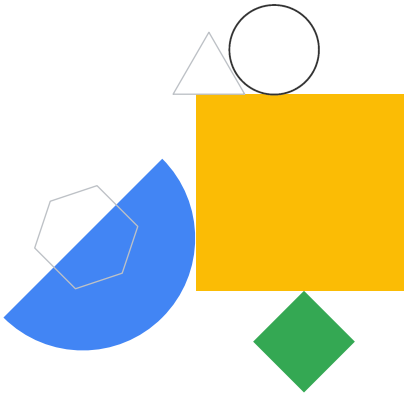
Google Cloud



Google Cloud Core Infrastructure Module 8

On-demand course
March 2022

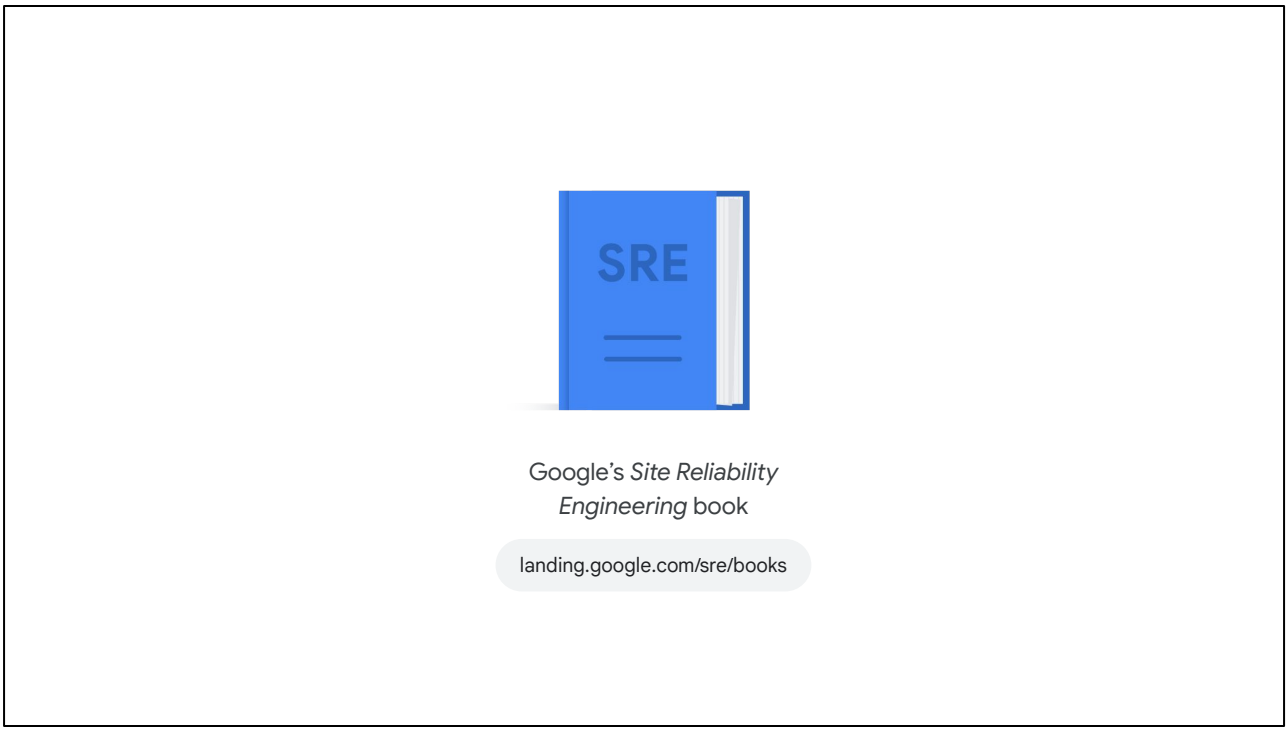
In this section of the course we'll transition our focus from developing and deploying in the cloud,

to logging and monitoring. Let’s begin with monitoring.

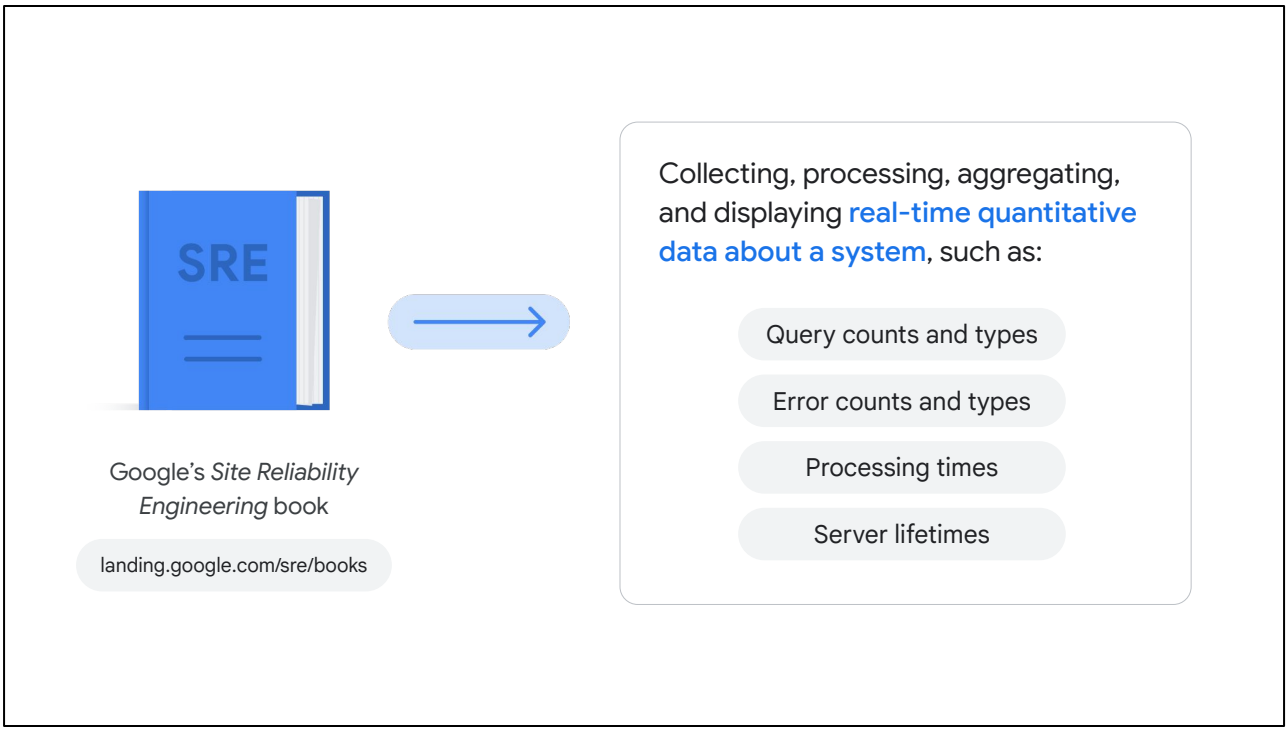
Monitoring is the foundation of product reliability.

- Reveals what needs urgent attention
- Shows trends in application usage patterns
- Helps improve an application experience

It reveals what needs urgent attention and shows trends in application usage patterns, which can yield better capacity planning and generally help improve an application client's experience and lessen their pain.

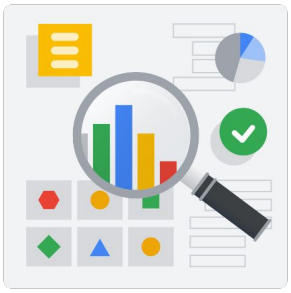


In Google's *Site Reliability Engineering* book, which is available to read at landing.google.com/sre/books,



monitoring is defined as:

"Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes."

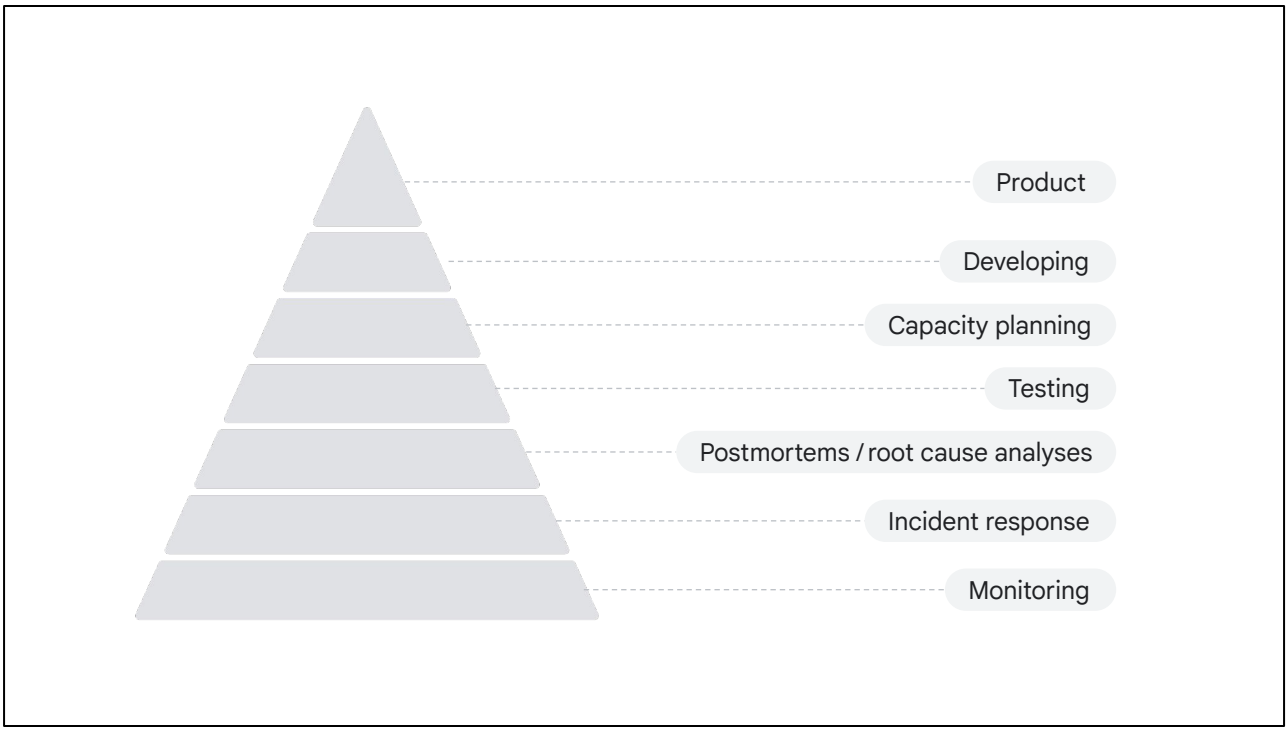


Monitoring

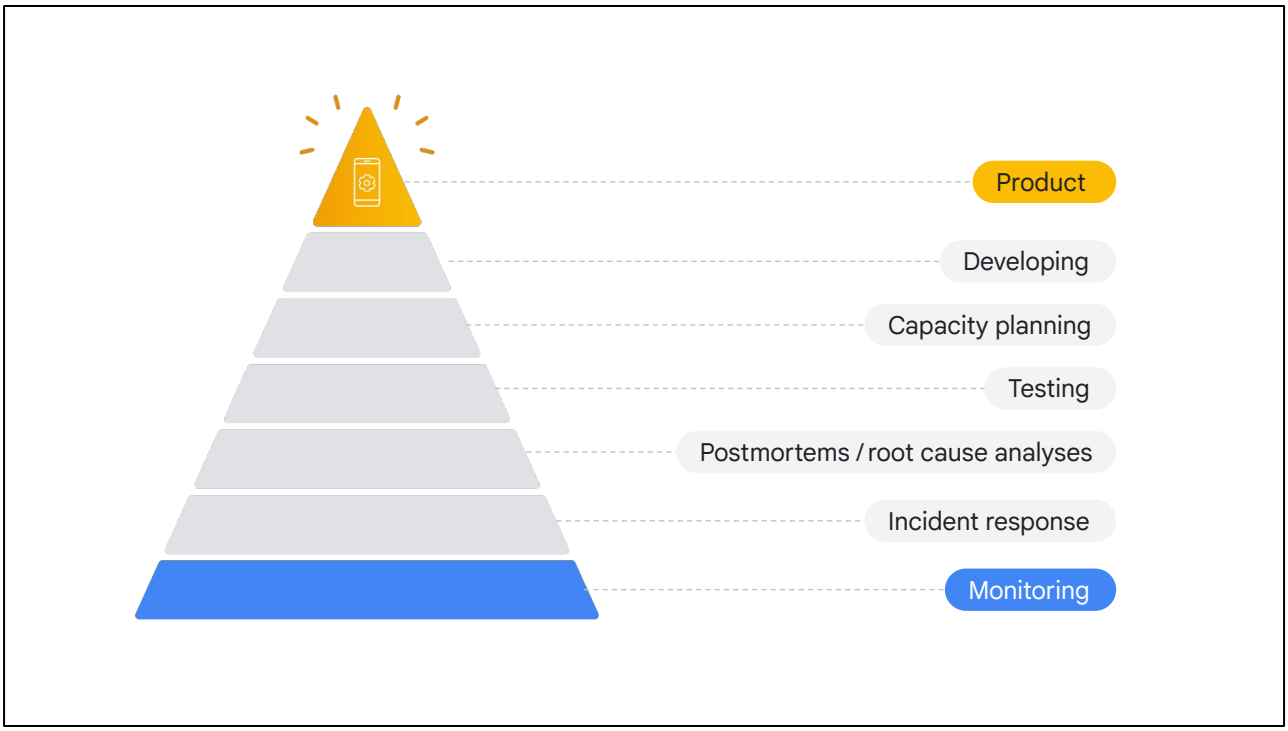
- Ensure continued system operations
- Uncover trend analyses over time
- Build dashboards
- Alert personnel when systems violate predefined SLOs
- Compare systems and systems changed
- Provide data for improved incident response

With monitoring, you can:

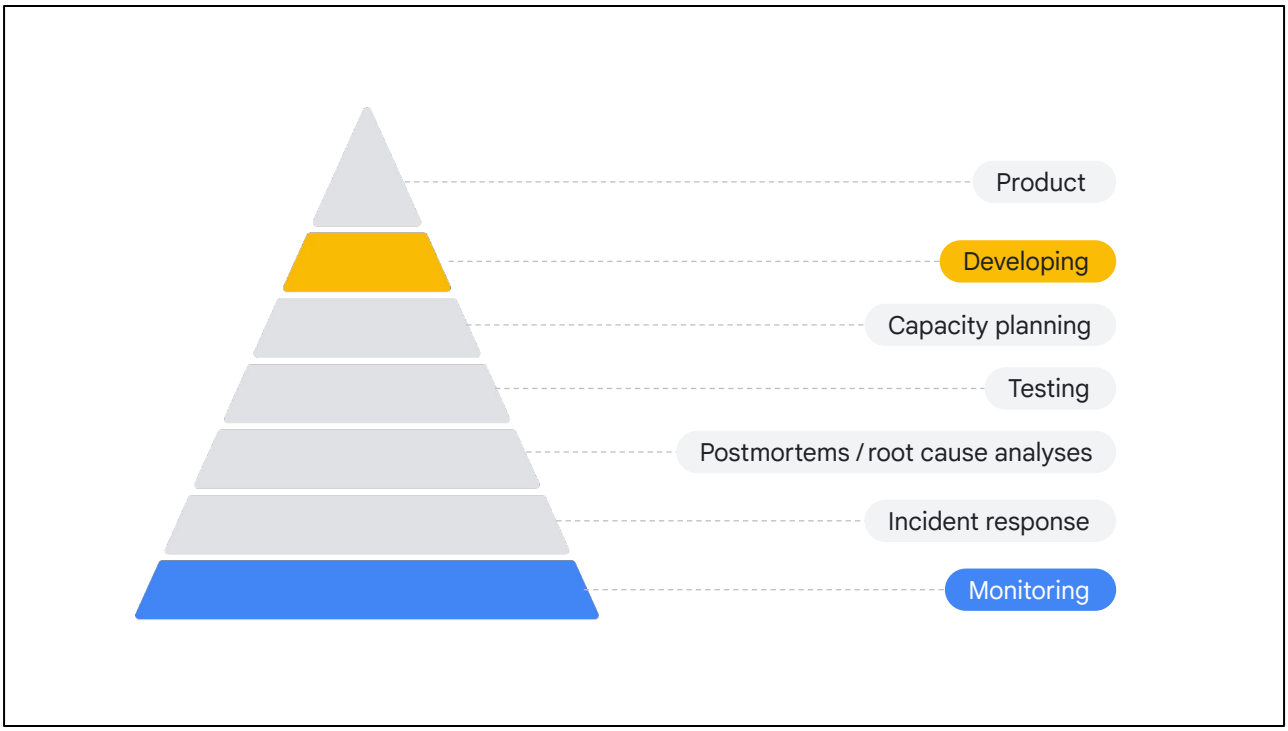
- ensure continued system operations,
- uncover trend analyses over time,
- build dashboards,
- alert personnel when systems violate predefined service level objectives (SLOs),
- compare systems and systems changed,
- and provide data for improved incident response–just to name a few tasks.



