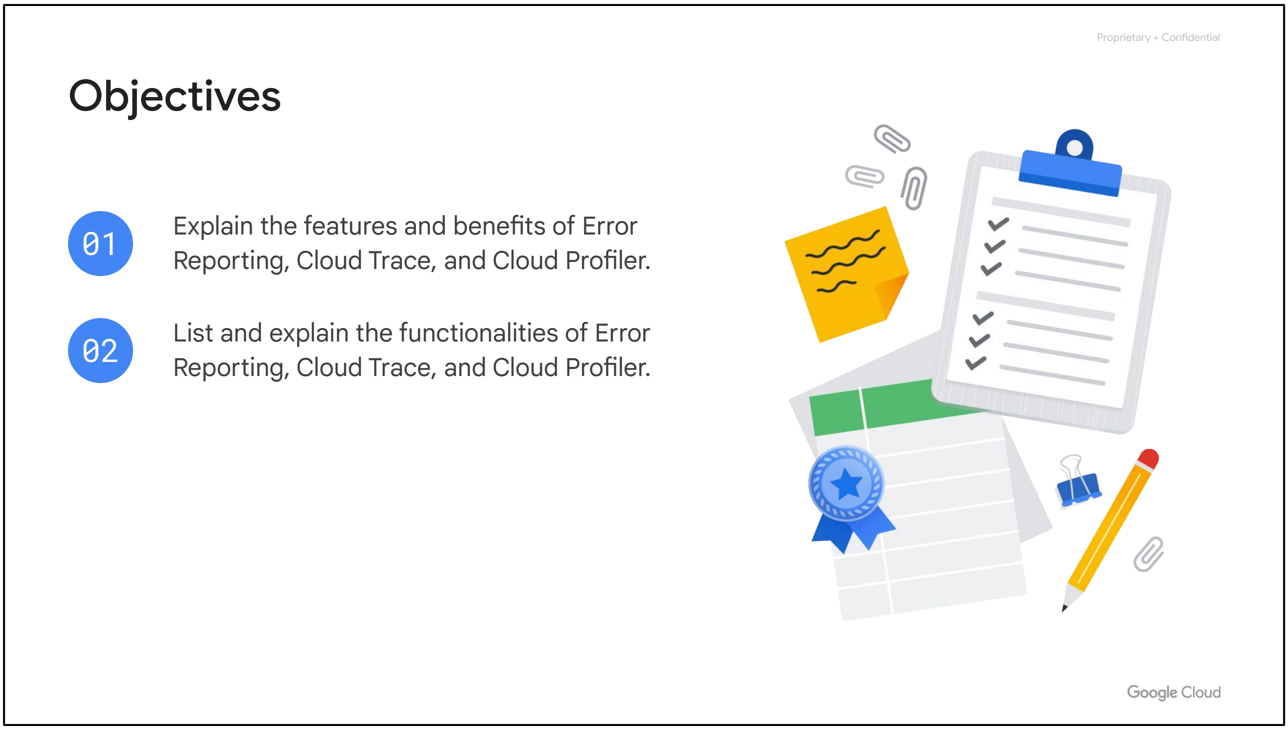
Google Cloud

Investigating Application Performance Issues

When deploying applications to Google Cloud, the Application Performance Management products (Cloud Trace and Cloud Profiler) provide insight into how your code and services are functioning. The tools also help troubleshoot where needed.



Proprietary + Confidential

Objectives

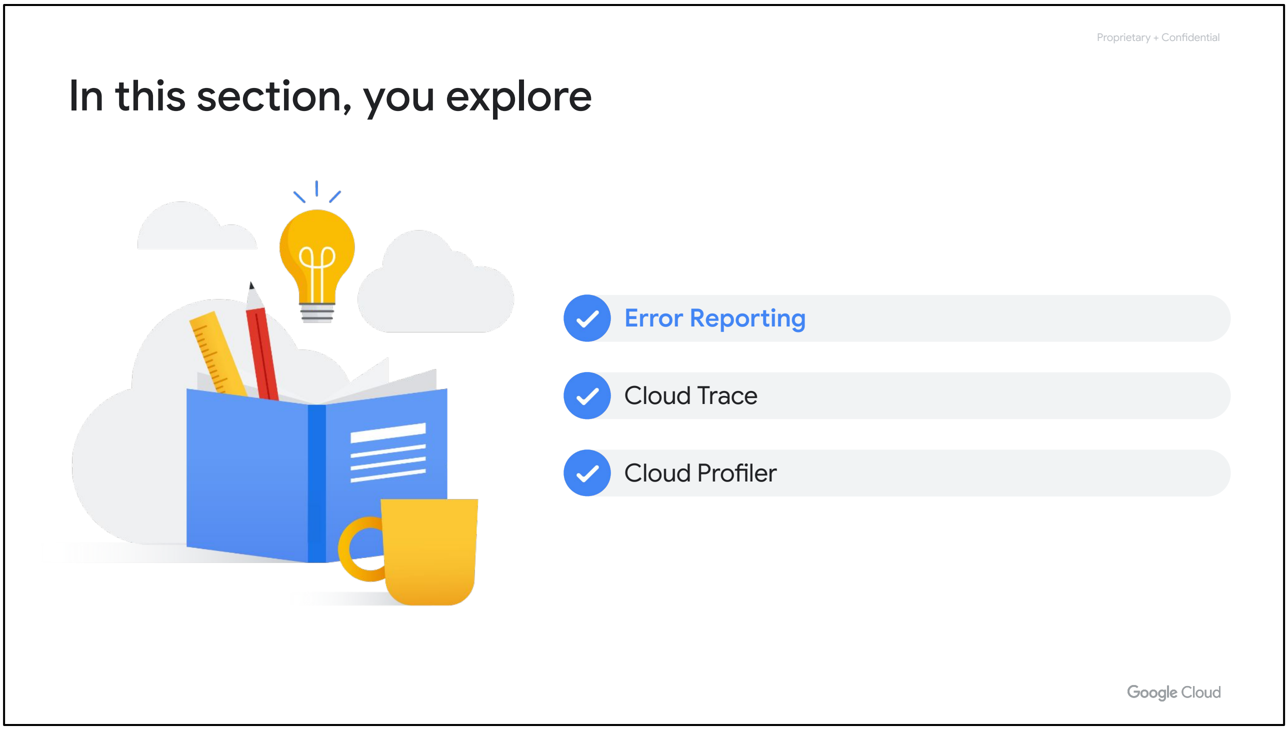
01 Explain the features and benefits of Error Reporting, Cloud Trace, and Cloud Profiler.

02 List and explain the functionalities of Error Reporting, Cloud Trace, and Cloud Profiler.

Google Cloud

In this module, you learn to:

- Explain the features and benefits of Error Reporting, Cloud Trace, and Cloud Profiler.
- List and explain the functionalities of the Error Reporting, Cloud Trace, and Cloud Profiler components.

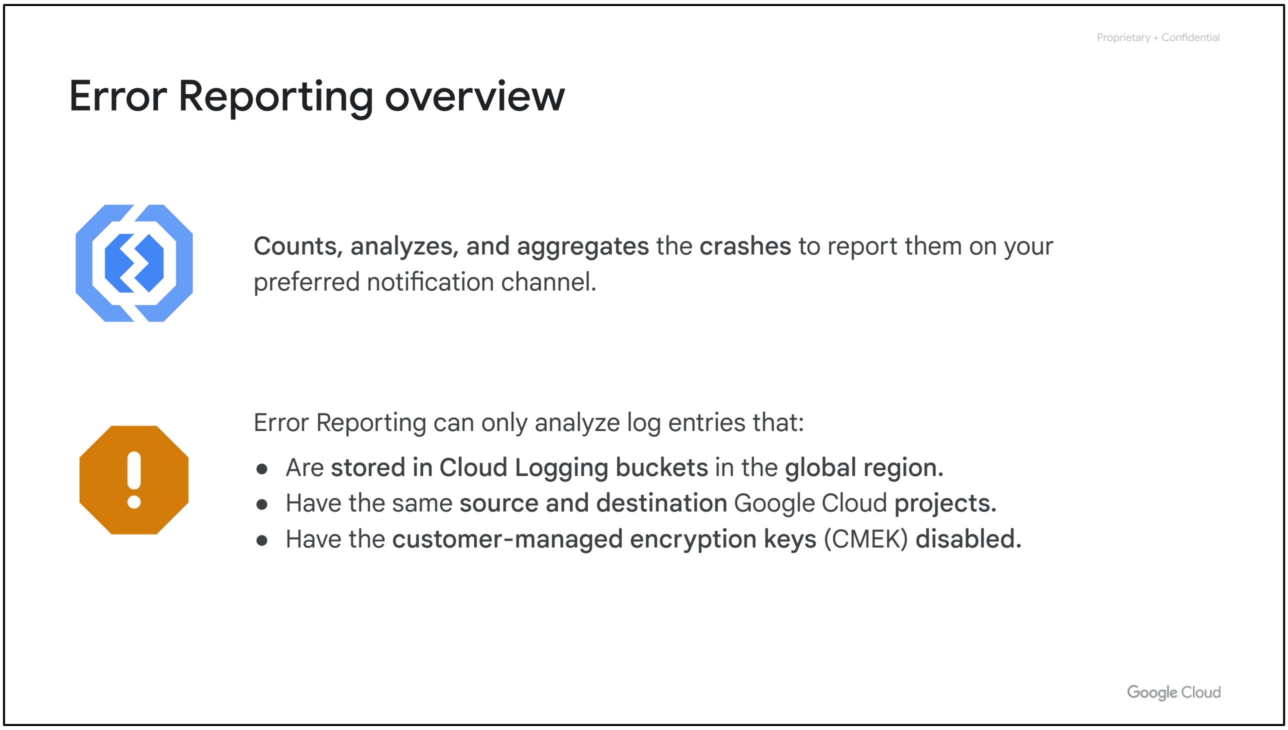


In this section, you explore

- ✓ Error Reporting
- ✓ Cloud Trace
- ✓ Cloud Profiler

Application Performance Management (APM) combines the monitoring and troubleshooting capabilities of Cloud Logging and Cloud Monitoring with Error Reporting, Cloud Trace, and Cloud Profiler. APM helps you reduce latency and cost so that you can run more efficient applications.

Let's start with Error Reporting.



Error Reporting overview

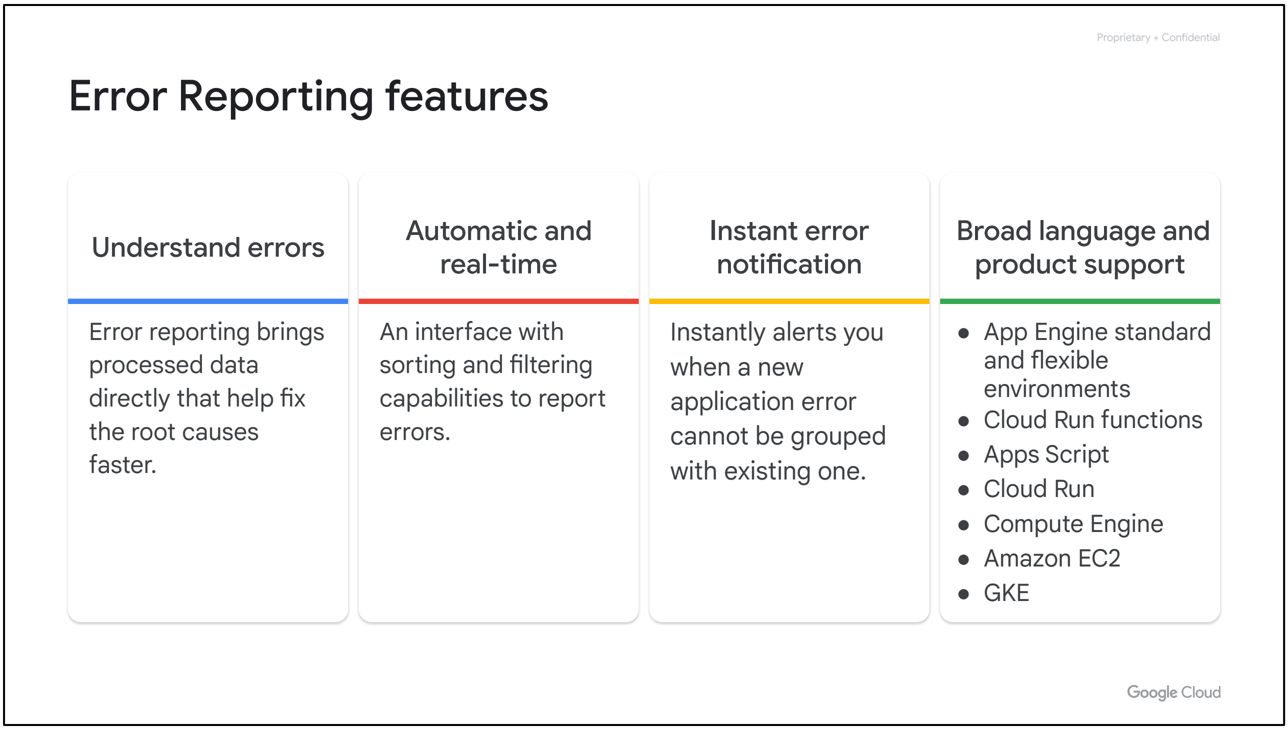
Counts, analyzes, and aggregates the **crashes** to report them on your preferred notification channel.

Error Reporting can only analyze log entries that:

- Are **stored in Cloud Logging buckets** in the **global region.**
- Have the same **source and destination** Google Cloud **projects.**
- Have the **customer-managed encryption keys** (CMEK) **disabled.**

Analyses and reports exceptions
Error Reporting looks through all the logs that your application and infrastructure has reported. It then counts, analyzes, and aggregates the exceptions to report them on your preferred notification channel such as email, mobile app, slack, or through web hooks.

Note: Error Reporting can only analyze log entries that are stored in Cloud Logging buckets that are in the global region. The source and destination Google Cloud projects must be the same, and customer-managed encryption keys (CMEK) must be disabled. If you route logs to a different Cloud project, regionalized buckets, or enable CMEK, then Error Reporting doesn't capture and analyze those logs.



Proprietary + Confidential

Error Reporting features

Understand errors

Error reporting brings processed data directly that help fix the root causes faster.

Automatic and real-time

An interface with sorting and filtering capabilities to report errors.

