



Advanced Logging and Analysis



In this module, we will examine some of Google Cloud's advanced logging and analysis capabilities.

Agenda

Strategic Logging

Labeling

Working with the Log Viewer

Using Logs-Based Metrics

Exporting and Analyzing Logs

Error Reporting



Specifically, in this module you will learn to:

- Identify and choose among resource tagging approaches because tags can make locating and tracking resources easier.
- Define log sinks (inclusion filters) to include specific log entries, and exclusion filters to exclude others.
- Create monitoring metrics based on log entries. For example, "this NGINX log entry has appeared x times in the last minute," so we can now tell how many requests our web site took.
- Link application errors to Logging and other operation tools using Error Reporting
- and Export logs to BigQuery for long term storage and SQL based analysis.

Agenda

Strategic Logging

Labeling

Working with the Log Viewer

Using Logs-Based Metrics

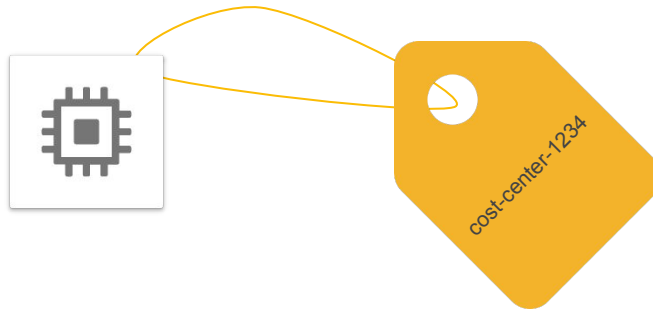
Exporting and Analyzing Logs

Error Reporting



Let's start with labeling.

Labels Help Identify Resources



Labels are key-value pairs that help organize and to identify Google Cloud Resources. They can be used to track linked resources, even if they span multiple projects, and are especially helpful when analyzing resource usage and billing.

Labels

What resources can be labeled?

- Cloud Spanner
- Cloud SQL
- Cloud Storage
- GCE
- GKE
- Cloud Run
- Networking
- Resource Manager
- Logs-based metrics
- BigQuery
- Bigtable
- Dataflow
- Dataproc
- Deployment Manager
- Cloud Functions
- KMS
- Pub/Sub

Considerations for applying labels

- Consistency
 - ✓ Apply labels programmatically if possible
- Simple labels with both technical and business value (ie, 'webserver', 'backups')
 - ✓ Try to stick with no more than 5 labels (up to 64)
- Follow json format of -l <key>:<value>
 - ✓ < 63 characters
 - ✓ Only contain lowercase letters, numbers, underscores, and dashes



On this slide, you can see a partial list of Google resources that support labeling.

Some considerations when applying labels:

- Since labels are arbitrary, key-value pairs, consistency can be challenging. If one user sets a label city:ny and another sets a label as city:nyc, GCP considers these two different labels, and text-based searching could easily find one and miss the other. Consider applying labels programmatically where possible.
- Choose label keys and values that are simple, and which use standard organizational terminology and knowledge.
- Labels follow the JSON format of key:value. Label keys and values must be less than 63 characters long and can only contain lowercase letters, numbers, underscores, and dashes.

Agenda

Strategic Logging

Labeling

Working with the Log Viewer

Using Logs-Based Metrics

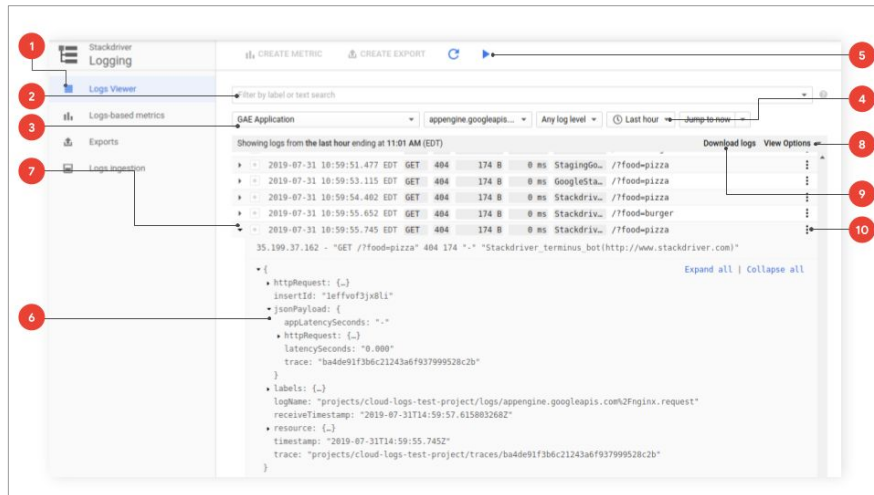
Exporting and Analyzing Logs

Error Reporting



Cloud Logging allows you to store, search, analyze, monitor, and alert on log data and events from Google Cloud. Cloud Logging is a fully managed service that performs at scale and can ingest application and system log data from thousands of VMs. Even better, you can analyze all that log data in real time.

Log Viewer



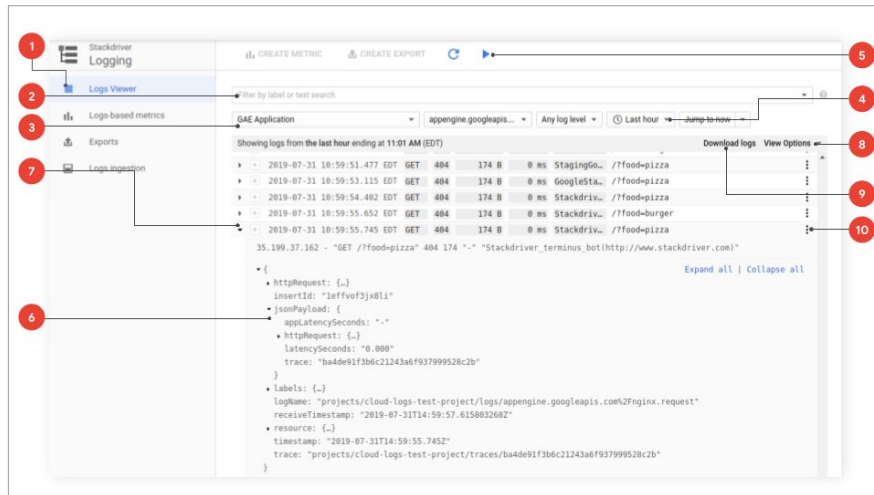
To start using Google Cloud Logging, start with the Log Viewer. There are two querying interfaces in the Logs Viewer:

- The basic query interface that you see on this slide lets you select logs from menus and has a simple search capability.
- The advanced query interface we will examine in a few slides. It lets you view log entries from multiple logs and has a more sophisticated search capability.

In the screenshot on this slide, you see the Logs Viewer's layout with the basic query interface active. In this example we see a list of log entries from a App Engine application. The UI allows log entries to display in different time zones, for globally distributed teams.

Newly added features include the ability to view data as Histograms (to see event distributions over time), and to view aggregated log data (resource.type , resource.labels , logName, and severity) using the new Logs Field Explorer.

Log Viewer



Let's take a tour.

The basic query interface has the following major components—indicated by red numbers in the screenshot above—some of which are shared with the advanced query interface:

- 1) The *window tabs* let you stay on **Logs** (the Logs Viewer page), or choose between other Logging features: **Metrics** (see [Logs-based metrics](#)), **Exports** (see [Exporting with the Logs Viewer](#)), and **Logs ingestion** (see [Logs exclusions](#)).
- 2) The *search-query box* in the basic query interface lets you query log entries by label or text search. The [basic query](#) is shown, and the drop-down arrow (▼) at the far right lets you switch to the [advanced query interface](#) or get a link to your query. Log queries are labelled as "filters" in the user interface, since they let you select a particular set of log entries.
- 3) The *basic selector menu* lets you choose resources, logs, and severity levels to display:
 - **Resources:** The resources available in your current project.
 - **Logs:** The log types available for the current resources in your project.

4) The *time-range selector* drop-down menus let you query for specific dates and times in the logs.

5) The *streaming selector*, at the top of the page, controls whether new log entries are displayed as they arrive.

6) The *log-entry table* contains the log entries available according to your current queries and [custom fields](#).

7) The *expander arrow* (►) in front of each log entry lets you look at the full contents of the entry. For more information, see [Expand log entries](#).

8) The **View Options** menu, at the far right, has additional display options.

9) The **Download logs** menu, at the far right, lets you download a set of log entries. For details, see [download log entries](#).

10) The **More** (⋮) option, displayed with each log entry, lets you place a pin on the log entry, show the log entry in its resource context, and copy a URL for the log entry to the clipboard.