An application client normally only sees the public side

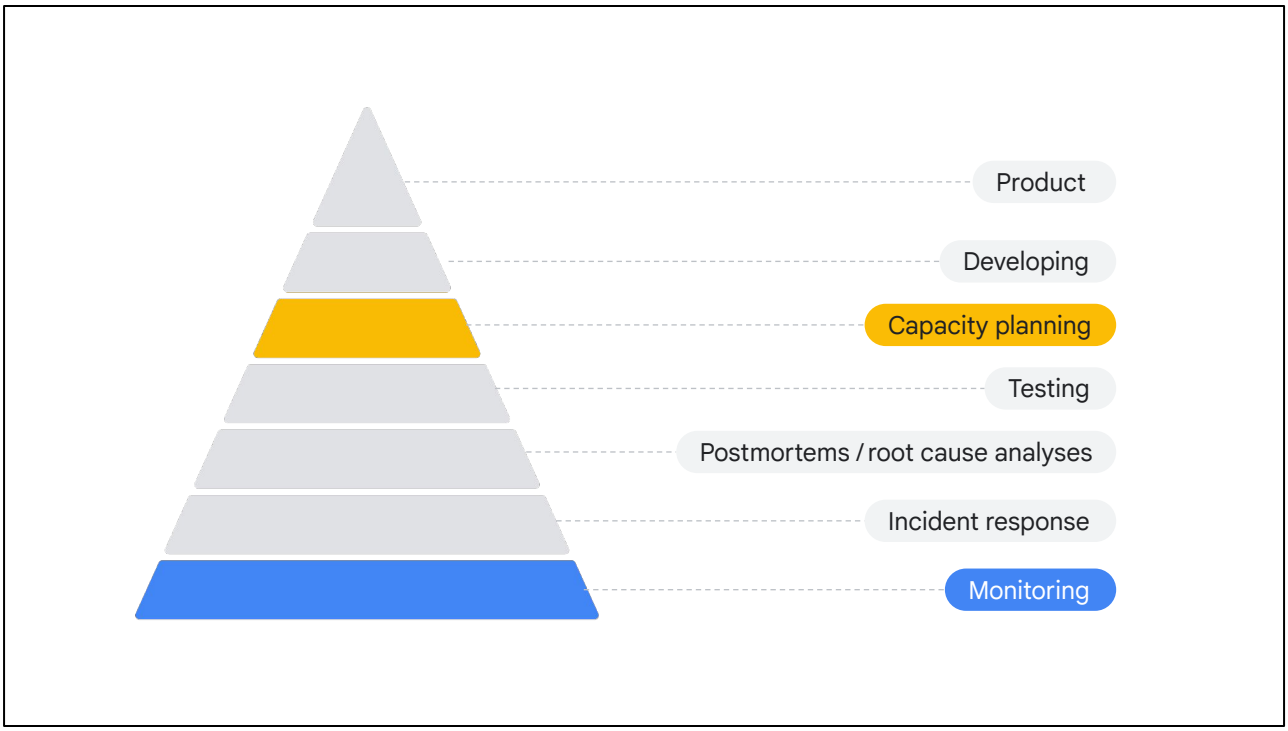


of a **product**, and as a result, developers and business stakeholders both tend to think that the most crucial way to make the client happy is by spending the most time and effort

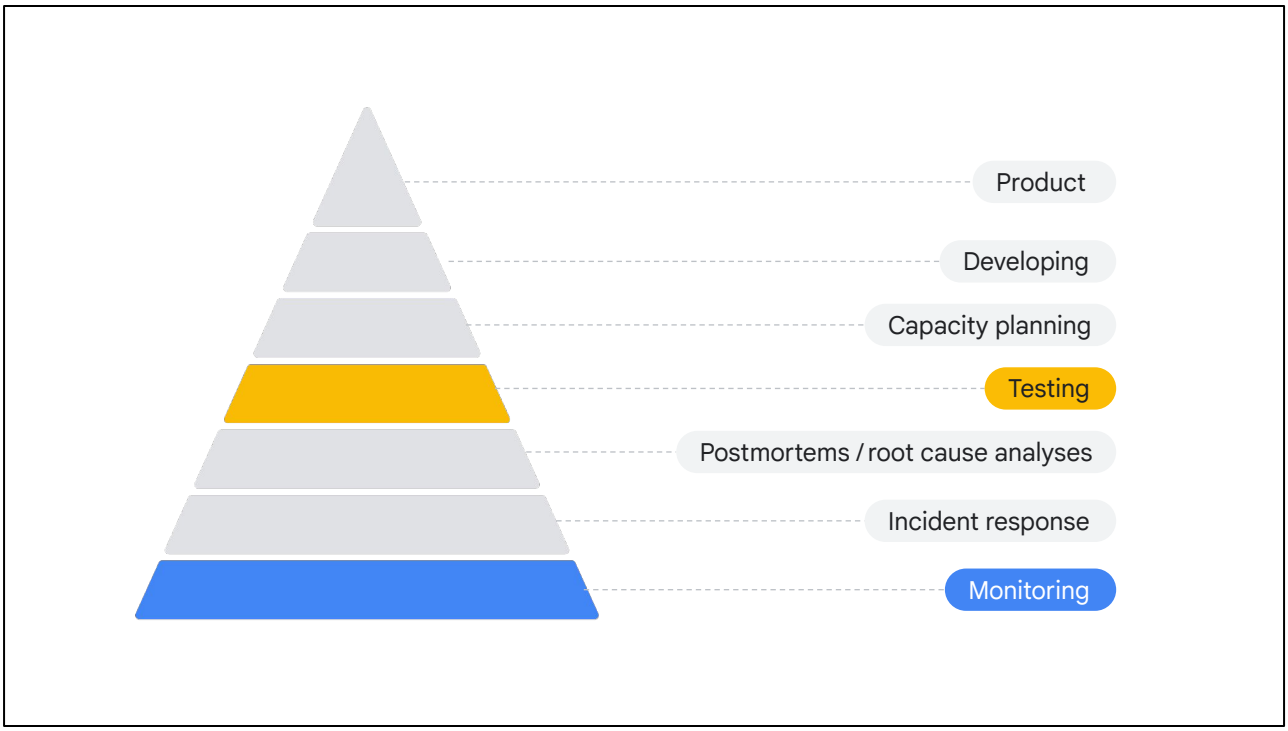


on **developing** that part of the product.

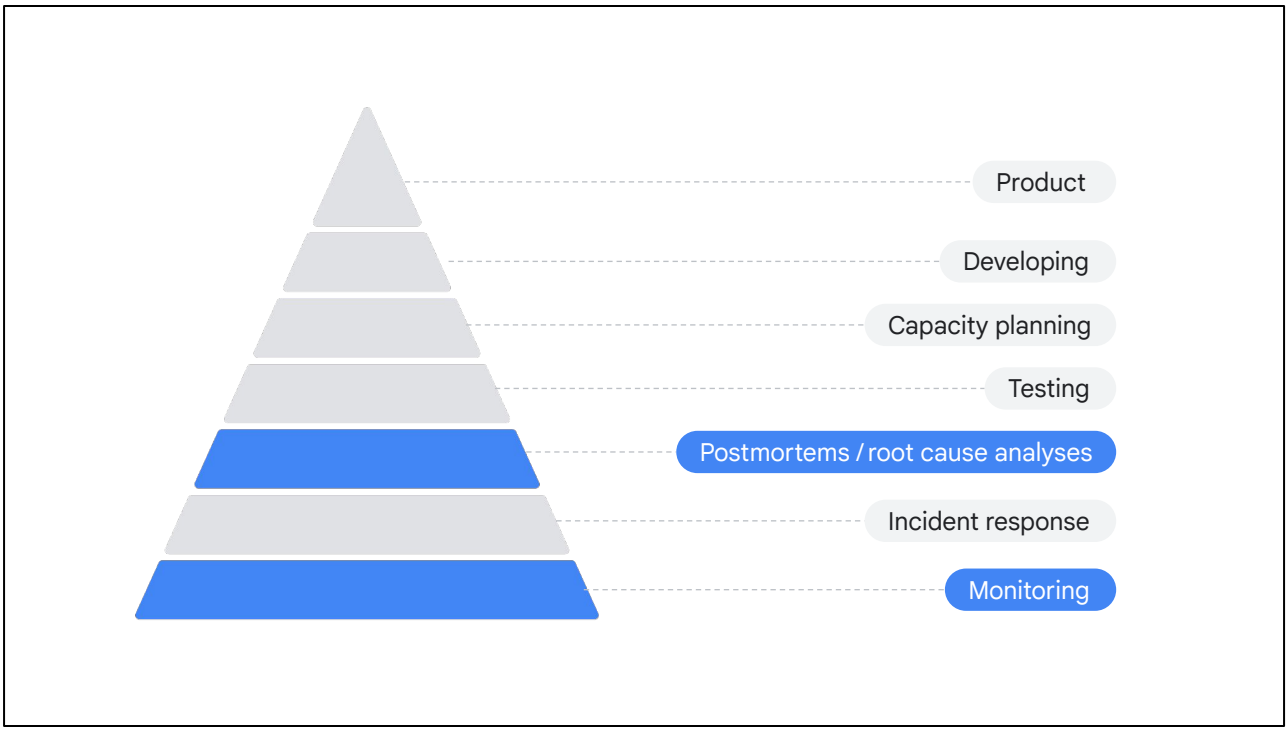
However, to be truly reliable, even the very best products still must be deployed



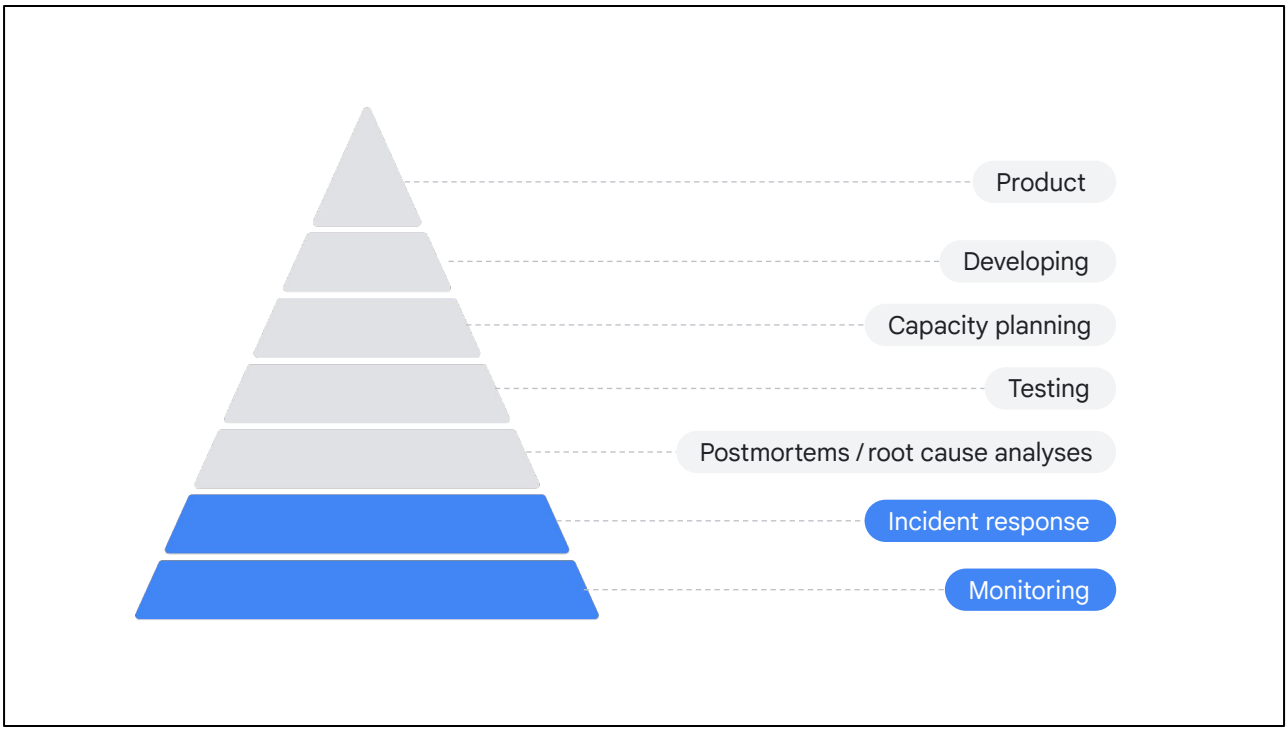
into environments with enough **capacity** to handle the anticipated client load.



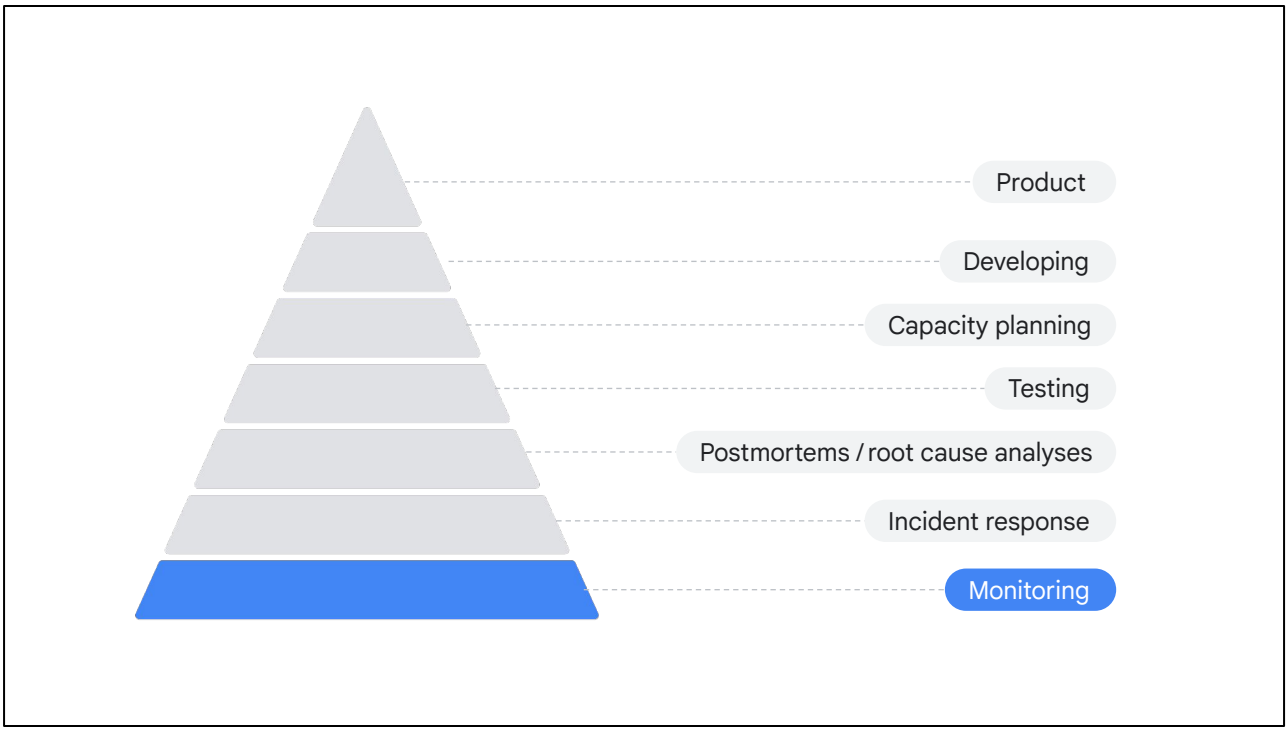
Great products also need thorough **testing**, preferably automated testing, and a refined continuous integration/continuous development (CI/CD) release pipeline.



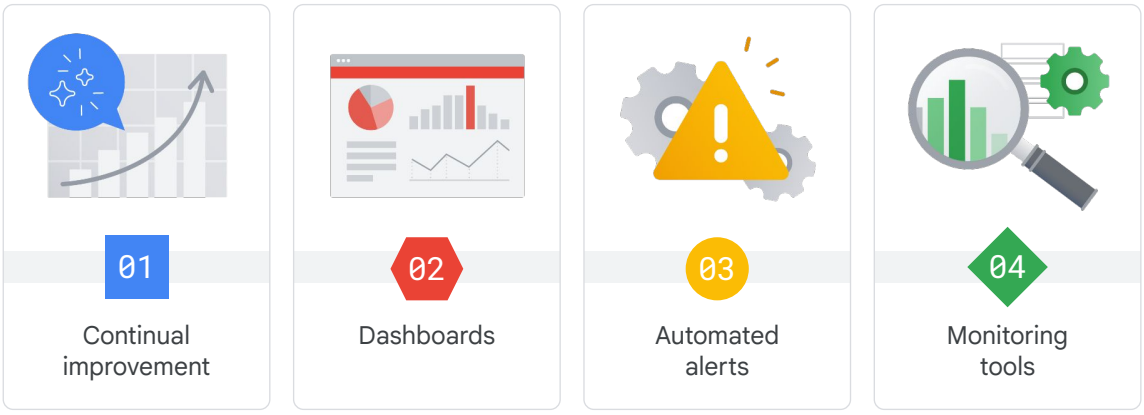
Postmortems and **root cause analyses** are the DevOps team's way of letting the client know



why an **incident** happened and why it's unlikely to happen again.



In this context we're discussing a system or software failure, but the term "incident" can also be used to describe a breach of security. Here, transparency is key to building trust.

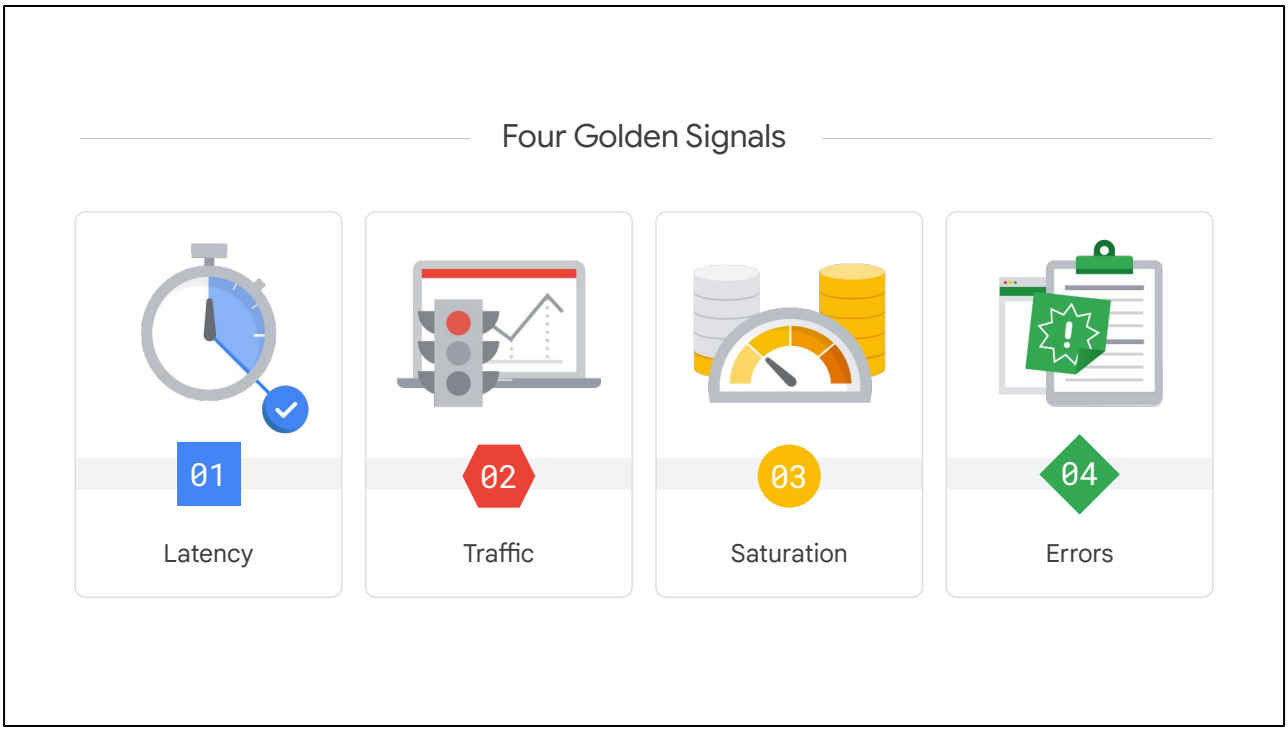


We need our products to **improve continually**, and we need data we can receive from monitoring to make sure that happens.

