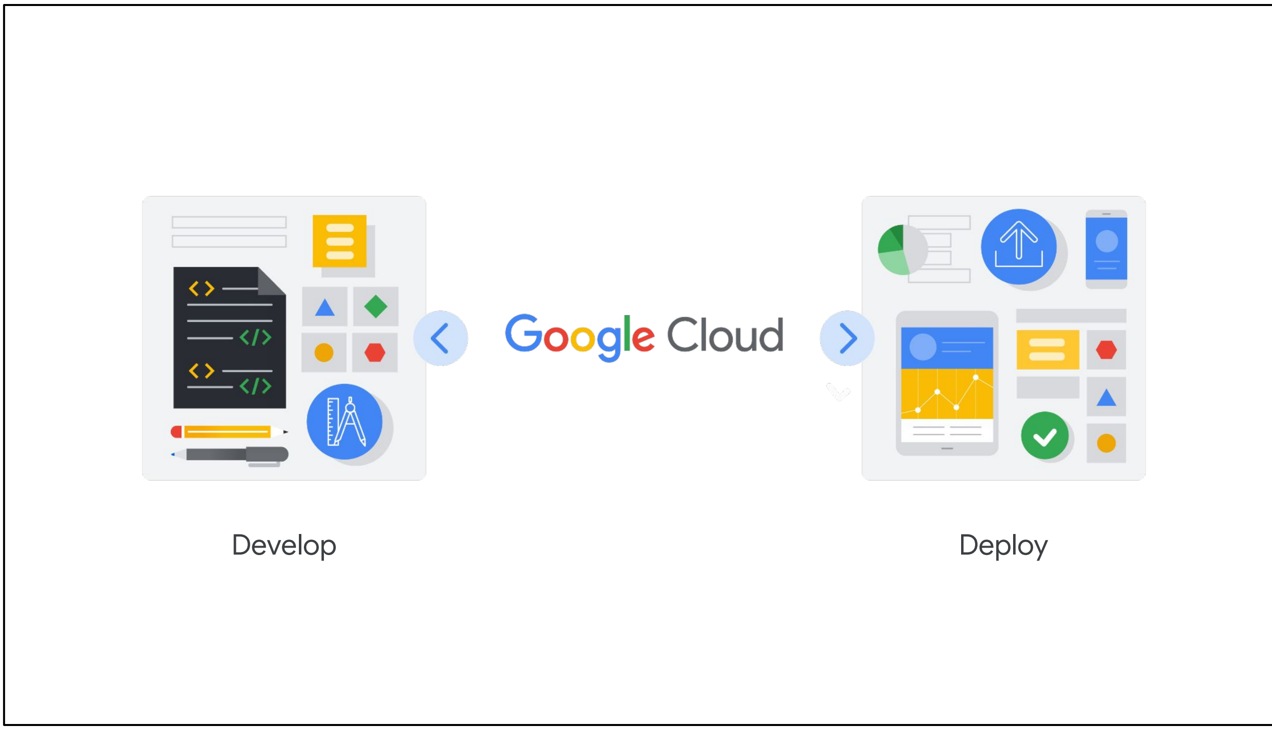
Google Cloud

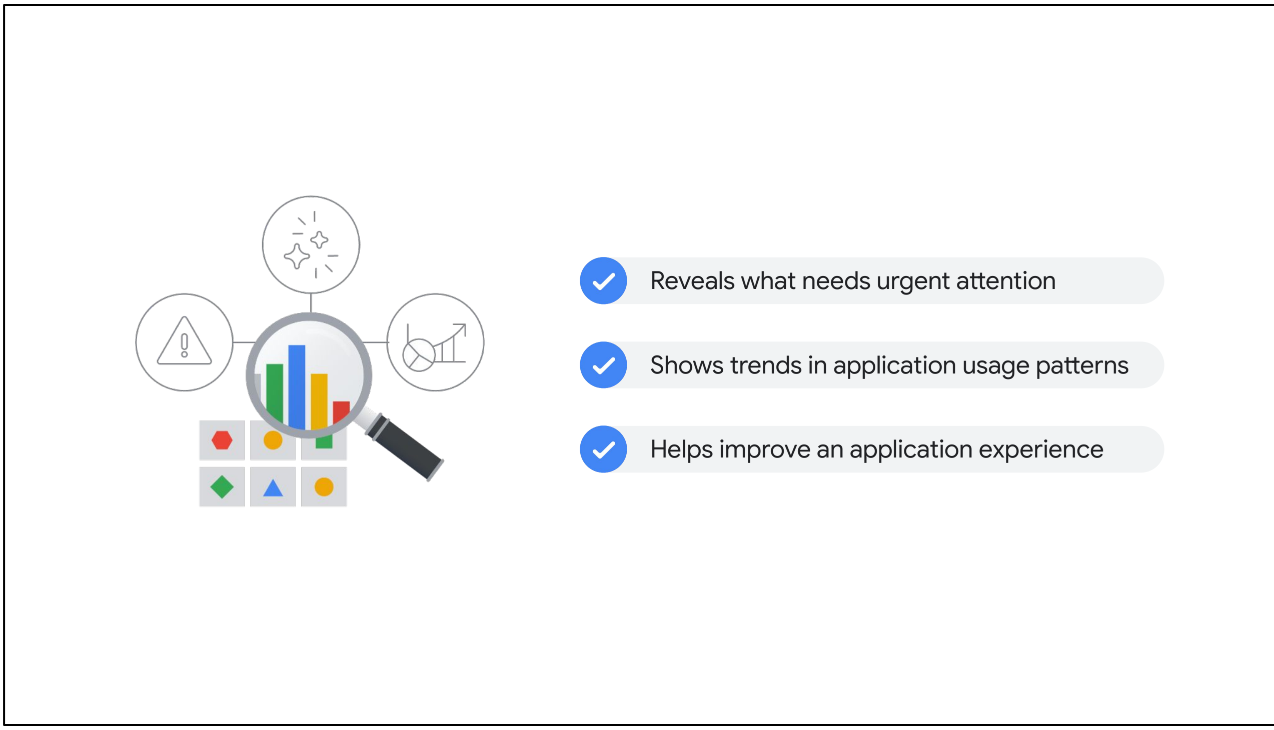
Google Cloud Core Infrastructure Module 8

On-demand course

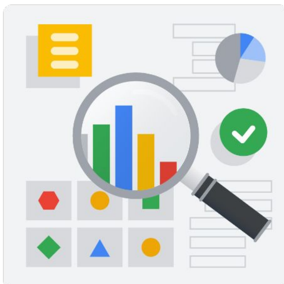
March 2022



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



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.

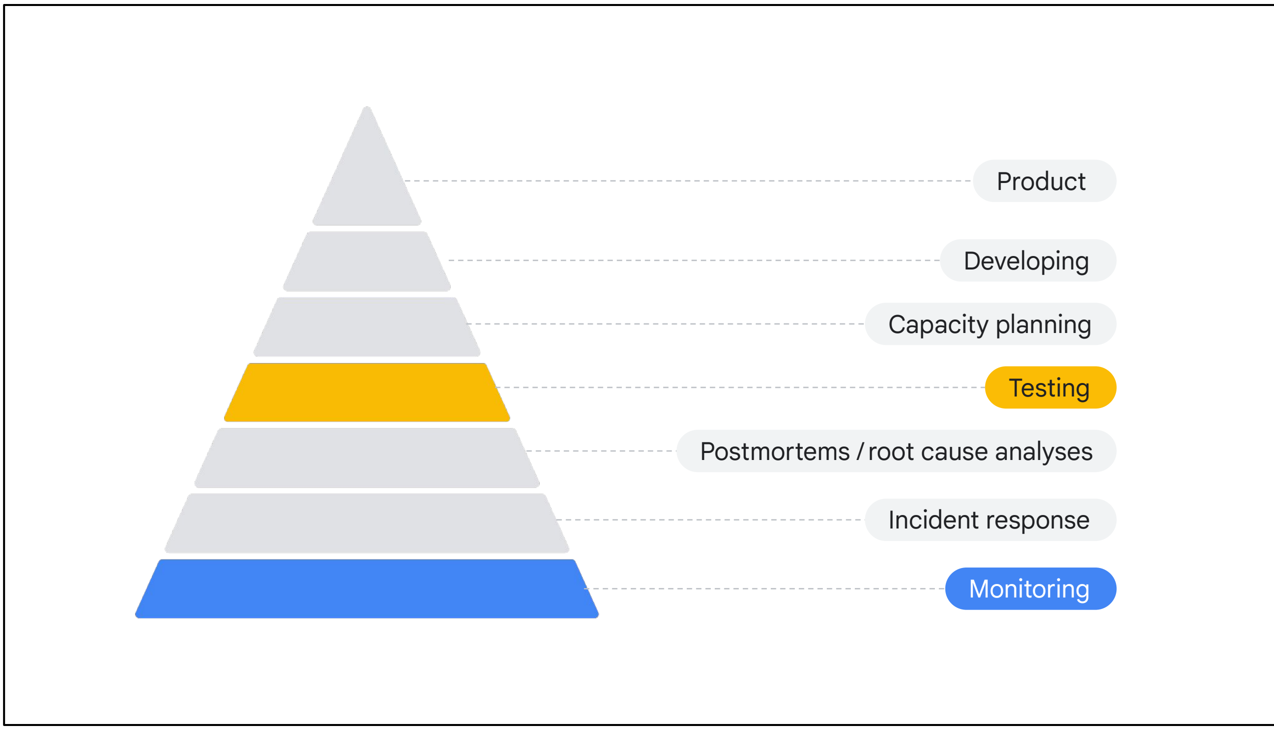


Monitoring

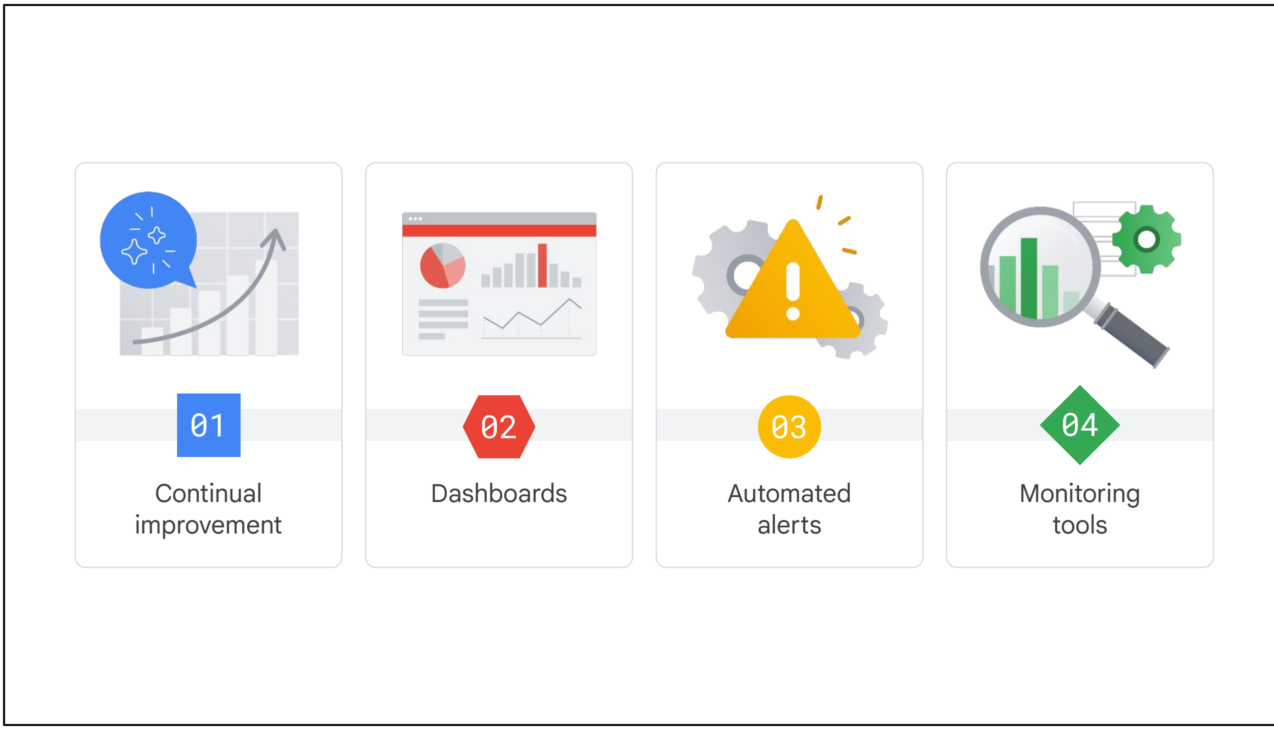
- Ensure continued system operations
- Uncover trend analyses over time
- Build dashboards
- Alert personnel when systems violate predefined SLOs
- Compare systems and systems changed
- Provide data for improved incident response

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.



