



---

## Debugging, Monitoring, and Performance Tuning

Google Cloud's operations suite combines metrics, logs, and metadata, whether you're running on Google Cloud, Amazon Web Services, on-premises infrastructure, or a hybrid cloud. You can quickly understand service behaviors and issues, from a single comprehensive view of your environment, and take action if needed.

## Google Cloud's operations suite, a multi-cloud service



### Cloud Logging

- Platform/system/app logs
- Log search/view/filter
- Logs-based metrics



### Cloud Monitoring

- Platform/system/app metrics
- Uptime/health checks
- Dashboards
- Alerts



### Error Reporting

- Error notifications
- Error dashboard



### Cloud Trace

- Latency reporting
- Per-URL latency sampling



### Cloud Debugger

- Production debug snapshots
- Conditional snapshots
- IDE integration



### Cloud Profiler

Low-impact profiling of applications in production

Let's start by looking at Google Cloud's operations suite, which includes features such as Cloud Logging, Cloud Monitoring, Error Reporting, Cloud Trace, and Cloud Debugger. These diagnostic features are well-integrated with each other. This helps you connect and correlate diagnostics data easily.

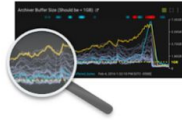
Cloud Profiler uses statistical techniques, and low-impact instrumentation that runs across all production application instances to provide a complete picture of an application's performance without slowing it down. Cloud Profiler helps you identify and eliminate potential performance issues.

## Cloud Monitoring enables you to increase application reliability



### Monitor Google Cloud, AWS, and multi-cloud environments

Get the insight that you need with minimal configuration. Monitor hosted services and cloud architectures.



### Identify trends and prevent issues

Visualize trends via flexible charts and dashboards. Identify risks using scoring, anomaly detection, and prediction.



### Reduce monitoring overhead

Spend less time correlating metrics, alerts, and logs across disparate systems. Don't worry about scaling tools.



### Improve signal-to-noise

Reduce false positives and alert fatigue with advanced alerting designed for modern distributed systems.

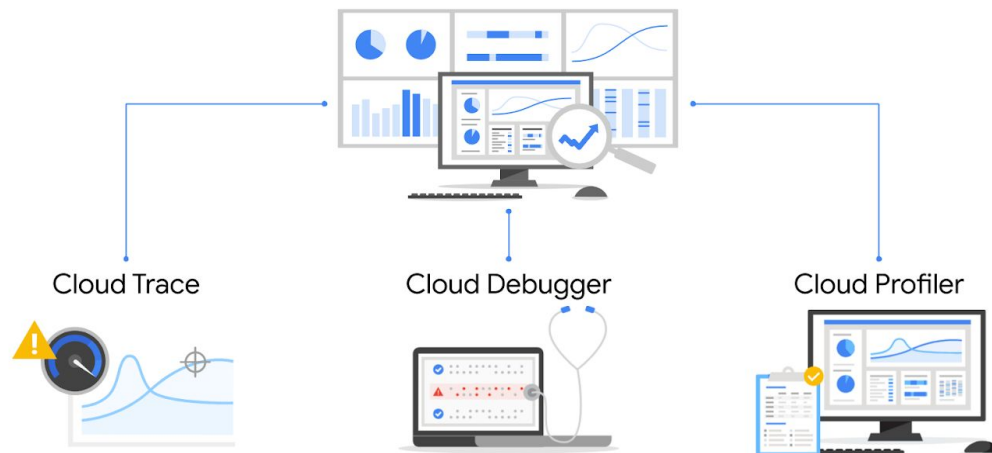


### Fix problems faster

Uptime and health checks notify you quickly when endpoints become inaccessible to your users. Drill down from alerts to dashboards to logs and traces to get to the root cause quickly.

Cloud Monitoring helps increase reliability by giving users the ability to monitor Google Cloud and multi-cloud environments to identify trends and prevent issues. With Cloud Monitoring, you can reduce monitoring overhead and improve your signal-to-noise ratio, allowing you to detect and fix problems faster.

## Application Performance Management (APM) tools



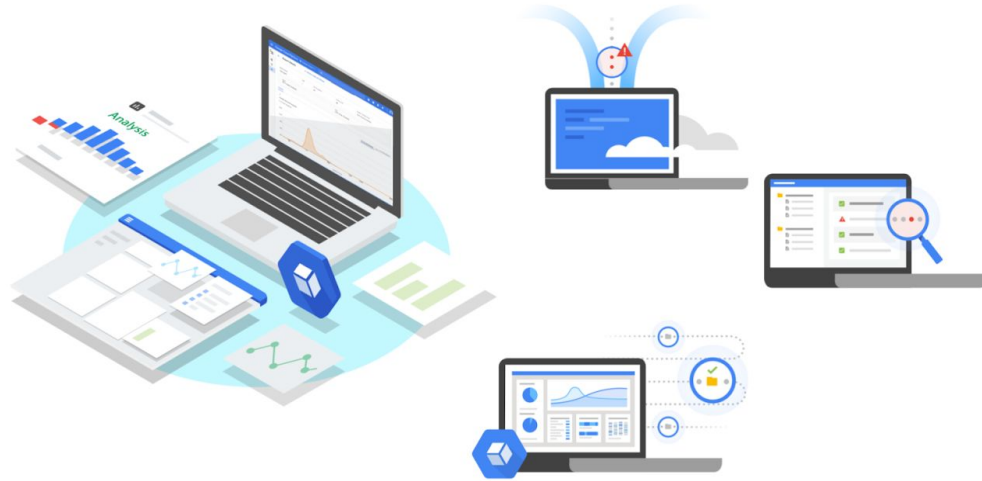
Let's talk about managing performance with Google Cloud's Application Performance Management, or APM tools.