Instant error notification

Instantly alerts you when a new application error cannot be grouped with existing one.

Broad language and product support

- App Engine standard and flexible environments
- Cloud Run functions
- Apps Script
- Cloud Run
- Compute Engine
- Amazon EC2
- GKE

Google Cloud

Understand errors
See at a glance the top or new errors for your application in a clear dashboard. Looking at a log stream to find important errors can slow you down when you're troubleshooting. Error Reporting brings you the processed data directly to help you understand and fix the root causes faster.

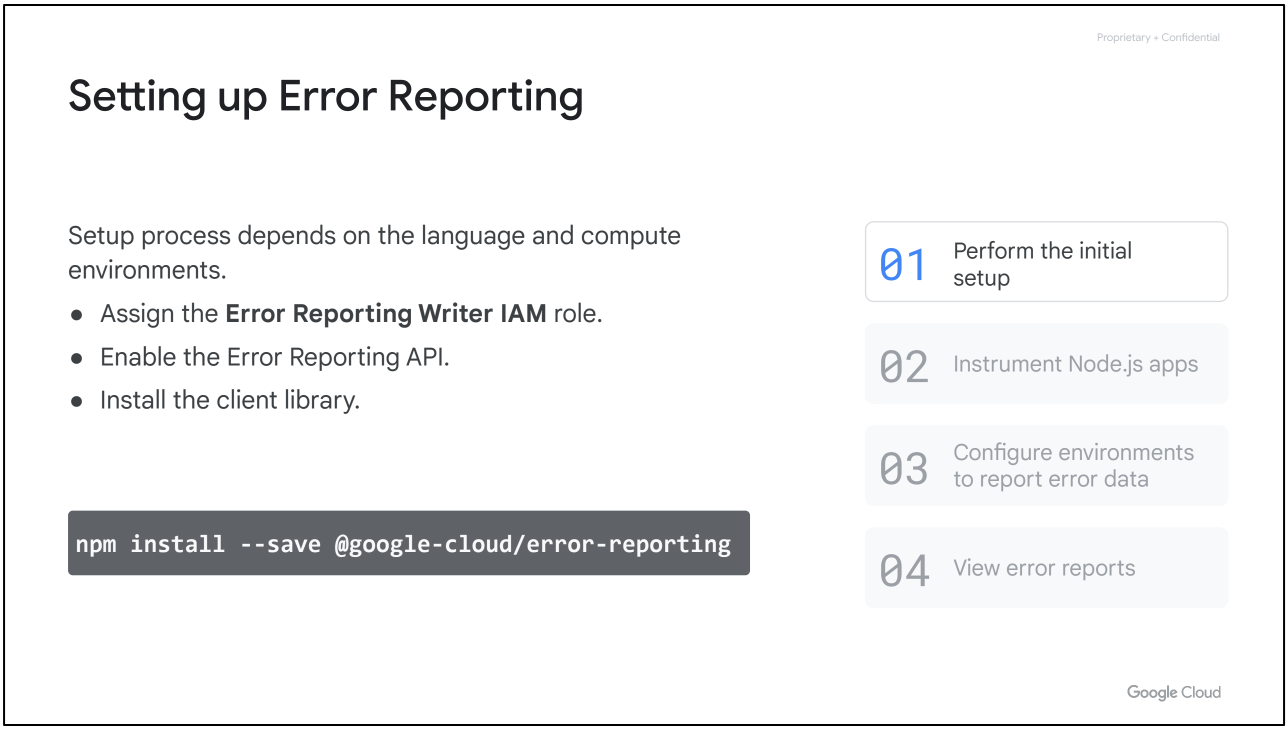
Real-time processing
Real production problems can be hidden in mountains of data. Error Reporting helps you see the problems through the noise by constantly analyzing your exceptions. Problems are intelligently aggregated into meaningful groups tailored to your programming language and framework.

Instant error notification
Do not wait for your users to report problems. Error Reporting is always watching your service and instantly alerts you when a new application error cannot be grouped with existing ones. Directly navigate from a notification to the details of the new error. Error Reporting is available on desktop and in the Google Cloud app for iOS and Android.

Popular language and product support
Support is available for many popular languages, including Go, Java, Node.js, PHP, Python, Ruby, or .NET. Use our client libraries, REST API, or send errors with Cloud Logging.

Error Reporting can aggregate and display errors for:

- App Engine standard environment and flexible environment
- Cloud Run functions
- Apps Script
- Cloud Run
- Compute Engine
- Amazon EC2
- And Google Kubernetes Engine (or GKE)



Proprietary + Confidential

Setting up Error Reporting

Setup process depends on the language and compute environments.

- Assign the **Error Reporting Writer IAM** role.
- Enable the Error Reporting API.
- Install the client library.

```
npm install --save @google-cloud/error-reporting
```

01 Perform the initial setup

02 Instrument Node.js apps

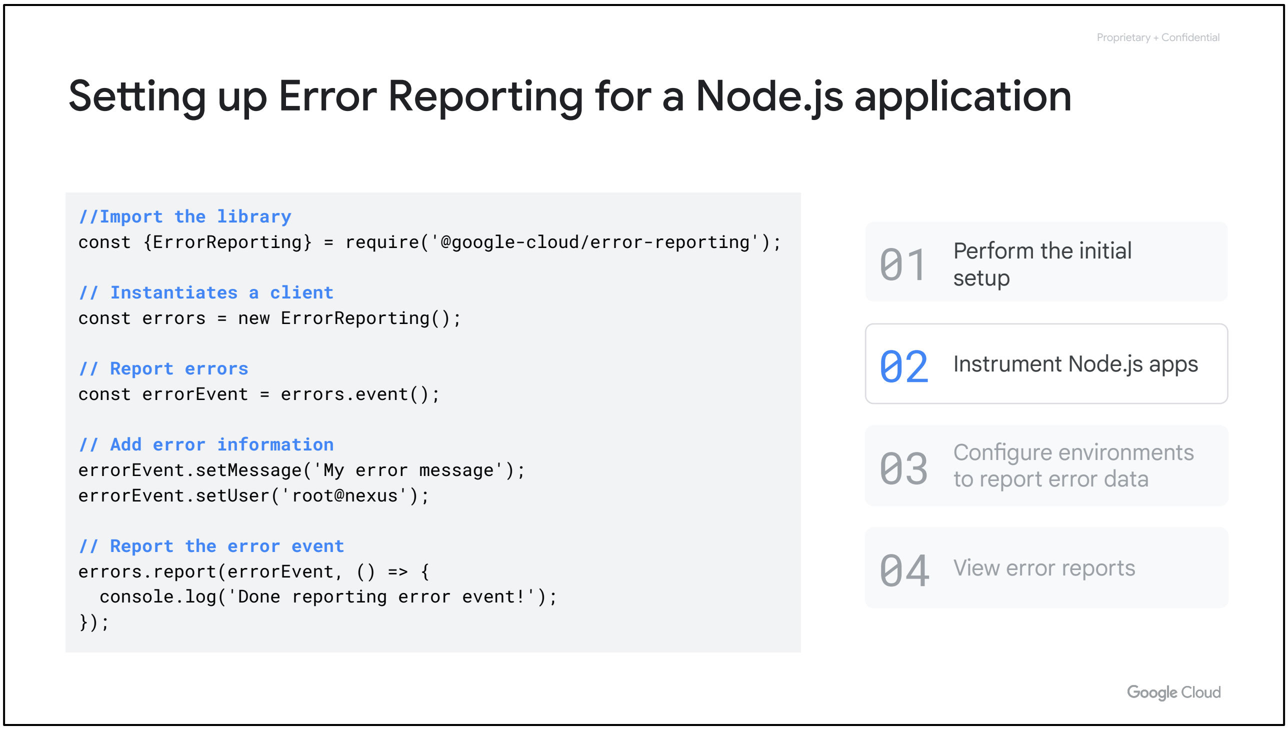
03 Configure environments to report error data

04 View error reports

Google Cloud

Setting up Error Reporting is simple and dependent on the language and compute environment. Here’s an error-catching example written in Node.js (JavaScript) and run on Cloud Run.

To report errors, the code needs the Error Reporting Writer Identity and Access Management (IAM) role. Enable the Error Reporting API, and install the client library by using npm.



Proprietary + Confidential

Setting up Error Reporting for a Node.js application

```
//Import the library
const {ErrorReporting} = require('@google-cloud/error-reporting');

// Instantiates a client
const errors = new ErrorReporting();

// Report errors
const errorEvent = errors.event();

// Add error information
errorEvent.setMessage('My error message');
errorEvent.setUser('root@nexus');

// Report the error event
errors.report(errorEvent, () => {
  console.log('Done reporting error event!');
});
```

01 Perform the initial setup

02 Instrument Node.js apps

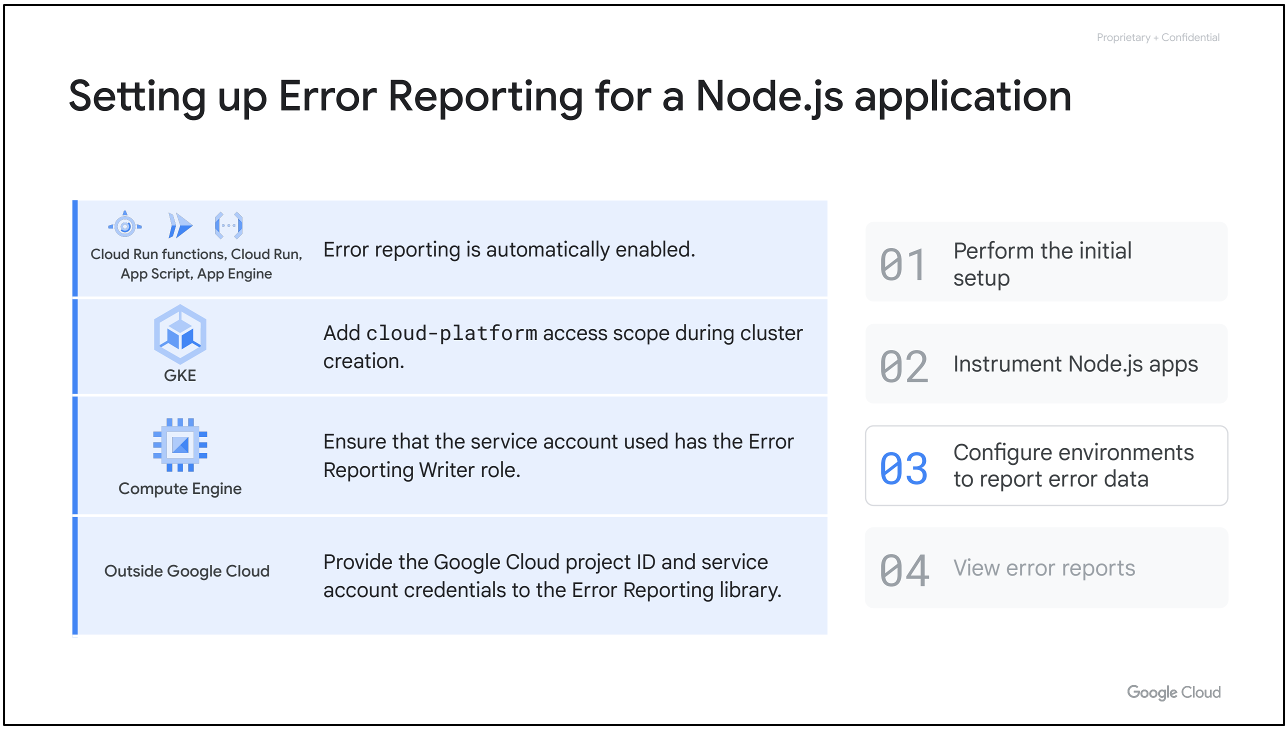
03 Configure environments to report error data

04 View error reports

The easiest way to manually log errors to Error Reporting in Node.js is to import the Error Reporting library.

- You then instantiate a client to start reporting errors to Error reporting. Optionally, you can also customize the behavior of the Error Reporting library for Node.js. These can be configured by passing objects to options.
- Use error message builder to customize all fields.
- Call the report method to manually report an error.

You can also integrate Error Reporting library for Node.js to web frameworks like Express.js. Refer to the documentation for more details.



Setting up Error Reporting for a Node.js application

Cloud Run functions, Cloud Run, App Script, App Engine	Error reporting is automatically enabled.
GKE	Add `cloud-platform` access scope during cluster creation.
Compute Engine	Ensure that the service account used has the Error Reporting Writer role.
Outside Google Cloud	Provide the Google Cloud project ID and service account credentials to the Error Reporting library.

01 Perform the initial setup

02 Instrument Node.js apps

03 Configure environments to report error data

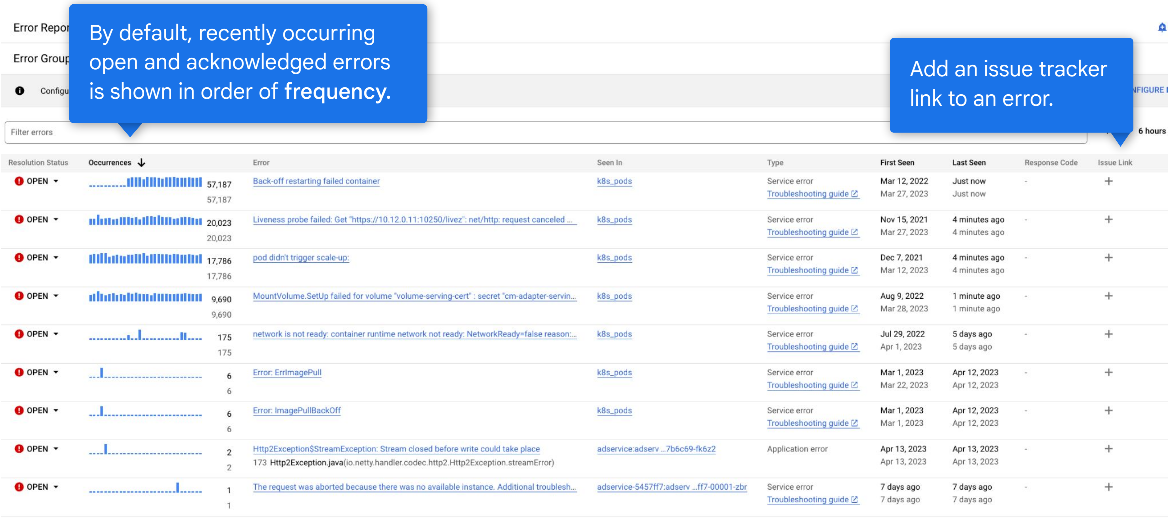
04 View error reports

Google Cloud

Error Reporting Library for Node.js can be configured on many Google Cloud environments. Let's explore the process for all.