Basic Filters



Good for basic field searches

- Entries that contain **foo** in any field
 - **text:foo** or **foo**
- Entries that contain **foo or bar**
 - **text:foo text:bar** or **foo bar**
- Find entries that contain **foo and bar**
 - **text:"foo bar"** or **"foo bar"**

The basic filters are good for fundamental field searches. Such as:

To search for entries that contain foo in any field, you can filter for **text:foo** or simply **foo**

To locate entries that contain foo or bar, then filter for **text:foo text:bar** or **foo bar**

To find entries that contain foo and bar, use quotes. Filter for **text:"foo bar"** or **"foo bar"**

Basic Filters



Unsupported filters

- Wildcard characters
 - text:foo* or foo*
- Searching the timestamp field
 - text:2017-02-05 or 2017-02-05
- Range operators
 - text:200..299 or 200..299
- Boolean operators
 - text:foo text:NOT text:bar or "foo NOT bar"



Basic features won't support:

Wildcard characters like in **foo***

Searching the timestamp field like **2020-02-05**

The use of range operators as in **200..299**

or using boolean operators in strings like in **"foo NOT bar"**

Log Entries

GCE VM Instance

All logs

Any log level

Last 24 hours

Jump to now

Showing logs from the beginning of time to 2:44 PM (EDT)

▶

2018-04-10 14:28:01.000 EDT

l66kxllfs3qete

Apr 10 18:28:01 instance-2 systemd[1]: Starting Cleanup of Temporary Directories.

⋮

▶

2018-04-10 15:01:46.000 EDT

fthh83g1bbjtvv

Apr 10 19:01:46 add-structured-log-resource systemd[1]: Starting Cleanup of Temporary Directories...

⋮

▶

2018-04-10 15:01:47.000 EDT

fthh83g1bbjtvv

Apr 10 19:01:47 add-structured-log-resource systemd-tmpfiles[12132]: [/usr/lib/tmpfiles.d/var.conf_

⋮

▶

2018-04-10 15:01:47.000 EDT

fthh83g1bbjtvv

Apr 10 19:01:47 add-structured-log-resource systemd[1]: Started Cleanup of Temporary Directories.

⋮

▶

2018-04-10 15:17:01.000 EDT

1lh3jxeglazkzr

Apr 10 19:17:01 add-unstructured-log-resource CRON[16498]: (root) CMD (cd / && run-parts --repor_

⋮

View Options

The log-entry table displays a summary line for each log entry by default.

Log Entries

GCE VM Instance	All logs	Any log level	Last 24 hours	Jump to now
Showing logs from the beginning of time to 2:44 PM (EDT)				
2018-04-10 14:28:01.000 EDT	166kxllfs3qete	Apr 10 18:28:01 instance-2 systemd[1]: Starting Cleanup of Temporary Directories.	View Options	
2018-04-10 15:01:46.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:46 add-structured-log-resource systemd[1]: Starting Cleanup of Temporary Directories...		
2018-04-10 15:01:47.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:47 add-structured-log-resource systemd-tmpfiles[12132]: [/usr/lib/tmpfiles.d/var.conf_		
2018-04-10 15:01:47.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:47 add-structured-log-resource systemd[1]: Starting Cleanup of Temporary Directories.		
2018-04-10 15:17:01.000 EDT	1lh3jxeglazkzr	Apr 10 19:17:01 add-unstructured-log-resource CRON[16498]: (root) CMD (cd / && run-parts --repor...		

The log entry summary line might contain highlighted fields. For example, custom fields are highlighted.

Log Entries

GCE VM Instance	All logs	Any log level	Last 24 hours	Jump to now
Showing logs from the beginning of time to 2:44 PM (EDT)				
2018-04-10 14:28:01.000 EDT	166kxllfs3qete	Apr 10 18:28:01 instance-2 systemd[1]: Starting Cleanup of Temporary Directories.		
2018-04-10 15:01:46.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:46 add-structured-log-resource systemd[1]: Starting Cleanup of Temporary Directories.		
2018-04-10 15:01:47.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:47 add-structured-log-resource systemd-tmpfiles[12132]: [/usr/lib/tmpfiles.d/var.conf_		
2018-04-10 15:01:47.000 EDT	fthh83g1bbjtvv	Apr 10 19:01:47 add-structured-log-resource systemd[1]: Starting Cleanup of Temporary Directories.		
2018-04-10 15:17:01.000 EDT	11h3jxegl1apzkzr	Apr 10 19:17:01 add-unstructured-log-resource CRON[16498]: (root) CMD (cd / && run-parts --repor_		



The fields included in the summary line are selected as subsets of the log entry fields. Certain fields are shown by default if they meet one or more of these criteria:

- The log entry has a well-known type, such as an App Engine request log.
- The log entry contains the [httpRequest](#) field.
- The log entry has a payload containing a field named message.

Log Entry Details

```
▼ {
  textPayload: "Dec  6 20:28:53 your-gce-instance collectd[4778]: match_throttle_metadata_keys: 269 history entries, 2
33 distinct keys, 21483 bytes server memory."
  insertId: "1dearxwf7j44nl"
  ▼ resource: {
    type: "gce_instance"
    ▼ labels: {
      project_id: "my-gcp-project-id"
      instance_id: "1428064241541024269"
      zone: "us-central1-a"
    }
  }
  timestamp: "2016-12-06T20:28:53Z"
  ▼ labels: {...}
  logName: "projects/my-gcp-project-id/logs/syslog"
}
```



To see the full details for one log entry, click the expander arrow (►) at the front of the summary line. To see the full details in a structured view for all the log entries available with your current query, click the **View Options** menu at the far right and then select **Expand All**

When you expand a summary line for a log entry, the Logs Viewer displays a structured (JSON) representation.

Primary Log Fields

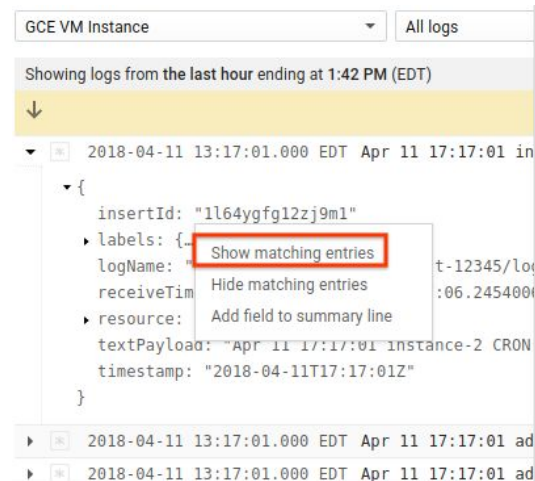
logName	Resource name of the log to which this log entry belongs (ex: projects/[PROJECT_ID]/logs/[LOG_ID])
insertId	Unique identifier
severity	Entry severity, defaults to LogSeverity.DEFAULT
timestamp/receiveTimestamp	The time the event described by the log entry occurred/was received by Logging
resource.type	The name of a resource type. Example: gce_instance
resource.labels.KEY	The value associated with a resource label key
httpRequest.FIELD	The value of a field in an HttpRequest object (method, url, size, status, etc.)
labels.KEY	The value associated with a label key
operation.FIELD	The value of a field in a LogEntryOperation object
protoPayload.FIELD	Log entry payload represented as a protocol buffer
jsonPayload.FIELD	The value of a field within a JSON object
textPayload	The log entry payload, represented as a Unicode string (UTF-8)



Log entries are based on Google's [LogEntry](#) datatype. They contain data like the logName, severity, resource.type, and various payload fields.

Note that jsonPayload and textPayload are mutually exclusive - you can only use one or the other of these fields, but not both.

Locate (or Hide) Similar Entries



You can click the value of a specific field in the expanded log entry view and then either show or hide all log entries with the same value. Doing so will also switch the Logs Viewer to the advanced mode.

Use Advanced Filters in the Logs Viewer

Filter by label or text search

GAE Application, Default Service

stdout, stderr, request...

Any log level


Convert to advanced filter
Get link to filter

Jump to now

1 resource.type="gae_app"
2 resource.labels.module_id="default"
3 logName=("projects/devproject-ashok/logs/appengine.googleapis.com%2Fstdout" OR "projects/devproject-ashok/logs/appengine.googleapis.com%2Fstderr"
OR "projects/devproject-ashok/logs/appengine.googleapis.com%2Frequest_log")
4 protoPayload.latency <= "0.01s"

Submit Filter | Last 7 days | Jump to now

"Control + Space" for autocomplete suggestions

 Google Cloud

The Logs Viewer advanced mode gives you the ability to create more advanced queries. The syntax of the queries is similar to the basic view, but not exactly the same.

Advanced Filter Comparison Operators

= Equals	<code>resource.type="gce_instance"</code>
!= Does not equal	<code>resource.labels.instance_id!="1234567890"</code>
<= Less than equal	<code>timestamp <= "2018-08-13T20:00:00Z"</code>
>= More than equal	<code>timestamp >= "2018-08-13T20:00:00Z"</code>
> More than	<code>timestamp > "2018-08-13T20:00:00Z"</code>
< Less than	<code>timestamp < "2018-08-13T20:00:00Z"</code>
: Has	<code>textPayload:"GET /check"</code>



The next several slides are includes for reference. Advanced queries support multiple comparison operators as seen here.

Advanced Filter Boolean Expressions

- AND `textPayload:"foo" AND textPayload:"bar"`
- NOT `textPayload:"foo" AND NOT textPayload:"bar"`
- OR `textPayload:"foo" OR textPayload:"bar"`



Advanced queries support the AND, OR, and NOT boolean expressions for joining queries.

Advanced Filter Syntax

Syntax:

- **() Grouping** `(foo | bar)`
- **[] Optional** `[foo]`
- **{ } Repeated zero or more times** `{ foo }`



And, advanced queries can support grouping, optional sections, and repeated values.