The three products that make up Application Performance Management are: Cloud Trace, Cloud Profiler, and Cloud Debugger. APM includes advanced tools to help developers reduce latency and cost for every application.

As a developer, how do you make your applications faster and more reliable? By understanding in detail how they behave in production. APM uses some of the same tools used by Google's own Site Reliability Engineering (SRE) teams, giving you insights into how your code runs. This lets you take action to optimize code and fix problems, regardless of the cloud you are using.

All of the APM tools work with code and applications running on any cloud, or even on-premises infrastructure. No matter where you run your application, you now have a consistent, reasonably priced APM toolkit to monitor and manage application performance. Google charges for APM tools based only on the amount of data collected. Generous free tiers make APM accessible for any project.

## Cloud Trace - Distributed tracing for everyone



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. Cloud Trace automatically analyzes all of your application's traces to generate in-depth latency reports, surfacing performance degradations. Cloud Trace can capture traces from all of your VMs, containers, and App Engine projects.

Using Trace, you can 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 causes. Cloud Trace is based on the tools used to keep Google's services running at extreme scale.

Trace continuously gathers and analyzes trace data from your project to automatically identify recent changes to your application's performance. These latency distributions, available through the Analysis Reports feature, can be compared over time or versions, and Cloud Trace will automatically alert you if it detects a significant shift in your app's latency profile.

Cloud Trace's language-specific SDKs can analyze projects running on VMs, even those not running in Google Cloud. The Trace SDK is currently available for Java, Node.js, Ruby, and Go, and the Trace API can be used to submit and retrieve trace data from any source. A Zipkin collector is also available, which allows Zipkin tracers to submit data to Cloud Trace. Traces are automatically captured for projects running on App Engine.

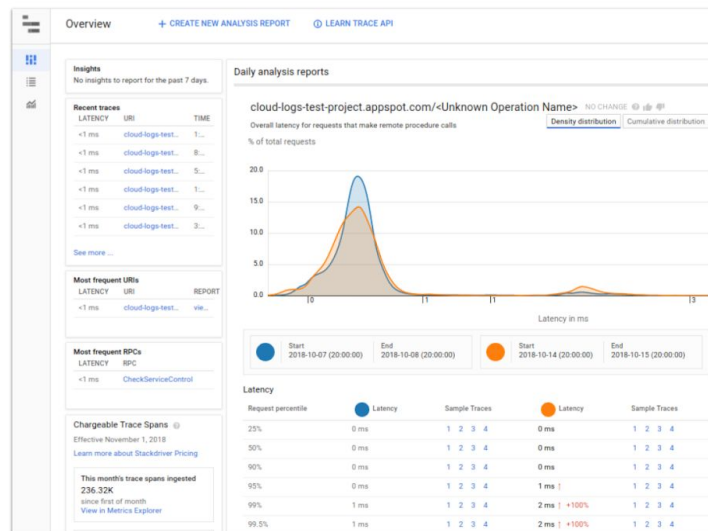
## Cloud Trace can help answer questions

- How long does it take my application to handle a given request?
- Why is it taking my application so long to handle a request?
- Why do some of my requests take longer than others?
- What is the overall latency of requests to my application?
- Has latency for my application increased or decreased over time?
- What can I do to reduce application latency?
- What are my application's dependencies?

Because Cloud Trace collects latency data from App Engine, HTTPS load balancers, and applications instrumented with the Cloud Trace API, it can help you answer the following questions:

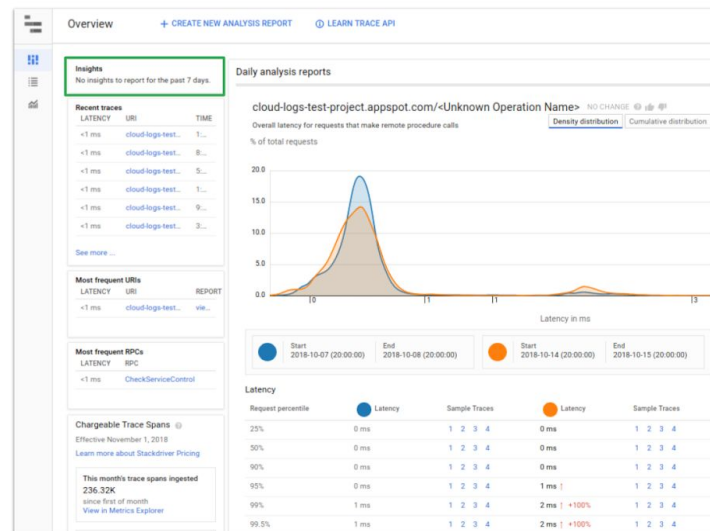
- 
- How long does it take my application to handle a given request?
- Why is it taking my application so long to handle a request?
- Why do some of my requests take longer than others?
- What is the overall latency of requests to my application?
- Has latency for my application increased or decreased over time?
- What can I do to reduce application latency?
- What are my application's dependencies?