- For App Engine flexible and standard environment, Cloud Run, Cloud Run functions, and AppScript, Error Reporting is automatically enabled.
- For Google Kubernetes Engine, add cloud-platform access scope during cluster creation. gcloud container clusters create example-cluster-name --scopes https://www.googleapis.com/auth/cloud-platform
- For Compute Engine, ensure service account used has the Error Reporting Writer role.
- Outside Google Cloud, provide the Google Cloud project ID and service account credentials to the Error Reporting library for Node.js.

View error reports



By default, recently occurring open and acknowledged errors is shown in order of **frequency.**

Add an issue tracker link to an error.

Resolution Status	Occurrences ↓	Error	Seen In	Type	First Seen	Last Seen	Response Code	Issue Link
OPEN	57,187 57,187	Back-off restarting failed container	k8s_pods	Service error Troubleshooting guide	Mar 12, 2022 Mar 27, 2023	Just now Just now	-	+
OPEN	20,023 20,023	Liveness probe failed: Get "https://10.12.0.11:10250/livez": net/http: request canceled ...	k8s_pods	Service error Troubleshooting guide	Nov 15, 2021 Mar 27, 2023	4 minutes ago 4 minutes ago	-	+
OPEN	17,786 17,786	pod didn't trigger scale-up:	k8s_pods	Service error Troubleshooting guide	Dec 7, 2021 Mar 12, 2023	4 minutes ago 4 minutes ago	-	+
OPEN	9,690 9,690	MountVolume.SetUp failed for volume "volume-serving-cert" : secret "cm-adapter-servin...	k8s_pods	Service error Troubleshooting guide	Aug 9, 2022 Mar 28, 2023	1 minute ago 1 minute ago	-	+
OPEN	175 175	network is not ready: container runtime network not ready: NetworkReady=false reason:...	k8s_pods	Service error Troubleshooting guide	Jul 29, 2022 Apr 1, 2023	5 days ago 5 days ago	-	+
OPEN	6 6	Error: ErrImagePull	k8s_pods	Service error Troubleshooting guide	Mar 1, 2023 Mar 22, 2023	Apr 12, 2023 Apr 12, 2023	-	+
OPEN	6 6	Error: ImagePullBackOff	k8s_pods	Service error Troubleshooting guide	Mar 1, 2023 Mar 1, 2023	Apr 12, 2023 Apr 12, 2023	-	+
OPEN	2 2	Http2Exception$StreamException: Stream closed before write could take place 173 Http2Exception.java(io.netty.handler.codec.http2.Http2Exception.streamError)	adservice:adserv ...7b6c69-fk6z2	Application error	Apr 13, 2023 Apr 13, 2023	Apr 13, 2023 Apr 13, 2023	-	+
OPEN	1 1	The request was aborted because there was no available instance. Additional troublesh...	adservice-5457ff7:adserv ...ff7-00001-zbr	Service error Troubleshooting guide	7 days ago 7 days ago	7 days ago 7 days ago	-	+

To see your errors, open the Error Reporting page in the Google Cloud console.

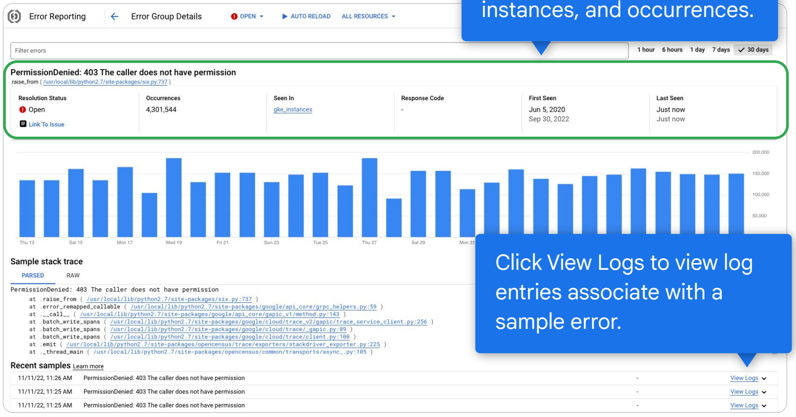
By default, Error Reporting shows you a list of recently occurring open and acknowledged errors in order of frequency.

Errors are grouped and de-duplicated by analyzing their stack traces. When *Auto reload* is turned on, Error Reporting automatically reloads the error list every 10 seconds.

Error Reporting recognizes the common frameworks used for your language and groups errors accordingly.

You can sort errors based on occurrences (first and last seen). You can also link an issue tracker link to an error group by clicking the + icon below **Insert link**.

View error details



PermissionDenied: 403 The caller does not have permission
.raise_from (/usr/local/lib/python2.7/site-packages/six.py:737)

Resolution Status	Occurrences	Seen In	Response Code	First Seen	Last Seen
● Open Link To Issue	4,301,544	gke_instances	-	Jun 5, 2020 Sep 30, 2022	Just now Just now

Sample stack trace

```
PermissionDenied: 403 The caller does not have permission
     at .raise_from ( /usr/local/lib/python2.7/site-packages/six.py:737 )
     at .error_remapped_callable ( /usr/local/lib/python2.7/site-packages/google/api_core/grpc_helpers.py:59 )
     at .__call__ ( /usr/local/lib/python2.7/site-packages/google/api_core/gapic_v1/method.py:143 )
     at .batch_write_spans ( /usr/local/lib/python2.7/site-packages/google/cloud/trace_v2/gapic/trace_service_client.py:256 )
     at .batch_write_spans ( /usr/local/lib/python2.7/site-packages/google/cloud/trace/_gapic.py:89 )
     at .batch_write_spans ( /usr/local/lib/python2.7/site-packages/google/cloud/trace/client.py:100 )
     at .emit ( /usr/local/lib/python2.7/site-packages/opencensus/trace/exporters/stackdriver_exporter.py:225 )
     at ._thread_main ( /usr/local/lib/python2.7/site-packages/opencensus/common/transports/async_.py:105 )
```

Recent samples

11/11/22, 11:26 AM	PermissionDenied: 403 The caller does not have permission	-	View Logs
11/11/22, 11:25 AM	PermissionDenied: 403 The caller does not have permission	-	View Logs
11/11/22, 11:25 AM	PermissionDenied: 403 The caller does not have permission	-	View Logs

You can examine the error group, specific error instances, and occurrences.

Click View Logs to view log entries associate with a sample error.

01 Perform the initial setup

02 Instrument Node.js apps

03 Configure environments to report error data

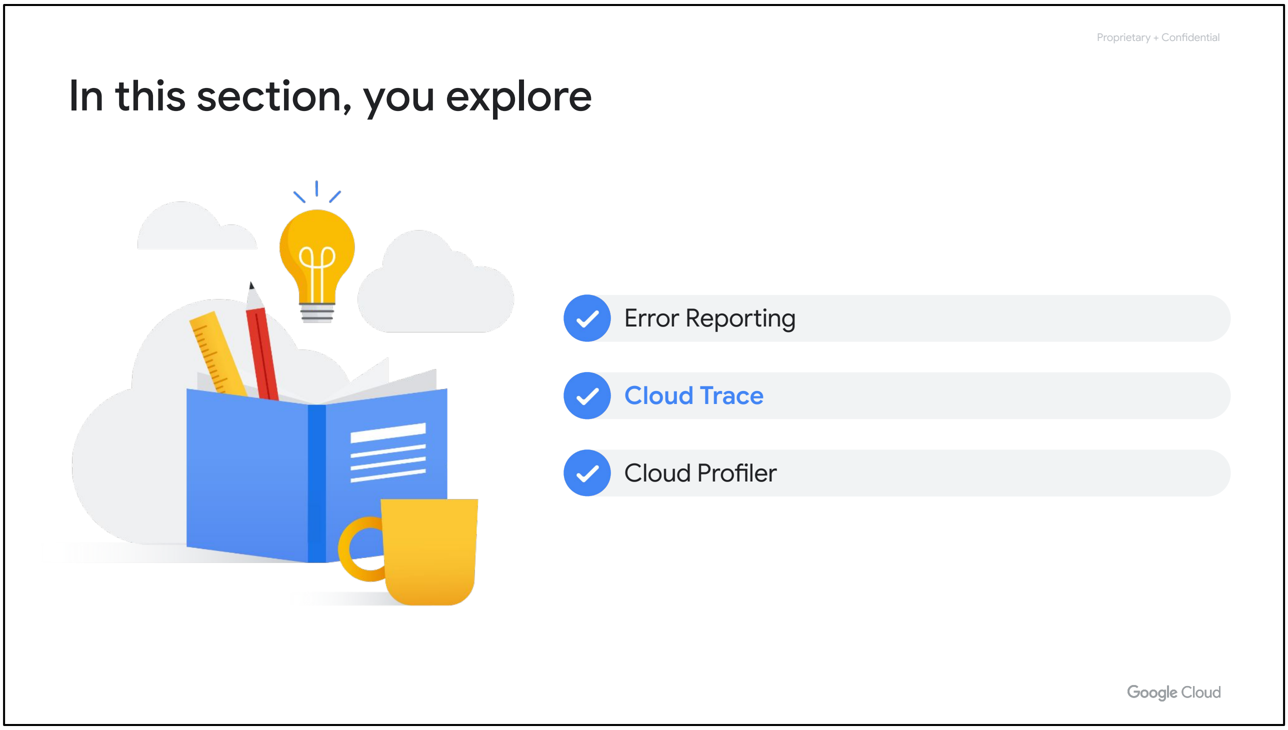
04 View error reports

Selecting an error entry will let allow you expand into the Error Details page.

On this page, you can examine information about the error group, including the history of a specific error, specific error instances, and diagnostic information.

To view the log entry associated with a sample error, click **View logs** from any entry in the **Recent samples** panel.

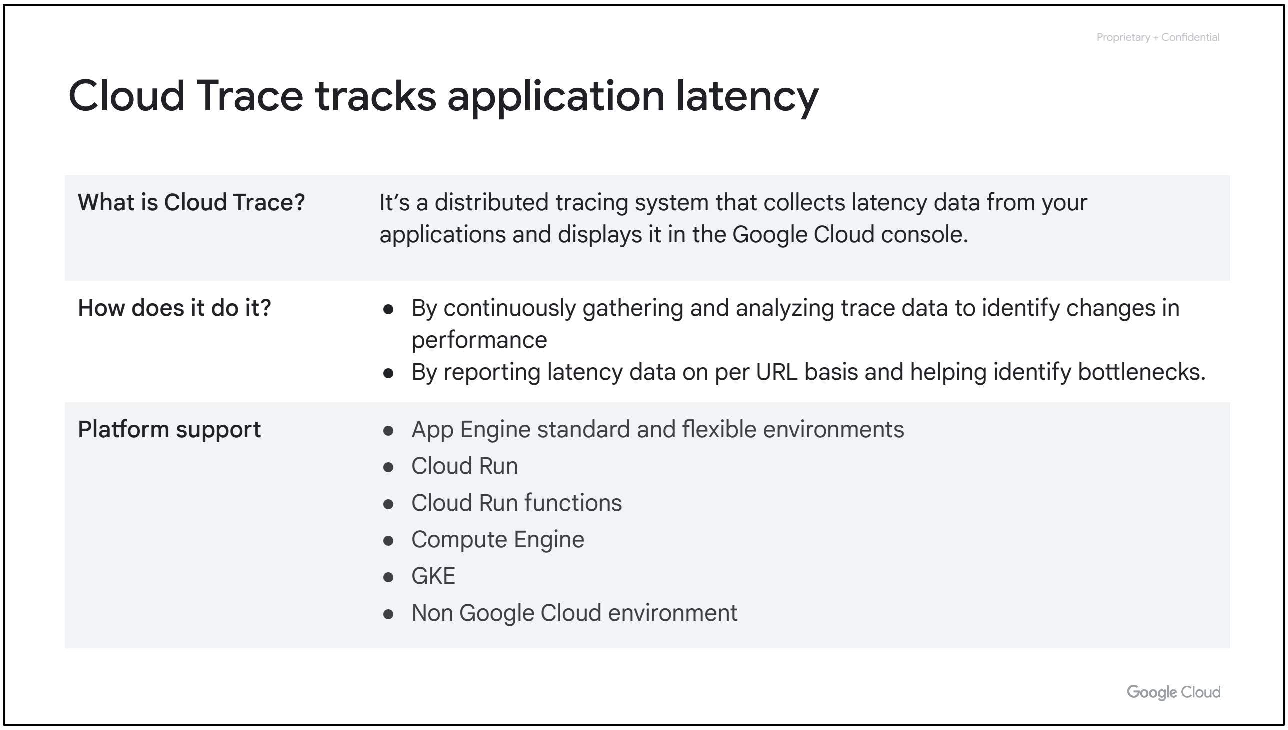
You’re taken to **Logs Explorer** in the Cloud Logging console.



In this section, you explore

- ✓ Error Reporting
- ✓ Cloud Trace
- ✓ Cloud Profiler

Now, let’s have a look at Cloud Trace.



