This page describes how to modify your Java 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 Java:

- CPU time
- Heap (Alpha, requires Java 11 or App Engine standard environment)
- Wall time (not available for Java 8 App Engine standard environment)

Supported Java language versions:

- OpenJDK and Oracle JDK for Java 7, 8, 9, or 11.

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 App Engine SDK (/appengine/docs/standard/java/download) version 1.9.64 or later)
- 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:

To list all versions of the agent available for downloading, run the following command:

The command response is similar to the following:

To download a specific version of the agent, pass its URL to the download command. For example, to download the agent built on 28 October 2019, you would use the following statement:

The version of the agent is logged during its initialization.

To profile your application, start Java as you normally would to run your program, but specify the agent-configuration options. You specify the path to the agent library, and you can pass options to the library.

For the App Engine standard environment, the agent is automatically loaded and configured. Skip ahead to Starting your program (#starting_your_program), for details on configuring, and starting, your program.

To configure the profiling agent, include the `-agentpath` flag when starting your application:

In this expression, `[INSTALL_DIR]` is the path to the profiling agent, while `[OPTION1]`, `[OPTION2]`, and `[OPTION3]` are agent configuration options. For example, if you replace `[OPTION1]` with `-cprof_service=myapp` in the previous expression, then you set the service name to `myapp`. There is no restriction on the number of options or their ordering. Supported configuration options are listed in the following table:

Agent option	Description
`-cprof_service`	If your application isn't running on App Engine, then you must use this configuration option to set the service name. For service name restrictions, see Service name and version arguments (#service_name_and_version_arguments).

Agent option	Description
`-cprof_service_version`	When you want the ability to analyze profiling data using the Profiler UI by the version of the service, use this option to set the version. For version restrictions, see Service name and version arguments (#service_name_and_version_arguments).
`-cprof_project_id`	When you are running outside of Google Cloud, use this option to specify your Google Cloud project ID. For more information, see Profiling applications running outside of Google Cloud (/profiler/docs/profiling-external).
`-cprof_zone_name`	When your application is running on Google Cloud, the profiling agent determines the zone (/compute/docs/regions-zones/) by communicating with the Compute Engine metadata service (/compute/docs/storing-retrieving-metadata). If the profiling agent can't communicate with the metadata service, then you need to use this option.
`-cprof_gce_metadata_server_retry_count` - `cprof_gce_metadata_server_retry_sleep_sec`	Together, these two options define the retry policy that the profiler agent uses when it communicates with the Compute Engine metadata service (/compute/docs/storing-retrieving-metadata). to gather your Google Cloud project ID and zone information. Default policy is to retry up to 3 times waiting 1 second between attempts. This policy is sufficient for most configurations.
`-cprof_cpu_use_per_thread_timers`	For the most accurate CPU time profiles, set this option to true. Use of this option results in increased per-thread overhead. Default value is false.
`-cprof_force_debug_non_safepoints`	By default, the profiling agent forces JVM to generate debugging information for all just in time (JIT) generated code, in addition to generating debug information for all safepoints. This results in the most accurate function and line-level location information for CPU time and heap profiles at the expense of additional agent overhead. You can disable the generation of debugging information for JIT code by setting this option to false. Default value is true.

Agent option	Description
`-cprof_wall_num_threads_cutoff`	By default, wall profiles aren't collected if the total number of threads in the application exceeds 4096. The limit ensures that during profile collection, the cost of traversing the stack of threads is minimal. If your service normally has more than 4096 threads and if you want to collect profiling data at the expense of additional overhead, use this flag to increase limit. Default limit is 4096 threads.

Alpha

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

`-cprof_enable_heap_sampling`	To enable heap profiling for Java 11 and higher, set **`-cprof_enable_heap_sampling`** to true. Heap profiling isn't supported for Java 10 and lower. Heap profiling is disabled by default. When you enable heap profiling, the sampling interval is set to 512 KiB by default. This interval is sufficient for most applications and incurs less than 0.5% overhead for the application. Sampling intervals from 256 KiB (262144) to 1024 KiB (1048576) are supported. For example, to set the sampling interval to 256 KiB, which doubles the sampling rate, add the agent option: Similarly, to set the sampling interval to 1024 KiB, which halves the sampling rate, add the agent option:

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`.

The profiling agent can report logging information for App Engine flexible environment, Compute Engine, and GKE. The profiling agent supports the following logging levels:

- `0`: Log all messages. Default logging level.
- `1`: Log warning, error, and fatal messages.
- `2`: Log error and fatal messages.
- `3`: Log only fatal messages and stop the application.

To enable writing logs to standard error with the default logging level, append `-logtostderr` to the `-agentpath` configuration.

To set the logging level to log only error and fatal messages, append `-minloglevel=2` to the `-agentpath` configuration.

For example, to enable logging of error and fatal messages to standard error, append `-logtostderr` and `-minloglevel=2` to the `-agentpath` configuration:

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)