## View and analyze trace data in the Cloud Trace interface



After the Cloud Trace agent has collected trace data, you can view and analyze that data in near real-time in the Cloud Trace interface. The interface contains three windows: Overview, Trace list, and Analysis reports.

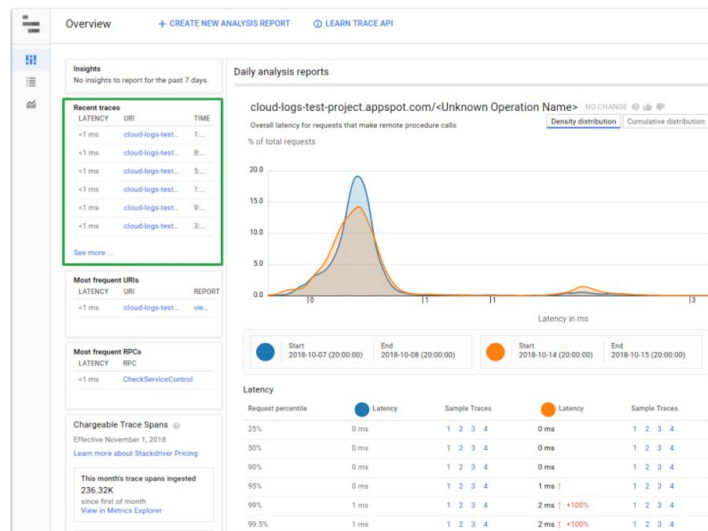
## View and analyze trace data in the Cloud Trace interface



The Insights pane displays a list of performance insights for your application, if applicable. This pane highlights common problems in applications, such as consecutive calls to a function, that, if batched, might be more efficient.

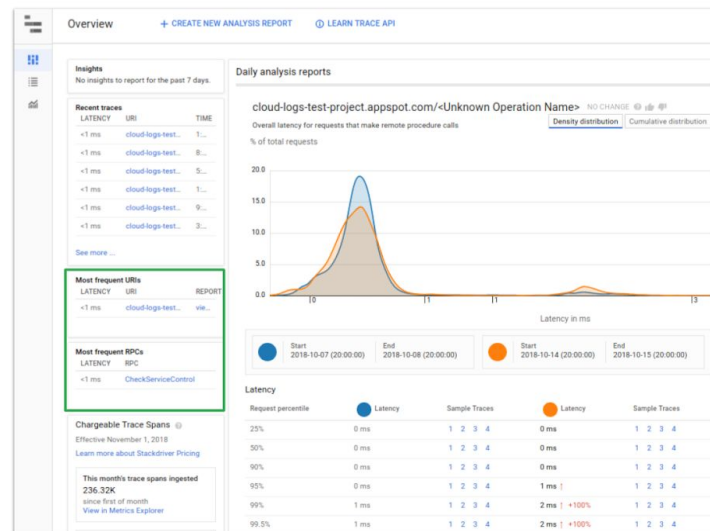


## View and analyze trace data in the Cloud Trace interface



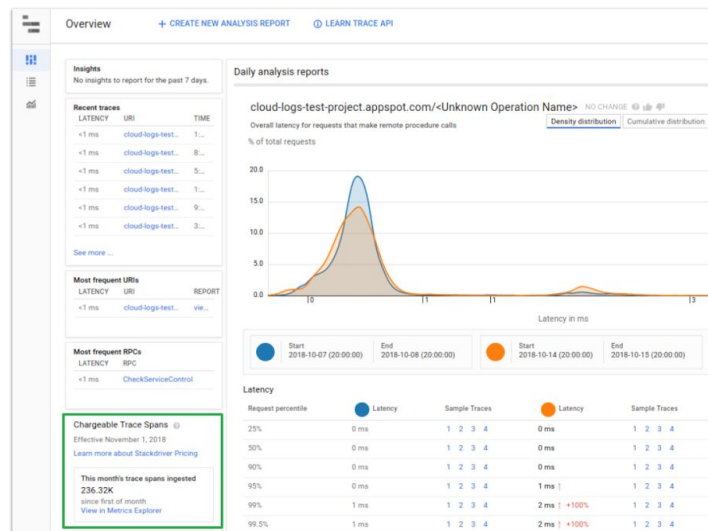
The Recent traces pane displays the most recent traces. For each, the latency, URI, and time are displayed. You can use this summary to understand the current activity in your application.