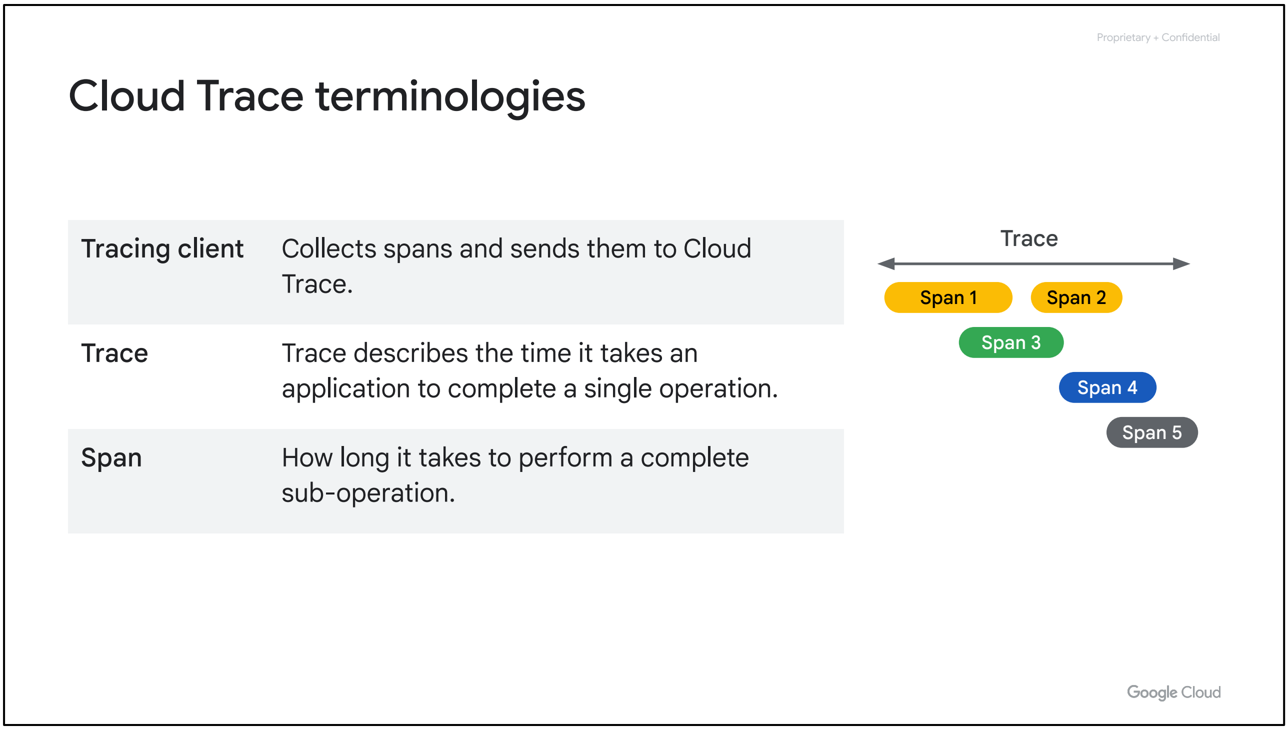
Cloud Trace tracks application latency

What is Cloud Trace?	It’s a distributed tracing system that collects latency data from your applications and displays it in the Google Cloud console.
How does it do it?	● By continuously gathering and analyzing trace data to identify changes in performance ● By reporting latency data on per URL basis and helping identify bottlenecks.
Platform support	● App Engine standard and flexible environments ● Cloud Run ● Cloud Run functions ● Compute Engine ● GKE ● Non Google Cloud environment

Cloud Trace is a distributed tracing system that collects latency data from your applications and displays it in the Google Cloud console. You can track how requests propagate through your application and receive detailed near-real time performance insights.

Trace automatically analyzes all of your application traces to generate in-depth latency reports to surface performance degradations.

- **Issue detection**
  Trace continuously gathers and analyzes trace data from your project to automatically identify recent changes to the performance of your application. These latency distributions, available through the Analysis Reports feature, can be compared over time or versions. If Trace detects a significant shift in the latency profile of your app, you’re automatically alerted.
- **Identification of performance bottlenecks**
  Cloud Trace help inspect detailed latency information for a single request or view aggregate latency for your entire application. Using the various tools and filters provided, you can quickly find where bottlenecks are occurring and more quickly identify their root cause.Trace is based off the tools used at Google to keep our services running at extreme scale.
- **Broad platform support**
  The language-specific SDKs of Trace can analyze projects that run on VMs (even VMs not managed by Google Cloud). The Trace SDK is available for Java, Node.js, Ruby, and Go. The Trace API can be used to submit and retrieve trace data from any source.



Proprietary + Confidential

Cloud Trace terminologies

Tracing client	Collects spans and sends them to Cloud Trace.
Trace	Trace describes the time it takes an application to complete a single operation.
Span	How long it takes to perform a complete sub-operation.

Google Cloud

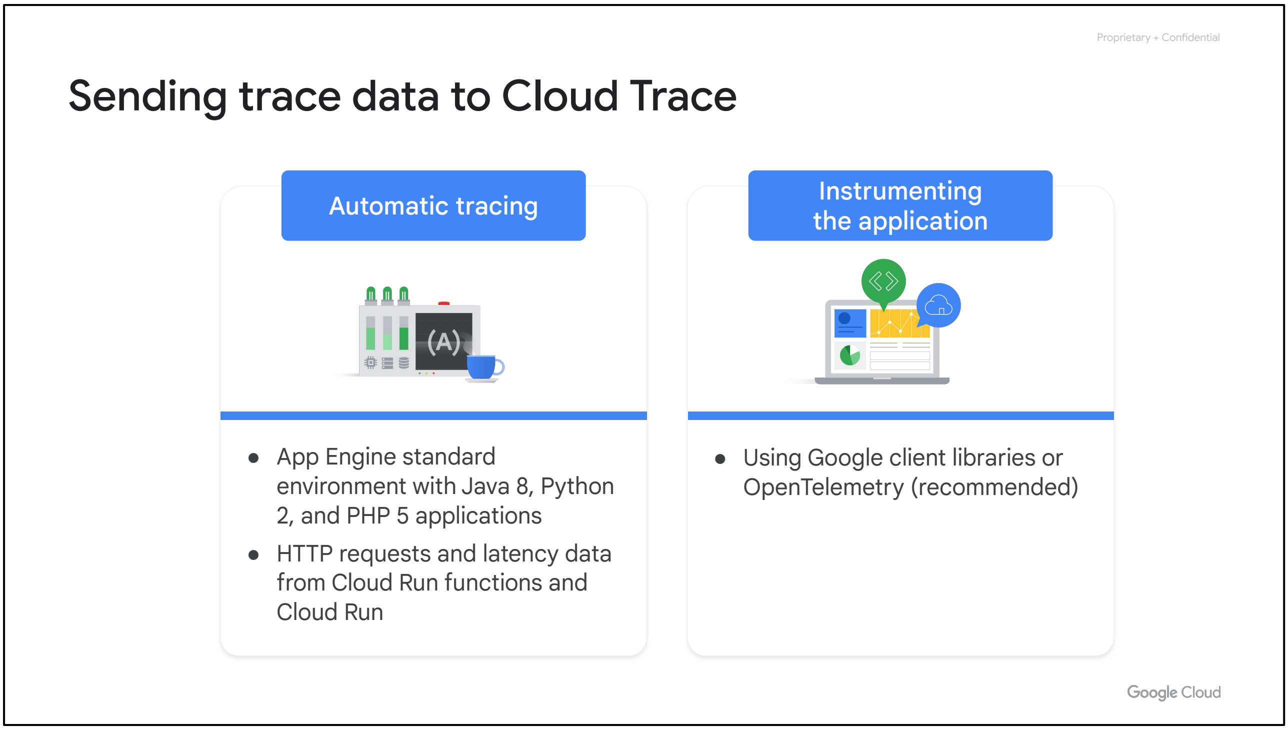
A Trace consists of a tracing client, which collects spans and sends them to Cloud Trace. You can then use the Google Cloud console to view and analyze the data collected by the agent.

A trace describes the time that it takes an application to complete a single operation. A trace is a collection of spans.

A span describes how long it takes to perform a complete suboperation. For example, a trace might describe how long it takes to process an incoming request from a user and return a response. A span might describe how long a particular RPC call requires.

If an OpenCensus library is available for your programming language, you can simplify the process of creating and sending trace data by using OpenCensus. In addition to being simpler to use, OpenCensus implements batching that might improve performance.

If an OpenCensus library doesn't exist, instrument your code by importing the Trace SDK library and using the Cloud Trace API. The Cloud Trace API collects trace data and sends it to your Google Cloud project.



Sending trace data to Cloud Trace

Automatic tracing

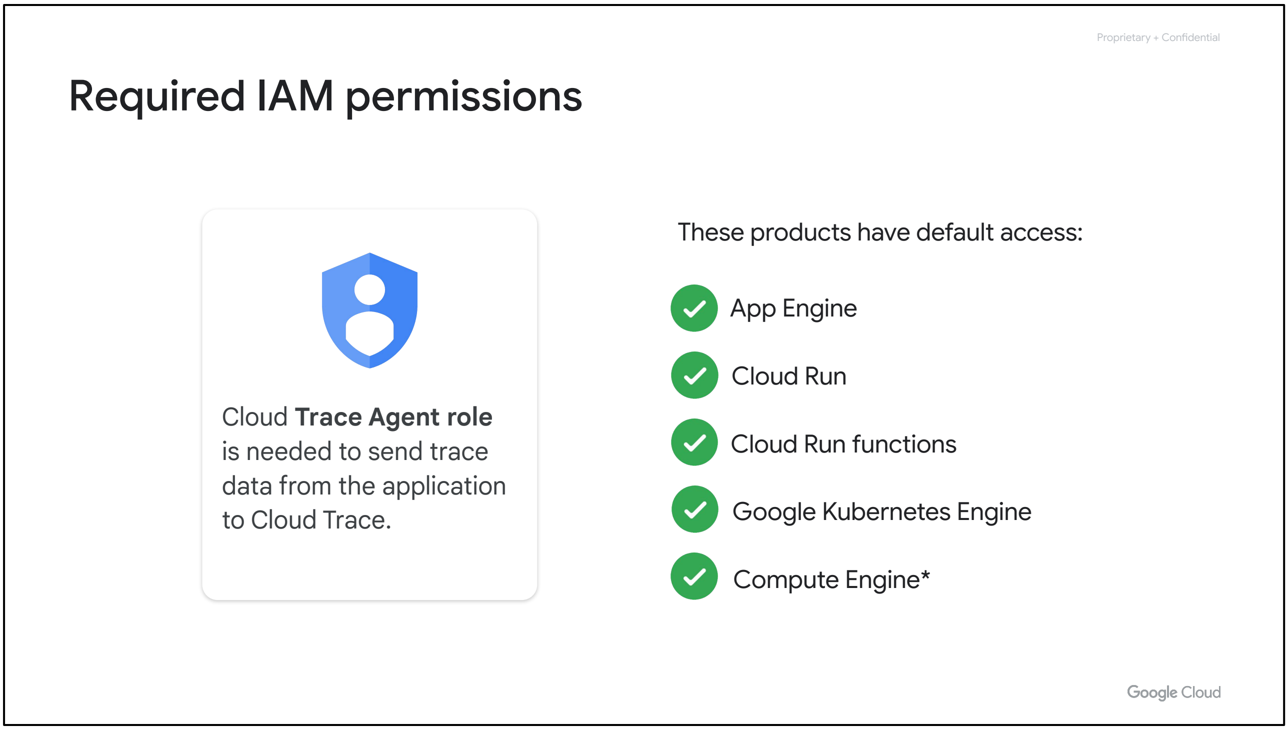
- App Engine standard environment with Java 8, Python 2, and PHP 5 applications
- HTTP requests and latency data from Cloud Run functions and Cloud Run

Instrumenting the application

- Using Google client libraries or OpenTelemetry (recommended)

There are two ways to send trace data to Cloud Trace:

- The first one is automatic tracing. Some configurations support automatic tracing. These include:
  - App Engine standard environment with Java 8, Python 2, and PHP 5 applications
  - HTTP requests and latency data from Cloud Run functions and Cloud Run
- And the second one is by instrumenting the application. You can do this by using Google client libraries or OpenTelemetry, which is the recommended option:



Proprietary + Confidential

Required IAM permissions

Cloud **Trace Agent role** is needed to send trace data from the application to Cloud Trace.

These products have default access:

- ✓ App Engine
- ✓ Cloud Run
- ✓ Cloud Run functions
- ✓ Google Kubernetes Engine
- ✓ Compute Engine*

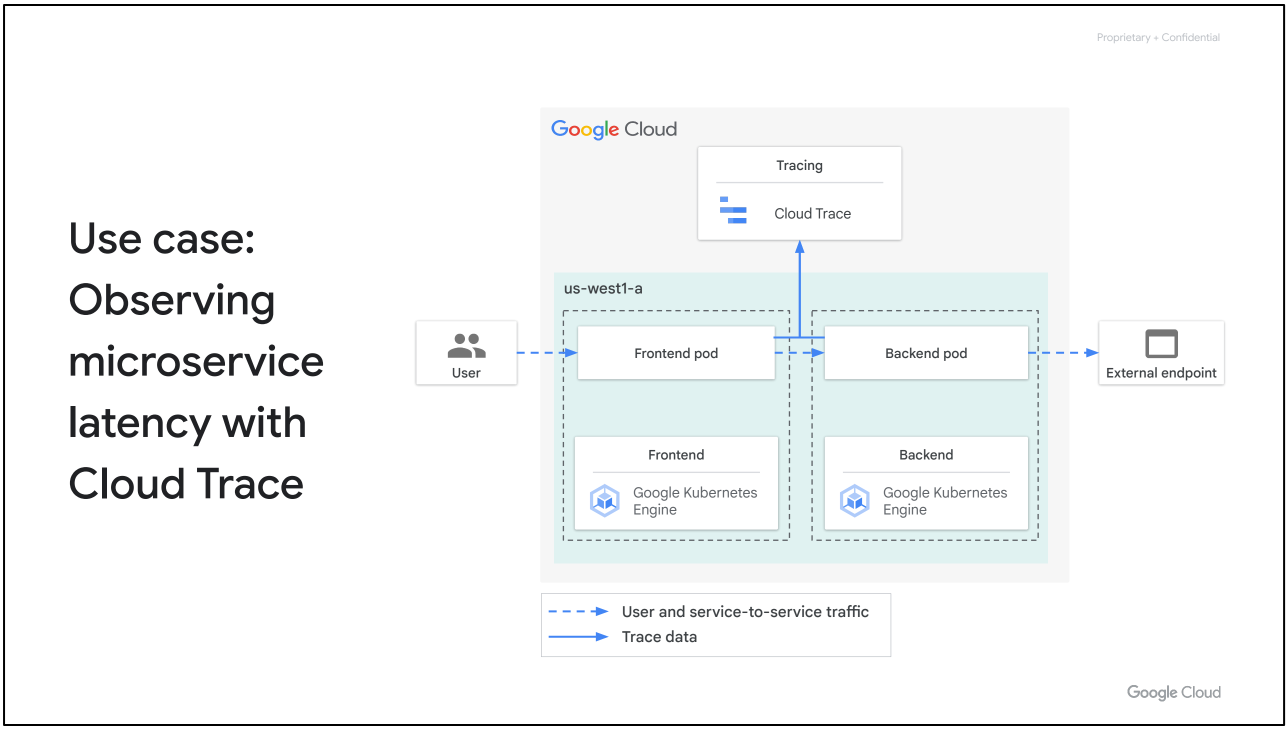
Google Cloud

Trace will need to offload tracing metrics to Google Cloud.