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

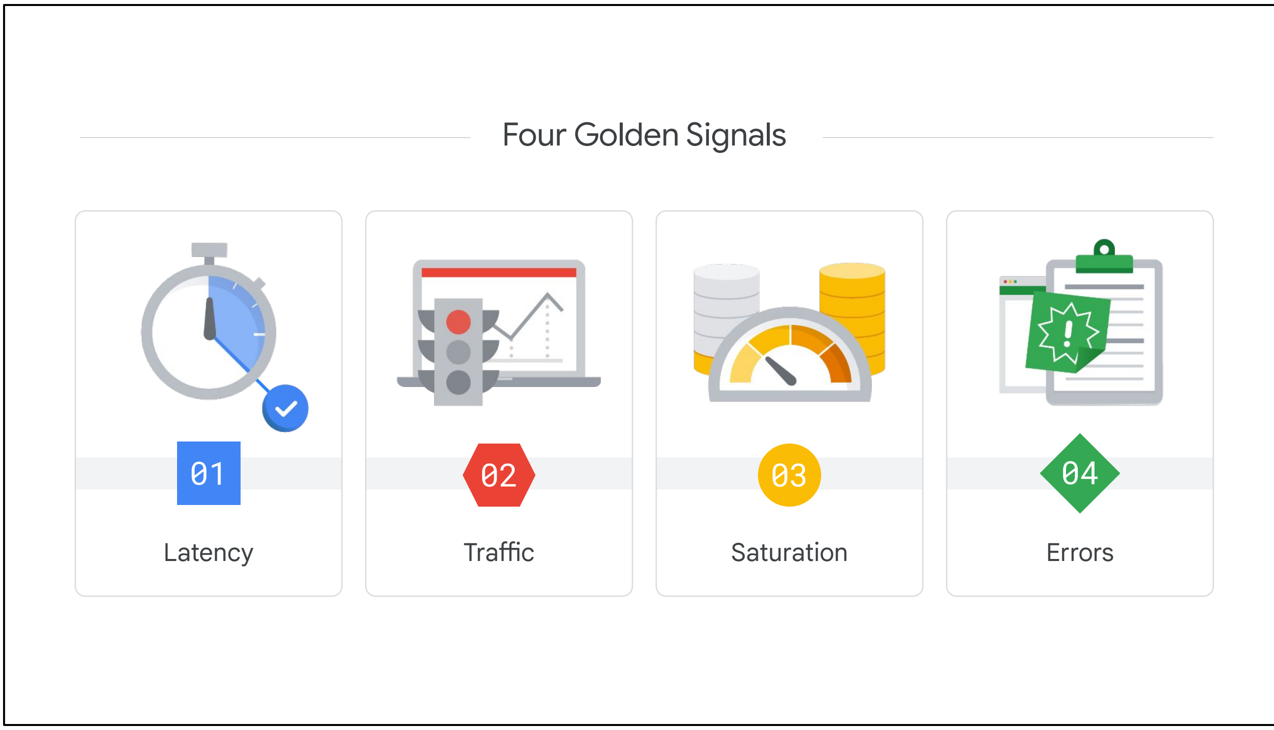


We need our products to **improve continually**, and we need data we can receive from monitoring to make sure that happens.

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

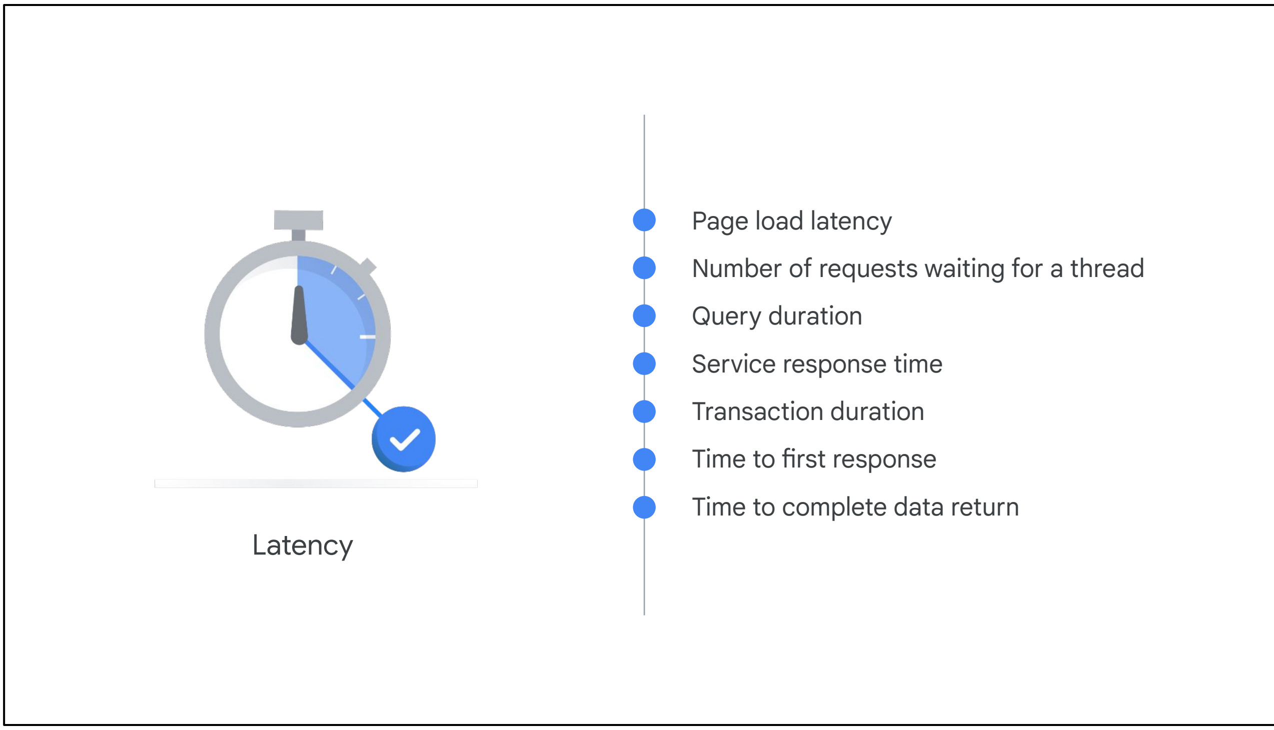
We need **automated alerts** because humans tend to look at things only when there's something important to look at. An even better option is to construct automated systems to handle as many alerts as possible so humans only have to look at the most critical issues.

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.

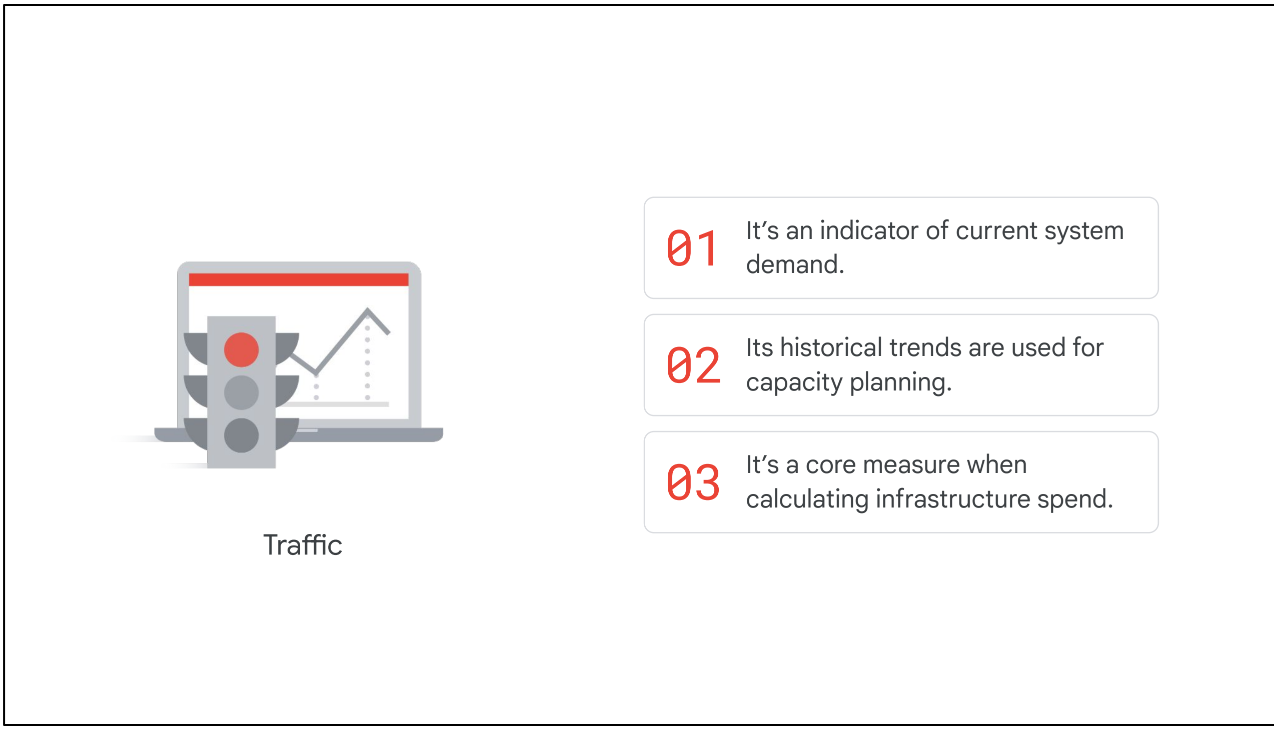


There are “Four Golden Signals” that measure a system’s performance and reliability.