We need **dashboards** to provide business intelligence so our DevOps personnel have the data they need to do their jobs.

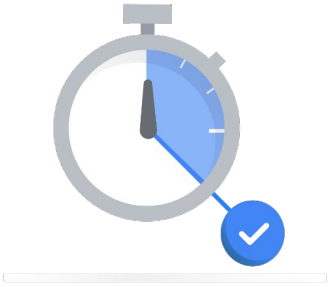
We need **automated alerts** because humans tend to look at things only when there's something important to look at. An even better option is to construct automated systems to handle as many alerts as possible so humans only have to look at the most critical issues.

Finally, we need **monitoring tools** that help provide data crucial to debugging application functional and performance issues. We'll look more closely at Google's integrated monitoring tools a bit later in this module.



There are “Four Golden Signals” that measure a system’s performance and reliability.

They are **latency**, **traffic**, **saturation**, and **errors**.



Latency

01 It directly affects the user experience.

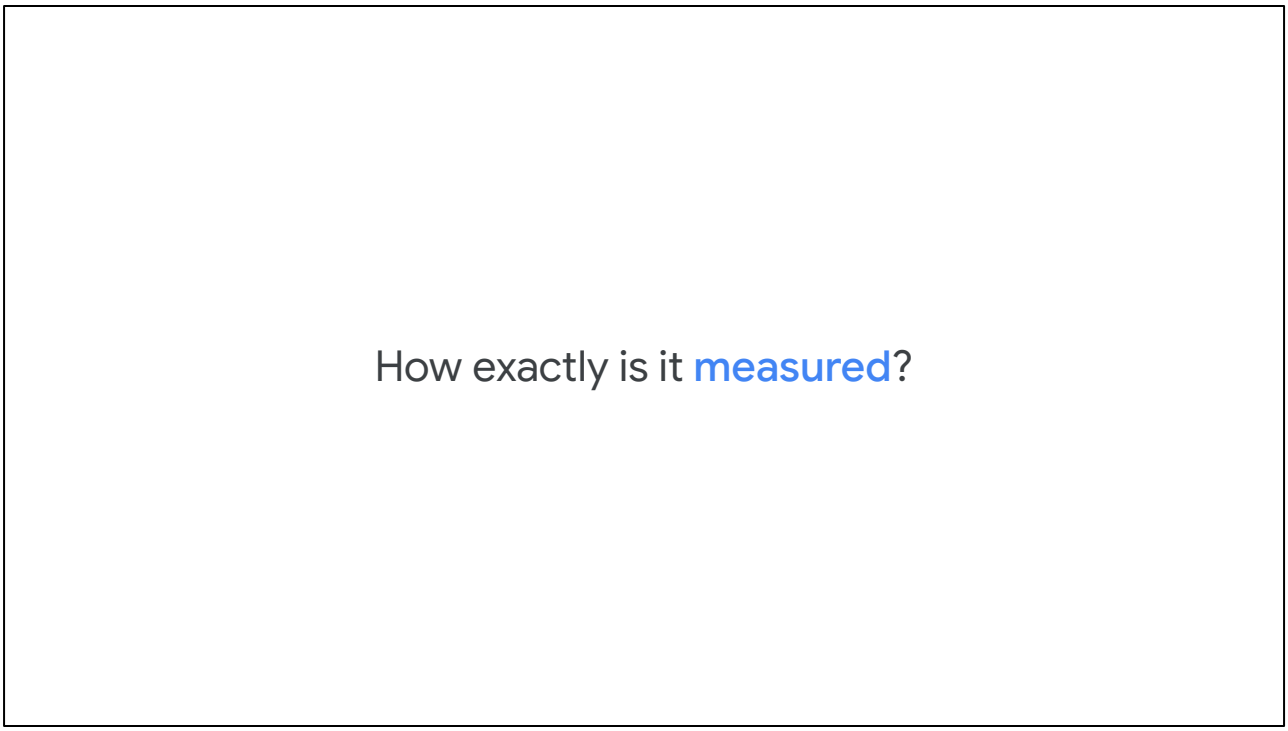
02 Changes in latency could indicate emerging issues.

03 Its values may be tied to capacity demands.

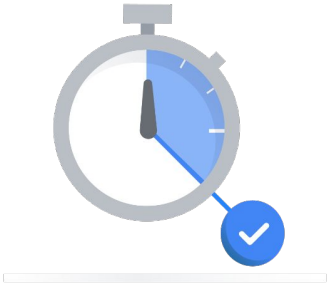
04 It can be used to measure system improvements.

Latency measures how long it takes a particular part of a system to return a result. Latency is important because:

- It directly affects the user experience.
- Changes in latency could indicate emerging issues.
- Its values may be tied to capacity demands.
- And it can be used to measure system improvements.



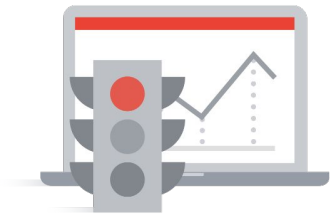
But how exactly is it measured? Sample latency metrics include:



Latency

- Page load latency
- Number of requests waiting for a thread
- Query duration
- Service response time
- Transaction duration
- Time to first response
- Time to complete data return

- Page load latency
- Number of requests waiting for a thread
- Query duration
- Service response time
- Transaction duration
- Time to first response
- Time to complete data return



Traffic

01 It's an indicator of current system demand.

02 Its historical trends are used for capacity planning.

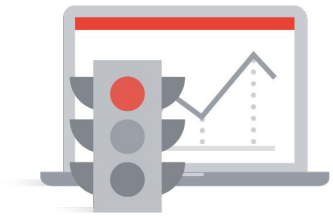
03 It's a core measure when calculating infrastructure spend.

The next signal is **traffic**, which measures how many requests are reaching your system.

Traffic is important because:

- It's an indicator of current system demand.
- Its historical trends are used for capacity planning.
- And it's a core measure when calculating infrastructure spend.

Sample traffic metrics include:



Traffic

- # HTTP requests per second
- # requests for static vs. dynamic content
- Network I/O
- # concurrent sessions
- # transactions per second
- # of retrievals per second
- # of active requests
- # of write ops
- # of read ops
- # of active connections

- number of HTTP requests per second
- number of requests for static vs. dynamic content
- Network I/O
- number of concurrent sessions
- number of transactions per second
- number of retrievals per second
- number of active requests
- number of write ops
- number of read ops
- And number of active connections

Saturation

01 It's an indicator of how full the service is.

02 It focuses on the most constrained resources.

03 It's frequently tied to degrading performance as capacity is reached.

The third signal is **saturation**, which measures how close to capacity a system is. It's important to note, though, that capacity is often a subjective measure that depends on the underlying service or application.

Saturation is important because:

- It's an indicator of how full the service is.
- It focuses on the most constrained resources.
- And it's frequently tied to degrading performance as capacity is reached.

Sample capacity metrics include:

Saturation

- % memory utilization
- % thread pool utilization
- % cache utilization
- % disk utilization
- % CPU utilization
- Disk quota
- Memory quota
- # of available connections
- And # of users on the system

- % memory utilization
- % thread pool utilization
- % cache utilization
- % disk utilization
- % CPU utilization
- Disk quota
- Memory quota
- number of of available connections
- And number of of users on the system

Errors

01 They may indicate that something is failing

02 They may indicate configuration or capacity issues

03 They can indicate service level objective violations

04 An error might mean it's time to send out an alert

The fourth signal is **errors**, which are events that measure system failures or other issues. Errors are often raised when a flaw, failure, or fault in a computer program or system causes it to produce incorrect or unexpected results, or behave in unintended ways.

Errors are important because:

- They may indicate that something is failing.
- They may indicate configuration or capacity issues.
- They can indicate service level objective violations.
- And an error might mean it's time to send out an alert.

Errors

- Wrong answers or incorrect content
- # 400/500 HTTP codes
- # failed requests
- # exceptions
- # stack traces
- Servers that fail liveness checks
- And # dropped connections

Sample error metrics include:

- Wrong answers or incorrect content
- Number of 400/500 HTTP codes
- Number of failed requests
- Number of exceptions
- Number of stack traces
- Servers that fail liveness checks
- And number of dropped connections

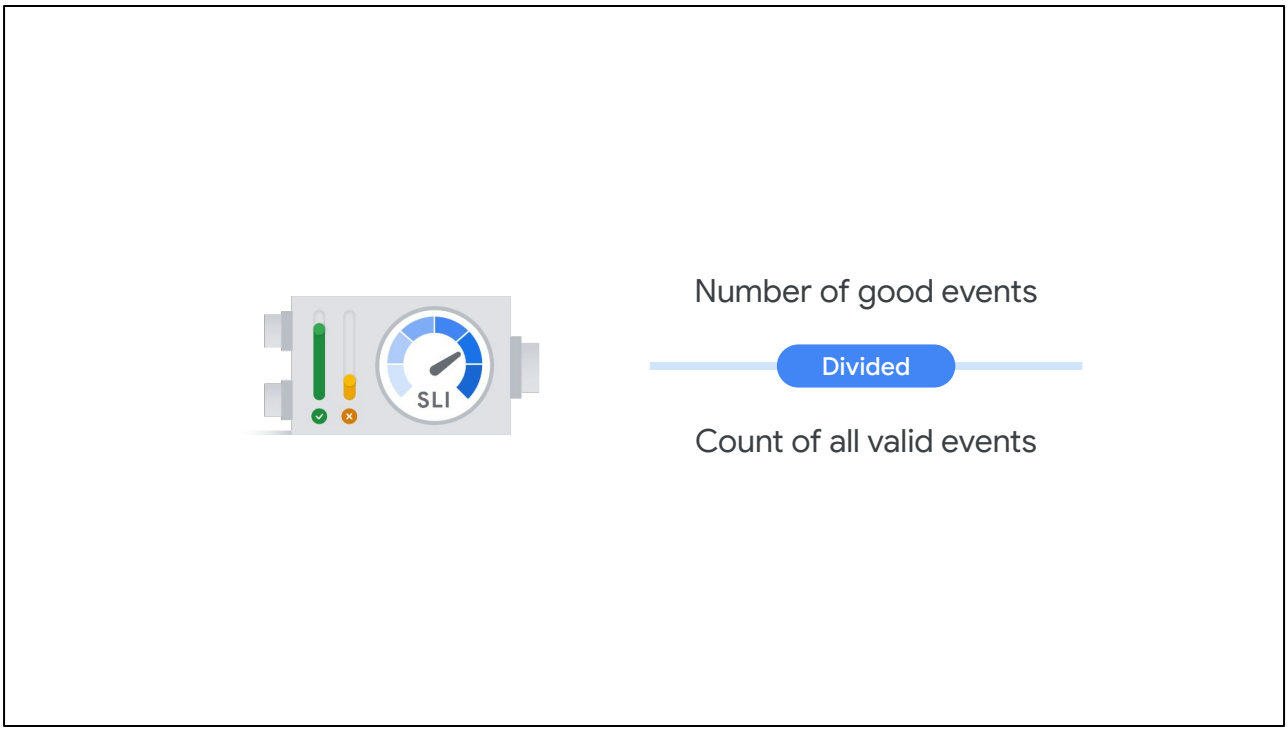
Service level indicator | Service level objective | Service level agreement

Now let's shift our focus to SLIs, SLOs and SLAs, which are all types of targets set for a system's Four Golden Signal metrics.

Service Level Indicator

Carefully selected monitoring metrics that measure one aspect of a service's reliability

Service level indicators, or SLIs, are carefully selected monitoring metrics that measure one aspect of a service's reliability.

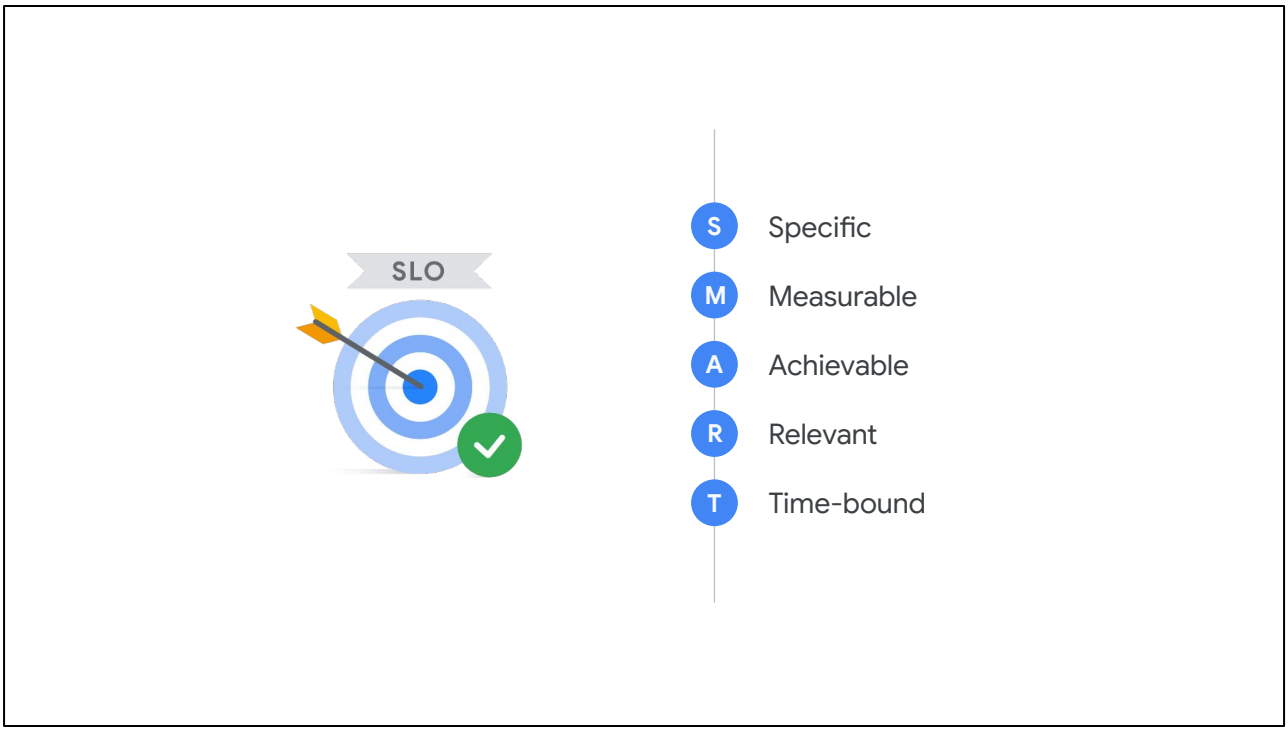


Ideally, SLIs should have a close linear relationship with your users' experience of that reliability, and we recommend expressing them as the ratio of two numbers: the number of good events divided by the count of all valid events.

Service Level Objective

Combines a service level indicator with a target reliability and will generally be somewhere just short of 100%, for example, 99.9% ("three nines")

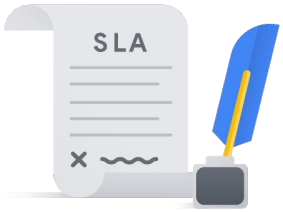
A **Service level objective, or SLO**, combines a service level indicator with a target reliability. If you express your SLIs as is commonly recommended, your SLOs will generally be somewhere just short of 100%, for example, 99.9%, or "three nines."



You can't measure everything, so when possible, you should choose SLOs that are **S.M.A.R.T**.

- SLOs should be **specific**. A question such as “Is the site fast enough for you?” is not specific; it's subjective. A statement such as “The 95th percentile of results are returned in under 100 milliseconds” is specific.
- SLOs need to be based on indicators that are **measurable**. A lot of monitoring is numbers, grouped over time, with math applied. An SLI must be a number or a delta; something we can measure and place in a mathematical equation.
- SLO goals should be **achievable**. "100% Availability" might sound good, but it's not possible to obtain, let alone maintain, over an extended window of time.
- SLOs should be **relevant**. Does it matter to the user? Will it help achieve application-related goals? If not, then it’s a poor metric.
- And SLOs should be **time-bound**. You want a service to be 99% available? That’s fine. Is that per year? Per month? Per day? Does the calculation look at specific windows of set time, from Sunday to Sunday for example, or is it a rolling period of the last seven days?

If we don't know the answers to those types of questions, it can’t be measured accurately.