As far as the required IAM permissions are concerned, for external systems, or Compute Engine and GKE environments that don't run under the default service account, ensure that they run under a service account with at least the Cloud Trace Agent role.

App Engine, Cloud Run, Cloud Run functions, Google Kubernetes Engine, and Compute Engine have default access.

*Compute Engine and GKE get that access through the default Compute Engine service account.



Use case: Observing microservice latency with Cloud Trace

Scenario:
Consider a scenario where:

- You have two GKE clusters that that run an app.
- The user traffic is sent to the frontend app on the frontend cluster.
- The frontend pod on the frontend cluster communicates to the backend pod on the backend cluster.
- The backend pod calls an external API endpoint.

The Application:

- The sample app in this tutorial is composed of two microservices written in Go.
- The frontend service accepts HTTP requests on the URL and calls the backend service. The address of the backend service is defined by the BACKEND environment variable. The value of that variable is set in a ConfigMap object that you create.
- The backend service accepts HTTP requests on the URL and makes an outbound call to an external URL as defined in the DESTINATION_URL environment variable. The value of the variable is set through a ConfigMap object. After the external call is completed, the backend service returns the HTTP status call, for example 200, back to the caller.

Challenge:
A challenge with tracing in distributed systems is that information about the original frontend request isn't automatically or inherently carried forward when subsequent

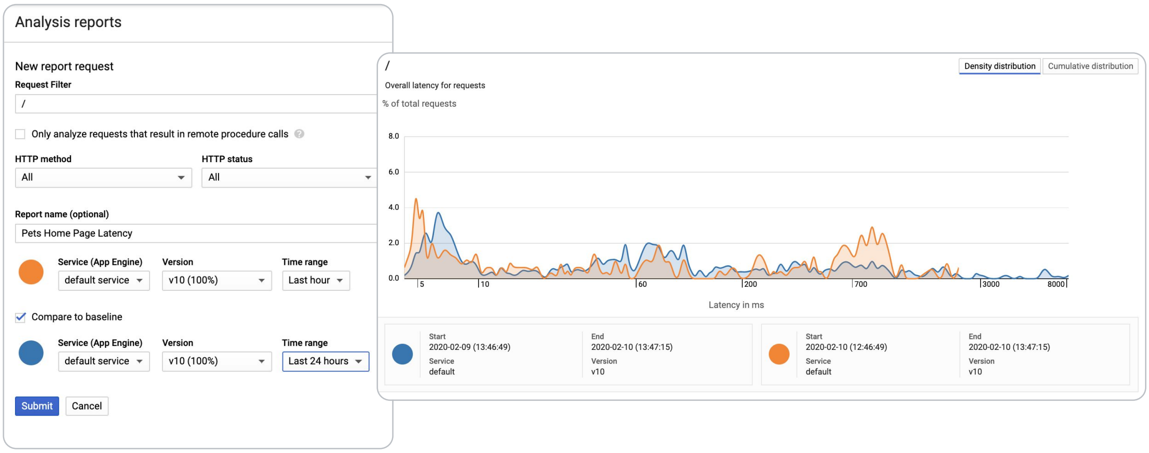
requests are made to various backend services.

Using Trace and Go:
In Go, you can make requests with context (the cluster IP address and credentials), meaning additional information is loaded in the HTTP header. The concept of context is extended with OpenCensus to include span context, where you can include information about the parent span with each subsequent request. You can then append child spans to compose the overall trace, so you can see how the user request traversed the system and was eventually served back to the user.

This trace contains three spans with the following names:

- The incoming call span captures the end-to-end latency observed by the client.
- The call to backend span captures the latency of the backend call observed by the frontend.
- The call remote endpoint span captures the latency of the external endpoint call observed by the backend.

Analysis reports show requests over time and compares versions and timespans



Analysis reports

New report request

Request Filter: /

□ Only analyze requests that result in remote procedure calls

HTTP method: All

HTTP status: All

Report name (optional): Pets Home Page Latency

● Service (App Engine): default service — Version: v10 (100%) — Time range: Last hour

☑ Compare to baseline

● Service (App Engine): default service — Version: v10 (100%) — Time range: Last 24 hours

Submit | Cancel

/

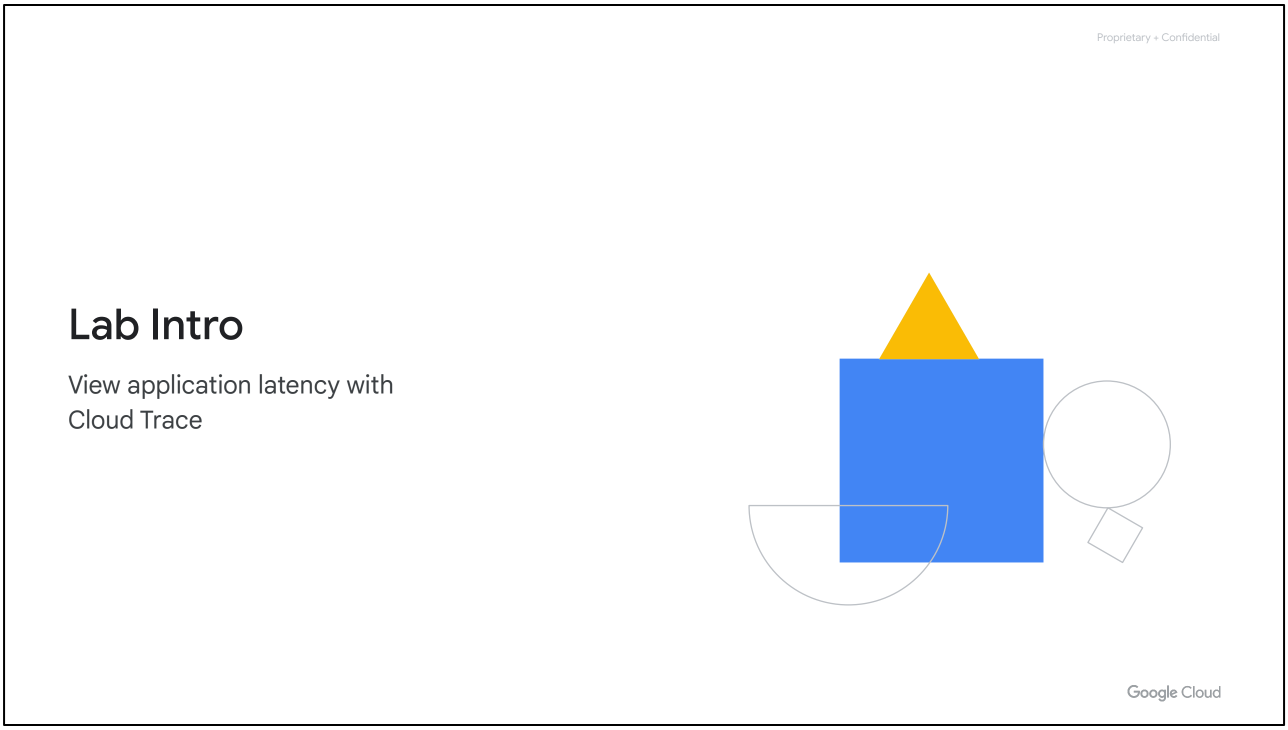
Overall latency for requests

●	Start 2020-02-09 (13:46:49) Service default	End 2020-02-10 (13:47:15) Version v10	
●	Start 2020-02-10 (12:46:49) Service default	End 2020-02-10 (13:47:15) Version v10	

Analysis reports in Trace show you an overall view of the latency for all requests, or a subset of requests, to your application. Analysis reports will be similar to the daily report viewed on the **Trace Overview** main page.

Custom reports can be created for particular request URIs and other search qualifiers.

The report can show results as both a density distribution, as in the screenshot, or as a cumulative distribution.



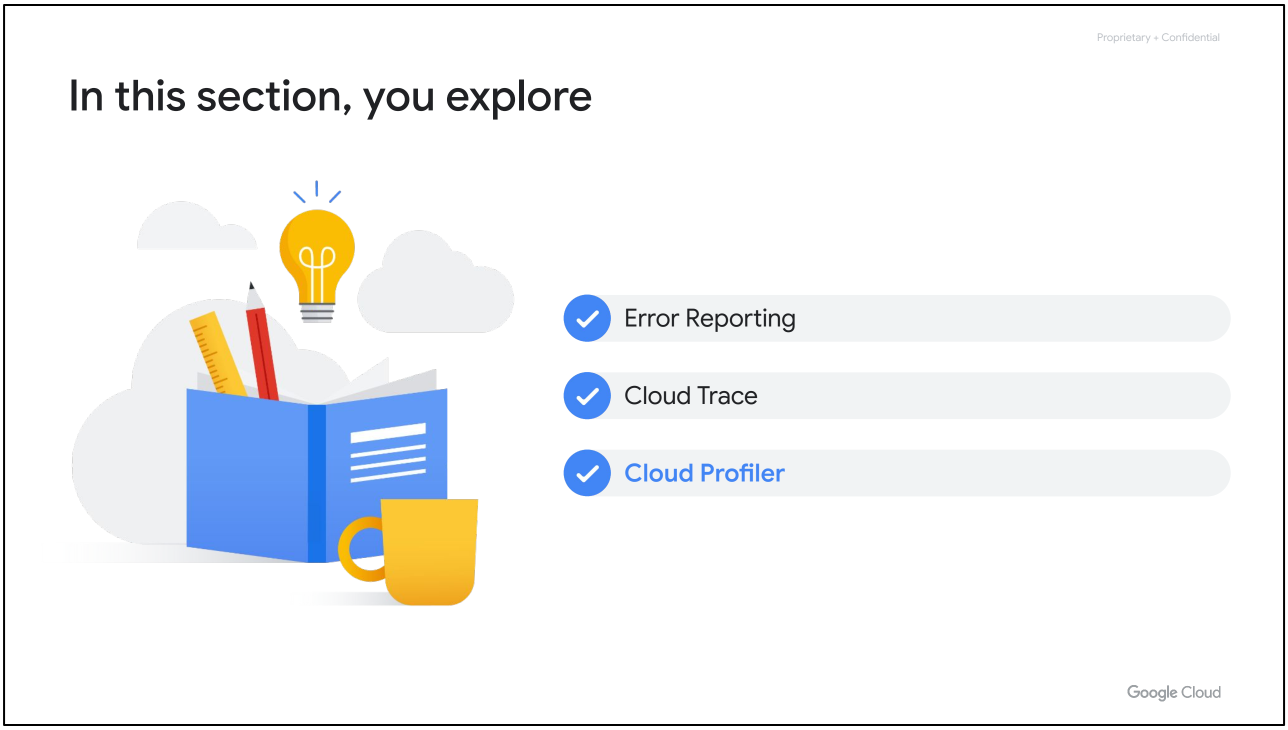
Lab Intro

View application latency with Cloud Trace

Let’s cover a quick activity on how to view application latency by using Cloud Trace.

In this lab you will learn how to use Cloud Trace by doing the following:

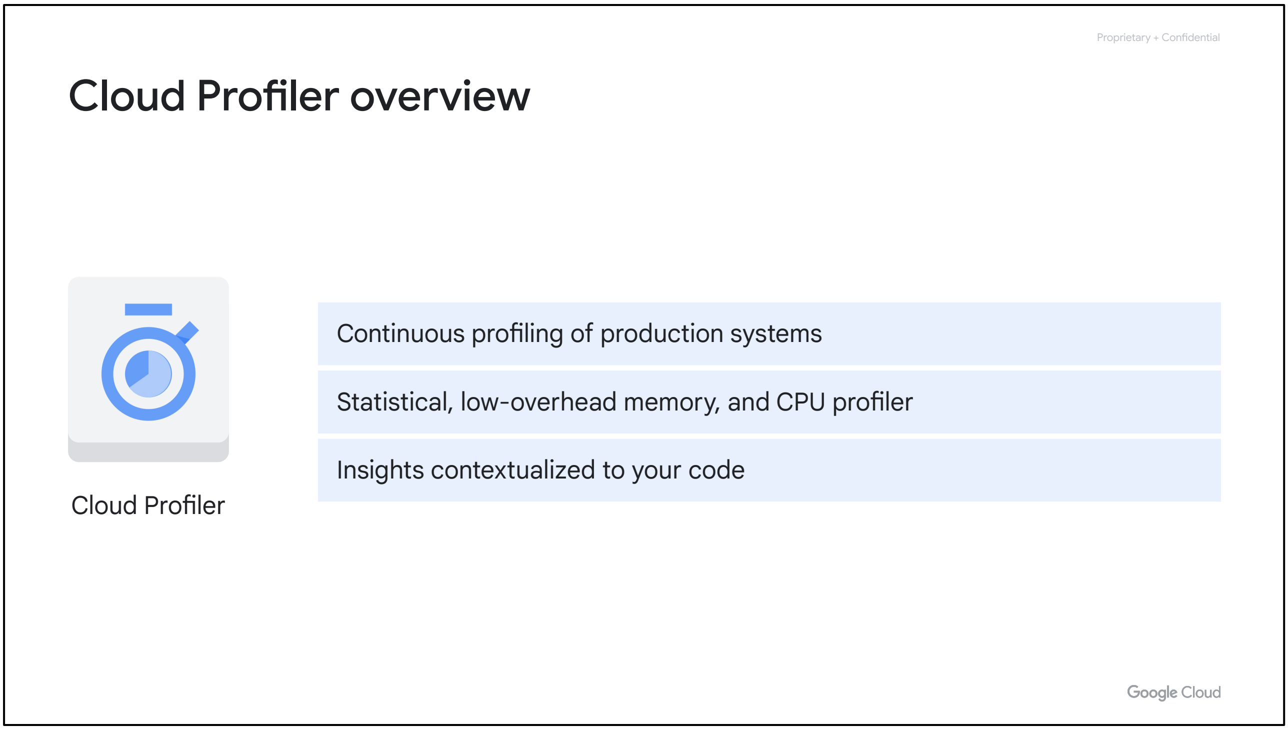
- Deploy a sample application to a Google Kubernetes Engine (GKE) cluster.
- Create a trace by sending an HTTP request to the sample application.
- Use the Cloud Trace interface to view the latency information of the trace you created.