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



- 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



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

Traffic is important because:

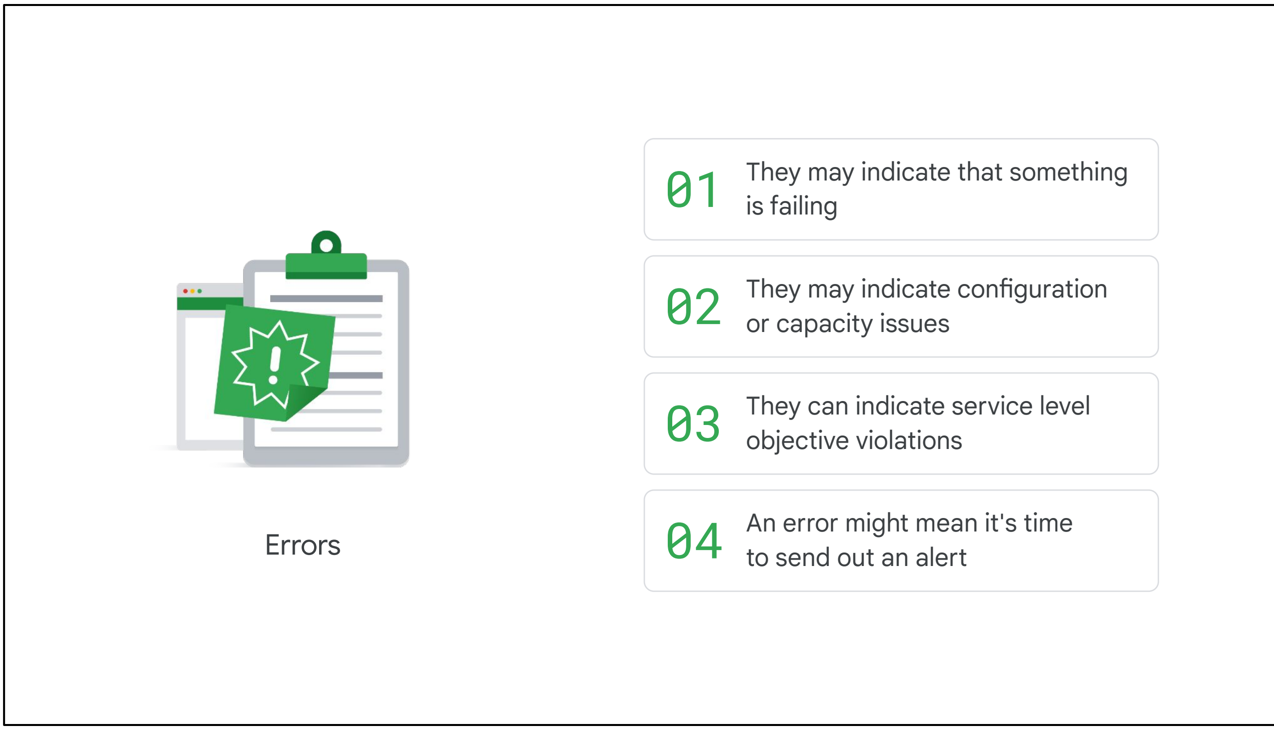
- It’s an indicator of current system demand.
- Its historical trends are used for capacity planning.
- And it’s a core measure when calculating infrastructure spend.

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

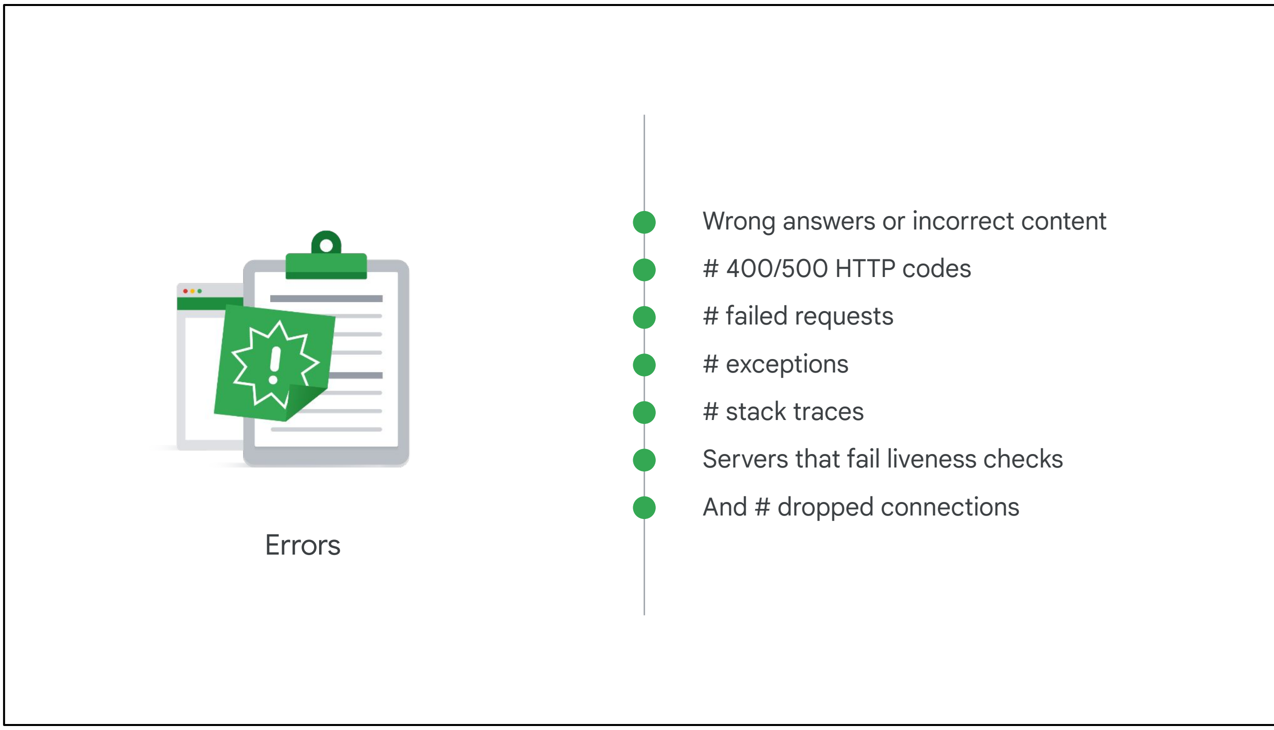
- number of HTTP requests per second
- number of requests for static vs. dynamic content
- Network I/O
- number of concurrent sessions
- number of transactions per second
- number of retrievals per second
- number of active requests
- number of write ops
- number of read ops
- And number of active connections



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.



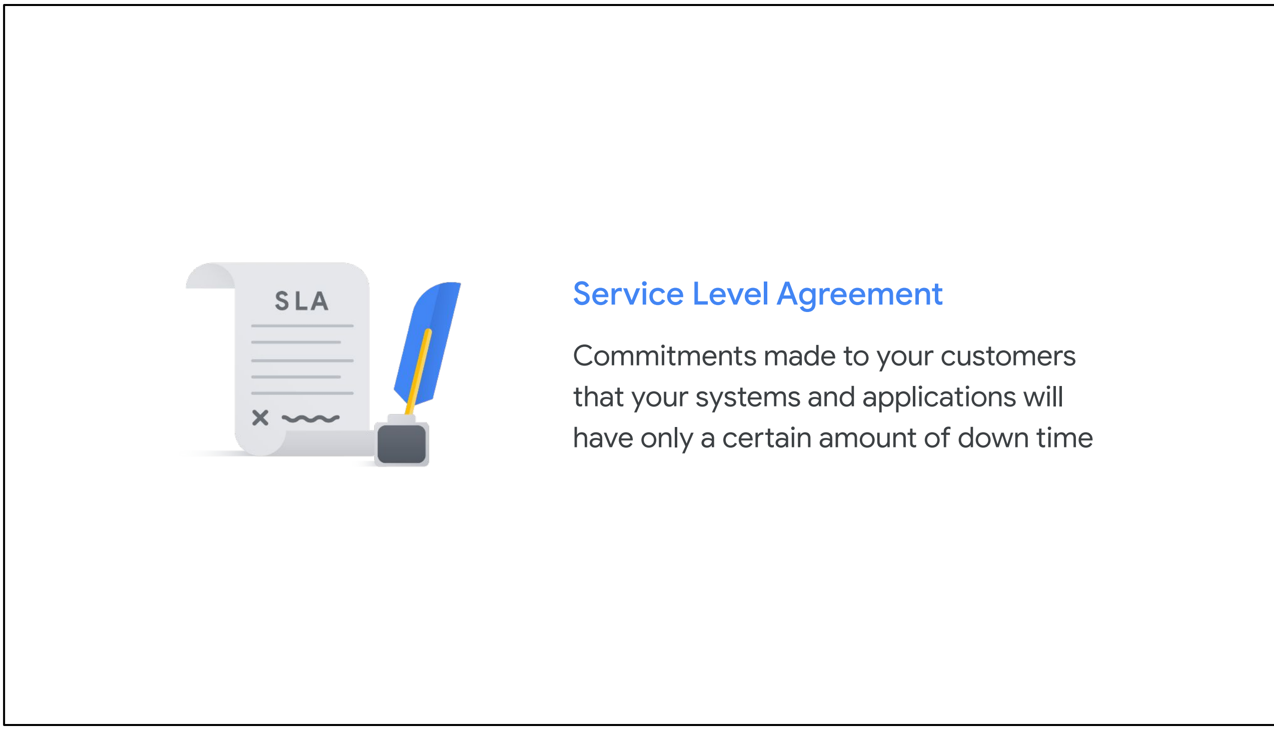
Sample error metrics include:

- 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

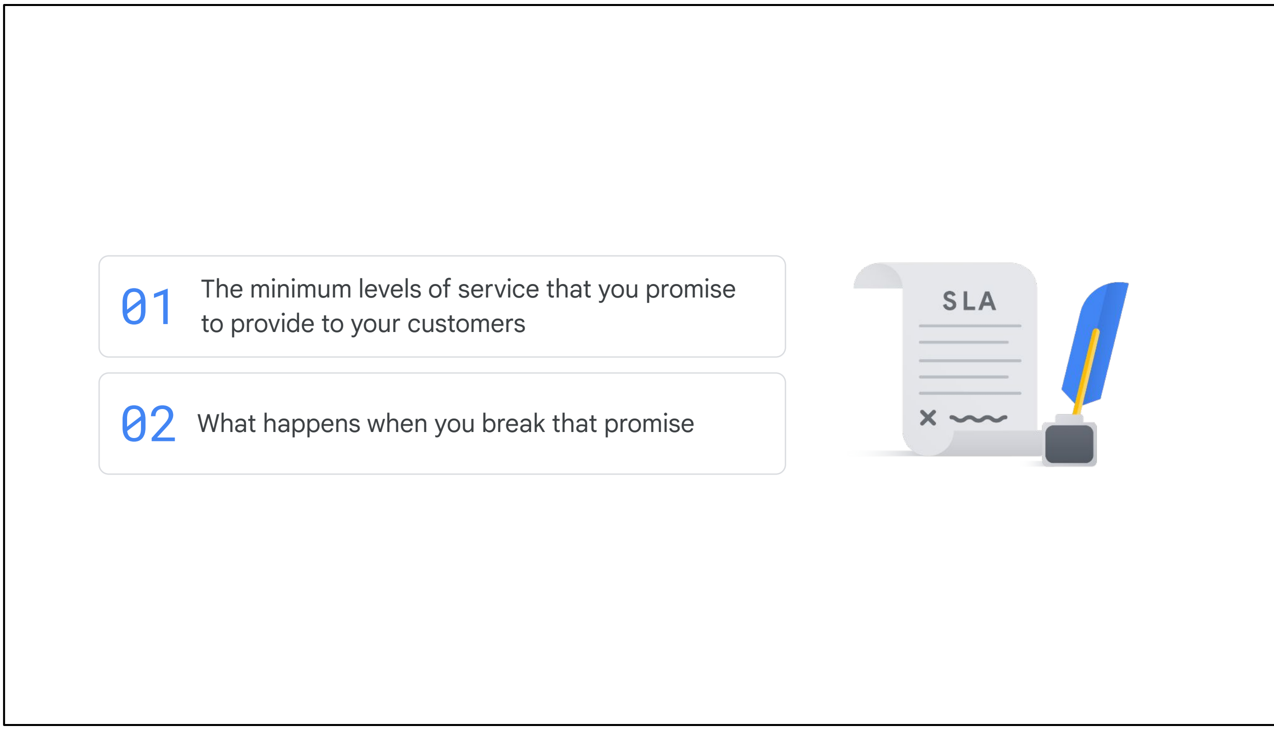
You can't measure everything, so when possible, you should choose SLOs that are **S.M.A.R.T**.

- 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.