Finding Entries Quickly

- Search on an indexed field
`logName, resource.type, timestamp, resource.labels`
- Be specific on which logs you are searching
`logName="projects/benkelly-test/logs/apache-access"`
- Limit the time range you are searching
`timestamp >= "2018-08-08T10:00:00Z" AND timestamp <= "2018-08-08T10:10:00Z"`
- Don't do global searches
`"Foo, Bar"` instead use `textPayload="Foo, Bar"`



Some tips on finding log entries quickly:

Search for specific values of indexed fields, like the log entry's name, resource type, and resource labels.

As seen in the example, be specific on which logs you are searching by referring to it or them by name.

Limit the time range you are searching to lessen the log data being queried.

Be wary of global searches; they add a lot of overhead. But, when all else fails, they are a great way of scanning everything.

Agenda

Strategic Logging

- Labeling

- Working with the Log Viewer

Using Logs-Based Metrics

- Exporting and Analyzing Logs

- Error Reporting



Now that we've seen how the Logs Viewer works, let's talk about generating monitoring metrics from logging data.

Key access control roles

- [Logging/Logs Configuration Writer](#)
 - List, create, get, update, and delete logs-based metrics
- [Logging/Logs Viewer](#)
 - View existing metrics
- [Monitoring Viewer](#)
 - Read the timeseries in logs-based metrics
- [Logging Admin, Editor, and Owner](#)
 - Broad-level roles that can create logs-based metrics



A refresher of the key IAM roles that relate to logging and monitoring.

First, on the logging side:

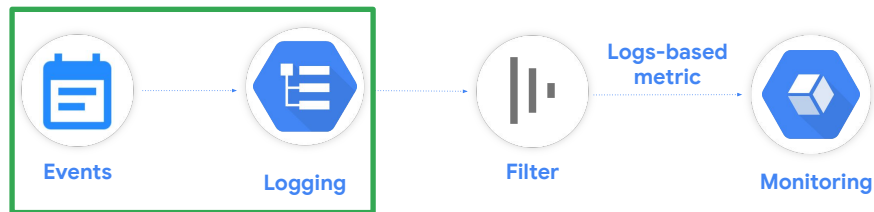
Logs Configuration Writers can List, create, get, update, and delete logs-based metrics

Logs Viewers can view existing metrics

On the monitoring side, **Monitoring Viewers** can read the timeseries in logs-based metrics

And finally, **Logging Admins, Editors, and Owners** are all broad-level roles that can create logs-based metrics

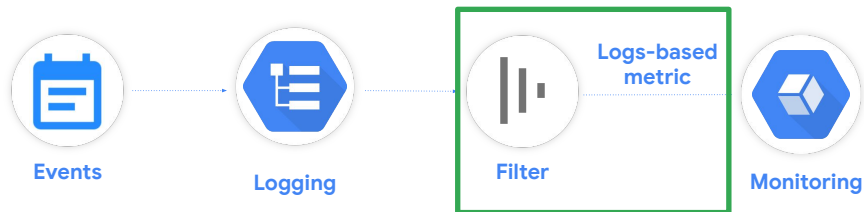
Logs-Based Metrics



Logs-based metrics are [Cloud Monitoring](#) metrics that are based on the content of log entries.

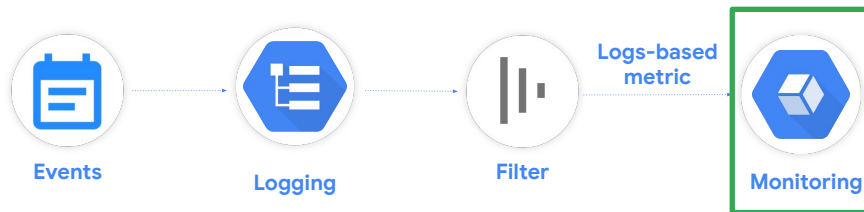
Resources generate logging events that are streamed into Google Cloud Logging.

Logs-Based Metrics



Logs-based metrics apply a **filter** to locate particular entries. For example, the metrics might record the number of log entries containing particular messages, or that were generated by a particular resource.

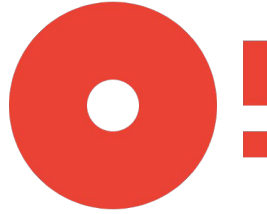
Logs-Based Metrics



Once created, you can use logs-based metrics in Cloud Monitoring charts and alerting policies.

Don't reinvent the wheel!

- Google has a curated list of over 1,000 [predefined metrics](#)
- Check there first!



Once again - don't reinvent the wheel.

Before creating your own custom logs-based metrics, take a look at Google's curated list of pre-existing metrics.

There are over 1,000, and what you're looking for might already be there.

Custom metrics should generally only be used if the metric you need is not already available, or if you need to create application metrics.

Using custom metrics may also increase your monitoring and logging costs. .

Basic test code

```
//Basic NodeJS app built with the express server
app.get('/score', (req, res) => {
  //Random score, the containerID is a UUID unique to each
  //runtime container (testing was done in Cloud Run).
  //funFactor is a random number 1-100
  let score = Math.floor(Math.random() * 100) + 1;
  //Using the Winston logging library with GCP extension
  logger.info(`/score called`,
    {score:""+score, containerID:containerID,
      funFactor:""+funFactor });
  //Basic message back to browser
  res.send(`Your score is a ${score}. Happy?`);
});
```



Before we create a simple logs-based metric, let's generate some logging entries. Here we see a basic NodeJS app built with the simple and lightweight Express web server, which we will run as a managed container on Google's Cloud Run service.

Basic test code

```
//Basic NodeJS app built with the express server
app.get('/score', (req, res) => {
  //Random score, the containerID is a UUID unique to each
  //runtime container (testing was done in Cloud Run).
  //funFactor is a random number 1-100
  let score = Math.floor(Math.random() * 100) + 1;
  //Using the Winston logging library with GCP extension
  logger.info(`/score called`,
    {score:""+score, containerID:containerID,
      funFactor:""+funFactor });
  //Basic message back to browser
  res.send(`Your score is a ${score}. Happy?`);
});
```



The code watches for a request to come into the server on the '/score' path.

Basic test code

```
//Basic NodeJS app built with the express server
app.get('/score', (req, res) => {
  //Random score, the containerID is a UUID unique to each
  //runtime container (testing was done in Cloud Run).
  //funFactor is a random number 1-100
  let score = Math.floor(Math.random() * 100) + 1;
  //Using the Winston logging library with GCP extension
  logger.info(`/score called`,
    {score:""+score, containerID:containerID,
      funFactor:""+funFactor });
  //Basic message back to browser
  res.send(`Your score is a ${score}. Happy?`);
});
```



When a /score request arrives, the code generates a random 0-99 **score**, and it then creates a log entry.

Earlier code, not shown on this slide, created a unique identifier for the container serving this request in containerID, a random value called funFactor, and it also loaded the Winston Node.js logging library, which was integrated with Google Cloud Logging.

Basic test code

```
//Basic NodeJS app built with the express server
app.get('/score', (req, res) => {
  //Random score, the containerID is a UUID unique to each
  //runtime container (testing was done in Cloud Run).
  //funFactor is a random number 1-100
  let score = Math.floor(Math.random() * 100) + 1;
  //Using the Winston logging library with GCP extension
  logger.info(`/score called`,
    {score:""+score, containerID:containerID,
      funFactor:""+funFactor });
  //Basic message back to browser
  res.send(`Your score is a ${score}. Happy?`);
});
```



The log entry contains the text "/score called" and a JSON object containing the random score, the container id, and the fun factor.

Basic test code

```
//Basic NodeJS app built with the express server
app.get('/score', (req, res) => {
  //Random score, the containerID is a UUID unique to each
  //runtime container (testing was done in Cloud Run).
  //funFactor is a random number 1-100
  let score = Math.floor(Math.random() * 100) + 1;
  //Using the Winston logging library with GCP extension
  logger.info(`/score called`,
    {score:""+score, containerID:containerID,
      funFactor:""+funFactor });
  //Basic message back to browser
  res.send(`Your score is a ${score}. Happy?`);
});
```



Lastly, a basic message, also containing the score, is sent back to the browser.

Note that when using the Winston logger for GCP, by default the entries are made into the `/projects/<project-id>/logs/winston_log`, which can be found in the Global category.

Logs-Based Metrics

The screenshot displays the Google Cloud Platform interface for managing log-based metrics. The left sidebar shows navigation options: Stackdriver Logging, Log Viewer, Log-based metrics (selected), Log Router, and Resource usage. The main content area is titled 'Log-based metrics' and includes 'CREATE METRIC' and 'DELETE' buttons. It is divided into two sections: 'System Metrics' and 'User-defined Metrics'.

System Metrics

Predefined logs-based system metrics for your project. These metrics record the number of events that occurred within a specific time period.

Name	Description
billing/bytes_ingested	The total number of billable bytes received in log entries.
billing/monthly_bytes_ingested	The total number of billable bytes received in log entries since the start of the month.
byte_count	The total number of bytes received in log entries.
excluded_byte_count	The total number of bytes excluded from log entries.
excluded_log_entry_count	The total number of log entries that were not counted because they are being excluded by a resource type exclusion or an exclusion filter.
exports/byte_count	The total number of bytes exported using sinks.
exports/error_count	The total number of log entries that were not exported due to errors.
exports/log_entry_count	The total number of log entries that were exported using sinks.
log_entry_count	The total number of log entries received.
logs_based_metrics_error_count	The total number of log entries that were not counted due to their timestamp being too old.
metric_throttled	The throttling status for logs-based metrics.
time_series_count	The estimated number of active time series for logs-based metrics.

