newer versions of the code.

The value of the service-version argument is a free-form string, but values for this argument typically look like version numbers, for example, `1.0.0` or `2.1.2`.

By default, the profiling agent logs messages with a severity level of `error`. To configure the agent to log messages with lower severity levels, specify the `verbose` parameter when starting the agent. There are four supported values for `verbose`:

- `0` : Error
- `1` : Warning
- `2` : Informational
- `3` : Debug

If you set the `verbose` parameter to `1` in your call to `start`, then messages with a severity level of `Warning` or `Error` are logged, while `Informational` and `Debug` messages are ignored.

To log all messages, set `verbose` to `3` when starting the agent:

This section lists limitations, exceptions, and known issues.

Profile type	Restrictions and limitations
CPU time	• Supported for Python 3.2 and higher.

Profile type	Restrictions and limitations
Wall time	• Supported for Python 3.6 and higher. • Not available for Python 3.0 to 3.5. • Available, but not supported, for Python 2. **Note:** Profiling Python 2 applications isn't recommended in production environments. For more information, see Known issues (#known-issues). • Main-thread profiling only. • The profile `start` function must be called from the main thread. • The Profiler signal handler executes only on the main thread. If the main thread cannot execute, no profiling data is captured. **Note:** Don't enable Wall profiling if your application contains a large number of threads. Use of Wall profiles in this configuration results in slow performance and no profile data.

Error	Cause	Solution
`NotImplementedError` thrown during `start`	Application executed in a non-Linux environment.	• Execute your application in a Linux environment.

Error	Cause	Solution
`ValueError` thrown during `start`	The `start` function arguments are invalid, necessary information can't be determined from the environment variables and arguments, or profiling if both CPU time profiling and Wall time profiling are disabled.	• Ensure the service name and version meet the requirements defined in Service name and version arguments (#svc-name-and-version). • If Wall profiling is enabled, make sure `start` is called from the main thread. • Ensure that you are using a supported version of Python and that CPU time or Wall time profiling is enabled. For more information, see `start` function (#start). • Check that you have specified the `project_id` parameter to `start` if you are running outside of Google Cloud. For more information, see `start` function (#start).

Behavior	Cause	Solution
System calls fail with Python 3.4 or earlier applications, when profiling CPU time or Wall time.	Python, version 3.4 and earlier, fails system calls interrupted by a signal with `EINTR`. Your application must handle this scenario. For more detailed information, see PEP 475 (https://www.python.org/dev/peps/pep-0475).	• Use Python 3.5 or later. • Have the application handle the `EINTR` response code.
Your Python 3.5 or earlier application isn't responding to signals when Wall profiling is enabled.	Python 3.5 and earlier contains a race condition in its signal handling. The effect is that applications stop responding to signals, including the kill signal `Ctrl-C`.	• Use Python 3.6 or later.

To learn about the Profiler graph and controls, go to Using the Stackdriver Profiler Interface (/profiler/docs/using-profiler). For advanced information, go to the following:

- Filtering profiles (/profiler/docs/filtering-profiles)
- Focusing the graph (/profiler/docs/focusing-profiles)

- Compare profiles (/profiler/docs/comparing-profiles)