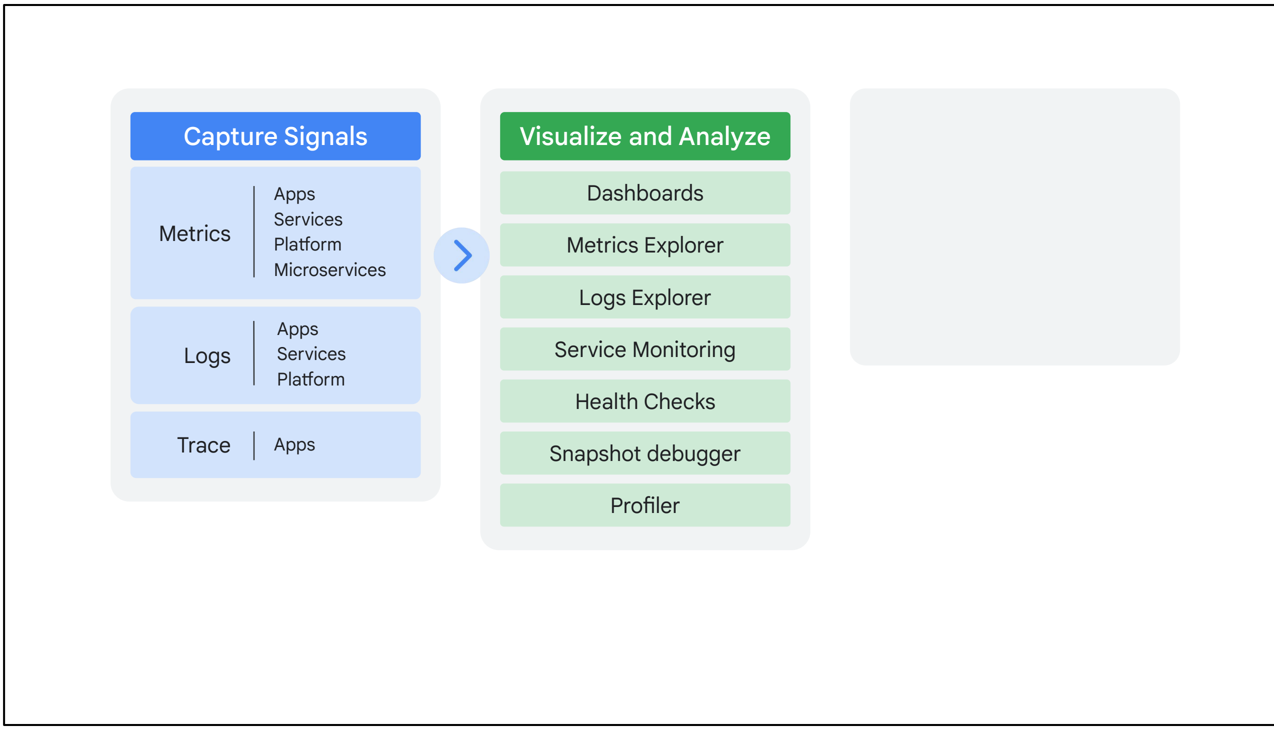
And then there are **Service Level Agreements, or SLAs**, which are commitments made to your customers that your systems and applications will have only a certain amount of "down time."



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

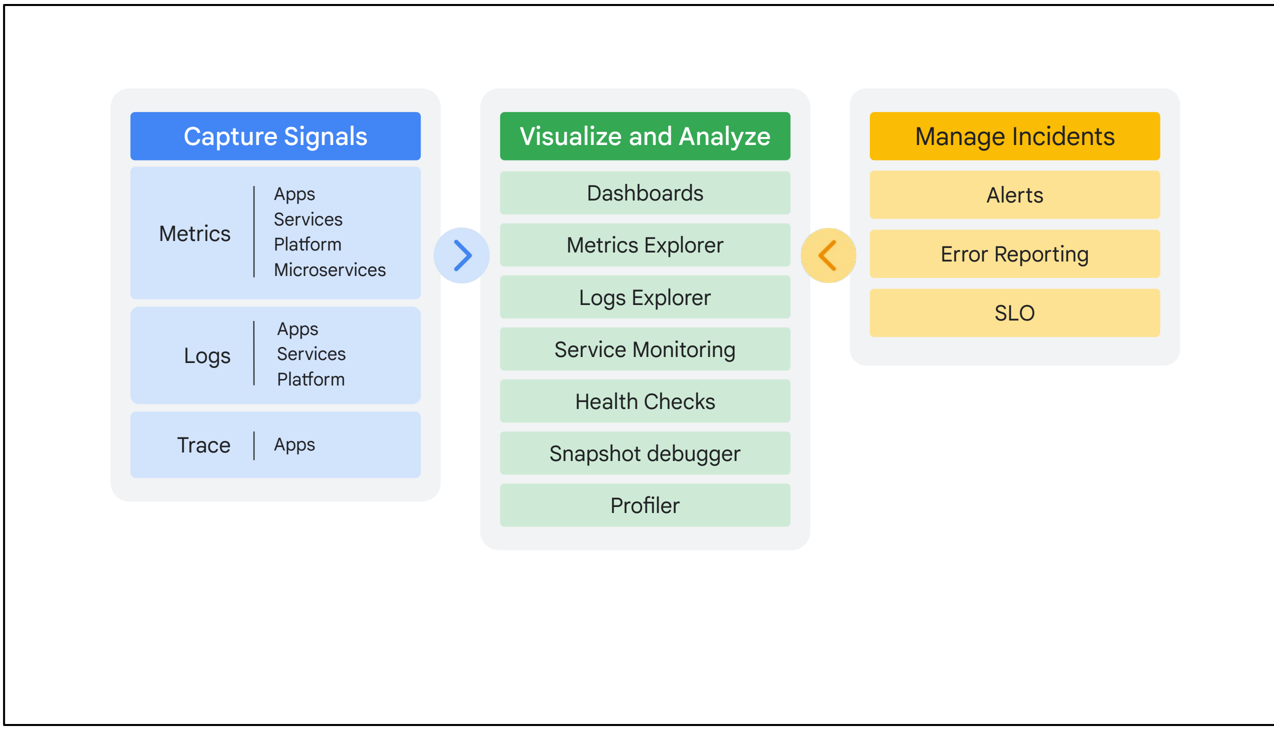
To improve service reliability, all parts of the business must agree that these are an **accurate measure of user experience** and must agree to use them as a **primary driver for decision making**

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.



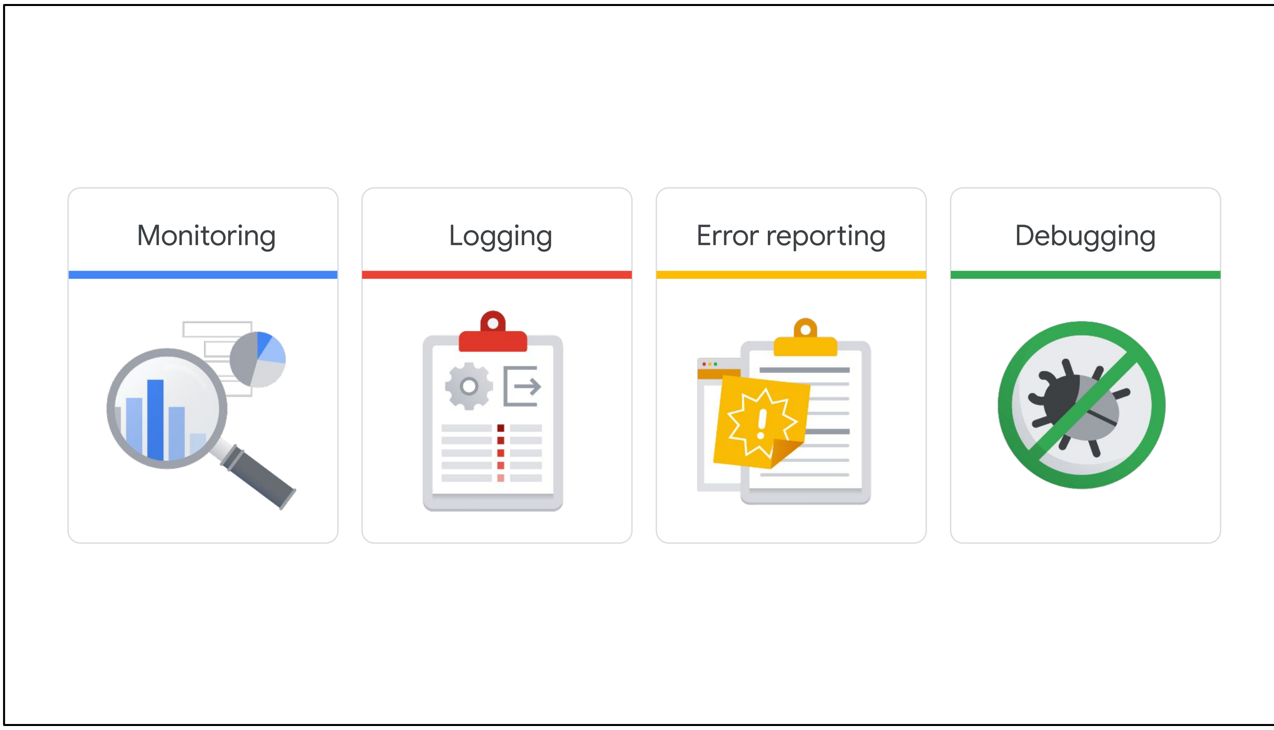
From those products:

- 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.

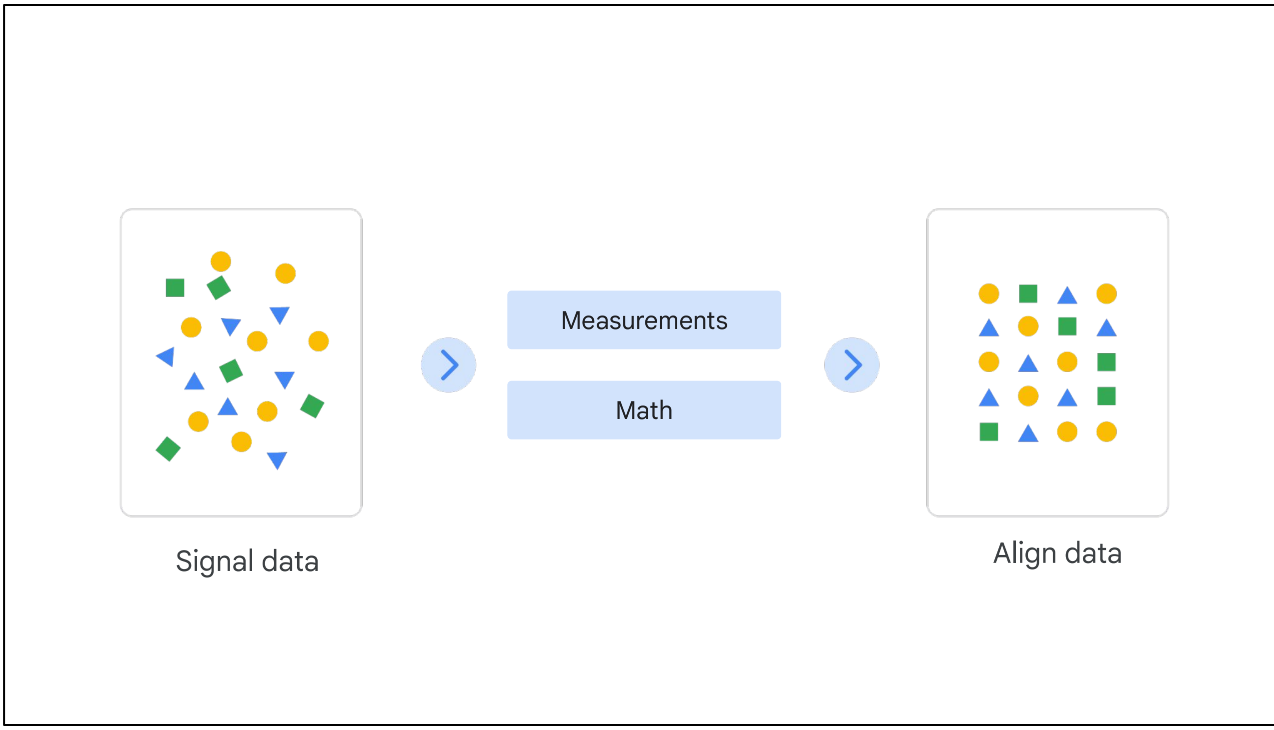


When incidents occur:

- 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.