User-defined Metrics

User-defined logs-based metrics that count the number of log entries that match a given filter.

Filter Metrics

Name	Type	Description	Previous Month Usage	Usage (MTD)	Filter
user/scene-fun	Distribution		0 B	7.89 KB	resource.type="global"

Context menu options for 'user/scene-fun':

- Edit metric
- Delete metric
- View logs for metric
- View in Metrics Explorer



Now, imagine we've generated some load on our Cloud Run sample application, and we'd now like to use the log events to generate a Logs-based metric.

There are two fundamental logs-based metric types:

Logs-Based Metrics

System Metrics

Predefined logs-based system metrics for your project. These metrics record the number of events that occurred within a specific time period.

Name	Description
billing/bytes_ingested	The total number of billable bytes received in log entries.
billing/monthly_bytes_ingested	The total number of billable bytes received in log entries since the start of the month.
type_count	The total number of bytes received in log entries.
excluded_byte_count	The total number of bytes excluded from log entries.
excluded_log_entry_count	The total number of log entries that were not counted because they are being excluded by a resource type exclusion or an exclusion filter.
exports/byte_count	The total number of bytes exported using sinks.
exports/error_count	The total number of log entries that were not exported due to errors.
exports/log_entry_count	The total number of log entries that were exported using sinks.
log_entry_count	The total number of log entries received.
logs_based_metrics_error_count	The total number of log entries that were not counted due to their timestamp being too old.
metric_throttled	The throttling status for logs-based metrics.
time_series_count	The estimated number of active time series for logs-based metrics.

User-defined Metrics

User-defined logs-based metrics that count the number of log entries that match a given filter.

Filter Metrics

Name	Type	Description	Previous Month Usage	Usage (MTD)	Filter
user/scene-fun	Distribution		0 B	7.89 KB	resource.type="global"

Actions for 'user/scene-fun': Edit metric, Delete metric, View logs for metric, View in Metrics Explorer



System logs-based metrics which are predefined by Google and are a standard part of Logs-Based Metrics.

Logs-Based Metrics

The screenshot displays the Google Cloud Platform interface for 'Logs-based metrics'. The left sidebar shows navigation options: Stackdriver Logging, Log Viewer, Logs-based metrics (selected), Log Router, and Resource usage. The main content area is titled 'Logs-based metrics' and includes 'CREATE METRIC' and 'DELETE' buttons. It is divided into two sections: 'System Metrics' and 'User-defined Metrics'.

System Metrics

Predefined logs-based system metrics for your project. These metrics record the number of events that occurred within a specific time period.

Name	Description
billing/bytes_ingested	The total number of billable bytes received in log entries.
billing/monthly_bytes_ingested	The total number of billable bytes received in log entries since the start of the month.
byte_count	The total number of bytes received in log entries.
excluded_byte_count	The total number of bytes excluded from log entries.
excluded_log_entry_count	The total number of log entries that were not counted because they are being excluded by a resource type exclusion or an exclusion filter.
exports/byte_count	The total number of bytes exported using sinks.
exports/error_count	The total number of log entries that were not exported due to errors.
exports/log_entry_count	The total number of log entries that were exported using sinks.
log_entry_count	The total number of log entries received.
logs_based_metrics_error_count	The total number of log entries that were not counted due to their timestamp being too old.
metric_throttled	The throttling status for logs-based metrics.
time_series_count	The estimated number of active time series for logs-based metrics.

User-defined Metrics

User-defined logs-based metrics that count the number of log entries that match a given filter.

Filter Metrics

Name	Type	Description	Previous Month Usage	Usage (MTD)	Filter
user/trace-span	Distribution		0 B	7.89 KB	resource.type="global"

Context menu for 'user/trace-span':

- Edit metric
- Delete metric
- View logs for metric
- View in Metrics Explorer



and then there are **User-defined logs-based metrics**, which are created by a user on a project.

These count the number of log entries that match a given query or keep track of particular values within the matching log entries.

Logs-Based Metrics

The screenshot shows the Google Cloud Platform interface for 'Log-based metrics'. The left sidebar contains navigation links: Stackdriver Logging, Log Viewer, Log-based metrics (selected), Log Router, and Resource usage. The main content area is titled 'Log-based metrics' and includes a 'CREATE METRIC' button (highlighted with a green box) and a 'DELETE' button. Below this, there are two sections: 'System Metrics' and 'User-defined Metrics'.

System Metrics

Predefined logs-based system metrics for your project. These metrics record the number of events that occurred within a specific time period.

Name	Description
billing/bytes_ingested	The total number of billable bytes received in log entries.
billing/monthly_bytes_ingested	The total number of billable bytes received in log entries since the start of the month.
byte_count	The total number of bytes received in log entries.
excluded_byte_count	The total number of bytes excluded from log entries.
excluded_log_entry_count	The total number of log entries that were not counted because they are being excluded by a resource type exclusion or an exclusion filter.
exports/byte_count	The total number of bytes exported using sinks.
exports/error_count	The total number of log entries that were not exported due to errors.
exports/log_entry_count	The total number of log entries that were exported using sinks.
log_entry_count	The total number of log entries received.
logs_based_metrics_error_count	The total number of log entries that were not counted due to their timestamp being too old.
metric_throttled	The throttling status for logs-based metrics.
time_series_count	The estimated number of active time series for logs-based metrics.

User-defined Metrics

User-defined logs-based metrics that count the number of log entries that match a given filter.

Filter Metrics

Name	Type	Description	Previous Month Usage	Usage (MTD)	Filter
user/scene-fun	Distribution		0 B	7.89 KB	resource.type="global"

Actions for the 'user/scene-fun' metric:

- Edit metric
- Delete metric
- View logs for metric
- View in Metrics Explorer



The latter is what we are creating now. You'll note the **Create Metric** button at the top of the interface.

Logs-based metric creation flow

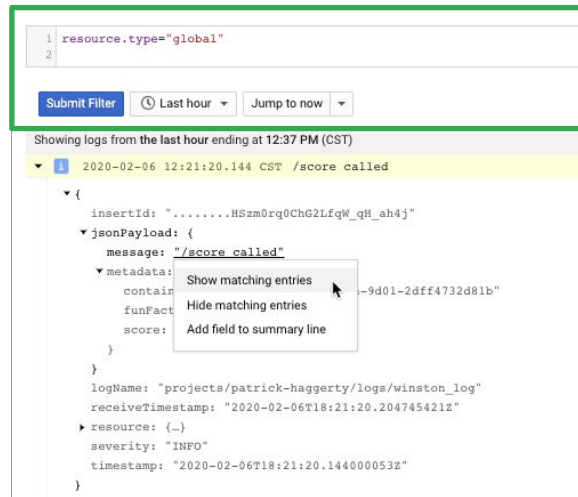
1. Find the log with the requisite data
2. Filter to the required entries
3. Pick a metric type (Counter or Distribution)
4. If Distribution, set configurations
5. Choose any secondary labels



The basic flow for creating logs-based metrics goes something like this.

1. You start by finding the log with the requisite data
2. Then you filter it to the required entries
3. Pick your metric type (Counter or Distribution)
4. If Distribution, then set configurations
5. And finally, add labels as needed

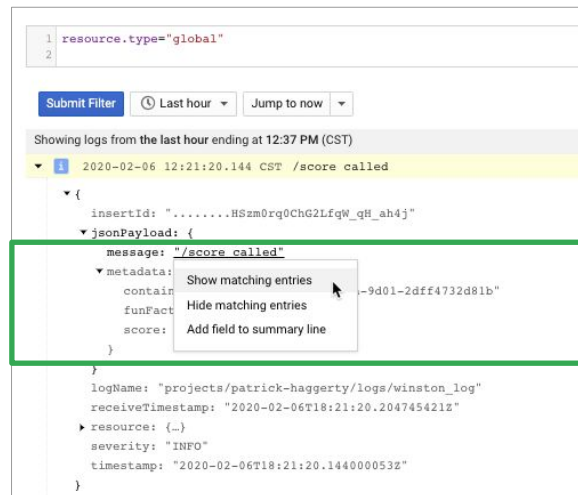
Filtering entries



You can see here that the Logs Viewer advanced view is being used.

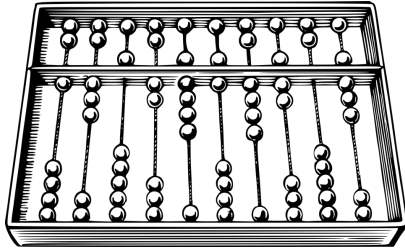
The **resource.type** has been set to **global**, since that's where the Winston log will appear.

Filtering entries

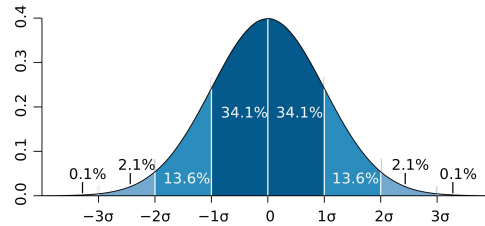


In the list of entries, we've located one of the `"/score called"` entries, and now we can filter to select those by clicking on `"/score called"` and selecting **Show matching entries**.