In this section, you explore

- ✓ Error Reporting
- ✓ Cloud Trace
- ✓ **Cloud Profiler**

Finally, let’s look at Cloud Profiler.



Cloud Profiler overview

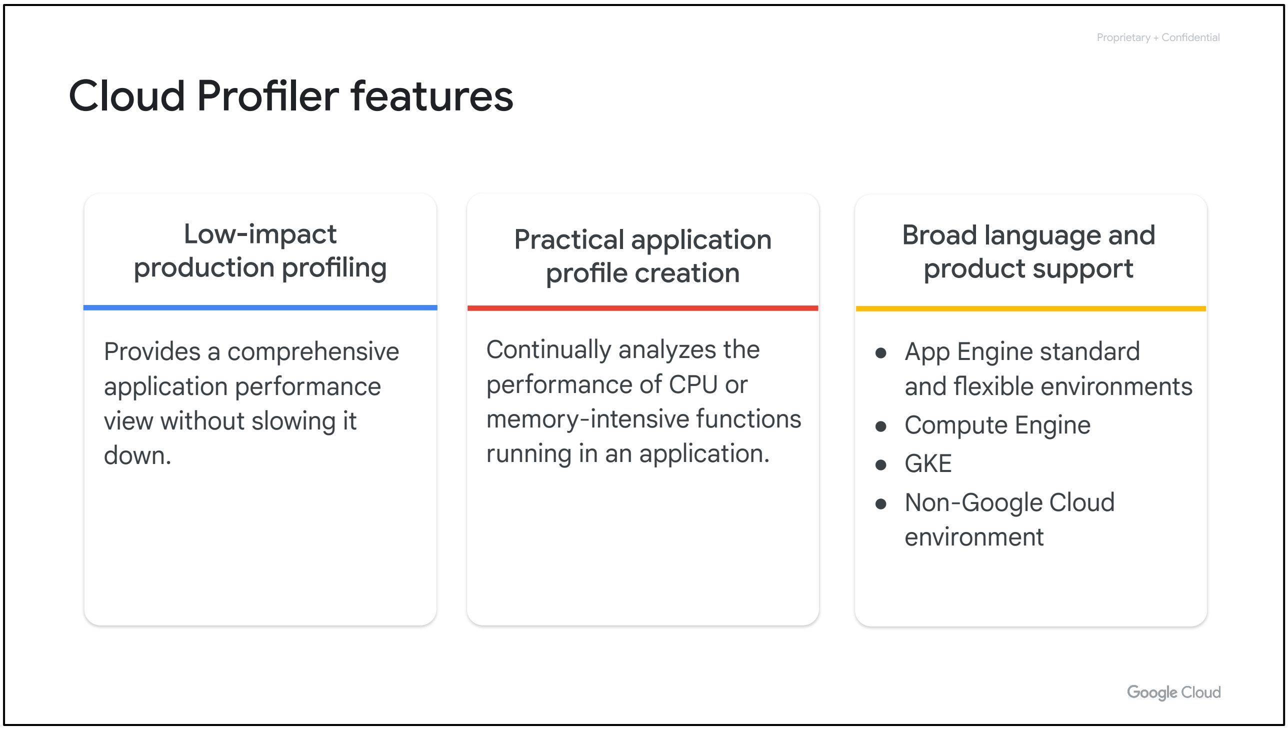
Cloud Profiler

- Continuous profiling of production systems
- Statistical, low-overhead memory, and CPU profiler
- Insights contextualized to your code

Understanding the performance of production systems is notoriously difficult. Attempting to measure performance in test environments usually fails to replicate the pressures on a production system.

Continuous profiling of production systems is an effective way to discover where resources like CPU cycles and memory are consumed when the service operates in its working environment.

Cloud Profiler is a statistical, low-overhead profiler that continuously gathers CPU usage and memory-allocation information from your production applications. It attributes that information to the source code that generated it, which helps you identify the parts of your application that are consuming the most resources. The insights provided illuminate the performance of your application characteristics.



Proprietary + Confidential

Cloud Profiler features

Low-impact production profiling

Provides a comprehensive application performance view without slowing it down.

Practical application profile creation

Continually analyzes the performance of CPU or memory-intensive functions running in an application.

Broad language and product support

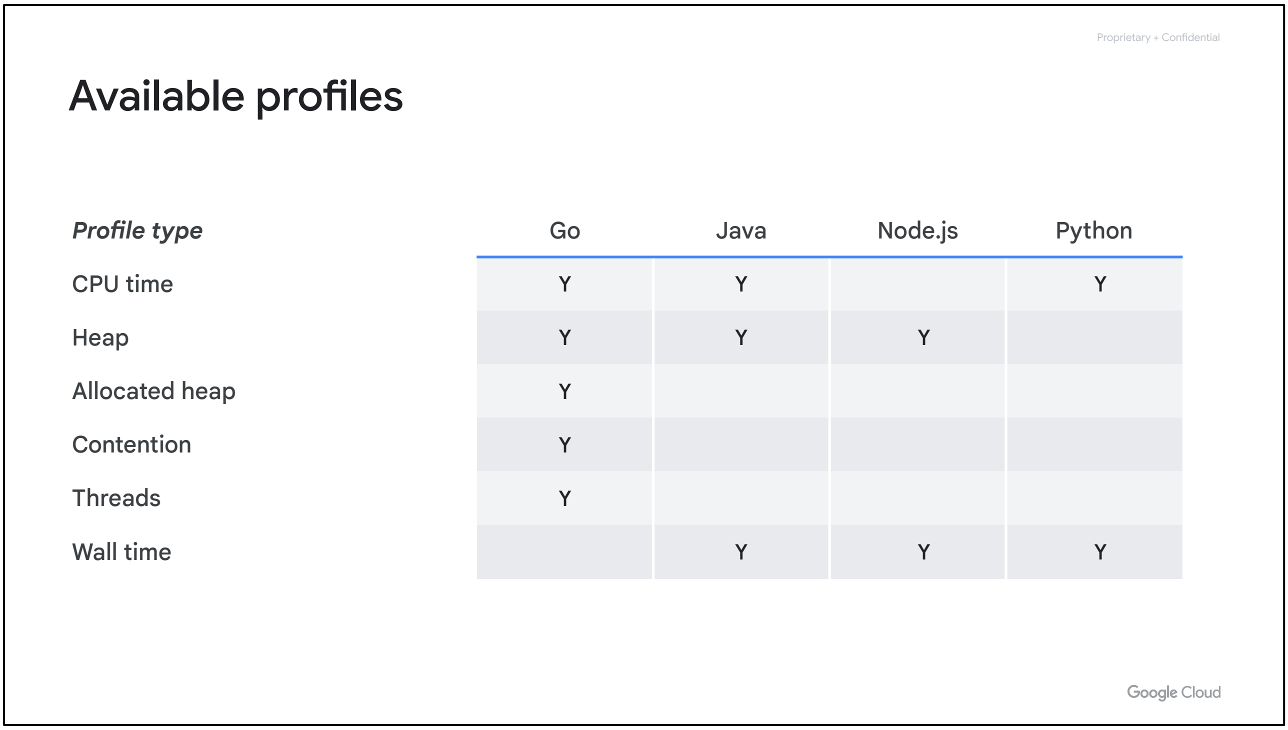
- App Engine standard and flexible environments
- Compute Engine
- GKE
- Non-Google Cloud environment

Google Cloud

Low-impact production profiling
Although it's possible to measure code performance in development environments, the results don't often show what happens in production. Many production profiling techniques slow down code execution or can only inspect a small subset of a codebase. Cloud Profiler uses statistical techniques and very low-impact instrumentation that runs across all production application instances. A comprehensive application performance view is provided without slowing it down.

Practical application profile creation
Poorly performing code increases latency and costs for web applications and services every day, without anyone knowing or doing anything about it. Cloud Profiler changes this situation by continually analyzing the performance of CPU or memory-intensive functions that run in an application. Cloud Profiler presents the call hierarchy and resource consumption of the corresponding function in an interactive flame graph. This graph helps developers figure out which paths are consuming the most resources and what are the different ways your code is called.

Broad language and product support
Cloud Profiler enables developers to analyze applications that run anywhere, including Google Cloud and other on-premises or cloud platforms that support Java, Go, Node.js, and Python. You can find a full explanation of language compatibility in the documentation.



Available profiles

Profile type	Go	Java	Node.js	Python
CPU time	Y	Y		Y
Heap	Y	Y	Y	
Allocated heap	Y			
Contention	Y			
Threads	Y			
Wall time		Y	Y	Y

The profiling types available vary by language. Check the Cloud Profiler documentation for the most recent options.

For the CPU metrics, you will find the following:

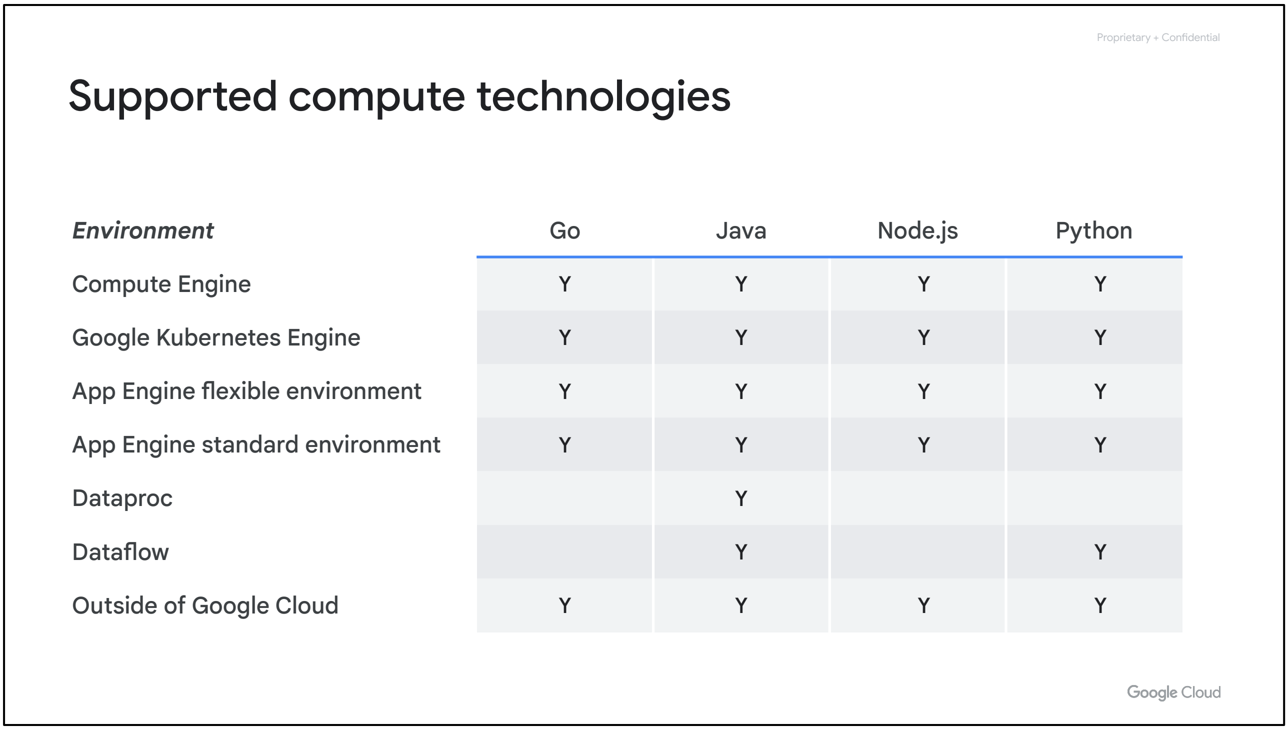
- CPU time is the time that the CPU spends executing a block of code. The time it was waiting or processing instructions for something else is not included.
- Wall time is the time that it takes to run a block of code, including all wait time, including that for locks and thread synchronization. The wall time for a block of code can never be less than the CPU time.

For heap, you have the following:

- **Heap** is the amount of memory allocated in the heap of the program when the profile is collected.
- **Allocated heap** is the total amount of memory that was allocated in the heap of the program. Allocated heap includes memory that has been freed and is no longer in use.

And for threads you have:

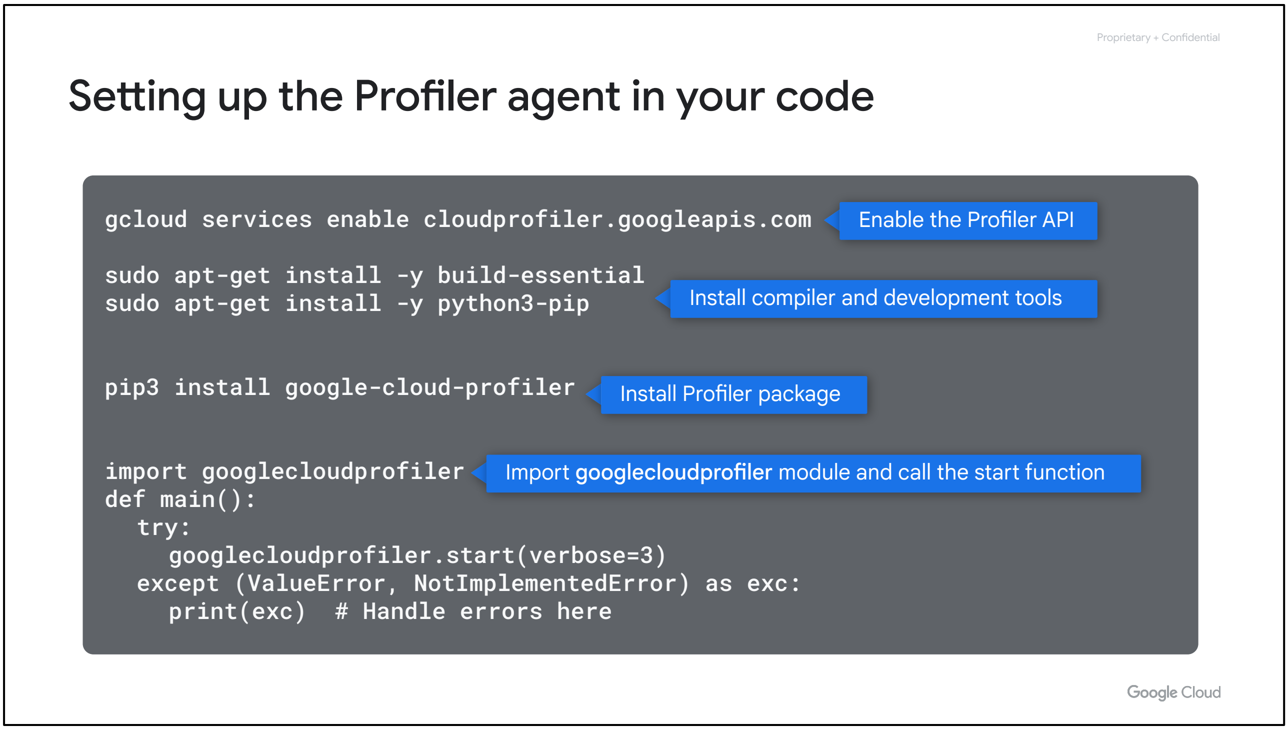
- **Contention,** which provides information about threads stuck waiting for other threads.
- **Threads**, which contain thread counts.