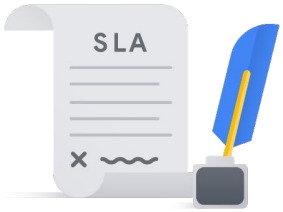
Service Level Agreement

Commitments made to your customers that your systems and applications will have only a certain amount of down time

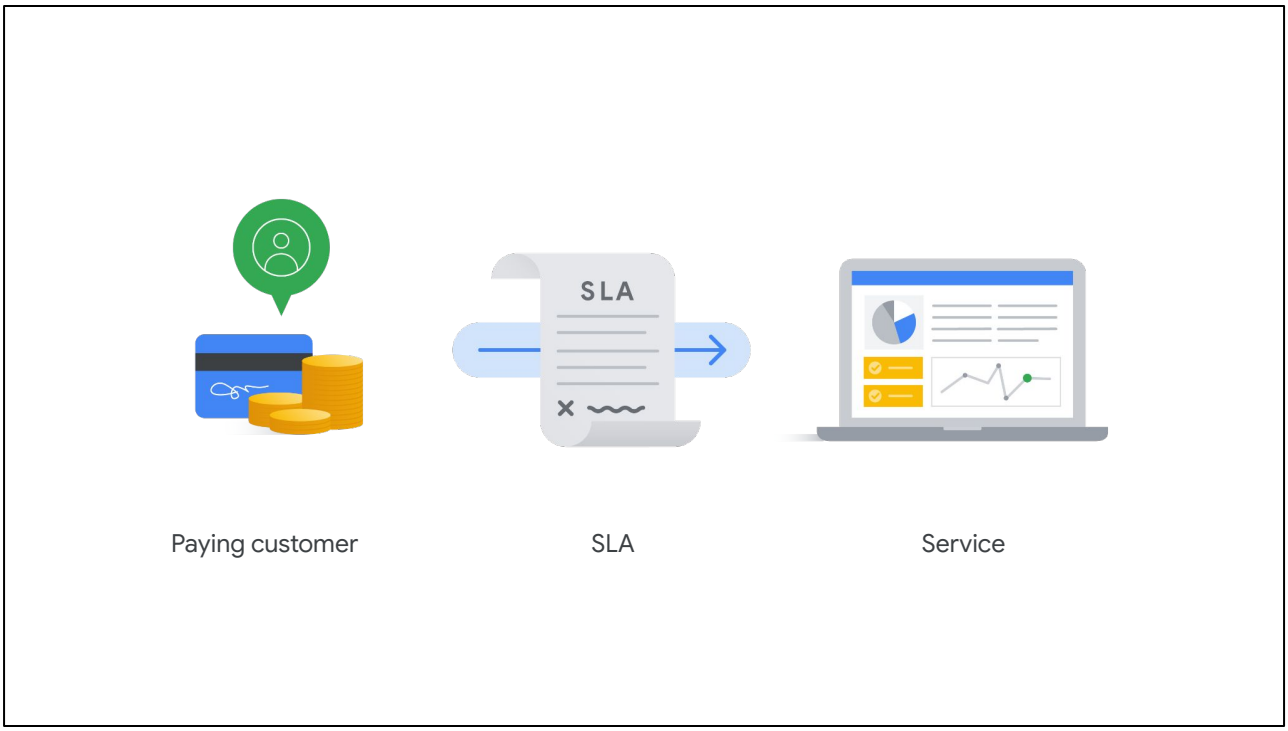
And then there are **Service Level Agreements, or SLAs**, which are commitments made to your customers that your systems and applications will have only a certain amount of “down time.”

01 The minimum levels of service that you promise to provide to your customers

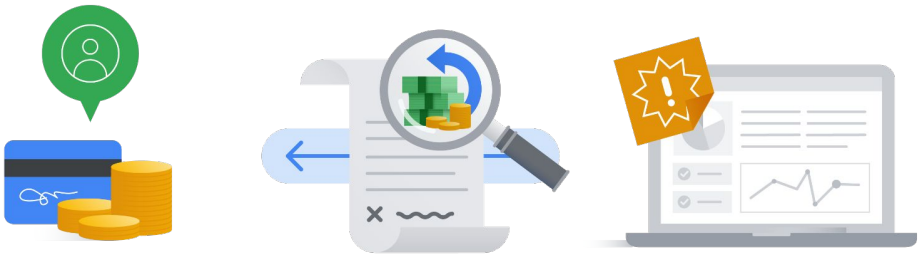
02 What happens when you break that promise



An SLA describes the minimum levels of service that you promise to provide to your customers and what happens when you break that promise.

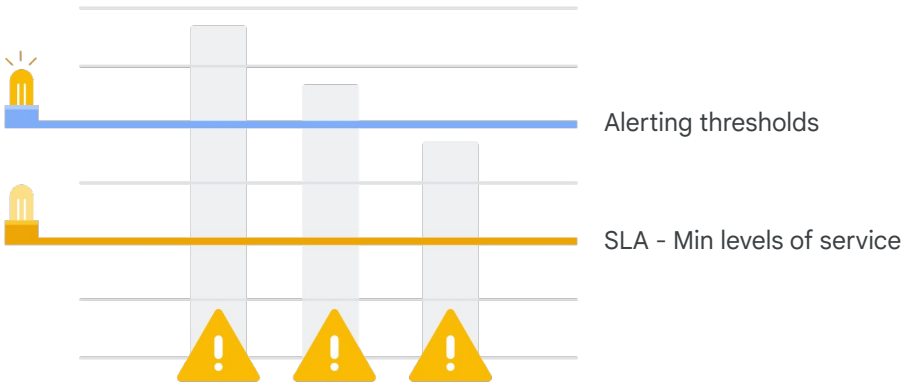


If your service has paying customers, an SLA may include some



Paying customer SLA compensation Service outage

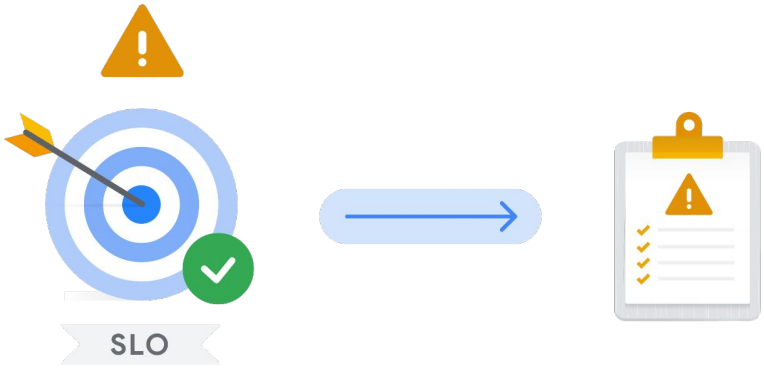
way of compensating them with refunds or credits when that service has an outage that is longer than this agreement allows.



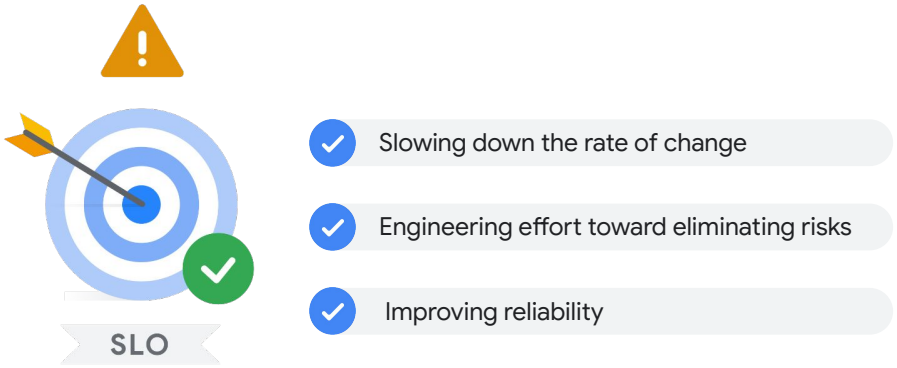
To give you the opportunity to detect problems and take remedial action before your reputation is damaged, your alerting thresholds are often substantially higher than the minimum levels of service documented in your SLA.

To improve service reliability, all parts of the business must agree that these are an accurate measure of user experience and must agree to use them as a primary driver for decision making

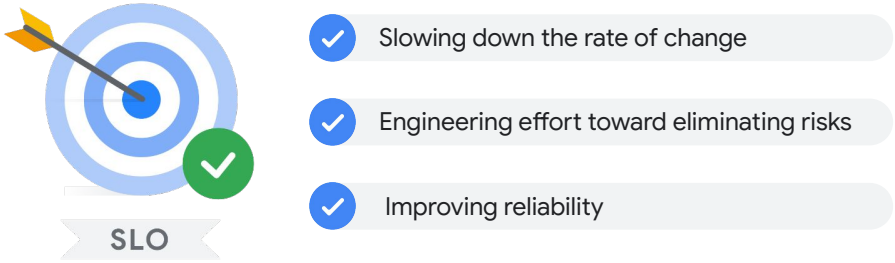
For SLOs, SLIs, and SLAs to help improve service reliability, all parts of the business must agree that they are an accurate measure of user experience and must also agree to use them as a primary driver for decision making.



Being out of SLO must have concrete, well-documented consequences, just as there are consequences for breaching SLAs.

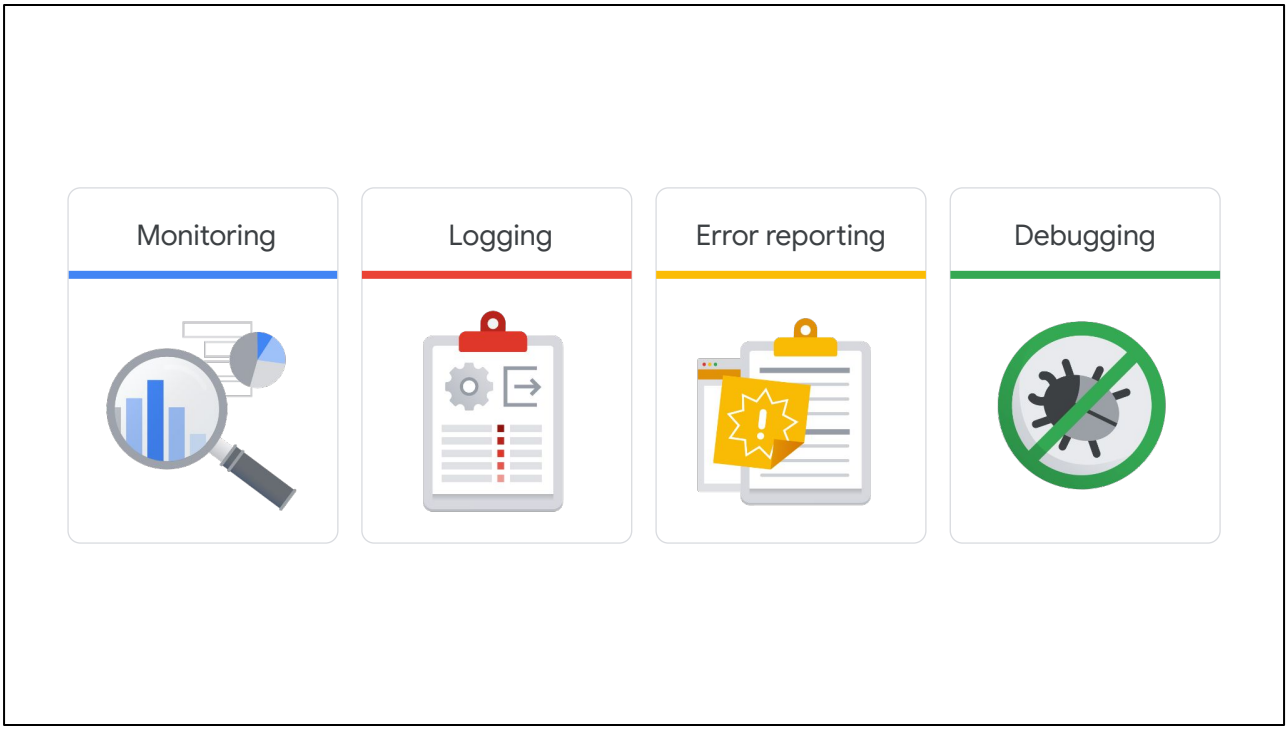


For example, slowing down the rate of change and directing more engineering effort toward eliminating risks and improving reliability

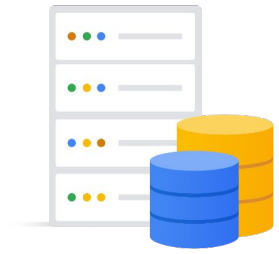


are actions that could be taken to get your product back to meeting its SLOs faster.

Operations teams need strong executive support to enforce these consequences and effect change in your development practice.

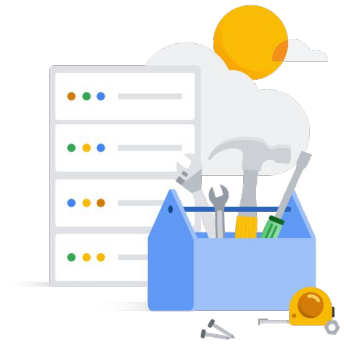


Let’s wrap up this section of the course by taking a look at Google Cloud’s integrated monitoring, logging, error reporting, and debugging tools.



On-premises

Physical check



Cloud

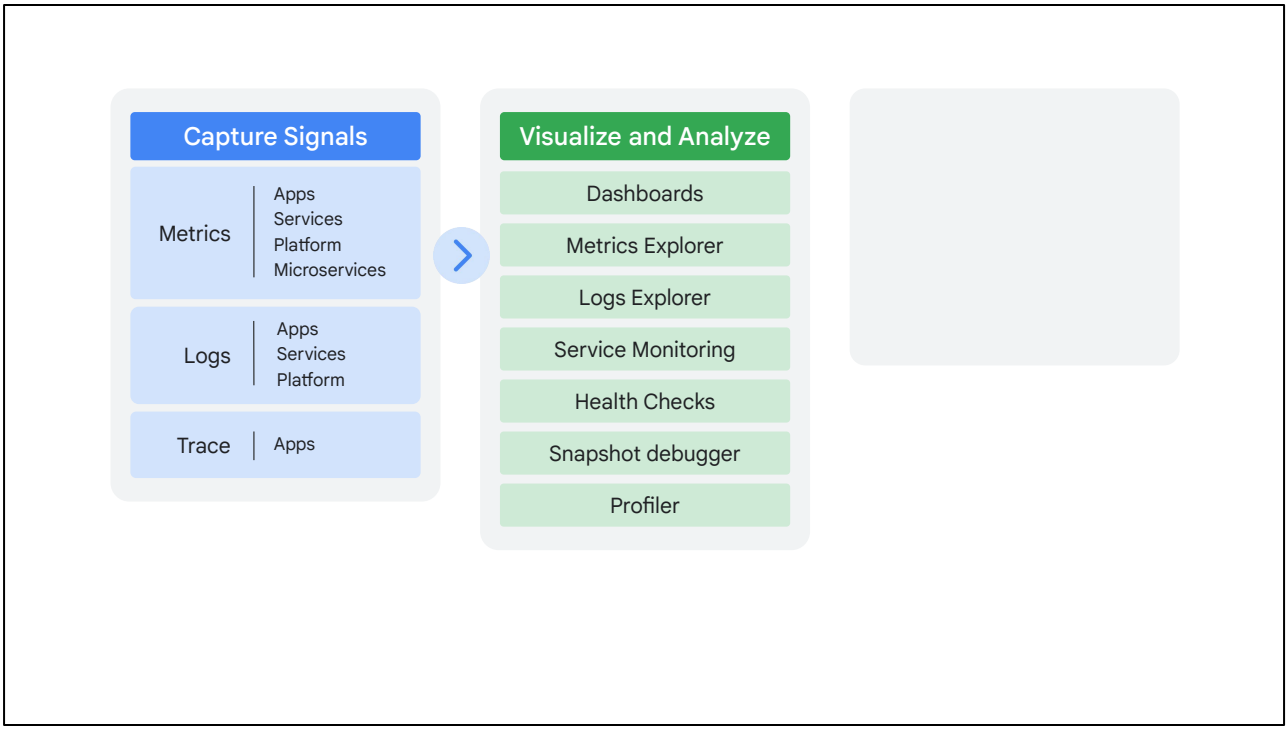
Observability tools

If you've ever worked with on-premises environments, you know that you can physically touch the servers. If an application becomes unresponsive, someone can physically determine why that happened.

In the cloud though, the servers aren't yours—they're Google's—and you can't physically inspect them. So the question becomes, how do you know what's happening with your server, or database, or application?

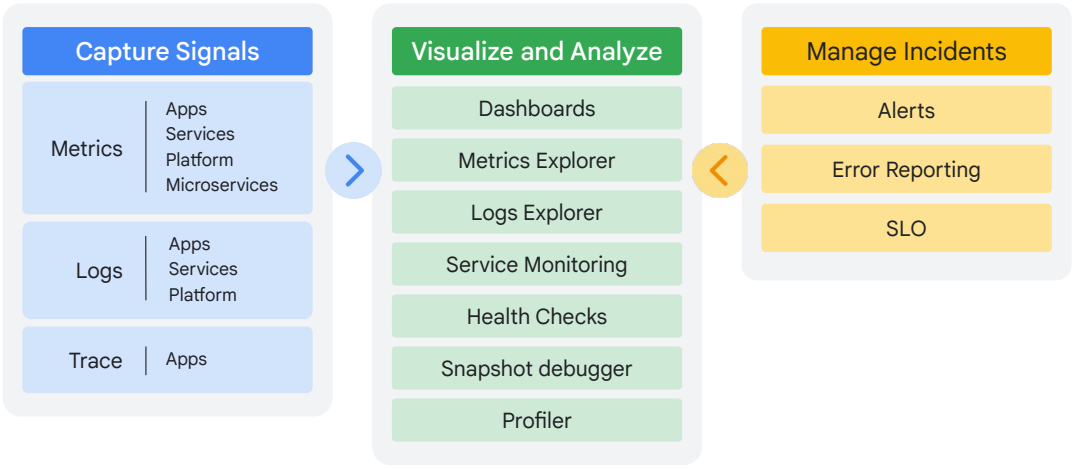
The answer is by using Google's integrated observability tools.