## View and analyze trace data in the Cloud Trace interface



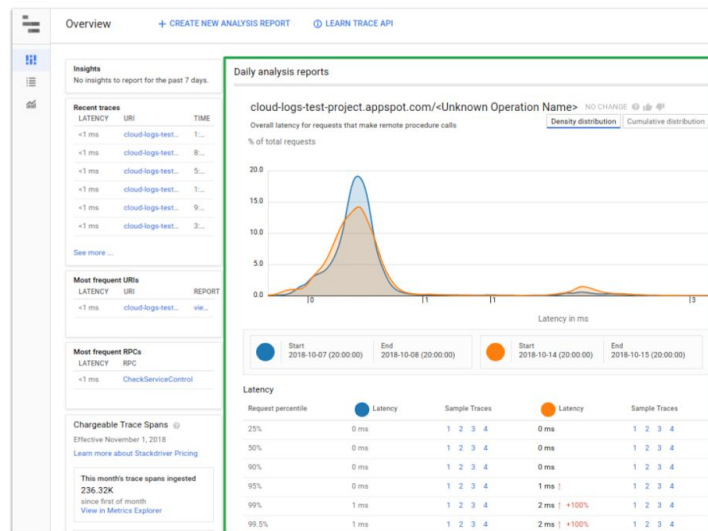
The Most frequent URIs and Most frequent RPCs from the previous day are listed, along with the average latency. If you click a link in either of these tables, you open a Trace list window, where you can view latency as a function of time, and investigate details of any individual trace.

## View and analyze trace data in the Cloud Trace interface



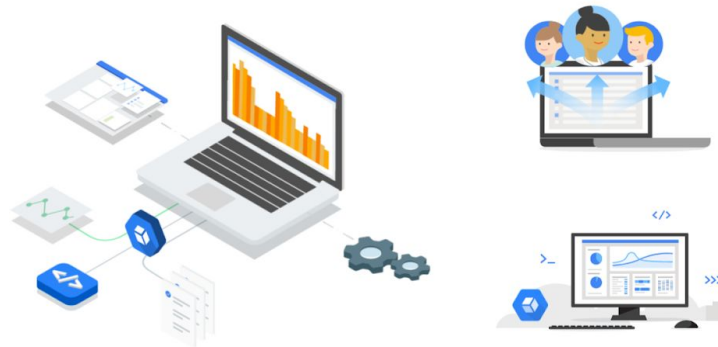
The Chargeable Trace Spans pane displays the number of spans ingested in the current calendar month, and the total for the previous month. You can use this information to monitor your costs for using Cloud Trace.

## View and analyze trace data in the Cloud Trace interface



The Daily analysis reports pane displays latency data for the previous day and compares it to the latency data from 7 days prior. You can also create your own analysis reports to select which traces you want included in the report.

## Cloud Profiler - Continuous profiling to improve performance and reduce costs



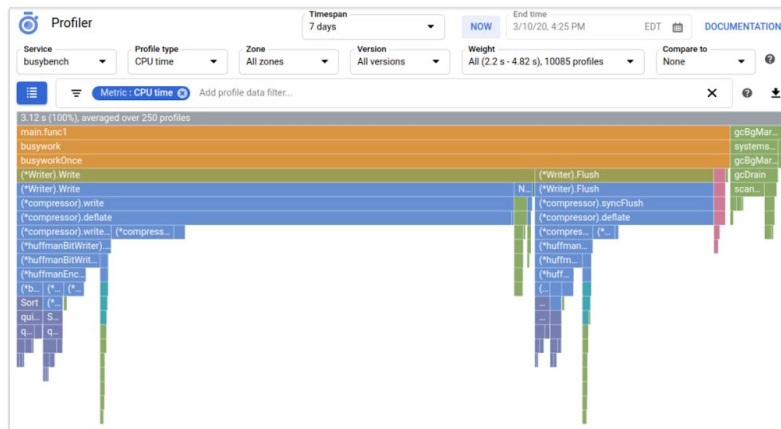
Cloud Profiler monitors CPU and heap to help you identify latency and inefficiency using interactive graphical tools, so you can improve application bottlenecks and reduce resource consumption. Google uses the same technology every day to identify inefficiently written code on services like Search and Gmail.

Poorly performing code increases the latency and cost of applications and web services every day, often without anyone knowing or doing anything about it. Cloud Profiler changes this by continuously analyzing the performance of CPU or memory-intensive functions executed across an application. Cloud Profiler presents the call hierarchy and resource consumption of the relevant function in an interactive flame graph that helps developers understand which paths consume the most resources and the different ways in which their code is actually called.

While it's possible to measure code performance in development environments, the results generally don't map well to what's happening in production. Many production profiling techniques either slow down code execution or only inspect a small subset of a codebase. Cloud Profiler uses statistical techniques and extremely low-impact instrumentation that runs across all production application instances to provide a complete picture of an application's performance without slowing it down.

Cloud Profiler allows developers to analyze applications running anywhere, including Google Cloud, other cloud platforms, or on-premises, with support for Java, Go, Node.js, and Python. A full explanation of language and platform support is available in the documentation.