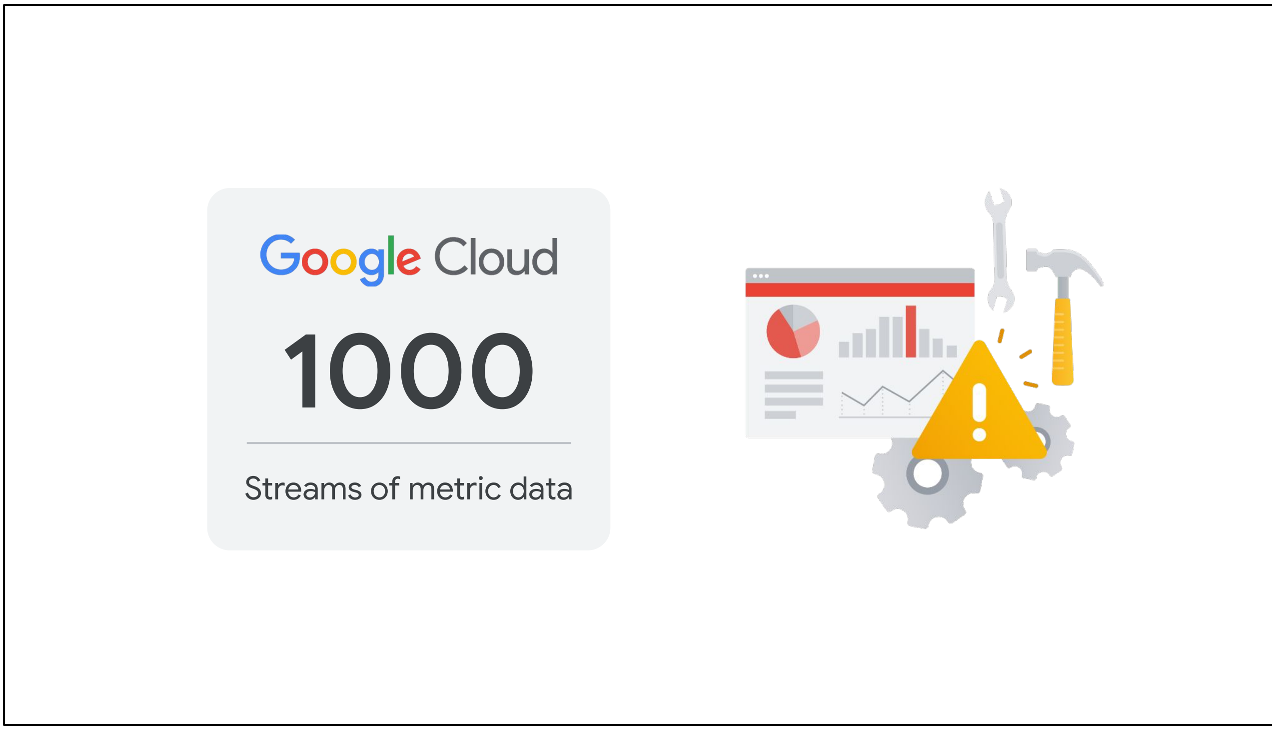


Over the next few videos we'll explore the products and tools offered by Google Cloud that are most applicable for those in operations roles that work with **monitoring**, **logging**, **error reporting**, and **debugging**.

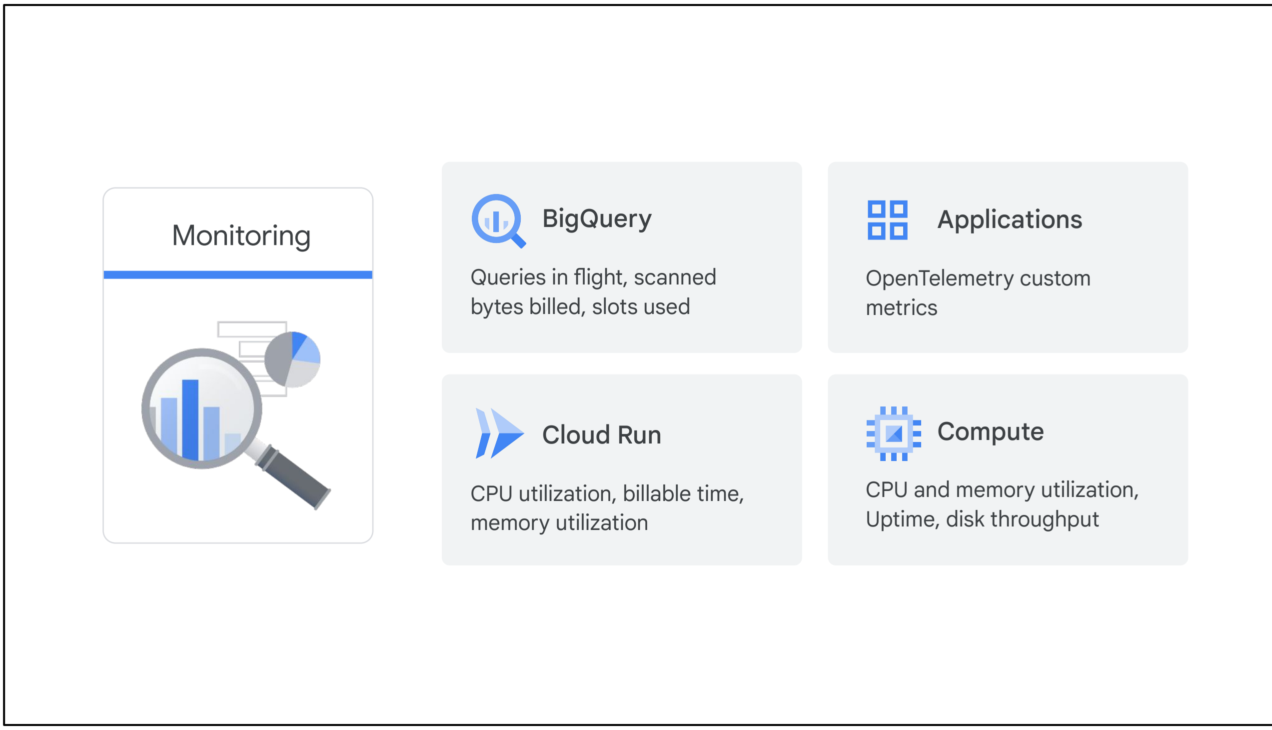


As we stated previously, monitoring starts with signal data. Metrics take measurements and use math to align those measurements over time.

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



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

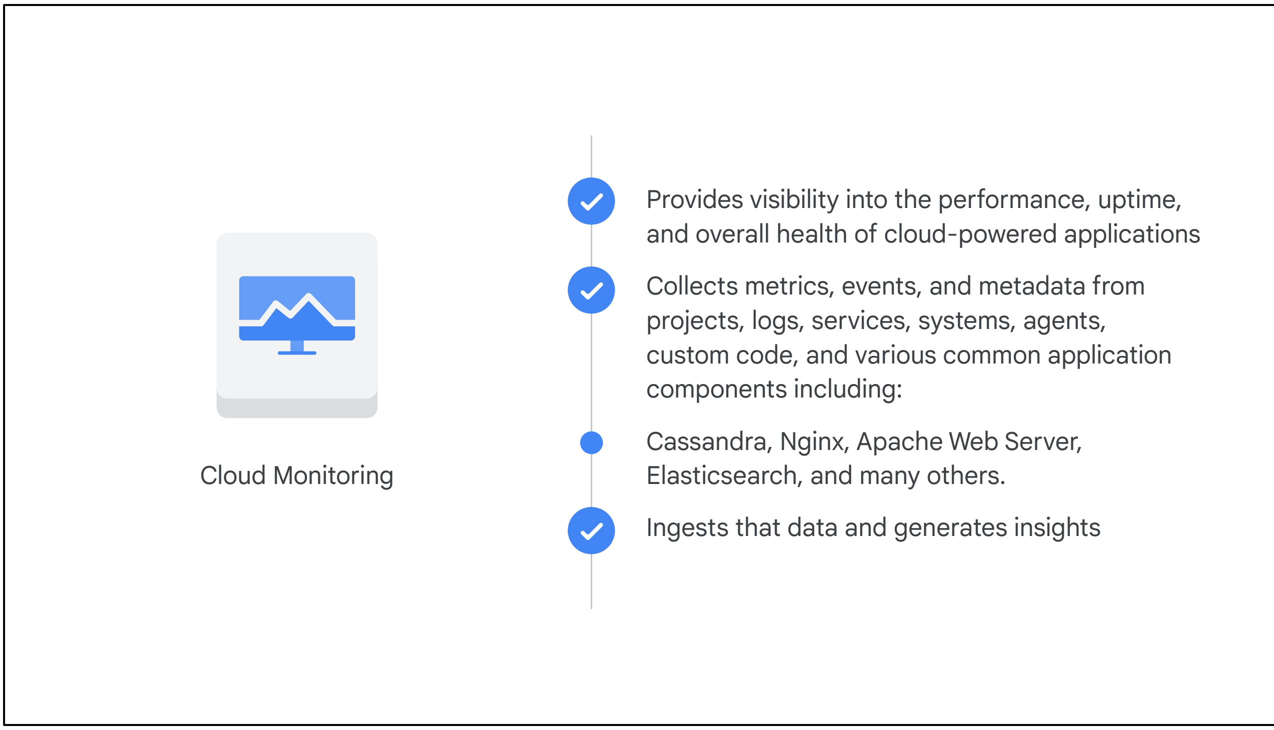


When data scientists run massive, scalable queries in **BigQuery**, it's important for them to know how many queries are currently in flight, how many bytes have been scanned and added to the bill, and data slot usage patterns.

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.

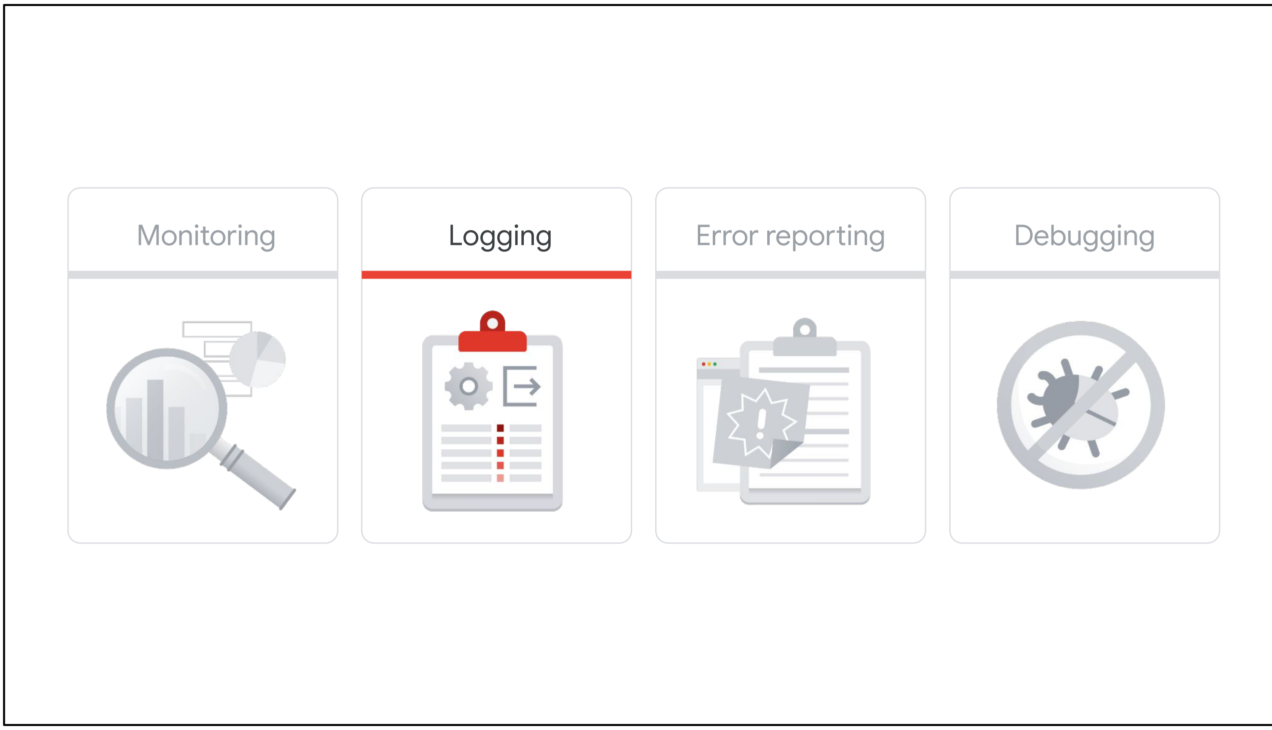
And workloads on **Compute Engine** will benefit from CPU and memory utilization data, along with uptime, disk throughput, and many other metrics.