Observability starts with signals, which are metric, logging, and trace data captured and integrated into Google products from the hardware layer up.



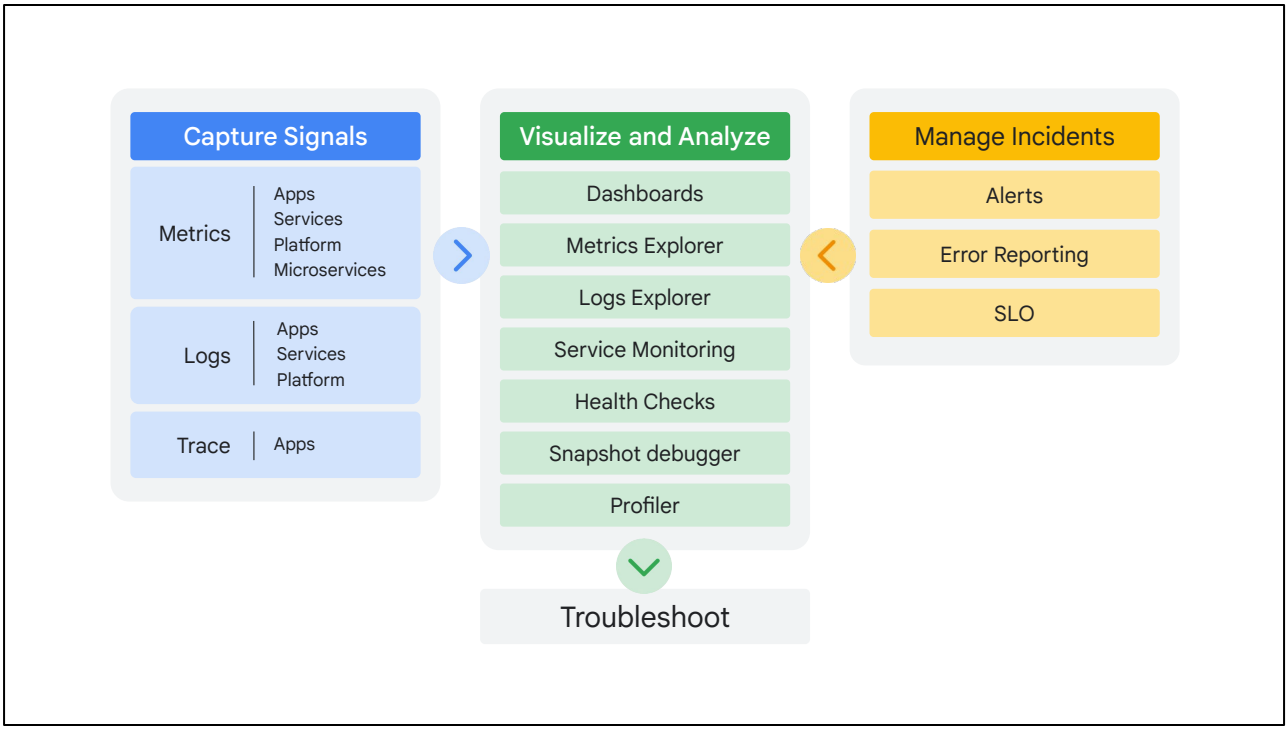
From those products:

- The signal data flows into the Google Cloud operation's tools where it can be visualized in dashboards and through the Metrics Explorer.
- Automated and custom logs can be dissected and analyzed in the Logs Explorer.
- Services can be monitored for compliance with service level objectives (SLOs), and error budgets can be tracked.
- Health checks can be used to check uptime and latency for external-facing sites and services.
- And running applications can be debugged And running applications can be debugged and profiled.



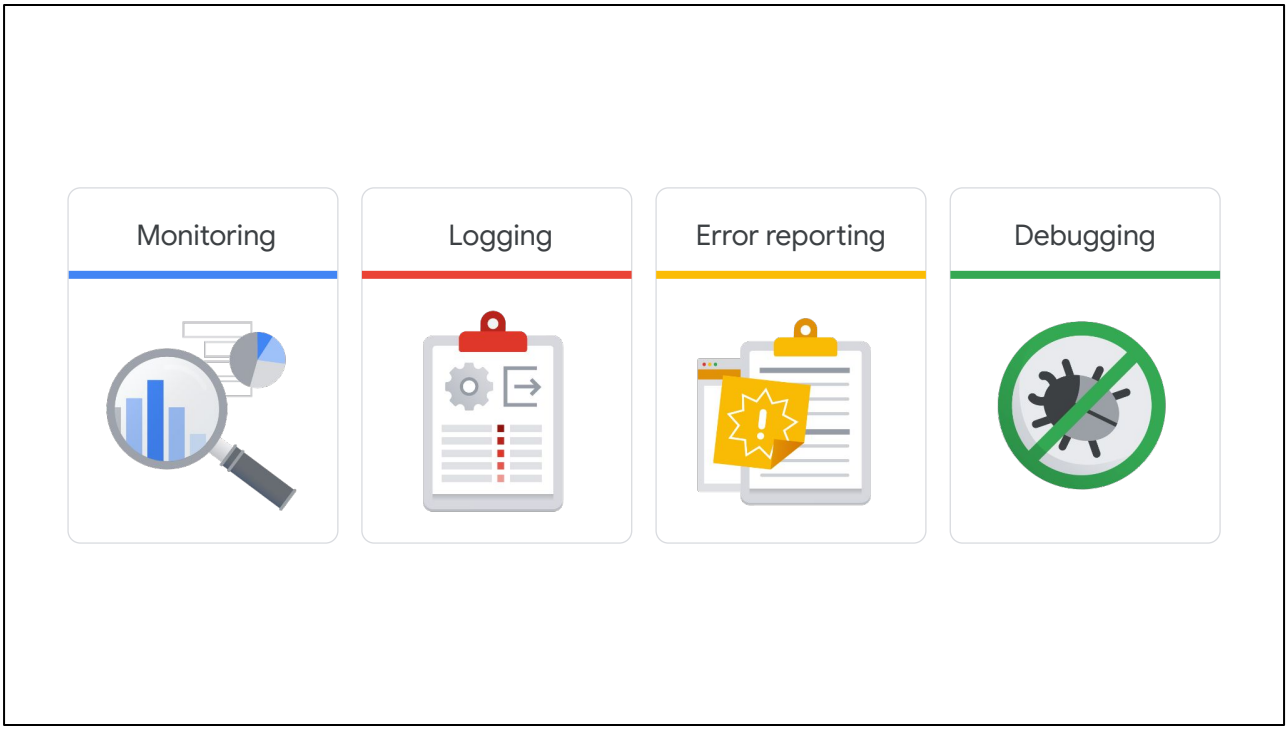
When incidents occur:

- Signal data can generate automated alerts to code or, through various information channels, to key personnel.
- Error Reporting can help operations and developer teams spot, count, and analyze crashes in cloud-based services.
- Service Level Objectives should be adhered to.

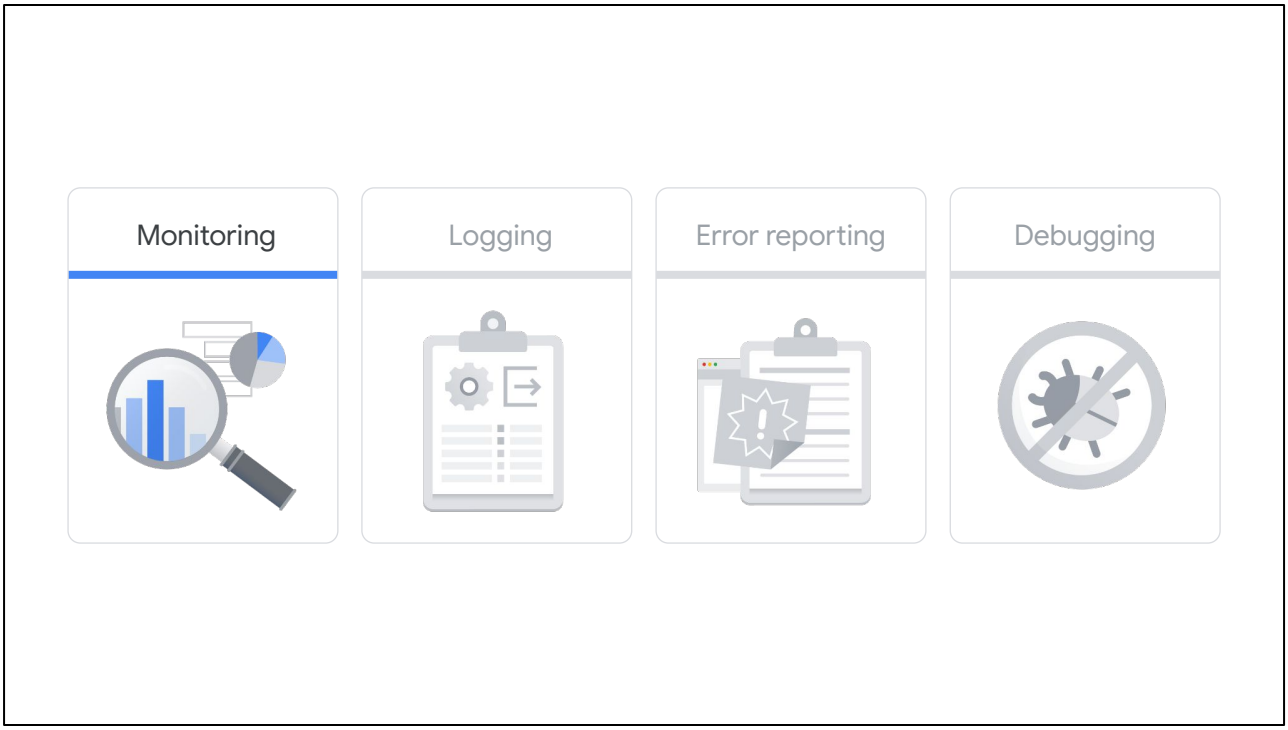


And the visualization and analysis tools can then help troubleshoot what's happening in Google Cloud.

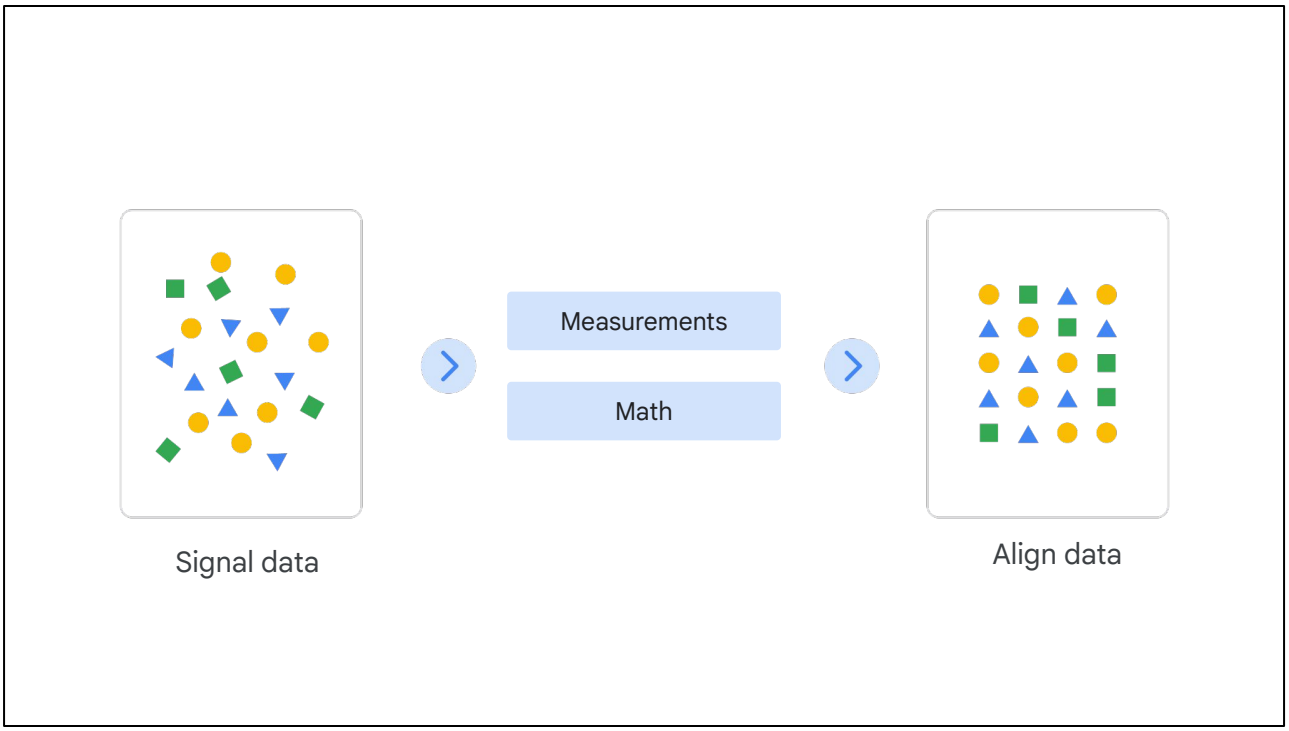
Ultimately, you won't miss that easy server access, because Google provides more precise insights into your Cloud install than you ever had on-premises.



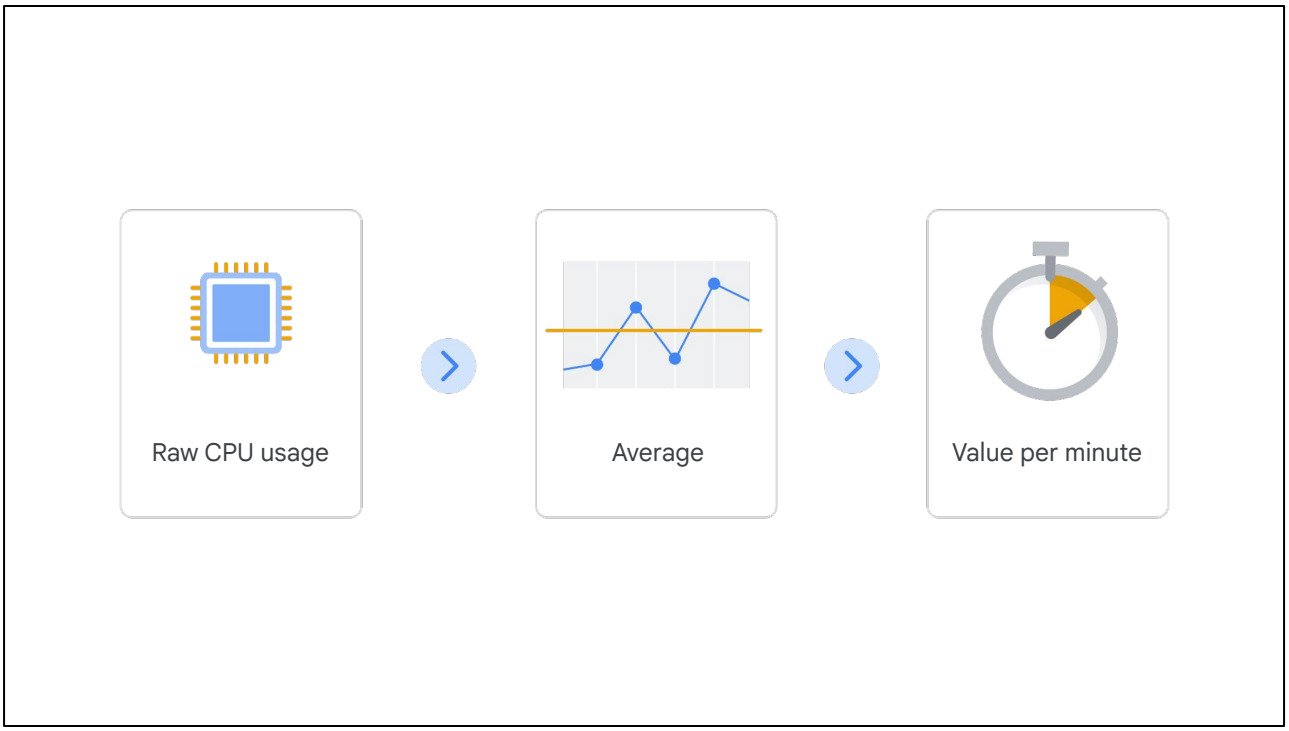
Over the next few videos we'll explore the products and tools offered by Google Cloud that are most applicable for those in operations roles that work with **monitoring**, **logging**, **error reporting**, and **debugging**.



When DevOps personnel want to track exactly what's happening inside Google Cloud projects, they often first think of **monitoring**.



As we stated previously, monitoring starts with signal data. Metrics take measurements and use math to align those measurements over time.

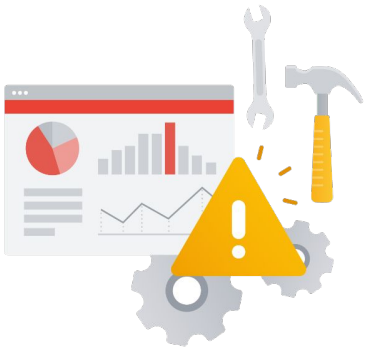


For example, it might be taking raw CPU usage measurement values and averaging them to produce a single value per minute.