The Cloud Profiler UI provides flame charts to correlate statistics with application areas and components



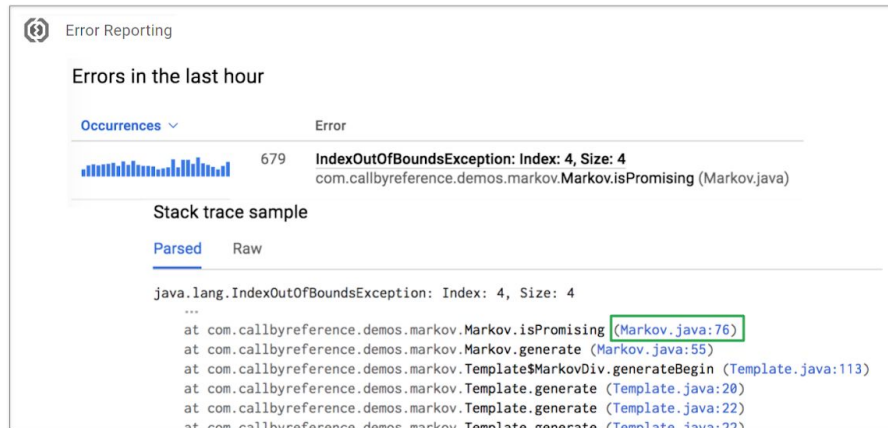
In order to use Cloud Profiler, you install the profiling agent on the virtual machines where your application runs. The agent typically comes as a library that you attach to your application when you run it. The agent collects profiling data as the app runs. Cloud Profiler is a statistical profiler, so the agent is not always active.

Cloud Profiler creates a single profile by collecting profiling data, usually for 10 seconds, once a minute for a single instance of the configured service in a single Compute Engine zone. After each instance of the agent starts, it notifies the Profiler backend that it's ready to capture data, and then the agent idles until it receives a reply from the backend specifying the type of profile to capture. If you have 10 instances of a service running in the same deployment, you create 10 profiling agents. However, these agents are idle most of the time. Over a 10-minute period, you can expect 10 profiles; each agent receives one reply for each profile type, on average. There is some randomization involved, so the actual number may vary.

The overhead of the CPU and heap allocation profiling at the time of the data collection is less than 5 percent. Amortized over the execution time and across multiple replicas of a service, the overhead is commonly less than 0.5 percent, making it an affordable option for always-on profiling in production systems.

After the agent has collected some profiling data, you can use the Profiler interface to see how the statistics for CPU and memory usage correlate with areas of your application.

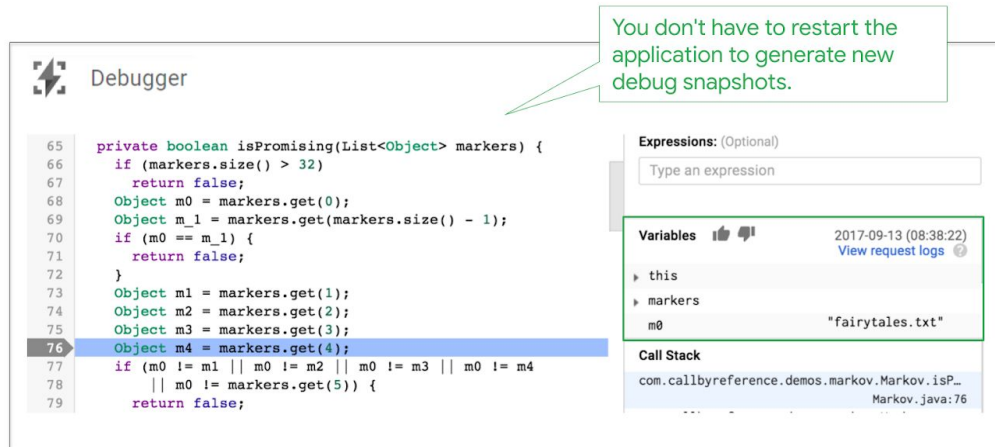
## Debug your application in development and production



With Error Reporting and Cloud Debugger, you can debug your application in development, and also troubleshoot errors in production.

Error Reporting displays errors that have occurred in your applications. You can view the stack trace to determine where the error occurred. Clicking the source code file in the stack trace takes you to Cloud Debugger and the line of code that has the problem.

## Debugger automatically creates debug snapshots



The screenshot displays the Cloud Debugger interface. On the left, a code editor shows a Java method `isPromising` with line numbers 65 to 79. Line 76, `Object m4 = markers.get(4);`, is highlighted. On the right, the sidebar contains three panels: 'Expressions' with a text input, 'Variables' showing `this`, `markers`, and `m0` (value `"fairytales.txt"`), and 'Call Stack' showing the current method `com.callbyreference.demos.markov.Markov.isP...`. A green callout bubble with an arrow points to the interface, containing the text: "You don't have to restart the application to generate new debug snapshots."

Debugger automatically creates a debug snapshot on the line that has the error. When the application hits the code again, Cloud Debugger creates a snapshot of the application state, including values of local variables. You can also manually select other lines of code where debug snapshots would be helpful.

You don't have to restart the application to generate new debug snapshots. This powerful feature of Cloud Debugger lets you inject logging information without interfering with the normal function of the application.





---

## Debugging Application Errors

Duration: 45 minutes

In this lab, you will debug application errors.