Metric types



Counter



Distribution

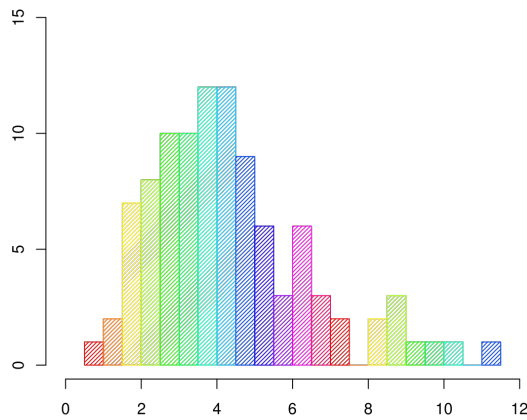


Logs-based metrics can be one of two metric types: **counter** or **distribution**. All predefined system logs-based metrics are the counter type, but user-defined metrics can be either counter or distribution types.

Counter metrics count the number of log entries matching an [advanced logs query](#). So if we simply wanted to know how many of our "/score called" entries were generated, we could create a counter.

Distribution metrics records the statistical distribution of the extracted log values in histogram buckets. The extracted values are not recorded individually, but their distribution across the configured buckets are recorded, along with the count, mean, and sum of squared deviations of the values.

A distribution as a histogram



- Linear: buckets of fixed width
- Exponential:
 - $N+2$ buckets
 - Upper: $\text{scale} * (\text{growthFactor} ^ i)$
 - Lower: $\text{scale} * (\text{growthFactor} ^ (i-1))$
- Explicit: Array of bucket boundaries
- (Up to 200 buckets)

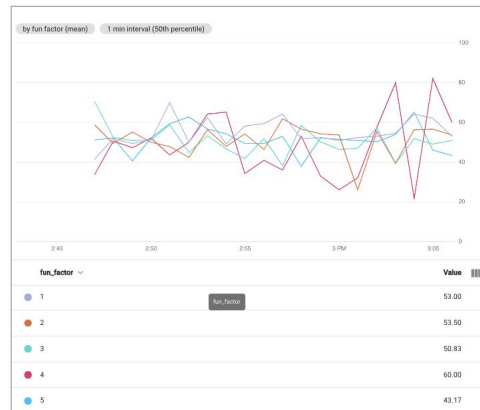


Distribution metrics include a histogram that counts the number of values that fall in specified ranges (buckets). There are three different ways to specify the boundaries between histogram buckets:

- **Linear:** Every bucket has the same width.
- **Exponential:** Bucket widths increase exponentially for higher values.
- **Explicit:** You list all the boundaries for the buckets in the *bounds* array. Bucket *i* has these boundaries:

Whichever the methodology, a distribution will support up to 200 buckets.

Labels (for example, group-by, or filter)



Like many cloud resources, labels can be applied to logs-based metrics. Their prime use is to help with group-by and filtering in Cloud Monitoring.

Labels and logs

- Allows for log-based metrics to contain a time series for each label
- Two types of labels applied:
 - Default
 - User-defined
- User-defined labels can be either of the following:
 - The entire contents of a named field in the LogEntry object
 - A part of a named field that matches a regular expression
- You can create up to 10 user-defined labels per metric
- A label cannot be deleted once created
 - And will grow time series significantly



Labels allow logs-based metrics to contain multiple time series—one for each label value.

All logs-based metrics come with some [default labels](#) and you can create additional user-defined labels in both counter-type and distribution-type metrics by specifying extractor expressions. An extractor expression tells Cloud Logging how to extract the label's value from log entries. You can specify the label's value as either of the following:

- The entire contents of a named field in the [LogEntry](#) object.
- A part of a named field that matches a regular expression (regexp).

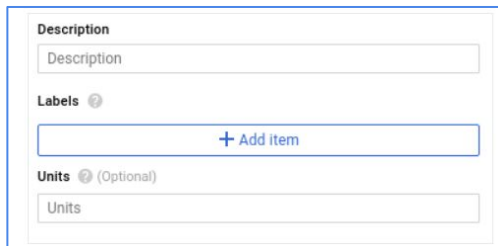
You can extract labels from the [LogEntry](#) built-in fields, such as `httpRequest.status`, or from one of the payload fields `textPayload`, `jsonPayload`, or `protoPayload`.

Label with care. A metric can support up to 10 user-defined labels, and once created, a metric cannot be removed. Also, each logs-based metric is limited to about 30,000 active time series.

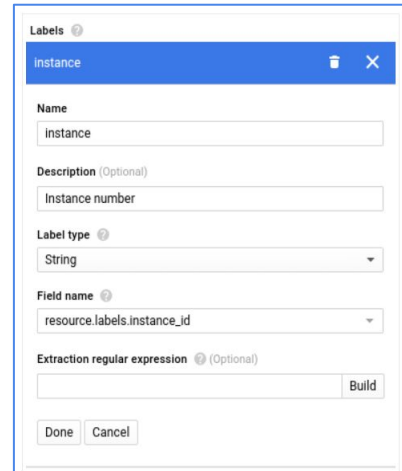
Each label can grow the time series count significantly. For example, if your log entries come from 100 resources such as VM instances, and you define a label with 20 possible values, then you can have up to 2,000 time series for your metric.

Creating user-defined labels

- User-defined labels can be created when creating a log-based metric



This screenshot shows the 'Labels' section of a form. It includes a 'Description' text field with the placeholder 'Description'. Below it is a 'Labels' section with a '+ Add item' button. At the bottom, there is a 'Units' section with a text field containing 'Units' and a '(Optional)' label.



This screenshot shows the 'Labels' dialog box. It has a title bar 'Labels' with a close button. The form includes: a 'Name' field with 'instance'; a 'Description (Optional)' field with 'Instance number'; a 'Label type' dropdown menu set to 'String'; a 'Field name' dropdown menu set to 'resource.labels.instance_id'; and an 'Extraction regular expression (Optional)' field with a 'Build' button. At the bottom are 'Done' and 'Cancel' buttons.

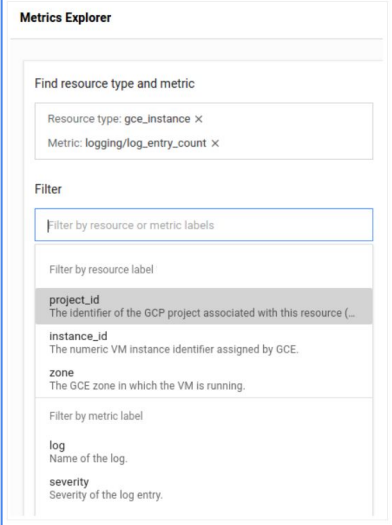


User-defined labels can be created when creating a log-based metric. The label form requires:

1. **Name:** The identifier which will be used when using the label in Monitoring.
2. **Description:** Describe the label. Try to be as specific as possible.
3. **Label type:** Choose **String**, **Boolean**, or **Integer**.
4. **Field name:** Enter the name of the log entry field that contains the label's value. This field supports autocomplete.
5. **Extraction regular expression:** If your label's value consists of the field's entire contents, then you can leave this field empty. Otherwise, specify a regular expression (regex) that extracts the label value from the field value.

Using labels

- User-based metrics can be seen by using filters within Metrics Explorer



The screenshot shows the 'Metrics Explorer' interface. At the top, it says 'Find resource type and metric'. Below this, there are two input fields: 'Resource type: gce_instance x' and 'Metric: logging/log_entry_count x'. Below these is a 'Filter' section. The first filter is 'Filter by resource or metric labels', which is expanded to show 'Filter by resource label'. Under this, there are three labels: 'project_id' (The identifier of the GCP project associated with this resource (...)), 'instance_id' (The numeric VM instance identifier assigned by GCE.), and 'zone' (The GCE zone in which the VM is running.). Below these is 'Filter by metric label', which shows 'log' (Name of the log.) and 'severity' (Severity of the log entry.).



Once created, the label and its time series will be available through the Metrics Explorer and other monitoring services. In this example, you can see the metric is set to **log_entry_count**, and at the bottom, you can filter by the **log** name or **severity** labels.

Agenda

Strategic Logging

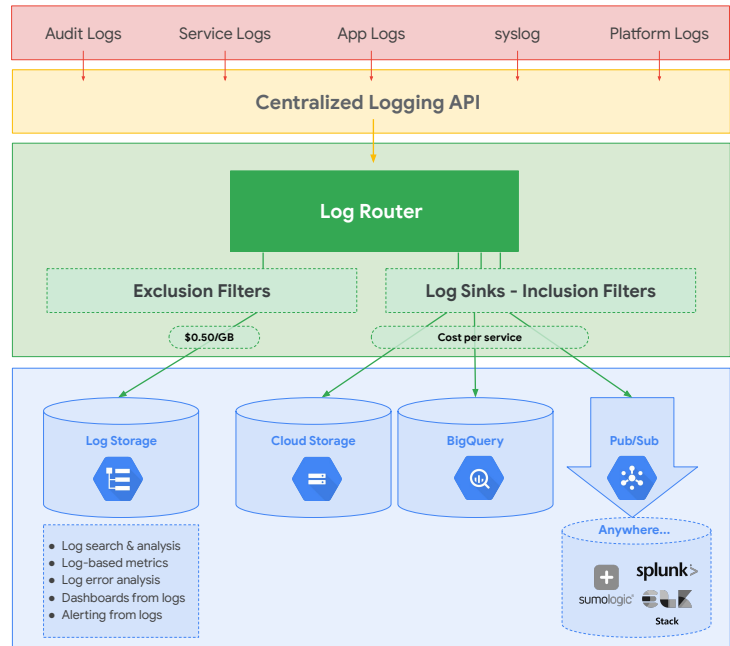
Exporting and Analyzing Logs

Error Reporting



Now that we understand the core elements involved in strategic logging and error reporting, let's look at how logs can be exported for long term storage and analysis.