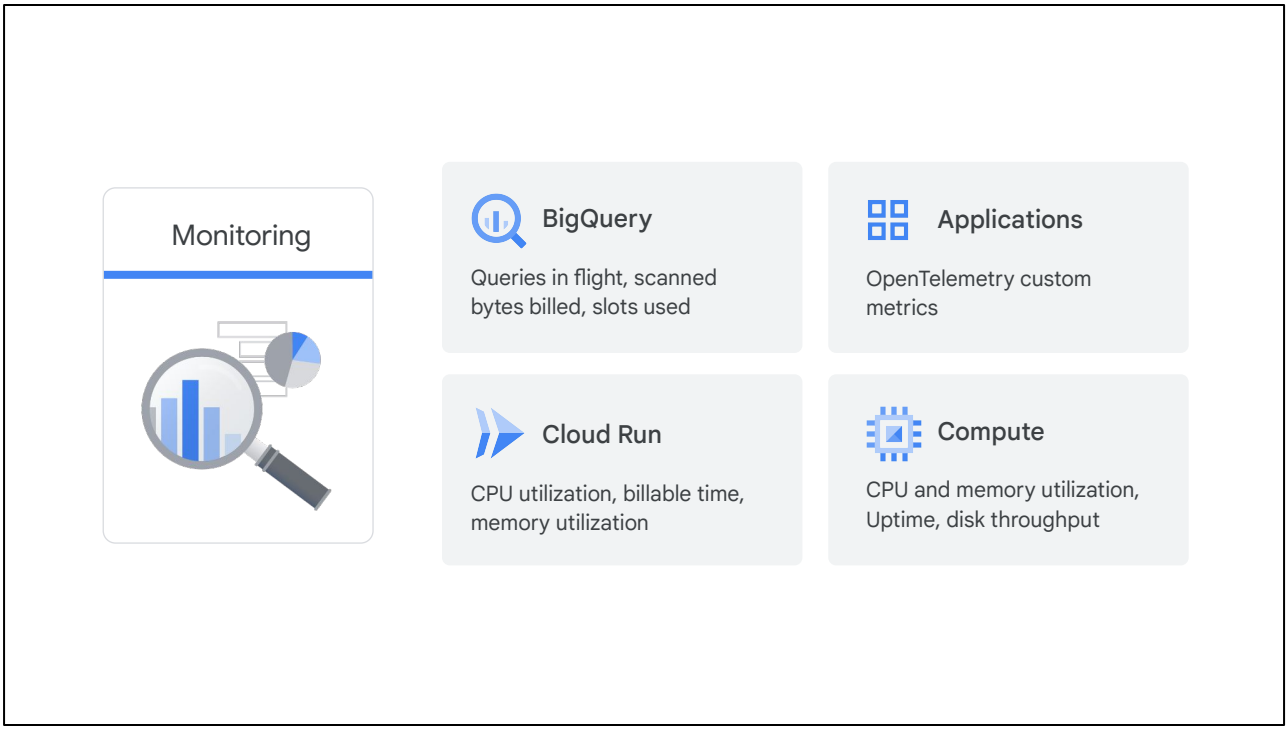
Google Cloud

1000

Streams of metric data



Google Cloud, by default, collects more than a thousand different streams of metric data, which can be incorporated into dashboards, alerts, and several other key tools.

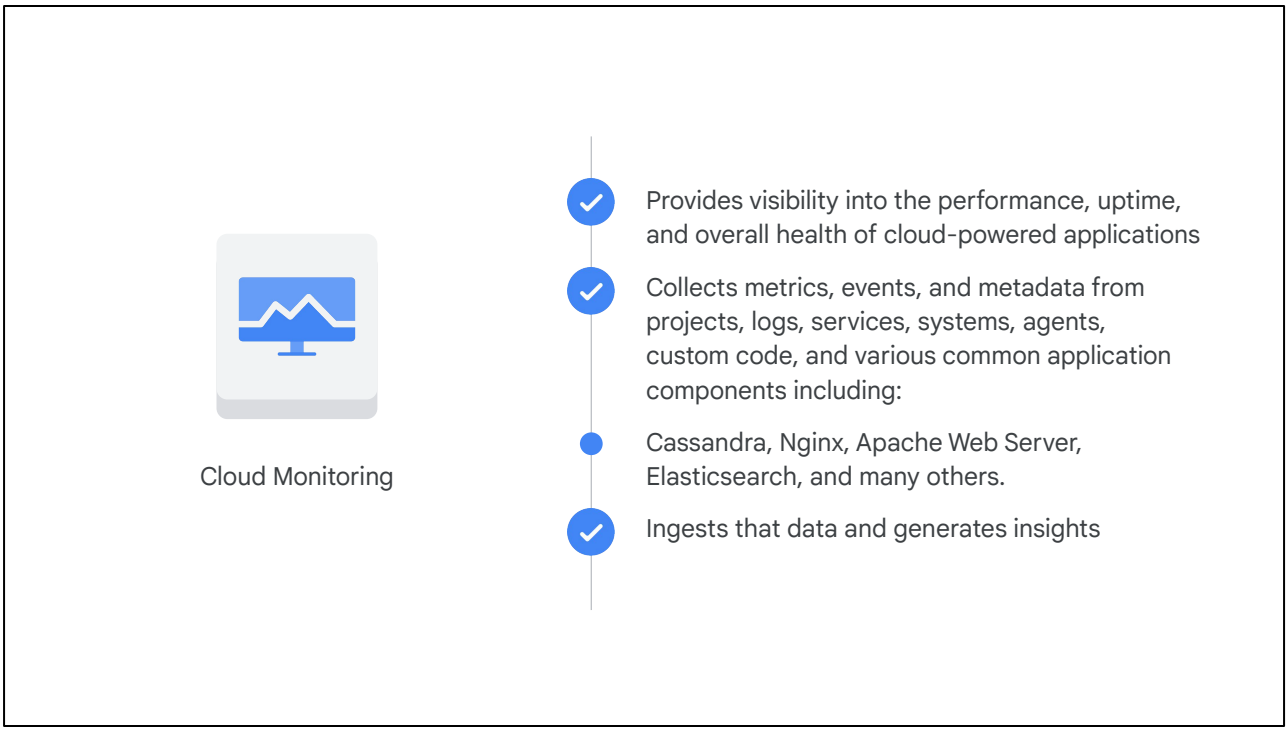


When data scientists run massive, scalable queries in **BigQuery**, it's important for them to know how many queries are currently in flight, how many bytes have been scanned and added to the bill, and data slot usage patterns.

It could also be critical to DevOps teams running containerized applications in **Cloud Run** to know CPU and memory utilization and app bill time.

If those same DevOps teams want to augment the signal metrics from their custom application wherever it's running, they could use the open-source **OpenTelementry** and create their own metrics.

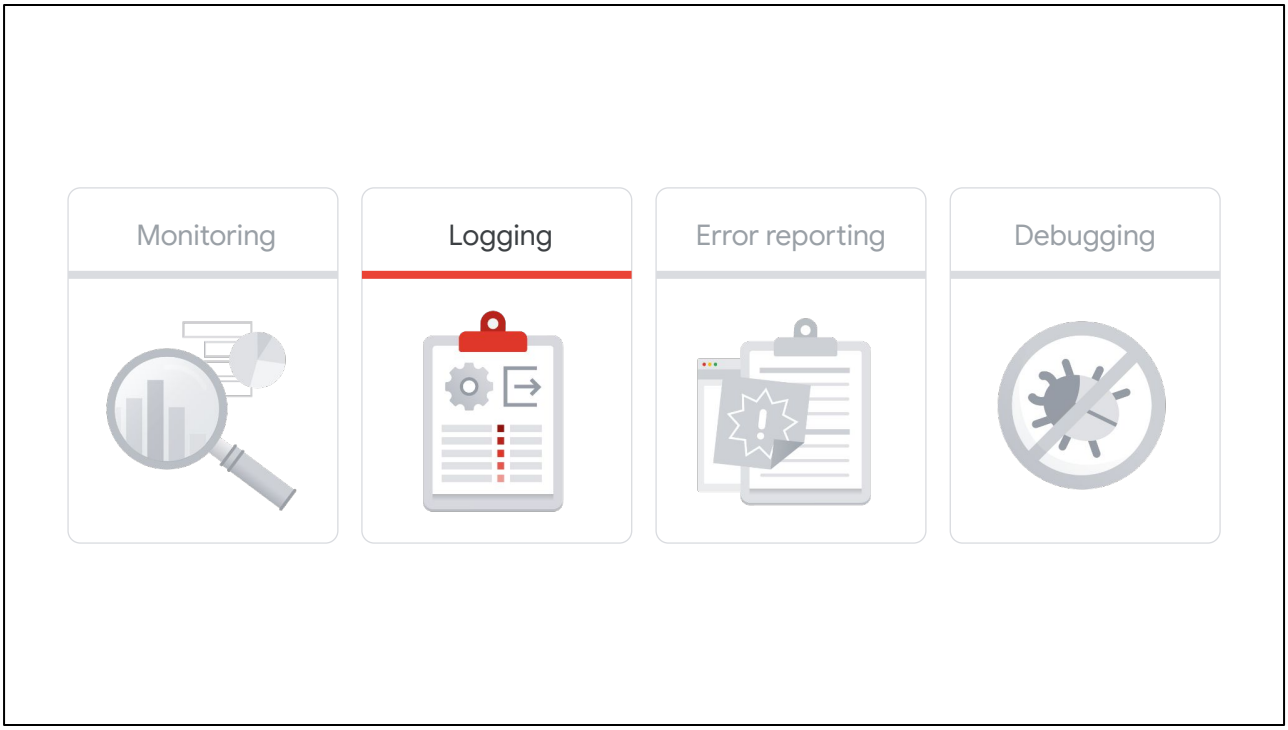
And workloads on **Compute Engine** will benefit from CPU and memory utilization data, along with uptime, disk throughput, and many other metrics.



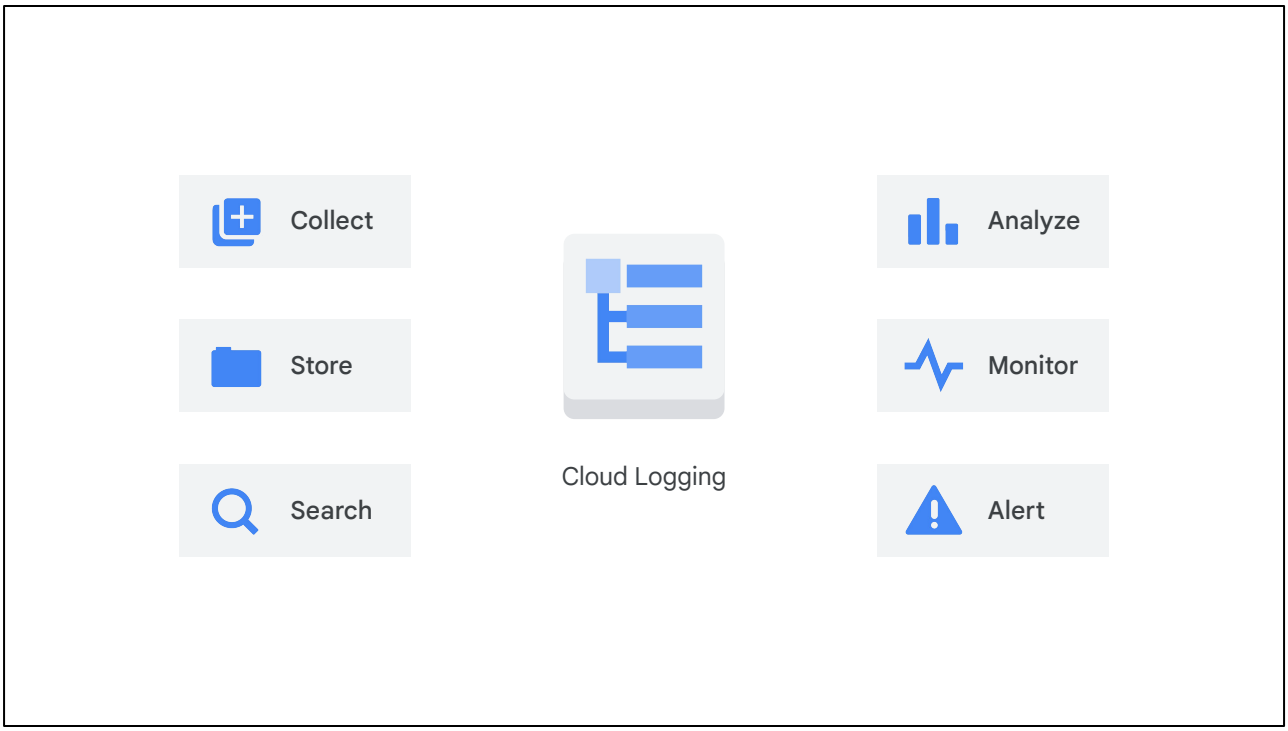
Cloud Monitoring provides visibility into the performance, uptime, and overall health of cloud-powered applications.

It collects metrics, events, and metadata from projects, logs, services, systems, agents, custom code, and various common application components, including Cassandra, Nginx, Apache Web Server, and Elasticsearch.

Cloud Monitoring ingests that data and generates insights via dashboards, Metrics Explorer charts, and automated alerts.



In this video we'll take a look at Google Cloud's integrated logging tools.



Cloud Logging allows users to collect, store, search, analyze, monitor, and alert on log entries and events. Automated logging is integrated into Google Cloud products like App Engine, Cloud Run, Compute Engine VMs running the logging agent, and GKE.

Cloud Logging

Log analysis

- Uses Google Cloud's integrated Logs Explorer
- Entries can be exported to several destinations
- Pub/Sub messages can be analyzed in near-real time using custom code or stream processing
- BigQuery allows examination of logging data through SQL queries
- Archived log files in Cloud Storage can be analyzed with several tools and techniques

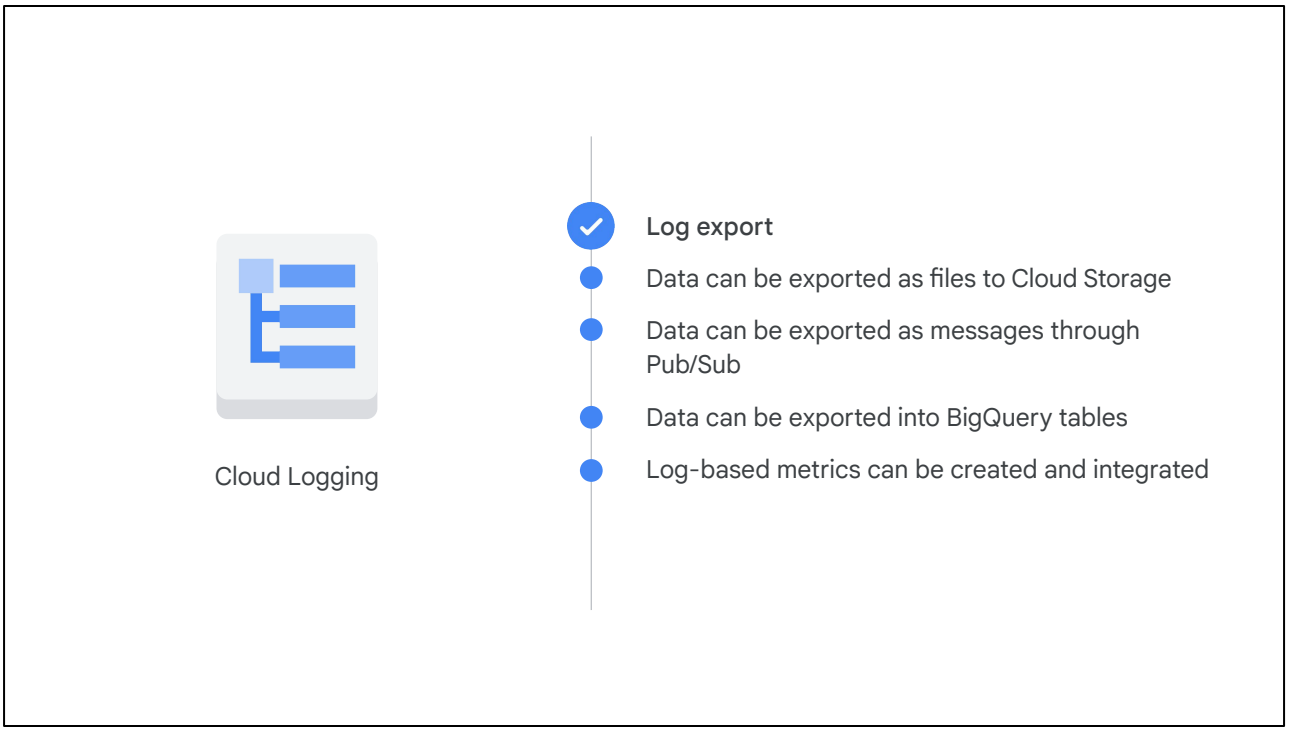
Most log analysis starts with Google Cloud's integrated Logs Explorer.

Logging entries can also be exported to several destinations for alternative or further analysis.

Pub/Sub messages can be analyzed in near-real time using custom code or stream processing technologies like Dataflow.

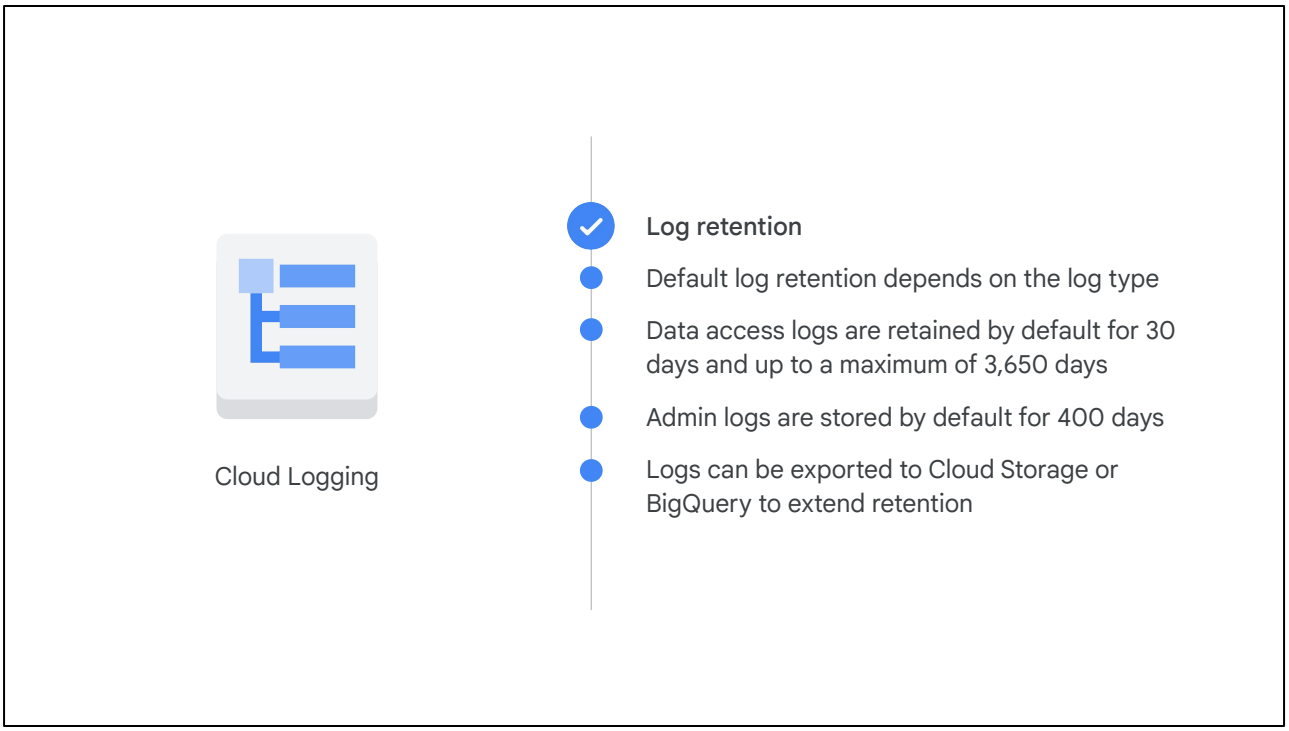
BigQuery allows analysts to examine logging data through SQL queries.