# Lab objectives

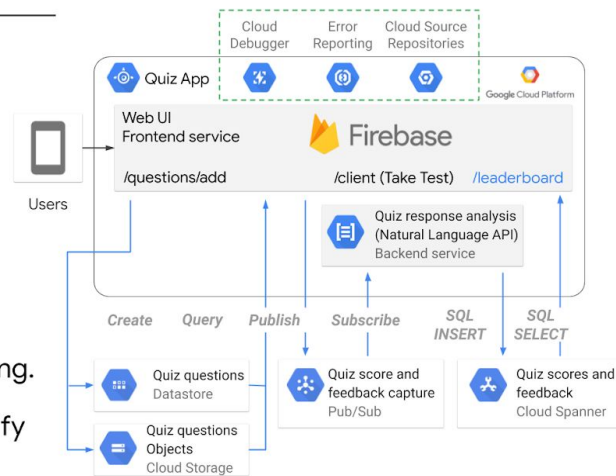
Create a repository and push application code to it.

Install and configure Cloud Debugger.

Harness Debug Snapshots and Logpoints to capture and display application variables.

Install and configure Error Reporting.

Leverage Error Reporting to identify application errors.



You will use Debug Snapshots and Logpoints to capture and display application variables.

You will use Error Reporting to identify application errors.

Google Cloud's operations suite is really cool in that you can troubleshoot application errors in development and production!

## Install the Cloud Logging agent to capture logs



Compute  
Engine

Amazon EC2

A robust system of logging is crucial for developer productivity and to help you understand the state of your application.

You can install the Cloud Logging agent on Compute Engine and Amazon EC2 instances to stream logs from third-party applications into Cloud Logging. The Logging agent is an application based on Fluentd. When you write your logs to existing log files, such as syslog on your VM instance, the logging agent sends the logs to Cloud Logging. Cloud Logging is pre-configured in other compute environments.

Cloud Logging is preconfigured in other compute environments



Dataflow



App Engine  
(Flexible and  
standard  
environments)



Cloud  
Functions



Google  
Kubernetes  
Engine

Dataflow, Cloud Functions and App Engine have built in support for logging. You can enable logging on Google Kubernetes Engine by simply enabling a checkbox in the Cloud Console when you set up a container cluster.

## Set up logs-based metrics and alerts



Cloud Logging and Cloud Monitoring

```
09:21:51.246GET2001.13 KB6 exampleapp/  
- - [14/Sep/2017:09:21:51 -0700] "GET / HTTP/1.1" 200 1156 - "exampleapp"  
"exampleapp-git.appspot.com" ms=6 cpu_ms=11 cpm_usd=1.2919299999999998e-7  
loading_request=0 instance=some_instance_id app_engine_release=1.9.54
```

[Expand all](#) | [Collapse all](#)

```
{  
  httpRequest: {  
    status: 200  
  }  
}
```

Create custom  
logs-based metric:  
HTTP\_Success

Alert if HTTP\_Success  
metric is:  
Below 400 per second  
for 5 minutes

In Cloud Logging, you can view your logs and search for particular types of messages. You can create custom logs-based metrics and alerts based on those metrics.

Let's take a look at the example here. There's a logs-based metric called HTTP\_Success that filters all log messages with an HTTP response code of 200.

We have then set an alert on that metric to notify you when the number of successful HTTP requests is below 400 per second for five minutes.

Logs-based metrics are a really powerful feature, in that they alert you to a problem so that you can react to it before it becomes a major issue.

## Monitor to analyze long-term trends

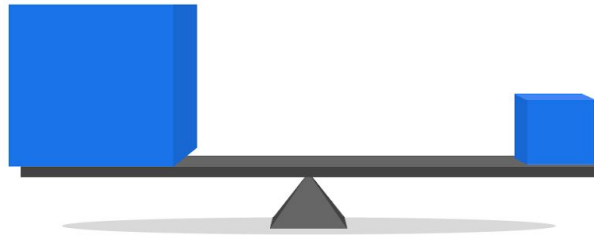


Let's take a look at why it's important to monitor your application, what resources you should monitor, and how you should go about troubleshooting performance for your application.

When you monitor your application and gather metrics, you can make improvements to the design of your application, increase reliability, detect and fix security issues, and reduce usage costs based on usage patterns.

Monitor to analyze long term trends and answer questions such as, how fast is my database growing? How many users have I added in the last four quarters? And so on.

Monitor to compare results over time or between experimental configurations



For example, is the site slower than it was last week? Are requests served faster with Apache or NGINX web server?

Monitor to raise alerts when something is broken or about to be broken



Monitor to raise alerts when something is broken and should be fixed urgently, or something is about to be broken, and can be addressed preemptively.



Monitor to perform ad hoc retrospective analysis



Monitor to perform ad hoc retrospective analysis. For example, the latency of your application just increased sharply. What else happened around the same time?

## Identify APIs and resources that you want to monitor

Examples:

- Public and private endpoints.
- Multi-cloud resources, such as Compute Engine VM instances, Cloud Storage buckets, Amazon EC2 instances, and Amazon RDS databases.

Identify APIs and resources that you want to monitor.

For example, there might be public and private endpoints that you want to monitor, or multi-cloud resources such as Compute Engine VM instances, Cloud storage buckets, Amazon EC2 instances, and databases that you want to keep your eye on.