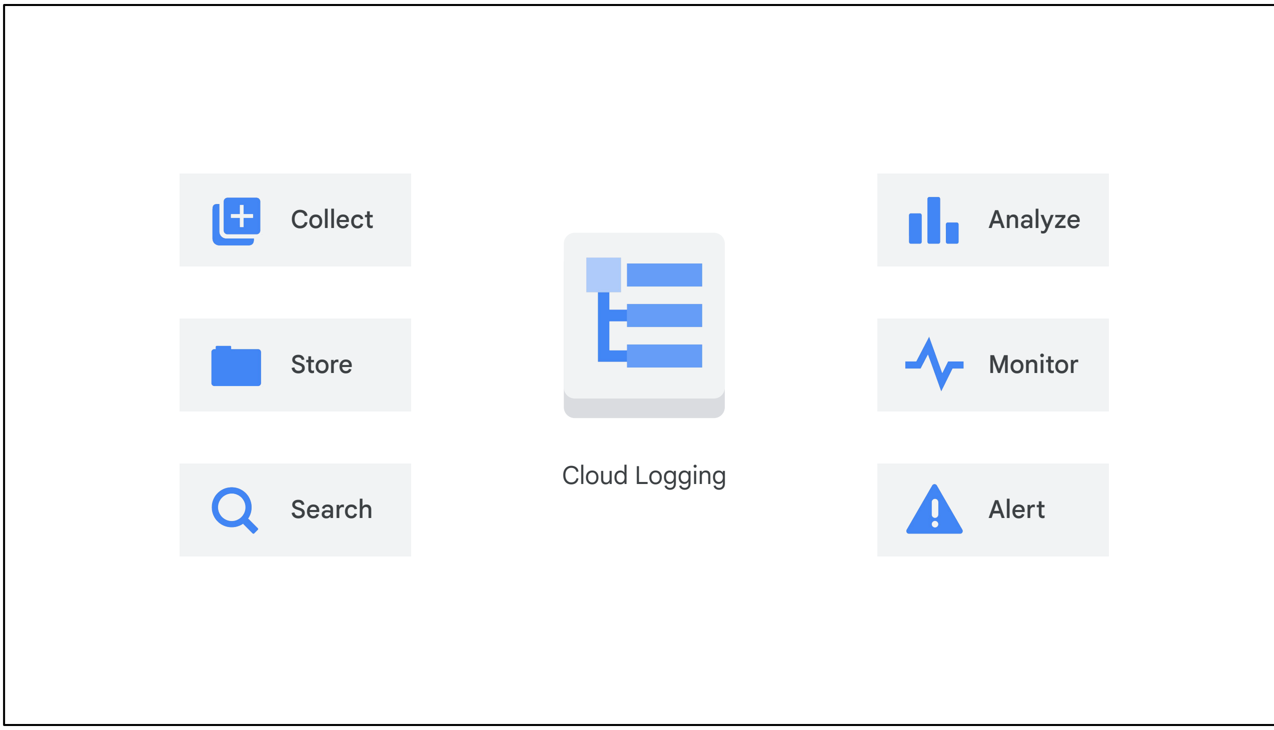
Cloud Monitoring provides visibility into the performance, uptime, and overall health of cloud-powered applications.

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

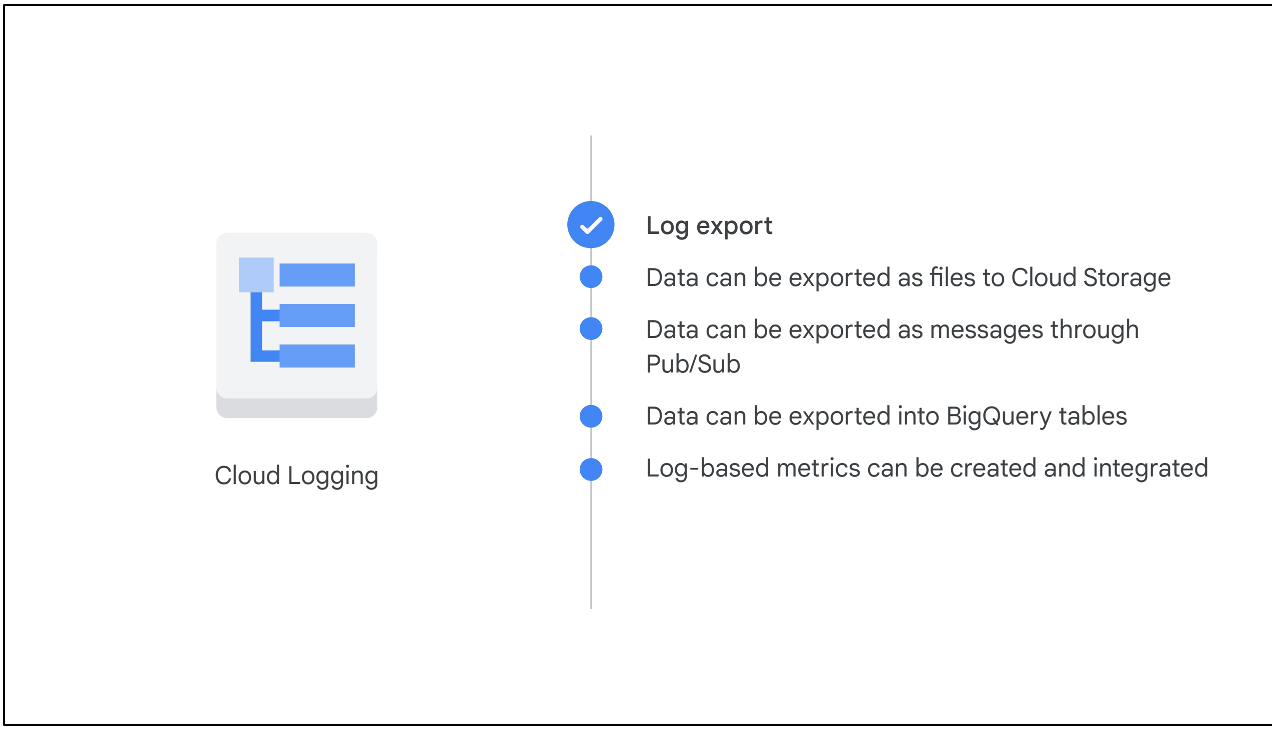
Most log analysis starts with Google Cloud's integrated Logs Explorer.

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.

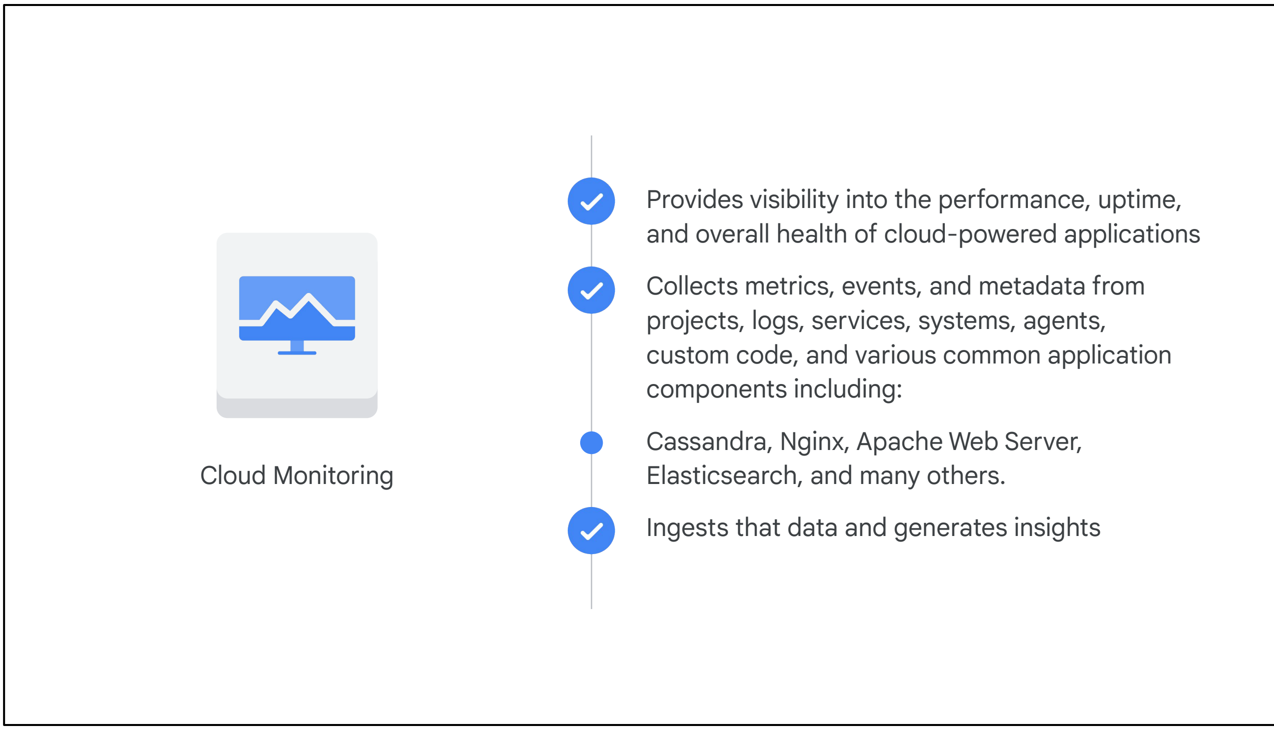
BigQuery allows analysts to examine logging data through SQL queries.

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.

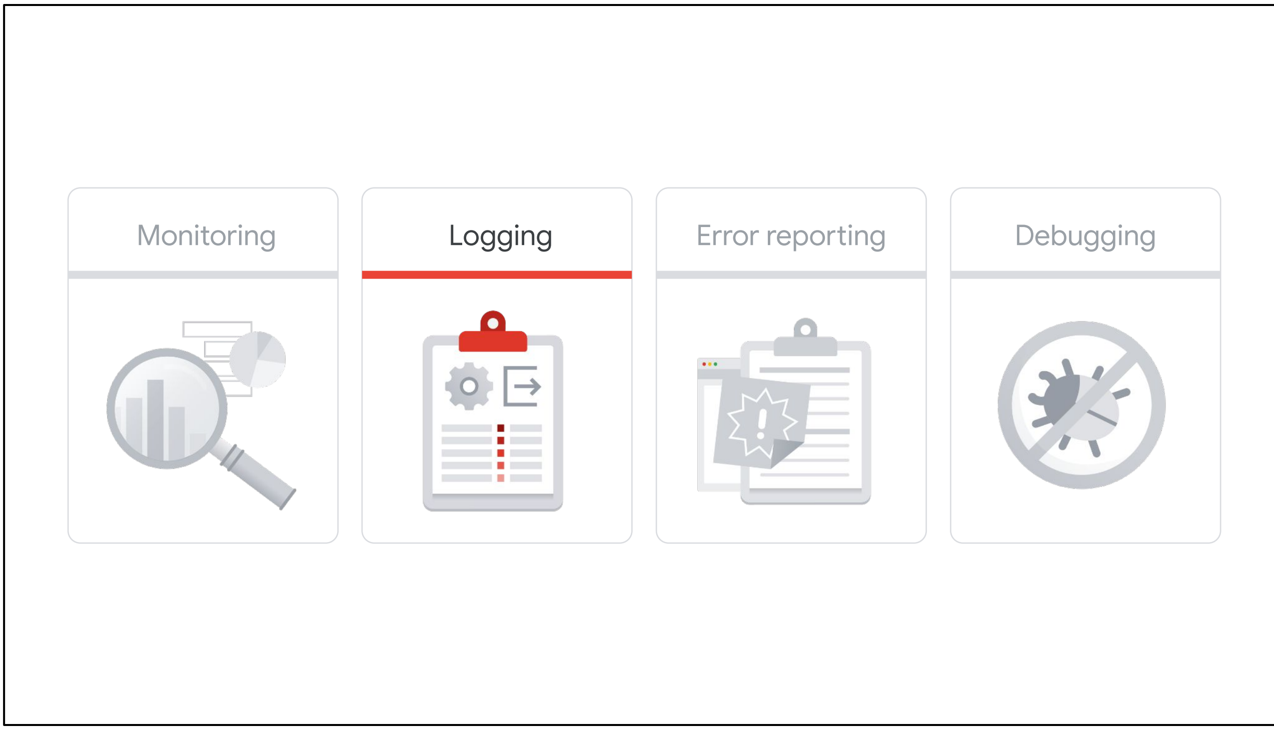
Log-based metrics can be created and integrated into Cloud Monitoring dashboards, alerts, and service SLOs.