And archived log files in Cloud Storage can be analyzed with several tools and techniques.



Log data can be exported as files to Cloud Storage, as messages through Pub/Sub, or into BigQuery tables.

Log-based metrics can be created and integrated into Cloud Monitoring dashboards, alerts, and service SLOs.

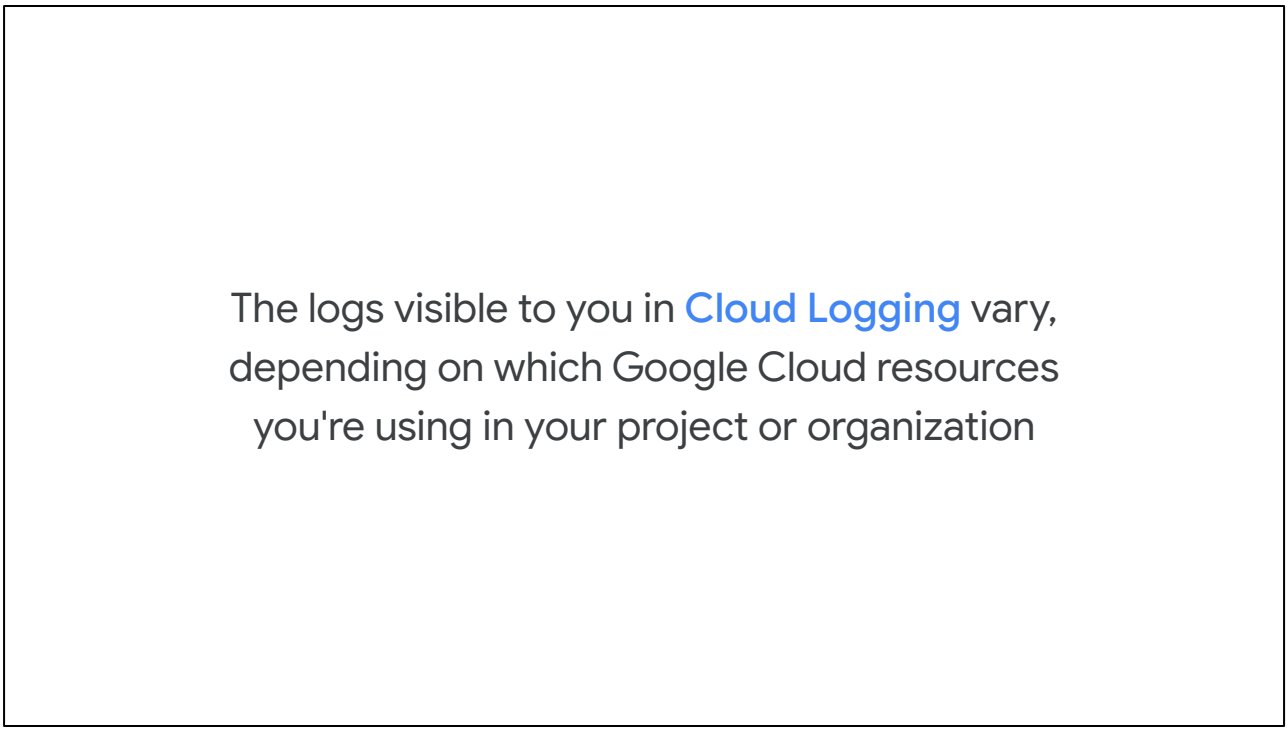


Default log retention in Cloud Logging depends on the log type.

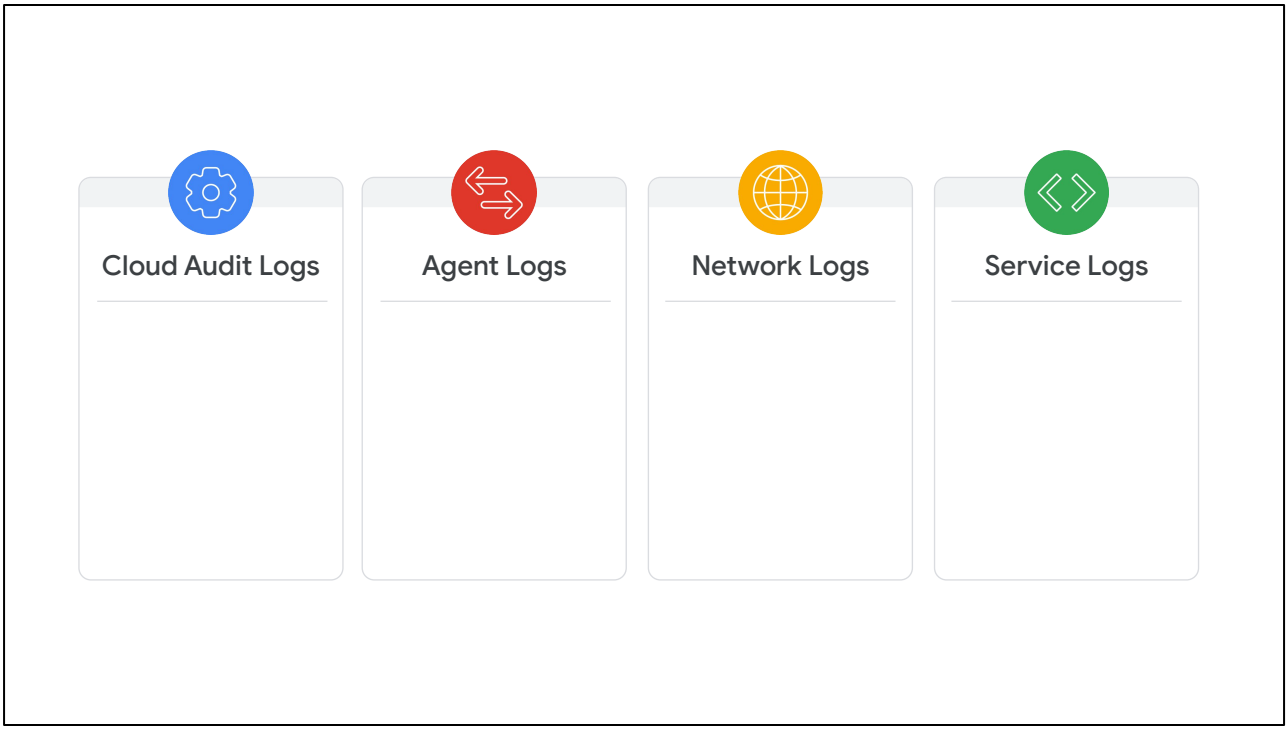
Data access logs are retained by default for 30 days, but this is configurable up to a maximum of 3,650 days.

Admin logs are stored by default for 400 days.

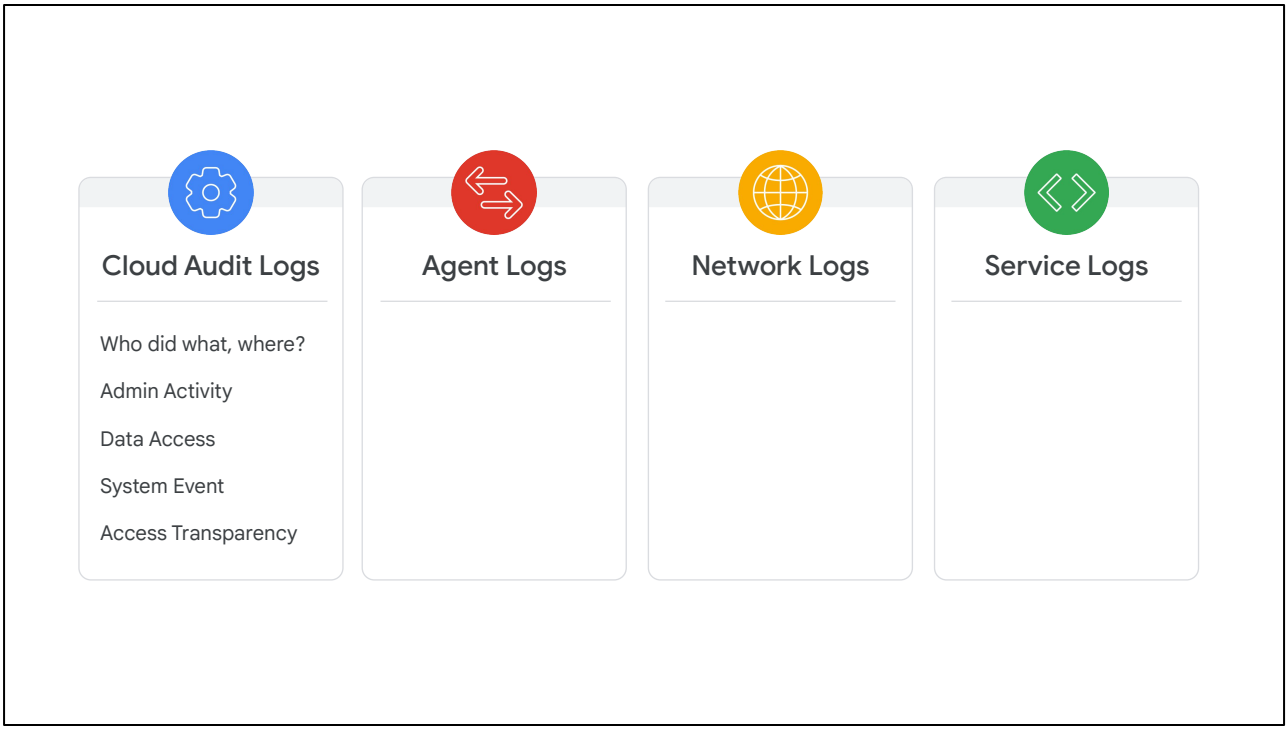
Logs can be exported to Cloud Storage or BigQuery to extend retention.



The logs visible to you in Cloud Logging vary, depending on which Google Cloud resources you're using in your project or organization.



Four key log categories are **audit logs**, **agent logs**, **network logs**, and **service logs**.



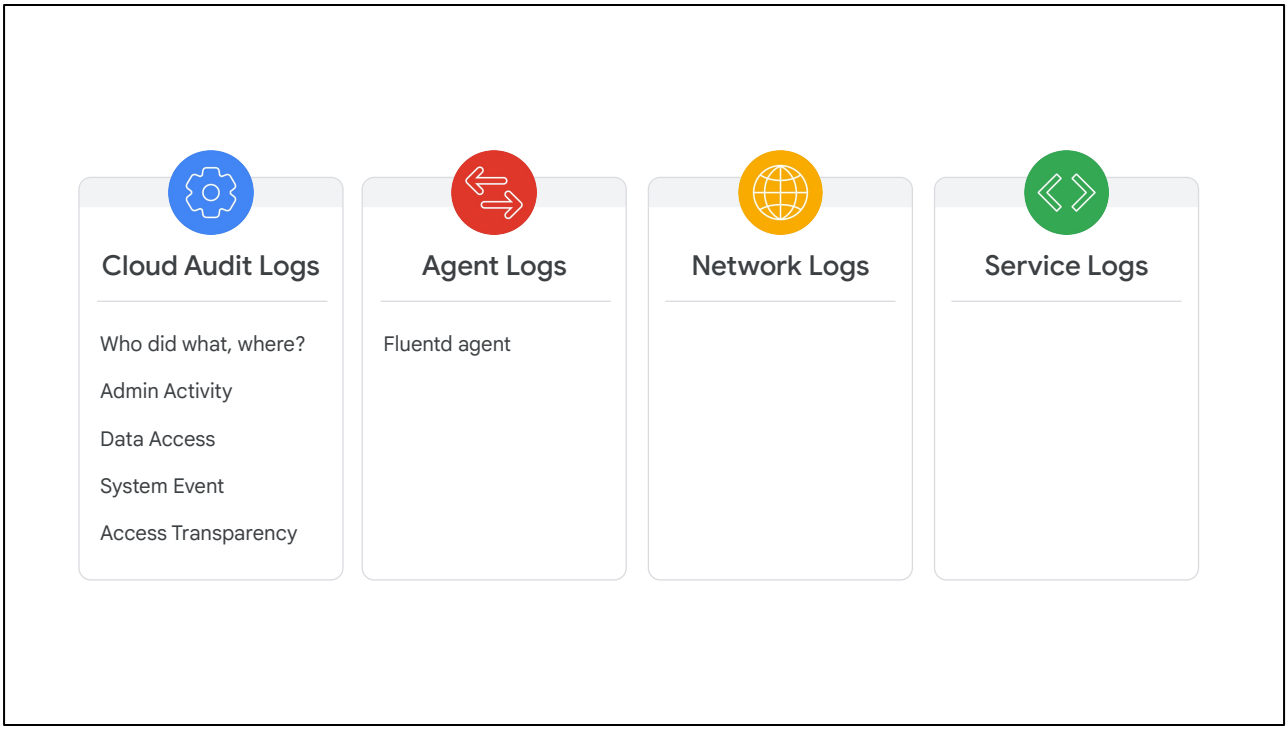
Cloud Audit Logs helps answer the question, "Who did what, where, and when?"

Admin activity tracks configuration changes.

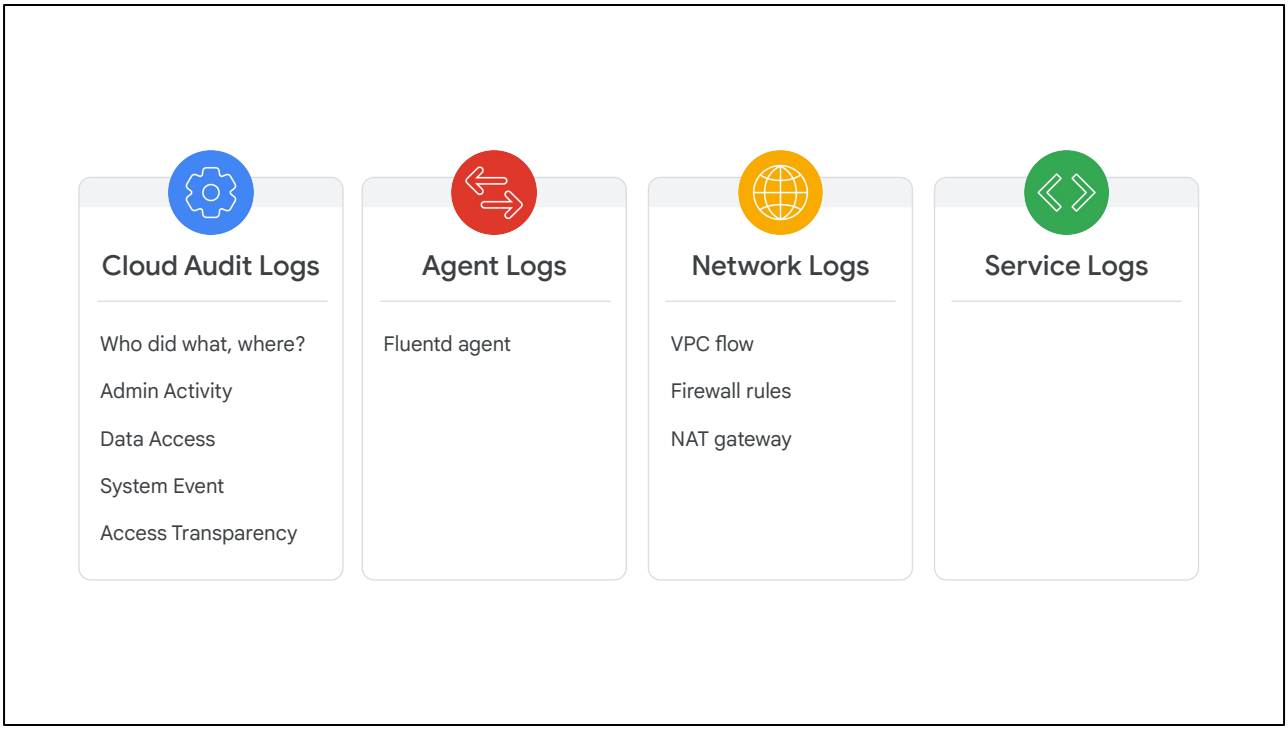
Data access tracks calls that read the configuration or metadata of resources and user-driven calls that create, modify, or read user-provided resource data.

System events are non-human Google Cloud administrative actions that change the configuration of resources.

And Access Transparency provides you with logs that capture the actions Google personnel take when accessing your content.