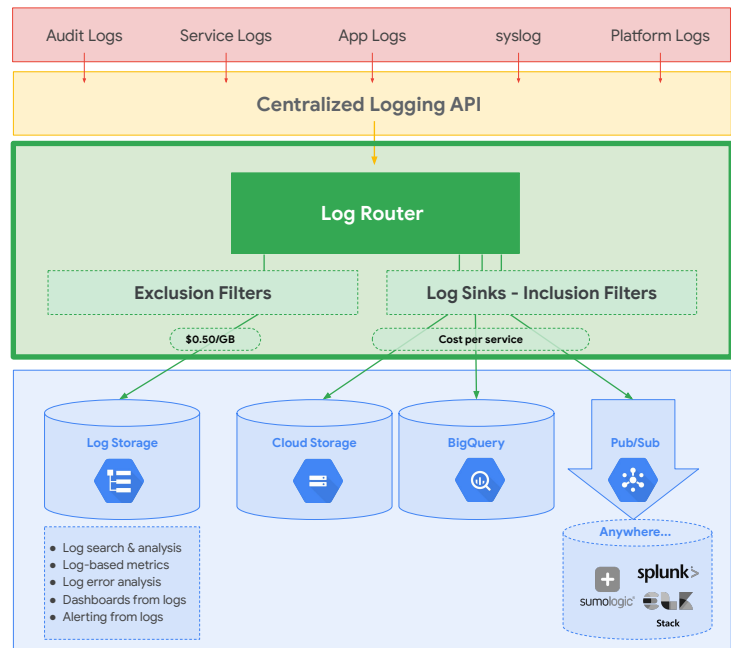
Logging Architecture



What we call Google Cloud Logging is actually a collection of components.

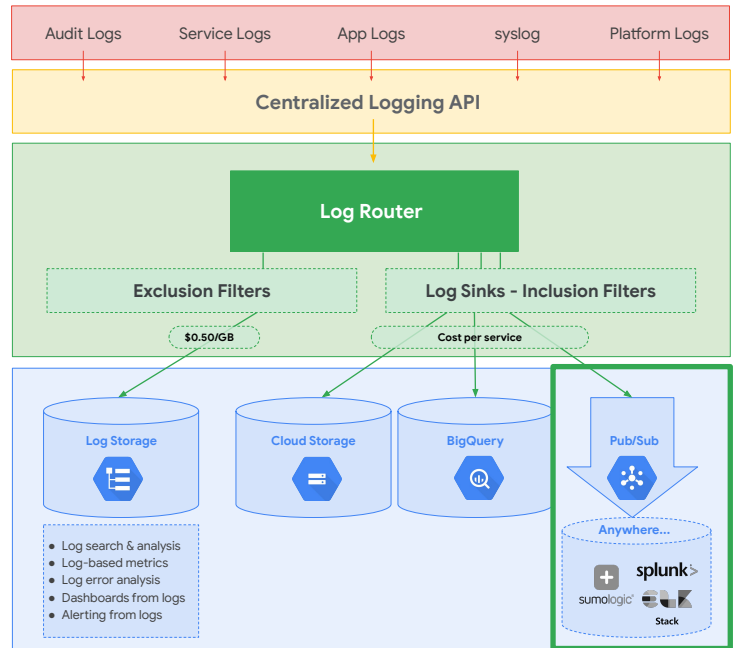
Users need to get logging events into the Logging system, so facing the outside is a centralized logging API.

Logging Architecture



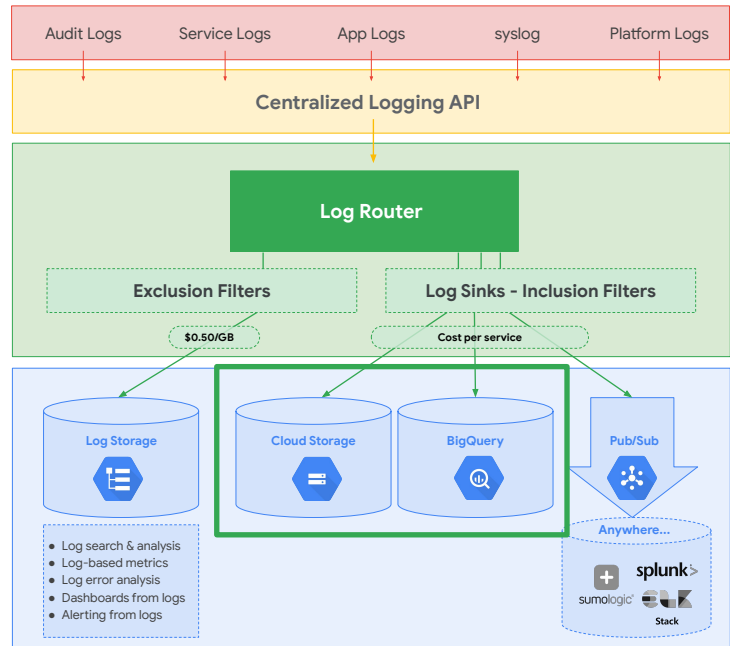
To give users a choice about where their logs go, Google created the Log Router. Log Router is optimized for processing streaming data, reliably buffering it, and sending it to any combination of BigQuery, GCS, PubSub, and of course, Log management.

Logging Architecture



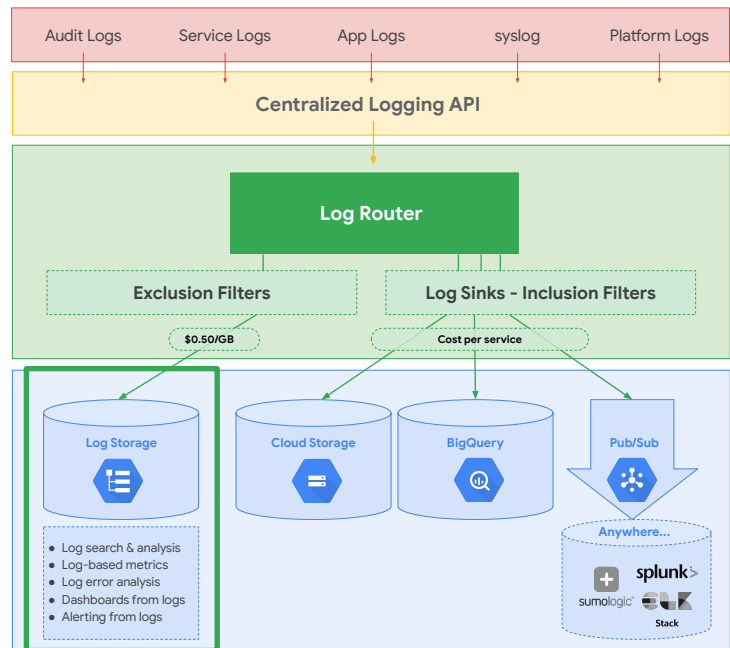
If users are already happily using Splunk, or some other external system or codebase, routing through Pub/Sub can easily make logging events available.

Logging Architecture



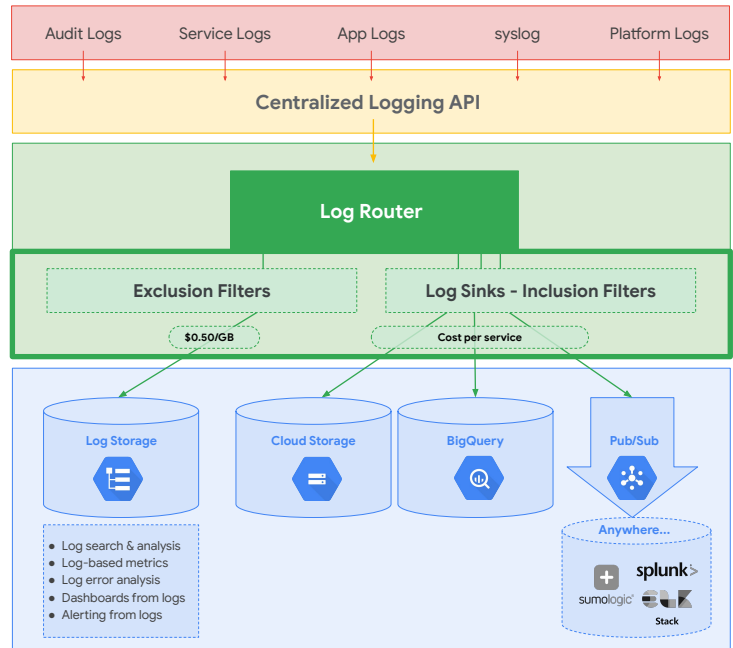
Log Router can also sink to BigQuery or Cloud Storage for long term storage, with BigQuery having the added benefit of being able to run massively scalable queries over those logs.

Logging Architecture



And finally, if users want to see their logs for operational or troubleshooting purposes, we have the cloud-based storage and interface abilities native to Google Cloud and covered earlier in this module.

Logging Architecture



Inclusion and exclusion filters can control exactly which logging entries end up at a particular destination, and which are ignored completely.