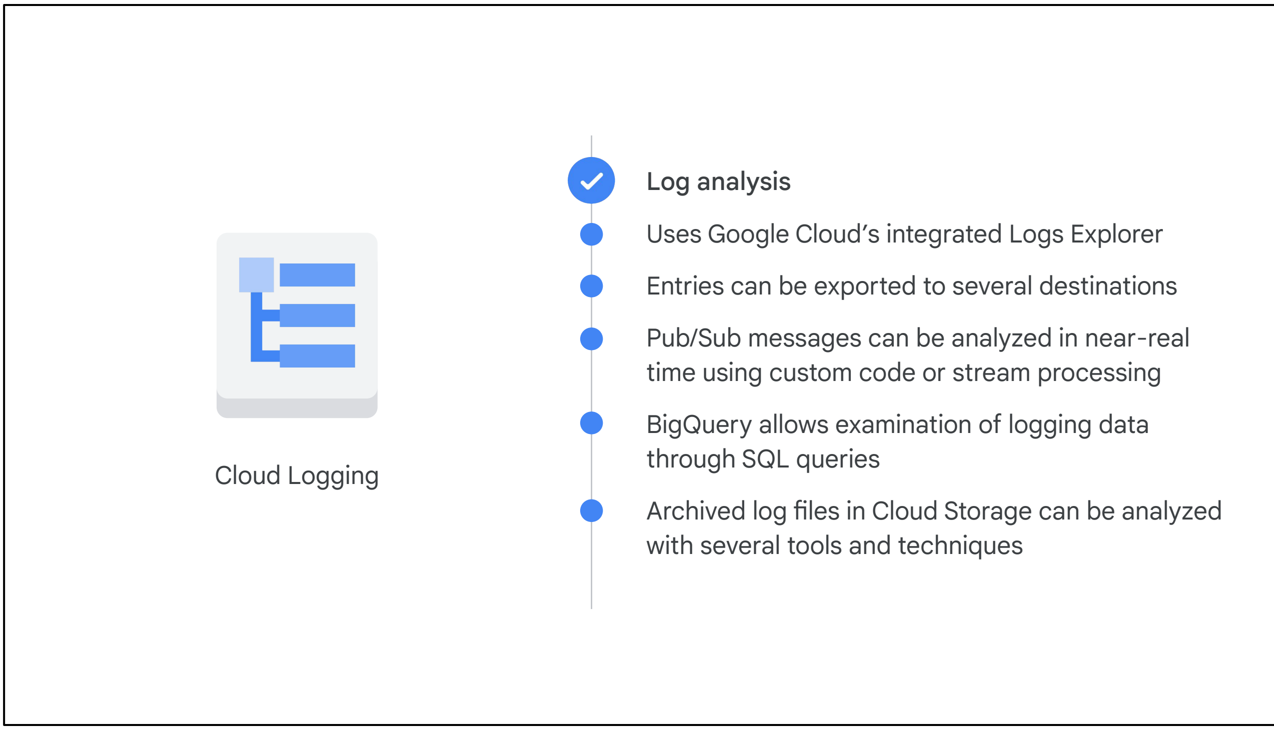
Cloud Monitoring provides visibility into the performance, uptime, and overall health of cloud-powered applications.

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.



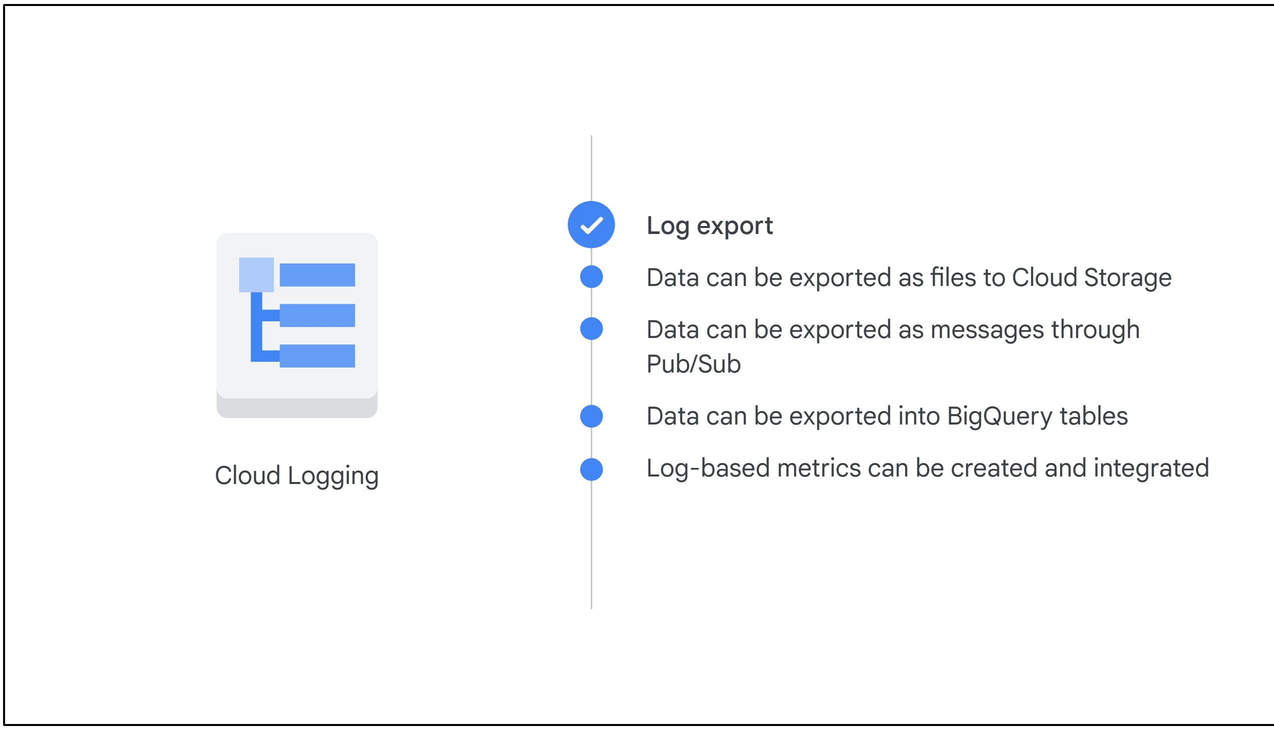
Most log analysis starts with Google Cloud's integrated Logs Explorer.

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.

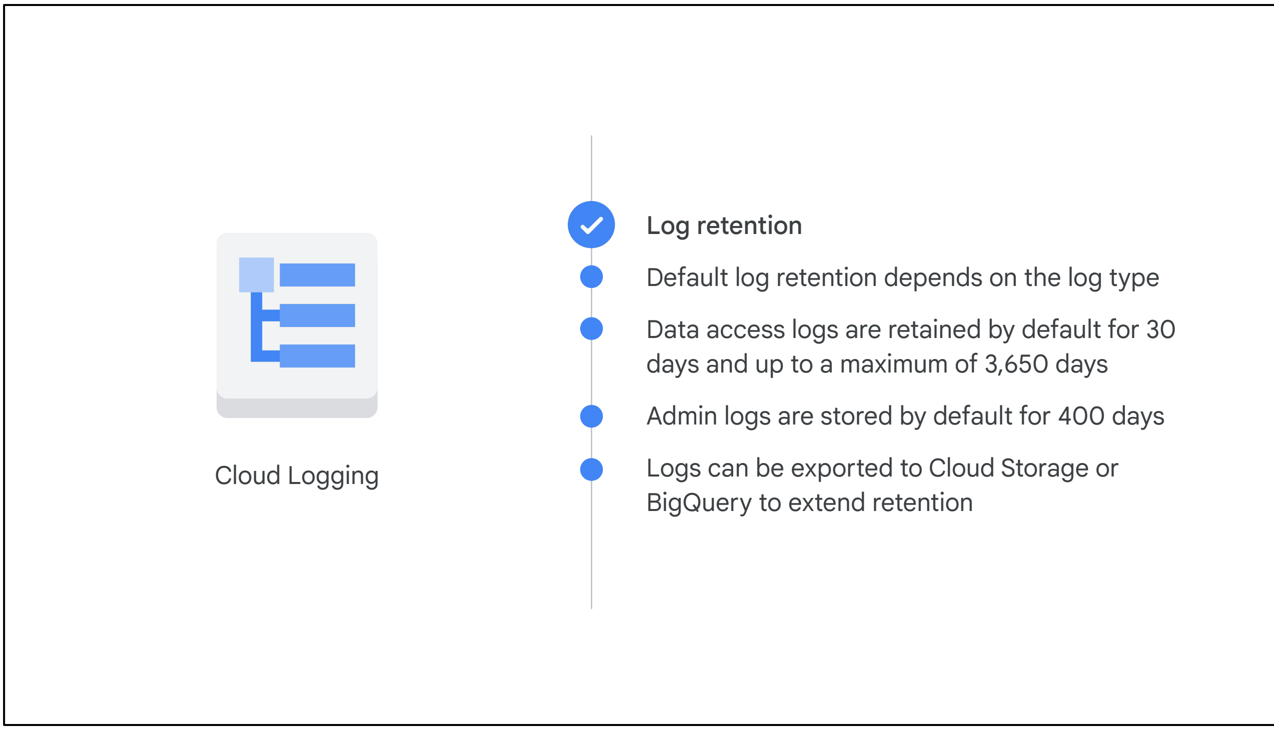
BigQuery allows analysts to examine logging data through SQL queries.

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.