## Identify service-level indicators and objectives

Service-Level  
Indicator (SLI):

Latency

---

Service-Level  
Objective (SLO):

99.9% of requests  
over 30 days have  
latency <100ms



Here, we begin to touch on Google's site reliability engineering or SRE principles. I strongly urge you to check out Google's SRE book online. It's available for free.

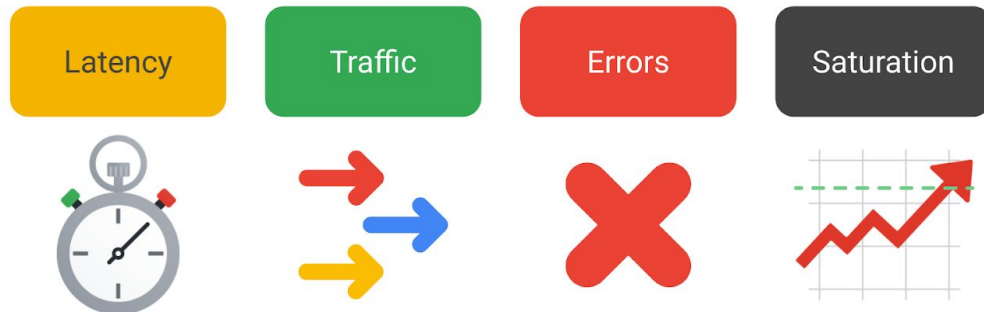
A service level indicator, or SLI, is a quantitative measure of some aspect of a service. A Service-Level Indicator might be a predefined metric or a custom metric, such as a logs-based metric.

A Service-Level Objective, or SLO, is a target value or range of values for a service level. Service-Level Objectives should include tolerance for small variations. Absolute limits will result in noisy pagers and require you to constantly tweak thresholds.

For example, latency is a service level indicator.

You can set a service level objective, indicating that 99% of requests over 30 days have latency less than 100 milliseconds. This is a great example of a Service-Level Objective that has a range of values for a service level.

## Create dashboards that include four golden signals



After you identify the resources to monitor, and define service level indicators and objectives, you can create dashboards to view metrics for your application. Create dashboards that include the four golden signals: Latency, Traffic, Errors, and Saturation.

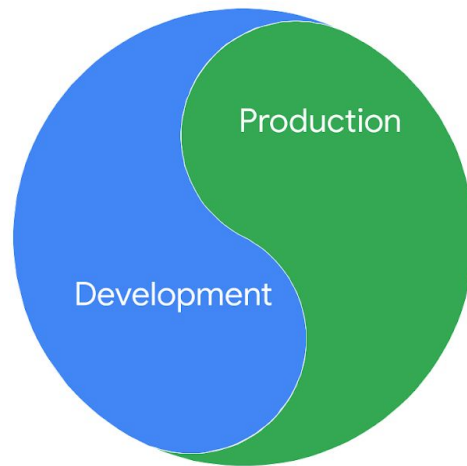
Latency is the amount of time it takes to serve a request. Make sure to distinguish between the latency of successful and unsuccessful requests. For example, an HTTP error that occurs due to a loss of connection to a database or another back end service, might be solved really quickly. However, because an HTTP 500 error indicates a failed request, including 500 errors in your overall latency might result in misleading metrics.

Traffic is a measure of how much demand is placed on your system. It's measured as a system-specific metric. For example, web server traffic is measured as the number of HTTP or HTTPS requests per second. Traffic to a NoSQL database is measured as the number of read or write operations per second.

Errors indicate the number of failed requests. Criteria for failure might be anything like an explicit error, such as a HTTP 500 error, or a successful HTTP 200 response but with incorrect content. It might also be a policy error. For example, your application promises a response time of one second, but some requests take over a second.

Saturation indicates how full your application is, or what resources are being stretched and reaching target capacity. Systems can degrade in performance before they achieve 100% utilization, so make sure to set utilization targets carefully.

## Monitor performance in development and production



Let's take a look at the areas that you can review to identify and troubleshoot performance issues.

It's important to monitor performance in the development phase, and in production.

## In Development: Add performance tests to your test suite



In Development, add performance tests to your test suite to ensure that the performance of your application doesn't degrade when you fix bugs, add new features, or change underlying software.

Response times and resource requirements can change significantly when you make changes to your application.

With performance tests, you'll be able to detect and address performance issues early in the development process.

## In Development: Check performance watchpoints related to incoming requests

Web authoring	Cold-boot performance	Self-inflicted load
Web page design and implementation	Operations during initial boot of VM	Service-to-service or browser-to-service calls

In Development, check performance watch points related to incoming requests. A watch point is a potential area of configuration or application code that could indicate a performance issue.

Performance issues may be a result of multiple watch points. Review metrics related to incoming requests, and check areas such as the ones shown here.

Review the design and implementation of your web pages.

You can use page speed insights to view information about the lack of caching headers, the lack of compression, too many HTTP browser requests, slow DNS response, and the lack of minification. If the application is not public facing, you can use Chrome DevTools with page speed insights to manually analyze your web pages.

Check for self-inflicted load. This is load caused by the application itself, such as service-to-service or browser-to-service calls. For example, check for polling cron jobs, batch requests, or multiple AJAX requests from the browser.

