This page describes how to modify your Go 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 Go:

- CPU time
- Heap
- Allocated heap
- Contention (Go mutex)
- Threads (Go goroutine)

Supported Go language versions:

- All officially maintained Go releases, unless otherwise noted. For more information, see Go language release policy (https://golang.org/doc/devel/release.html#policy).

Supported operating systems:

- Linux. Profiling Go applications is supported for Linux kernels whose standard C library is implemented with `glibc` or with `musl`. For configuration information specific to Linux Alpine kernels, see Running on Linux Alpine (#running_with_linux_alpine).

Supported environments:

- Compute Engine
- Google Kubernetes Engine (GKE)
- App Engine flexible environment
- App Engine standard environment (requires Go 1.11 or higher)
- 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:

In all of the supported environments, you use the Profiler by importing the package in your application and then initializing the Profiler as early as possible in your application.

You can enable Mutex-contention profiling (“Contention” in the interface) by setting the `MutexProfiling` configuration option to `true`.

For more information on the Profiler API, including all the configuration options, see the public API docs (https://godoc.org/cloud.google.com/go/profiler).

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`.

The profiling agent can report debug information in its logs. By default, agent logging is disabled.

To enable agent logging, set the `DebugLogging` option to `true` when starting the agent:

If you use Docker images that run with Linux Alpine (https://alpinelinux.org) (such as `golang:alpine` or just `alpine`), you might see the following authentication error:

Note that to see the error you must have agent logging enabled. By default the agent for Go doesn't output any log messages.

The error indicates that the Docker images with Linux Alpine don't have the root SSL certificates installed by default. Those certificates are necessary for the profiling agent to communicate with the profiler API. To resolve this error, add the following `apk` commands to your Dockerfile:

You then need to rebuild and redeploy your application.

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)