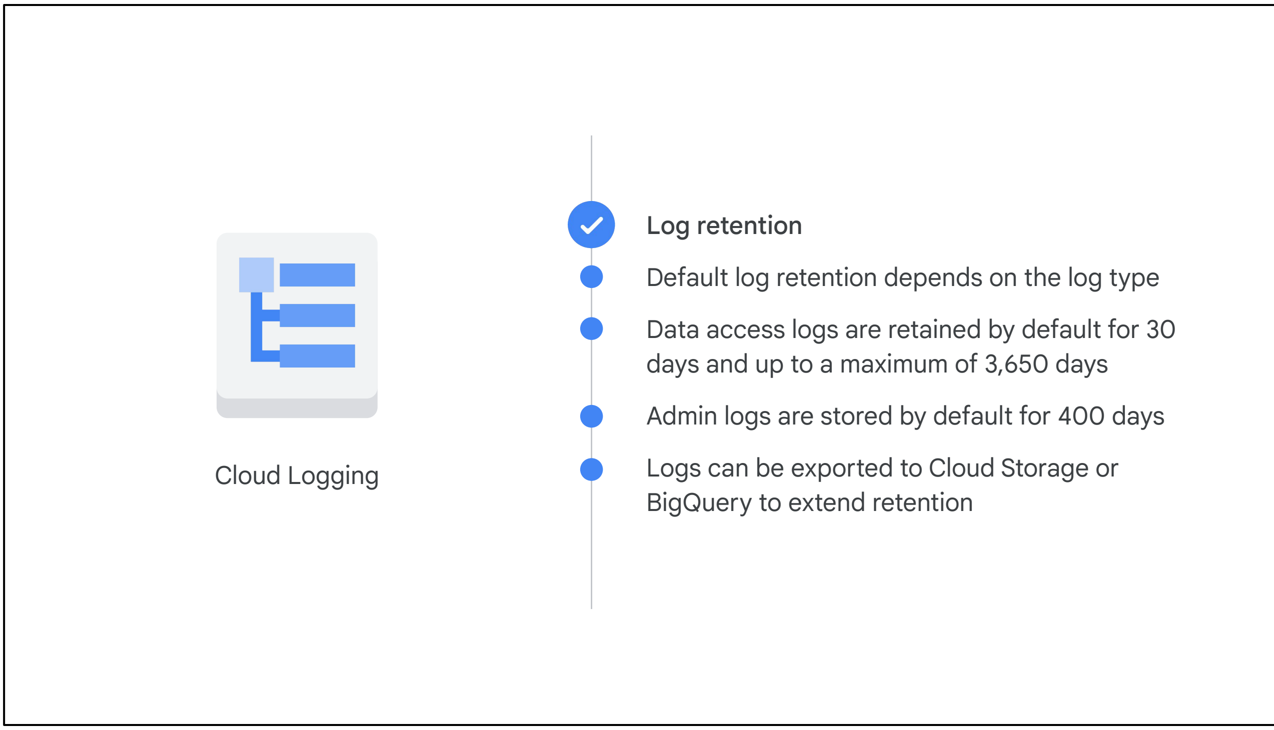


Default log retention in Cloud Logging depends on the log type.

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.



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

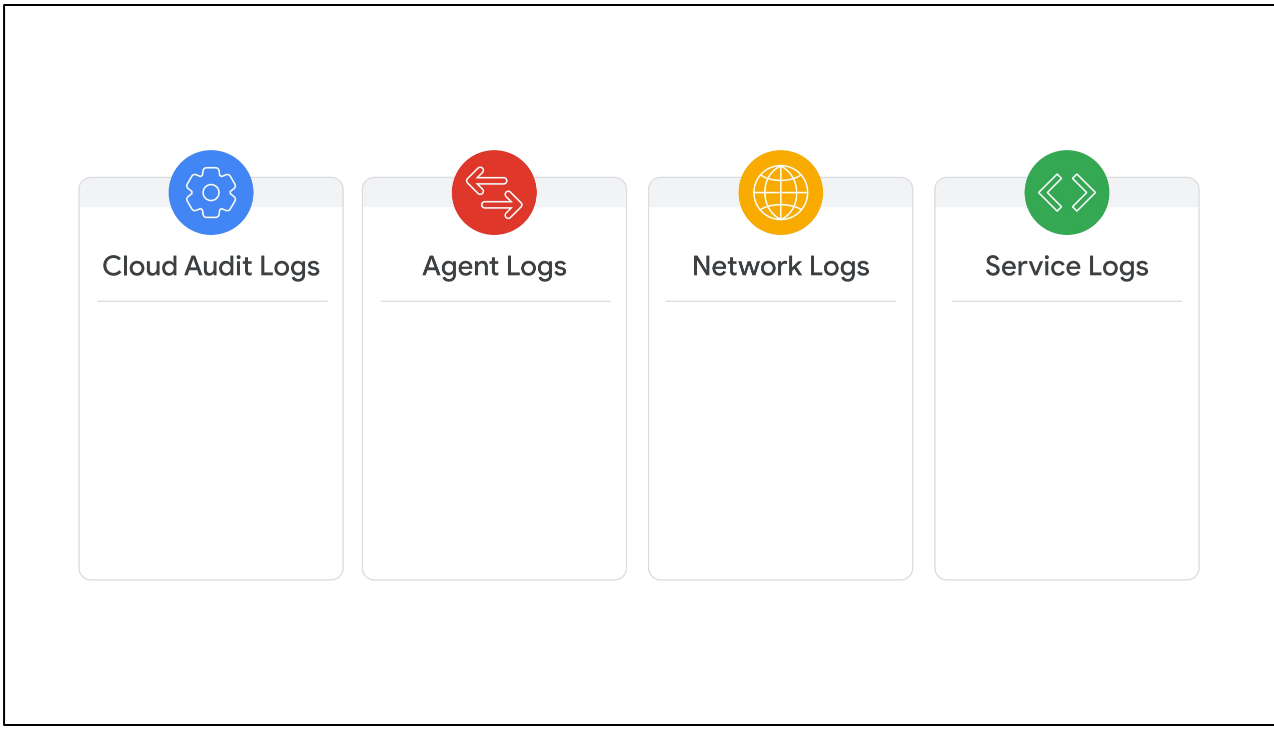
Data access logs are retained by default for 30 days, but this is configurable up to a maximum of 3,650 days.

Admin logs are stored by default for 400 days.

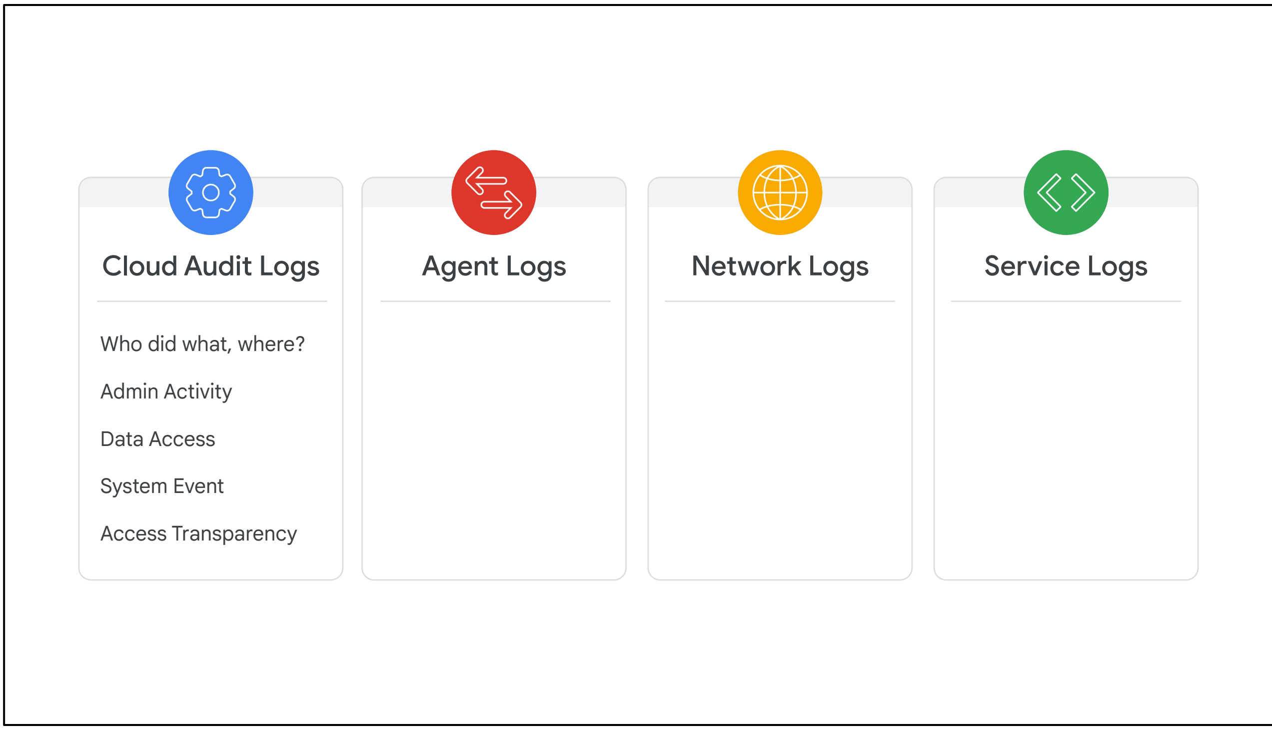
Logs can be exported to Cloud Storage or BigQuery to extend retention.

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

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.

Error Reporting

- Counts, analyzes, and aggregates the crashes in your running cloud services.
- 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.
- Create alerts to receive notifications on new errors.

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

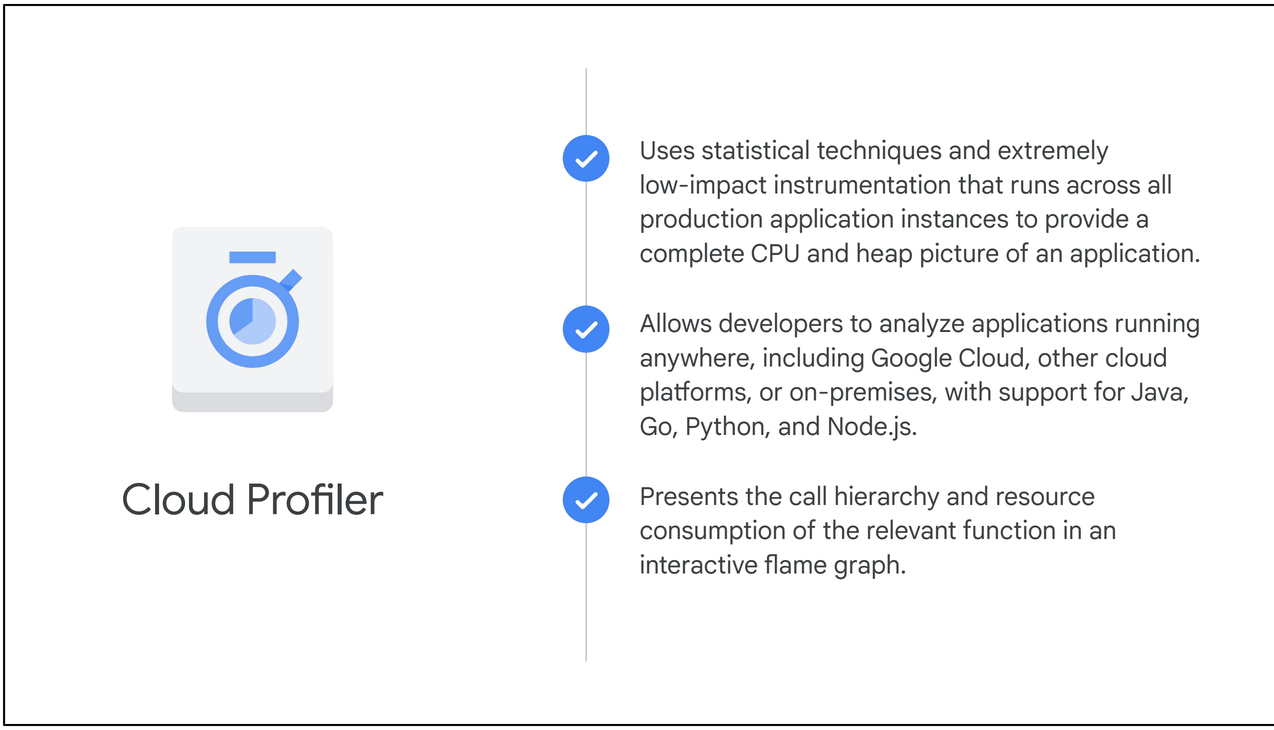
- Collects latency data from distributed applications and displays it in the Google Cloud console.
- Captures traces from applications deployed on App Engine, Compute Engine VMs, and Google Kubernetes Engine containers.
- Performance insights are provided in near-real time.
- Automatically analyzes all of your application's traces to generate in-depth latency reports to surface performance degradations.
- Continuously gathers and analyzes trace data to automatically identify recent changes to application performance.

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.



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.