Supported compute technologies

Environment	Go	Java	Node.js	Python
Compute Engine	Y	Y	Y	Y
Google Kubernetes Engine	Y	Y	Y	Y
App Engine flexible environment	Y	Y	Y	Y
App Engine standard environment	Y	Y	Y	Y
Dataproc		Y		
Dataflow		Y		Y
Outside of Google Cloud	Y	Y	Y	Y

Profiler instruments applications that run in most Google and non-Google compute technologies. Note that Windows guest OS is not supported.



Setting up the Profiler agent in your code

```
gcloud services enable cloudprofiler.googleapis.com

sudo apt-get install -y build-essential
sudo apt-get install -y python3-pip


pip3 install google-cloud-profiler


import googlecloudprofiler
def main():
  try:
    googlecloudprofiler.start(verbose=3)
  except (ValueError, NotImplementedError) as exc:
    print(exc)  # Handle errors here
```

Enable the Profiler API

Install compiler and development tools

Install Profiler package

Import **googlecloudprofiler** module and call the start function

Like with other Google application performance management products, the exact setup steps vary by language, so check the documentation to find more information. Here, we are sticking with our Python application, which will run on App Engine.

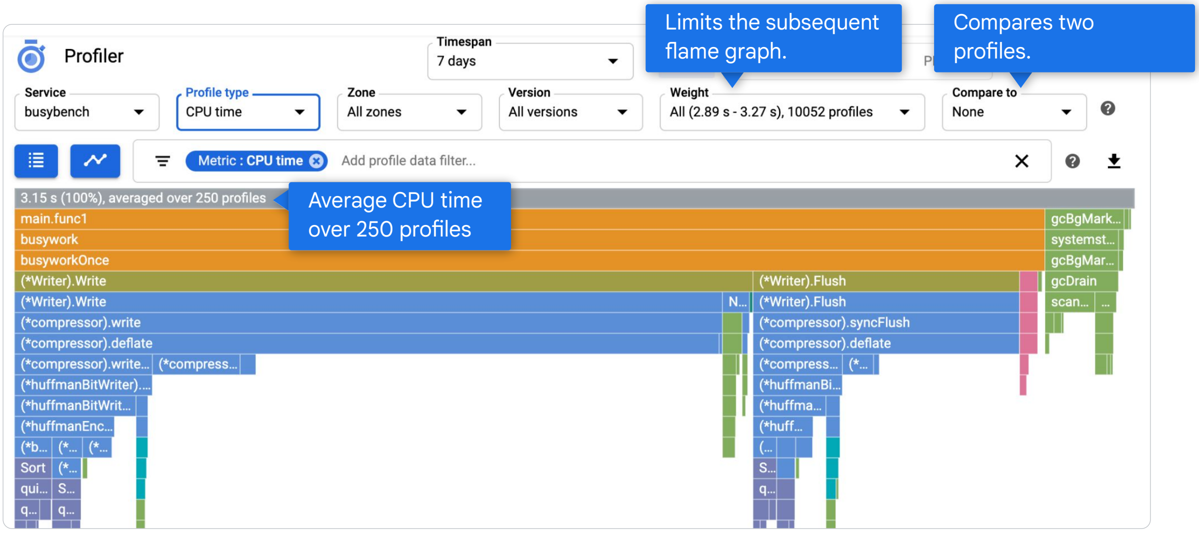
Before you start, ensure that the **Profiler API** is enabled in your project.

Start by importing the *googlecloudprofiler* package.

Install the C/C++ compiler and development tools, pip, and Profiler package.

Then, early as possible in your code, start the profiler. In this example, we are setting the logging level (verbose) to 3, or debug level. That setting will log all messages. The default would be 0 or error only.

Profiler interface



Profiler

Timespan: 7 days

Service: busybench

Profile type: CPU time

Zone: All zones

Version: All versions

Weight: All (2.89 s - 3.27 s), 10052 profiles

Compare to: None

Metric : CPU time

Add profile data filter...

3.15 s (100%), averaged over 250 profiles

Limits the subsequent flame graph.

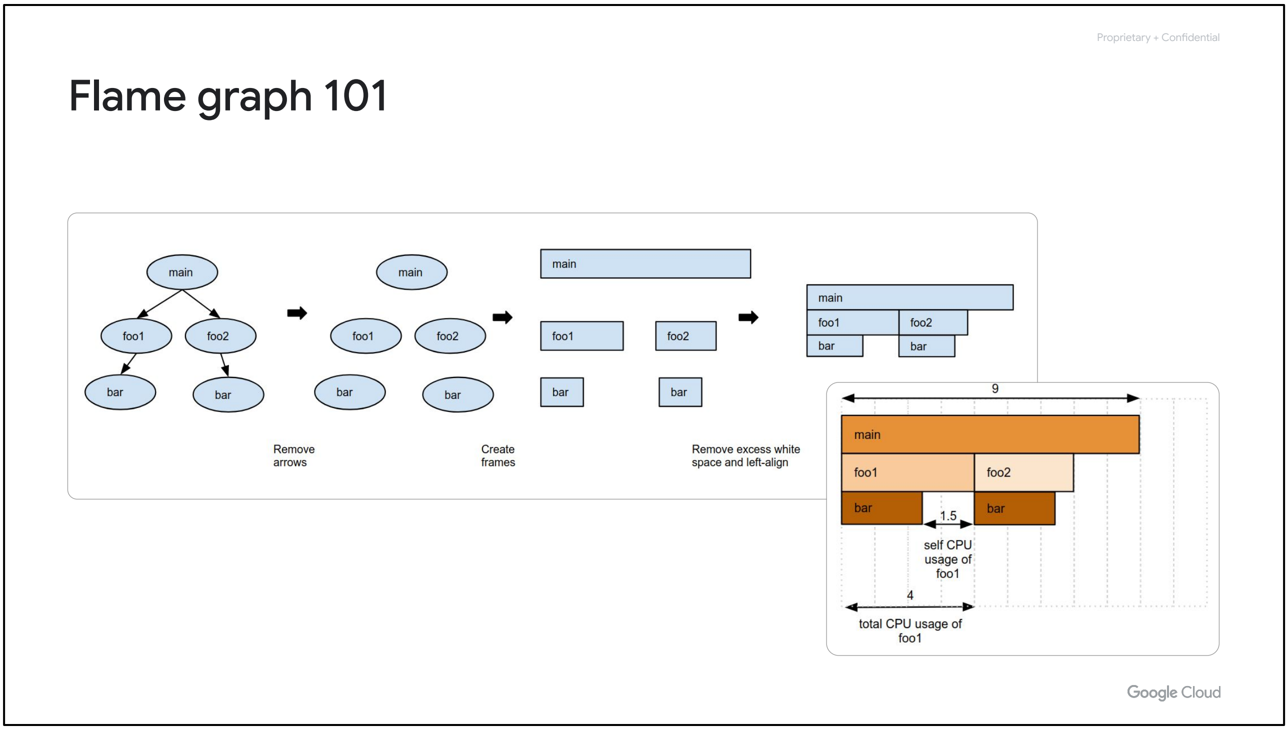
Compares two profiles.

Average CPU time over 250 profiles

At the top of the Profiler interface, you can select the profile that you want to analyze. You can select by options including **Timespan**, **Service**, **Profile type**, **Zone**, and **Version**.

The **Weight** will limit the subsequent flame graph to particular peak consumptions. Top 5%, for example.

Compare to allows the comparison of two profiles. Profiler then randomly selects a maximum of 250 profiles from this set, and uses those to construct the flame graph.



Proprietary + Confidential

Flame graph 101

Remove arrows

Create frames

Remove excess white space and left-align

self CPU usage of foo1

total CPU usage of foo1

Google Cloud

Cloud Profiler displays profiling data by using [Flame Graphs](). Unlike trees and standard graphs, flame graphs use of screen space efficiently by representing a large amount of information in a compact and readable format.

Look at this example. We have a basic application with a *main* method, which calls *foo1*, which in turn calls *bar*. Then *main* calls *foo2*, which also calls *bar*.

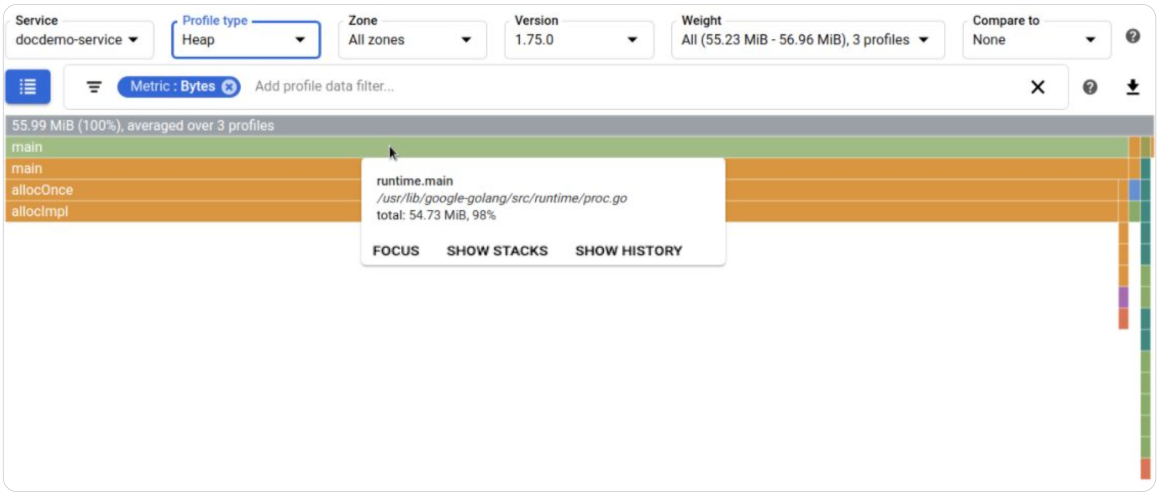
As you move through the graphic left to right, you can see how the information is collapsed. First, by removing arrows, then by creating frames, and finally by removing spaces and left-aligning.

In the bottom view, you see the result as as it appears in the Profiler. If you look at CPU time, then you can see that the *main* method takes a total of *9* seconds.

Beneath the *main* bar, you can see how that CPU time was spent: some in *main* itself, but most in the calls to *foo1* and *foo2*. And most of the *foo* time was spent in *bar*.

So, if we could make *bar* faster, we could really save some time in *main*.

Point to a frame



Service: docdemo-service

Profile type: Heap

Zone: All zones

Version: 1.75.0

Weight: All (55.23 MiB - 56.96 MiB), 3 profiles

Compare to: None

Metric : Bytes

Add profile data filter...
55.99 MiB (100%), averaged over 3 profiles
main
main
allocOnce
allocImpl

runtime.main
/usr/lib/google-golang/src/runtime/proc.go
total: 54.73 MiB, 98%

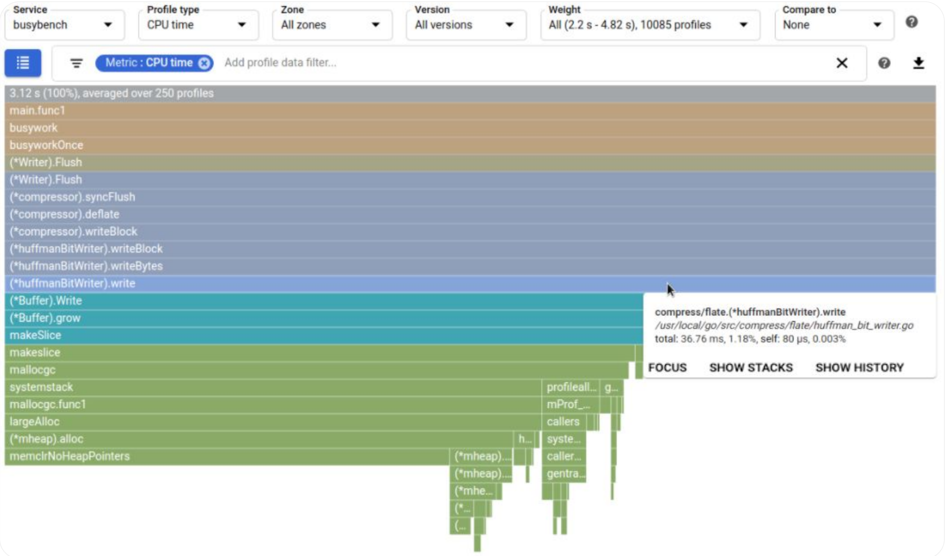
FOCUS
SHOW STACKS
SHOW HISTORY

Here we have a full example.

When you hold the pointer over a frame, a tooltip opens and displays additional information, which includes:

- The function name
- The source file location
- And some metric consumption information

Selecting a frame



Service: busybench
Profile type: CPU time
Zone: All zones
Version: All versions
Weight: All (2.2 s - 4.82 s), 10085 profiles
Compare to: None

Metric : CPU time
Add profile data filter...