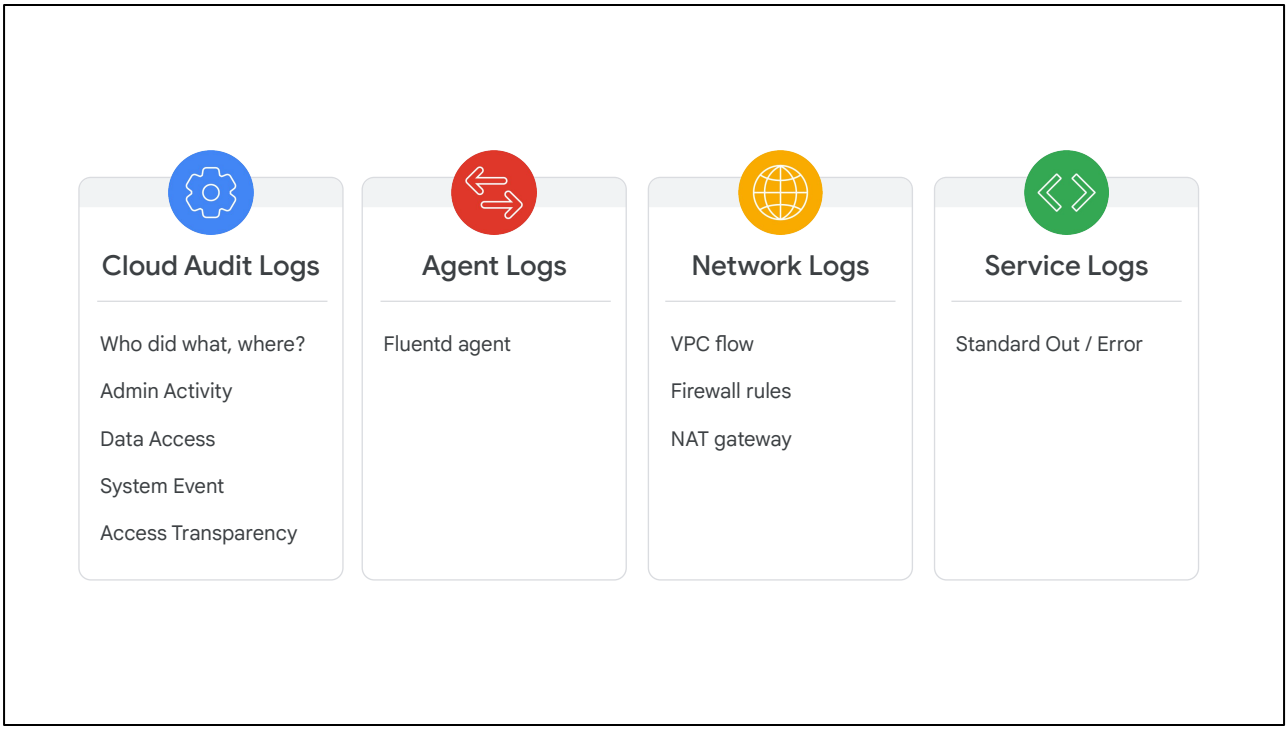
Agent logs use a Google-customized and packaged Fluentd agent that can be installed on any AWS or Google Cloud VM to ingest log data from Google Cloud instances–for example, Compute Engine, Managed VMs, or Containers–and AWS EC2 instances.



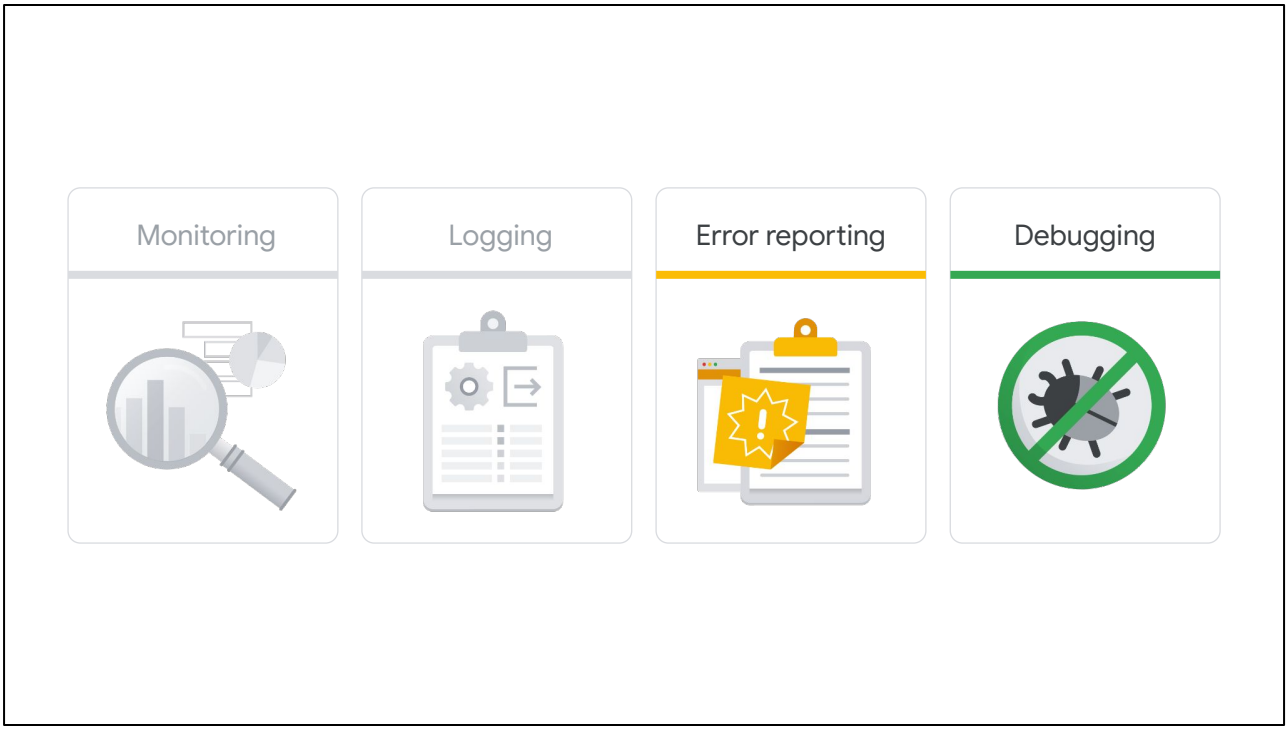
Network logs provide both network and security operations with in-depth network service telemetry. VPC Flow Logs records samples of VPC network flow and can be used for network monitoring, forensics, real-time security analysis, and expense optimization.

Firewall Rules Logging allows you to audit, verify, and analyze the effects of your firewall rules.

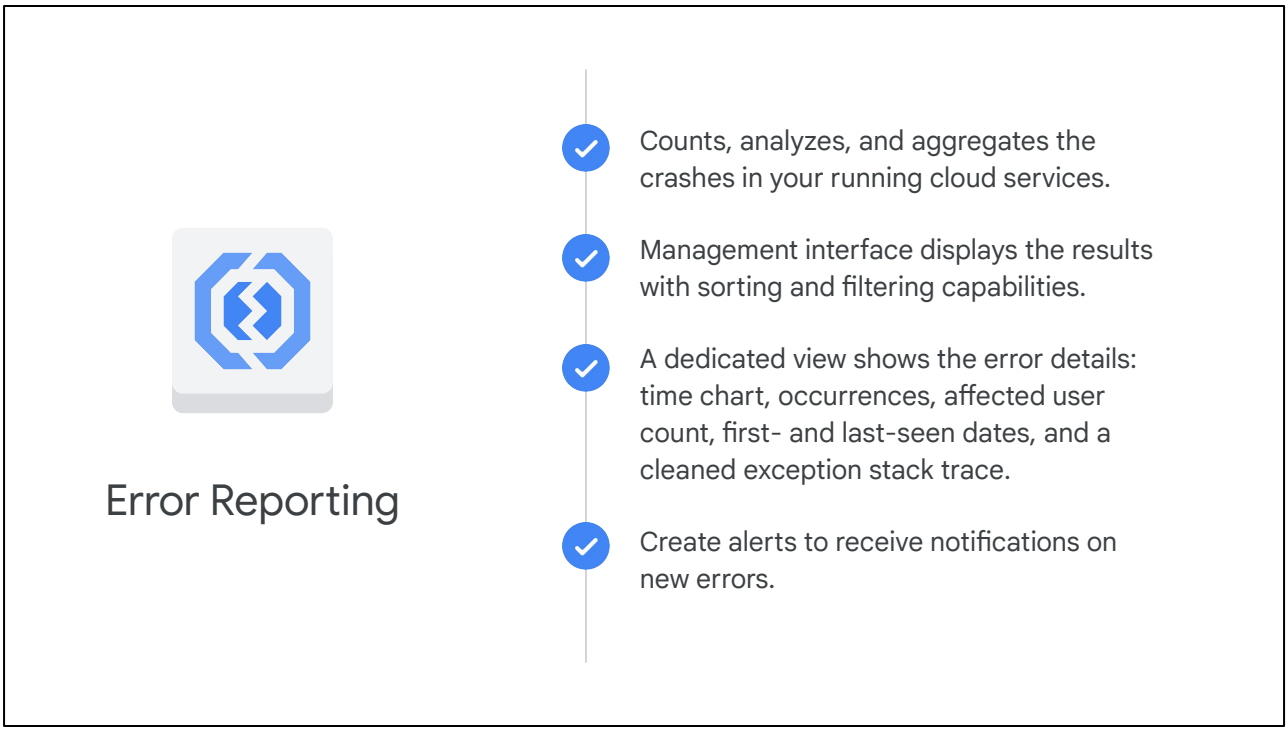
NAT Gateway logs capture information on NAT network connections and errors.



Service logs provide access to logs created by developers deploying code to Google Cloud. For example, if they build a container using Node.js and deploy it to Cloud Run, any logging to Standard Out or Standard Error will automatically be sent to Cloud Logging for easy, centralized viewing.



Let's round off this section of the course by taking a look at the tools offered by Google Cloud for error reporting and debugging.

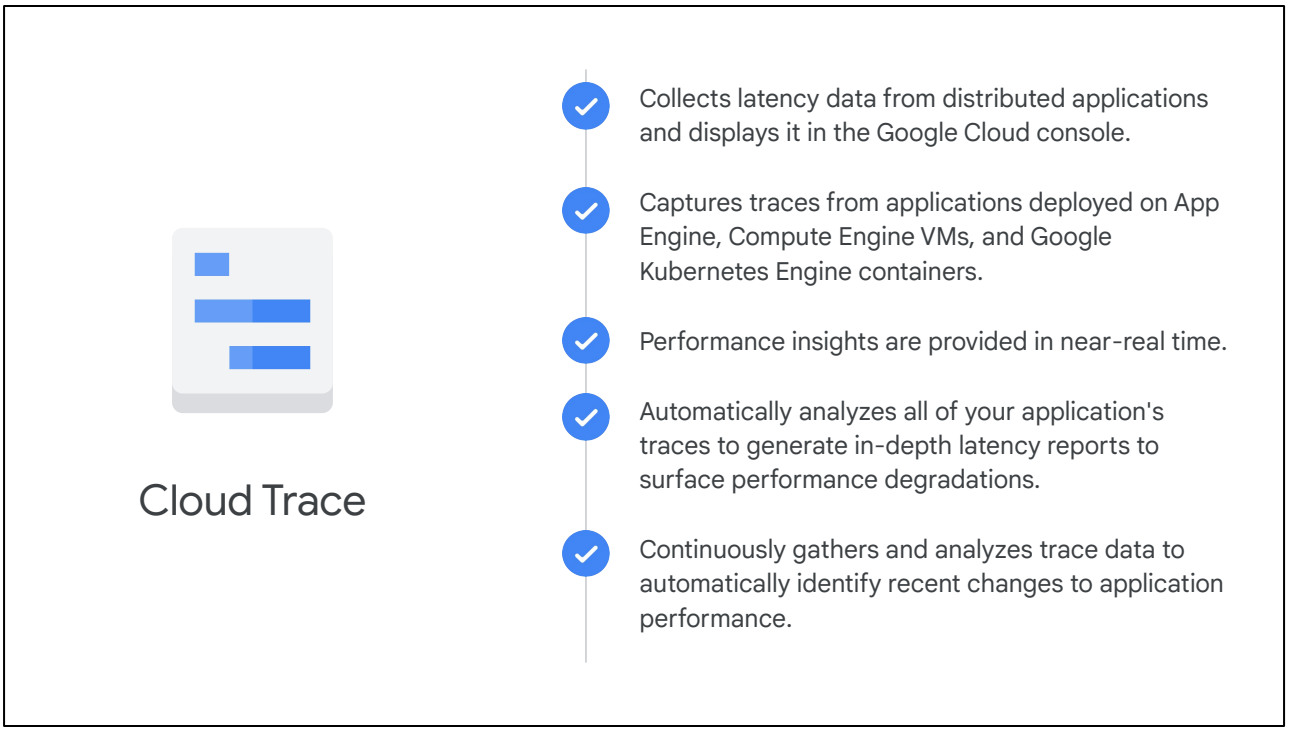


Error Reporting counts, analyzes, and aggregates the crashes in your running cloud services. Crashes in most modern languages are “Exceptions,” which aren’t caught and handled by the code itself.

Its management interface displays the results with sorting and filtering capabilities.

A dedicated view shows the error details: time chart, occurrences, affected user count, first- and last-seen dates, and a cleaned exception stack trace.

You can also create alerts to receive notifications on new errors.

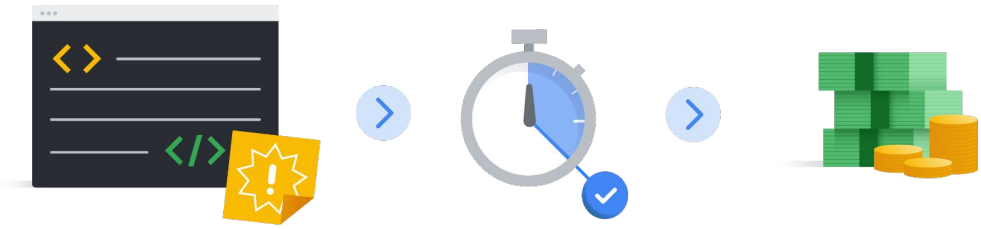


Cloud Trace, based on the tools Google uses on its production services, is a tracing system that collects latency data from your distributed applications and displays it in the Google Cloud console.

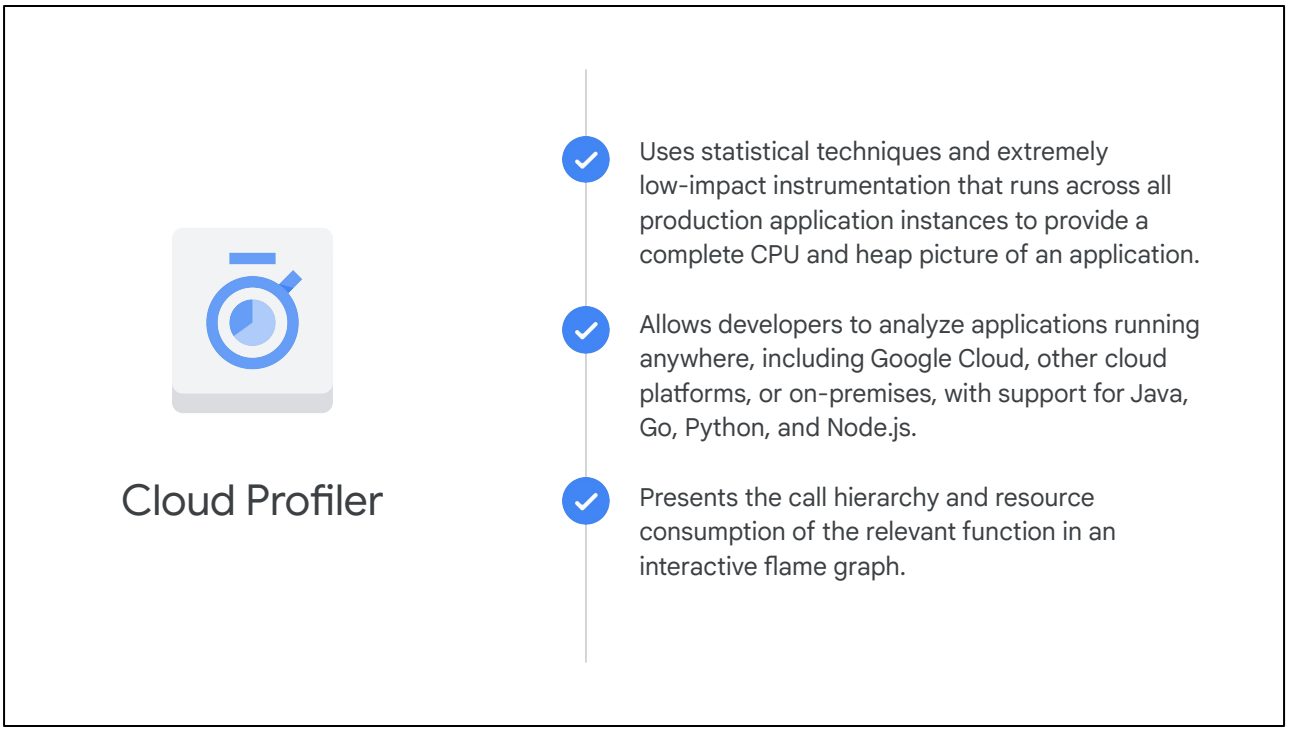
Trace can capture traces from applications deployed on App Engine, Compute Engine VMs, and Google Kubernetes Engine containers.

Performance insights are provided in near-real time, and Trace automatically analyzes all of your application's traces to generate in-depth latency reports to surface performance degradations.

Trace continuously gathers and analyzes trace data to automatically identify recent changes to your application's performance.



Poorly performing code increases the latency and cost of applications and web services every day, without anyone knowing or doing anything about it.



Cloud Profiler changes this by using statistical techniques and extremely low-impact instrumentation that runs across all production application instances to provide a complete CPU and heap picture of an application without slowing it down.

With broad platform support that includes Compute Engine VMs, App Engine, and Kubernetes, it allows developers to analyze applications running anywhere, including Google Cloud, other cloud platforms, or on-premises, with support for Java, Go, Python, and Node.js.

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.