Resource Usage

Stackdriver

Logging

Logs Viewer

Logs-based Metrics

Logs Router

Resource usage

LOGS ENABLED

CREATE USAGE ALERT

Last month's ingested log volume
25.88 MiB
Total for the month of January. [See bill](#)

This month's ingested log volume
427.89 MiB
since the first of the month.

Excluded log volume
0 B
since the first of the month.

Projected ingestion log volume
2.14 GiB
by end of month

Ingestions

Exclusions

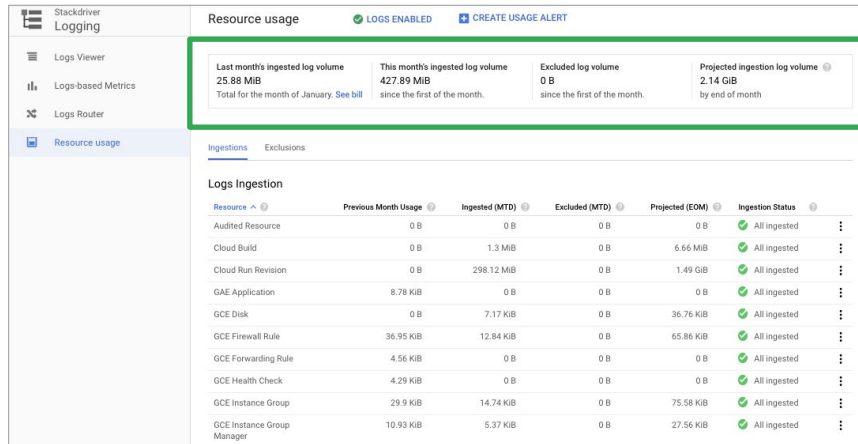
Logs Ingestion

Resource	Previous Month Usage	Ingested (MTD)	Excluded (MTD)	Projected (EOM)	Ingestion Status
Audited Resource	0 B	0 B	0 B	0 B	All ingested
Cloud Build	0 B	1.3 MiB	0 B	6.66 MiB	All ingested
Cloud Run Revision	0 B	298.12 MiB	0 B	1.49 GiB	All ingested
GAE Application	8.78 KiB	0 B	0 B	0 B	All ingested
GCE Disk	0 B	7.17 KiB	0 B	36.76 KiB	All ingested
GCE Firewall Rule	36.95 KiB	12.84 KiB	0 B	65.86 KiB	All ingested
GCE Forwarding Rule	4.56 KiB	0 B	0 B	0 B	All ingested
GCE Health Check	4.29 KiB	0 B	0 B	0 B	All ingested
GCE Instance Group	29.9 KiB	14.74 KiB	0 B	75.58 KiB	All ingested
GCE Instance Group Manager	10.93 KiB	5.37 KiB	0 B	27.56 KiB	All ingested



To track your project's logs volume, go to the **Resource usage** page in the Cloud Logging console.

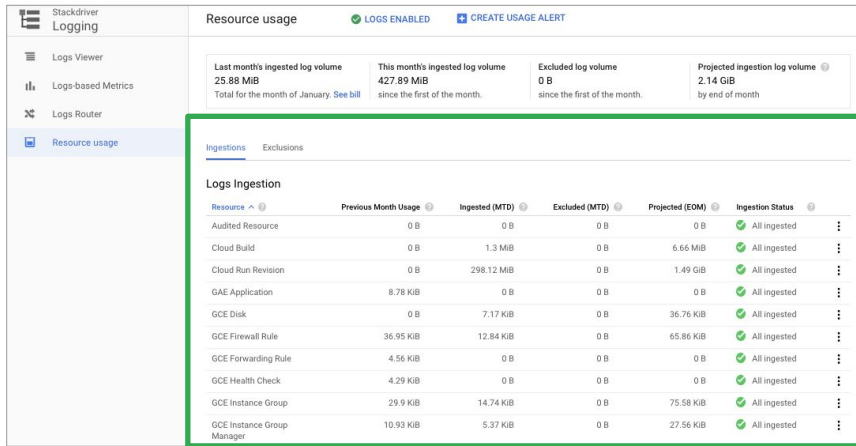
Resource Usage



The top of the page displays a summary of statistics for the logs that your project is receiving, including:

- **Last month's ingested log volume:** The amount of logs your project received in the last calendar month.
- **This month's ingested log volume:** The amount of logs your project has received since the first date of the current month.
- **Excluded log volume:** The amount of logs that you have excluded from your project since the first date of the current month. This number is not included in **This month's ingested log volume**. Excluding logs is covered on the next slide.
- **Projected ingestion log volume:** The estimated amount of logs your project will receive by the end of the current month, based on current usage.

Resource Usage



Below that, you have two tabs showing the Ingestions and Exclusions by type.

One note: The log volumes don't include Admin Activity audit logs or all System Event audit logs.

Those logs are free and cannot be excluded or disabled.

To create logging exclusions, start by clicking the Exclusions tab.

Exclusions

SHOW LIBRARY

Exclusion Editor

Filter by label or text search

Cloud Run Revision, hello-logging

All logs

Last hour

Jump to now

Showing logs from the last hour ending at 5:45 PM

Download

No older entries found matching current filter in the last hour. Load more

2020-02-06 17:15:24.200 CST	Cloud Run	ReplaceService
2020-02-06 17:15:31.554 CST	{ "textPayload": "", "insert: }	
2020-02-06 17:15:31.554 CST	> hello-logging-fun@1.0.0	
2020-02-06 17:15:31.554 CST	> node index.js	
2020-02-06 17:15:31.554 CST	{ "textPayload": "", "insert: }	

To create a logs exclusion, edit the filter on the left to only match logs you do not want to be included in Stackdriver Logging. After creating an exclusion, matched logs will no longer be accessible in Stackdriver Logging. [Learn more](#)

Name

Description

Percent to Exclude

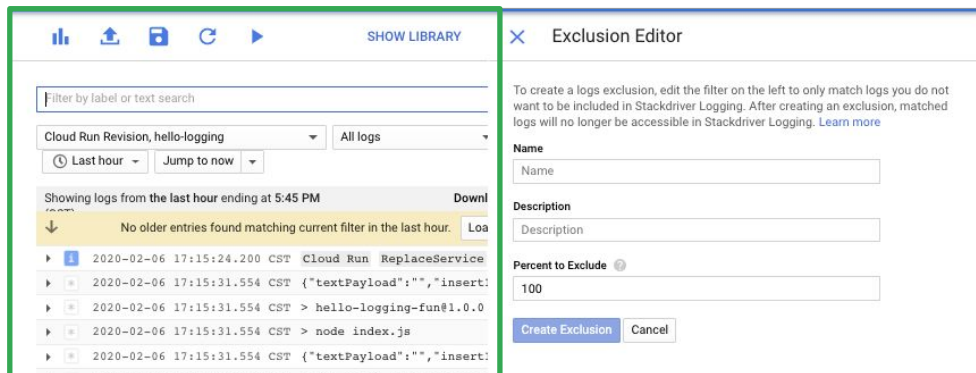
100

Create Exclusion Cancel



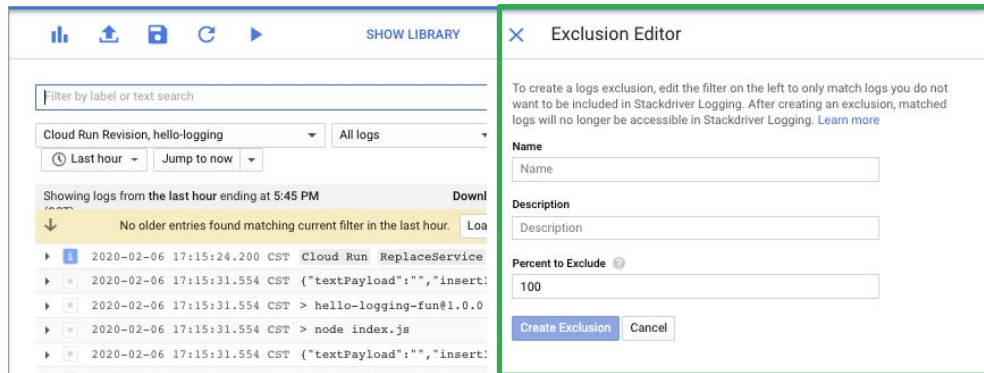
On the Logs Ingestion | Exclusions tab, click the **Create exclusion** button. You'll be presented with a split UI.

Exclusions



On the left is the Logs Viewer. Use it to correctly identify the logs you want to exclude. Take care here, because excluded log events will be lost forever.

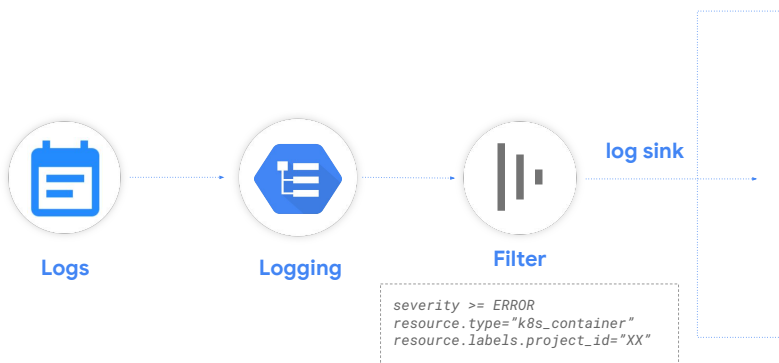
Exclusions



Once the entries are identified, and the query built, give the exclusion a name and description, and decide the percentage of log entries to exclude. It might be helpful to leave some representative events, depending on the exclusion.