3.12 s (100%), averaged over 250 profiles

compress/flate.(*huffmanBitWriter).write
/usr/local/go/src/compress/flate/huffman_bit_writer.go
total: 36.76 ms, 1.18%, self: 80 µs, 0.003%

If you click a frame, the graph is redrawn, which makes the call stack of the selected method more visible.

Knowledge Check

Quiz | Question 1

Question

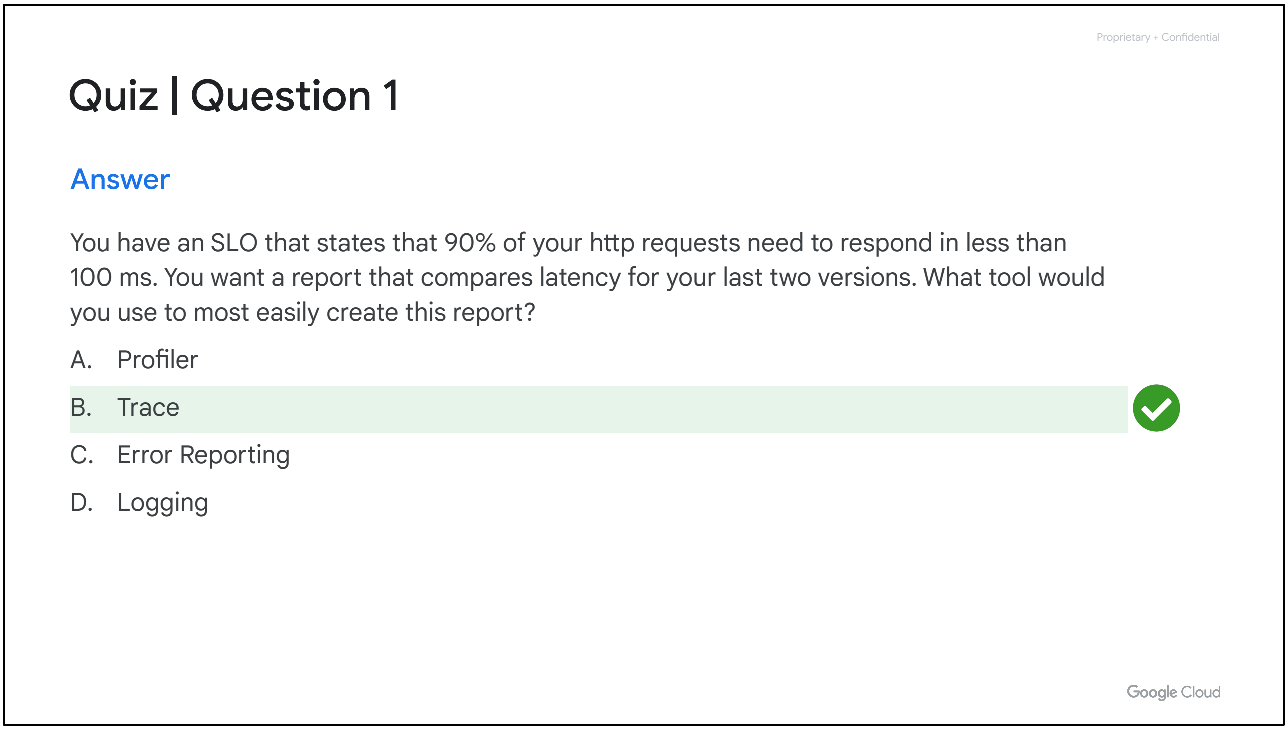
You have an SLO that states that 90% of your http requests need to respond in less than 100 ms. You want a report that compares latency for your last two versions. What tool would you use to most easily create this report?

A. Profiler

B. Trace

C. Error Reporting

D. Logging



Quiz | Question 1

Answer

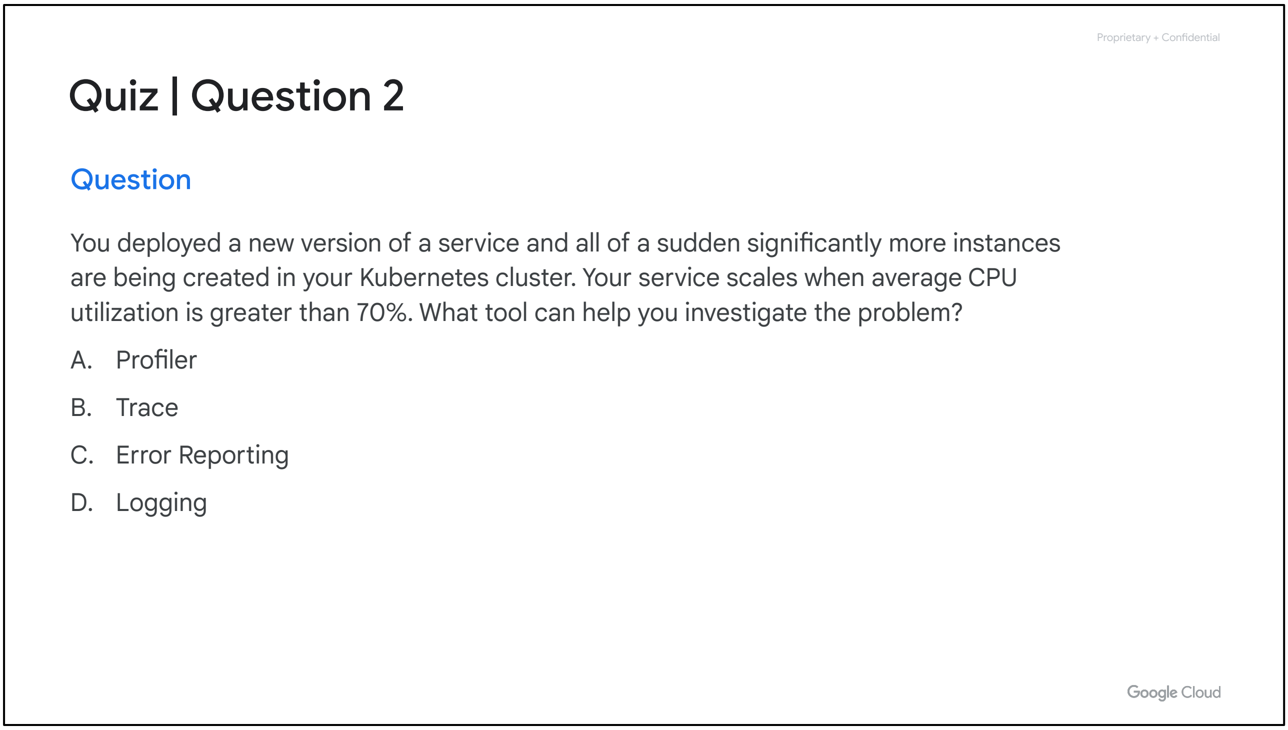
You have an SLO that states that 90% of your http requests need to respond in less than 100 ms. You want a report that compares latency for your last two versions. What tool would you use to most easily create this report?

A. Profiler

B. Trace ✓

C. Error Reporting

D. Logging



Quiz | Question 2

Question

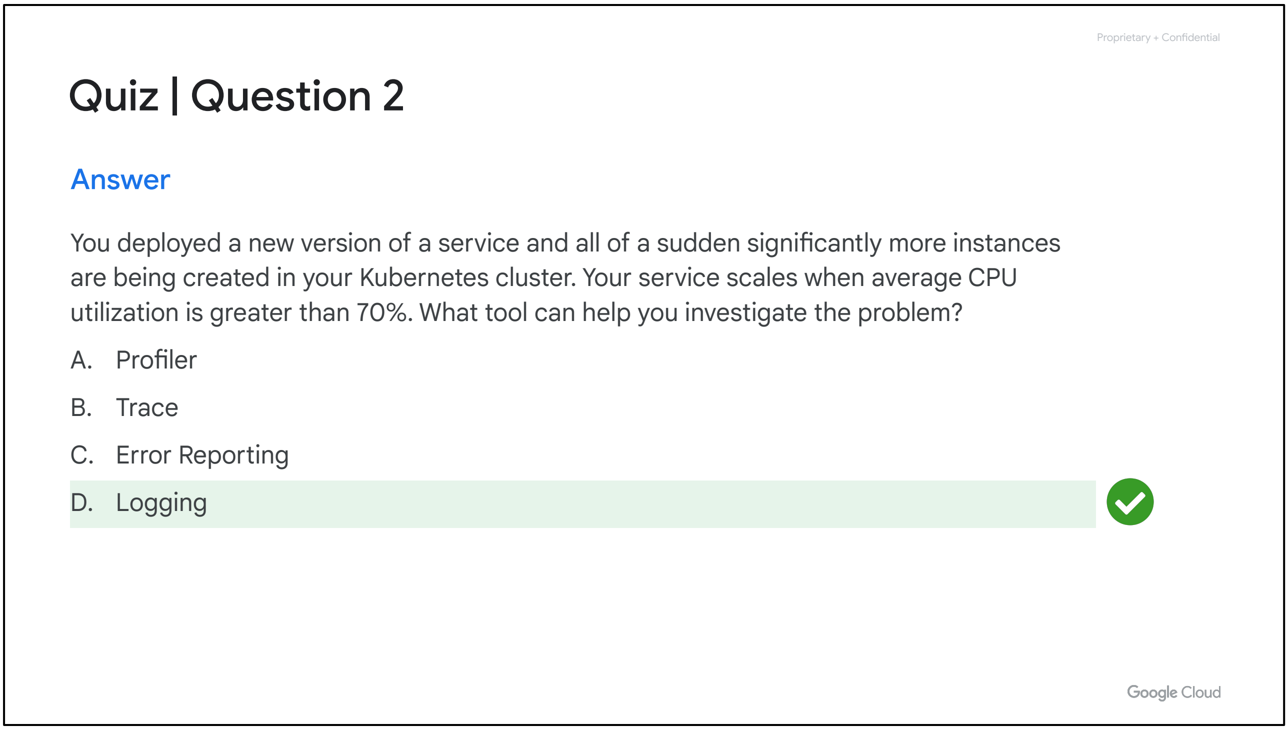
You deployed a new version of a service and all of a sudden significantly more instances are being created in your Kubernetes cluster. Your service scales when average CPU utilization is greater than 70%. What tool can help you investigate the problem?

A. Profiler

B. Trace

C. Error Reporting

D. Logging



Quiz | Question 2

Answer

You deployed a new version of a service and all of a sudden significantly more instances are being created in your Kubernetes cluster. Your service scales when average CPU utilization is greater than 70%. What tool can help you investigate the problem?

A. Profiler

B. Trace

C. Error Reporting

D. Logging ✓

Quiz | Question 3

Question

Which profile signifies the amount of memory allocated in the heap of the program?

A. Heap

B. Allocated heap

C. CPU time

D. Wall time

Quiz | Question 3

Answer

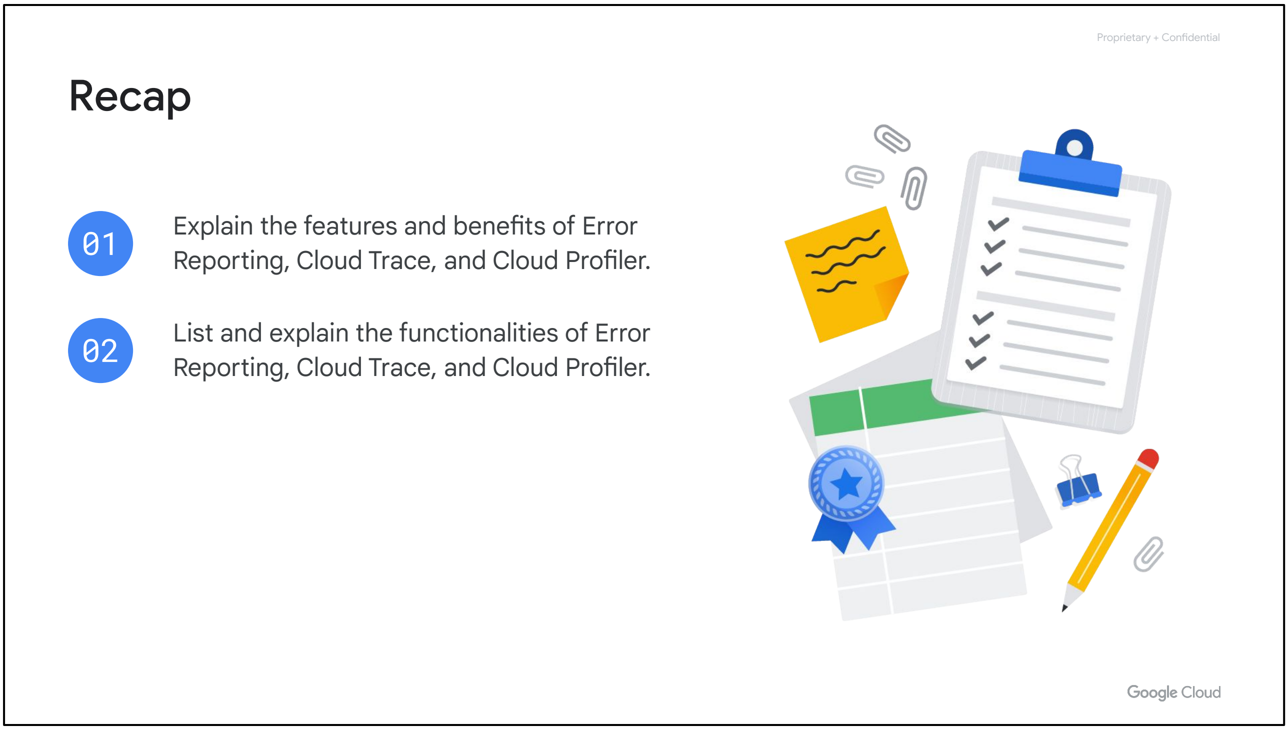
Which profile signifies the amount of memory allocated in the heap of the program?

A. Heap ✓

B. Allocated heap

C. CPU time

D. Wall time



Proprietary + Confidential

Recap

01 Explain the features and benefits of Error Reporting, Cloud Trace, and Cloud Profiler.

02 List and explain the functionalities of Error Reporting, Cloud Trace, and Cloud Profiler.

Google Cloud

In this module, you learned to:

- Explain the features and benefits of Error Reporting, Cloud Trace, and Cloud Profiler.
- List and explain the functionalities of the Error Reporting, Cloud Trace, and Cloud Profiler components.