You can use client-side tools, such as Chrome DevTools, and server side load analysis tools, such as Cloud Trace, to find the source of your problem.

## In Development: Review application code and logs

<b>Application errors</b> HTTP errors and other exceptions	<b>Runtime code gen.</b> Aspect-oriented programming	<b>Static resources</b> Static web pages, images
<b>Caching</b> Database retrieval and computation	<b>One-at-a-time retrieval</b> Multiple serial requests	<b>Error-handling</b> Exponential backoff

In Development, review application code and logs to check for performance issues.

Check logs for application errors such as HTTP errors and exceptions. Identify the root cause of the log messages and confirm that they are not related to periodic load or performance issues. Prioritize investigation by the frequency of errors. Because this data is historical, some errors might have been intermittent or already resolved. If a log message is not reproducible, it might be better to defer the investigation.

Check for runtime code generation. Aspect-oriented programming practices can sometimes cause reduced application performance. Consider compile time code generation instead.

Don't serve static resources from your application. Instead use a content delivery network such as Cloud CDN with Google Cloud Storage.

Consider caching frequently accessed values that are retrieved from a database or require significant compute resources to recalculate. You can also cache generated HTML fragments for later use.

Look for areas where data is retrieved from a database or service with multiple requests in serial. Replace these individual requests with a single batch request, or send the request in parallel.

Don't retry constantly on errors. Instead, retry with exponential backoff. Implement a circuit breaker to stop retries after a certain number of failures. Note that you should only retry in case of certain errors such as connection timeouts or too many requests. Don't retry in case of errors like 5xx errors and malformed URL errors.



## In Production: Check performance watchpoints related to incoming requests in production

External user load	Periodic load	Malicious load
Most frequent and slowest requests.	Traffic over an extended period of time.	Source of traffic is expected and legitimate.

In Production, monitor incoming requests to check the following areas:

**External user load:** Analyze the most frequent requests and slowest requests. Confirm that the requests are expected. Determine the cause for the slowest requests.

**Periodic load:** Analyze traffic over an extended period of time to determine which periods have higher levels of usage. Confirm that there is a business reason for this load.

**Malicious load:** Confirm that all load is expected and legitimate. If you have a web application, make sure that all load is coming from a web or mobile client. You can further segment the load by user to understand whether the majority of requests are coming from a small number of users.

## In Production: Review deployment settings

Scaling	Region	Cron Jobs
Autoscaling	Source of traffic	Schedule

In Production, review deployment settings as follows:

**Scaling:** Make sure that you have set up load balancing and autoscaling policies as appropriate for the traffic volumes for your application. Set target utilization levels conservatively to ensure that your application continues to handle traffic while new VM instances come online.

**Region:** Determine where the bulk of your traffic is coming from. Deploy resources in the appropriate regions to reduce latency.

**Cron jobs:** Make sure that your cron jobs are scheduled accurately.

Review and implement best practices for each of the services that you are using in your application.

Traditional SRE blessing

May the queries  
flow, and the  
pager stay silent

Remember, SRE stands for Site Reliability Engineering. "May the queries flow, and the pager stay silent".



## Harnessing Stackdriver Trace and Monitoring

Duration: 60 minutes

In this lab, you will use Cloud Trace and Cloud Monitoring.

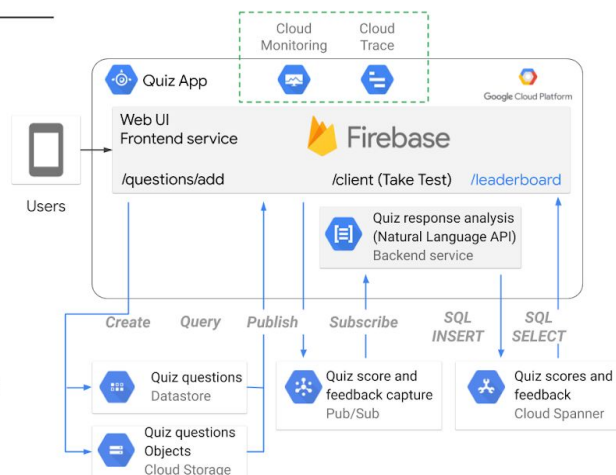
### Lab objectives

Enable, install, and configure Cloud Trace.

View Trace information to diagnose a performance issue.

Fix the performance issue and verify the performance improvement.

Monitor Google Cloud products using Cloud Monitoring.



You will use Cloud Trace to diagnose and fix a performance issue in the running Quiz application and use Cloud Monitoring to view performance metrics for the application.

Remember, Google Cloud's operations suite and APM tools are a multi-cloud service, so you can view metrics for all your multi-cloud resources in a single pane of glass.



---

## Summary

Google Cloud's operations suite is a multi-cloud service.

You can use Error Reporting and Cloud Debugger to debug and troubleshoot your application in development and production.

It is important to develop a strong suite of performance tests to monitor your application's performance as it evolves.

Identify service-level indicators and objectives. Set up dashboards with the four golden signals: latency, traffic, errors, and saturation.

If you see performance issues, apply the techniques that you have learned to analyze performance watchpoints and address issues.