Create the exclusion and it will go into effect immediately.

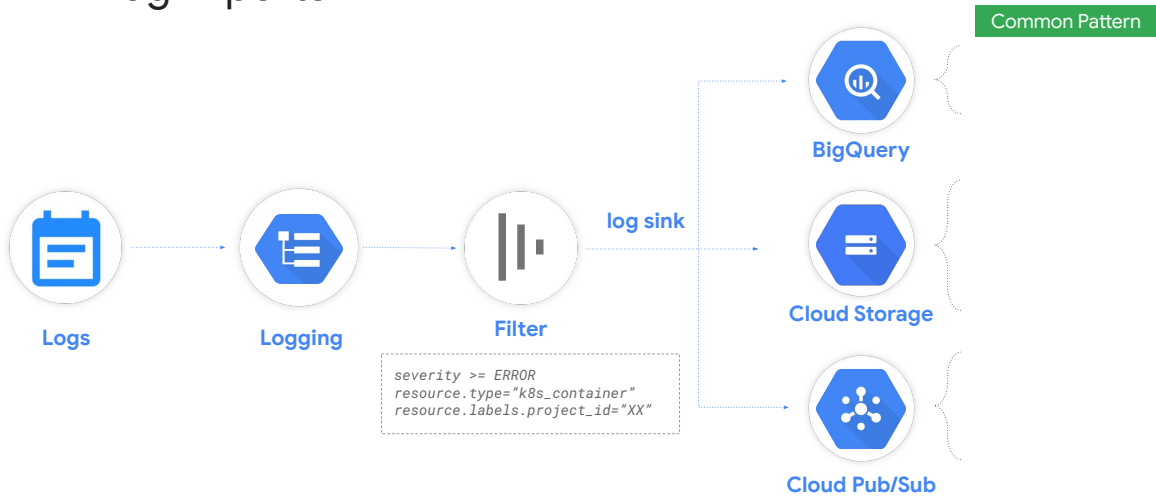
Log Exports



Log Exports can be used to forward copies of some or all of your log entries outside of Cloud Logging. Common reasons for exporting include:

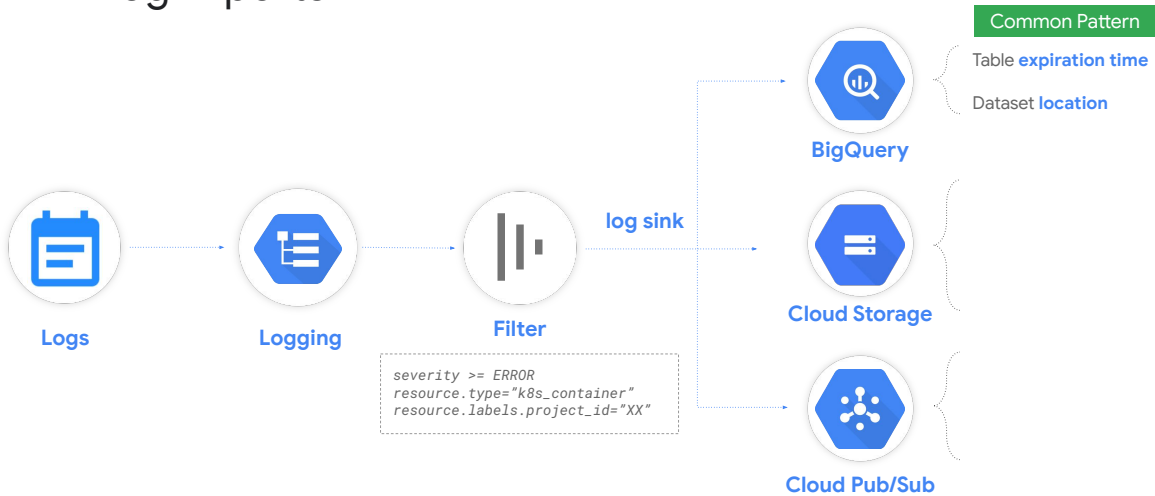
- Storing logs for extended periods.
- The use of big-data analysis tools on your logs, and
- Streaming your logs to other applications, other repositories, or to third parties.

Log Exports



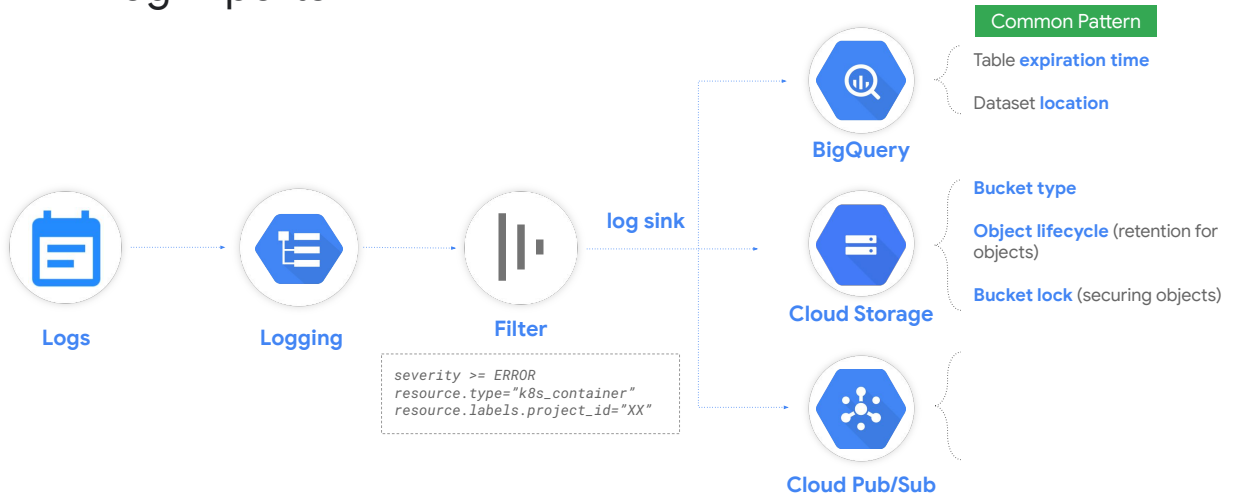
Log Sinks currently support three main targets, BigQuery, Cloud Storage, and messaging through Cloud Pub/Sub.

Log Exports



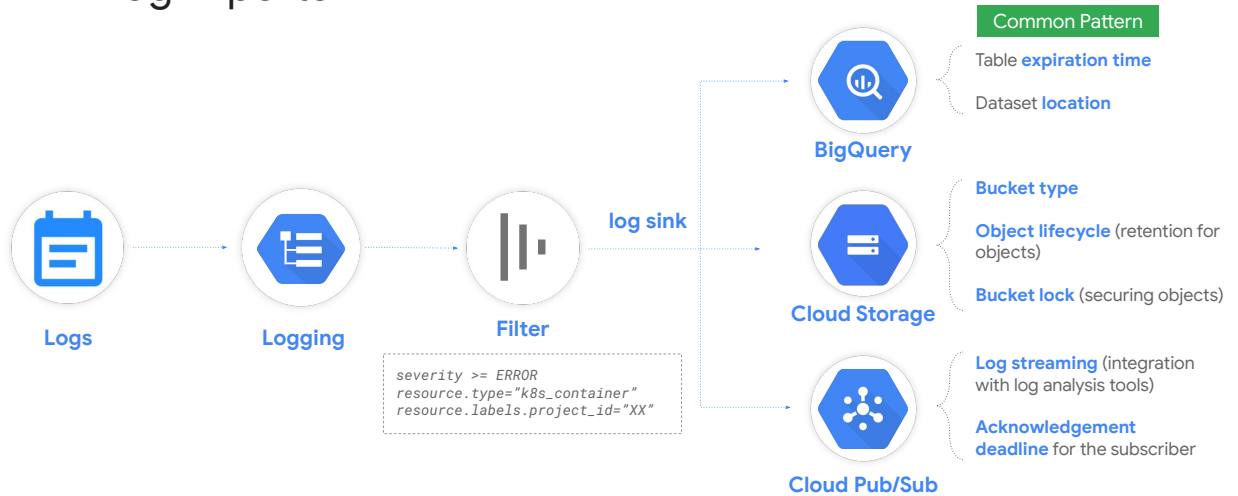
BigQuery is a common log sink as it allows both long term storage and the ability to implement powerful analytics using SQL queries.

Log Exports



Cloud Storage supports inexpensive, unlimited retention periods, lifecycle control, and locks.

Log Exports



Lastly, Pub/Sub messages allow logging events to be streamed to any system or code that can support HTTP based messages.

This option tends to be used to integrate third-party tools, or to trigger automated actions

Create a Log Sink

The screenshot shows the Google Cloud Platform interface with the 'Create Sink' dialog box open. The dialog box has a 'Sink Name' field, a 'Sink Service' dropdown menu (set to BigQuery), and a 'Sink Destination' dropdown menu (set to Custom destination). The 'Create Sink' button is highlighted. Below the dialog box, a confirmation message states 'Sink created' and provides details about the service account and destination.

Sink created

Export sink fun_sink was successfully created.

A unique service account, p1055281703932-044810@gcp-sa-logging.iam.gserviceaccount.com, has been created with permissions to write logs to the destination: bigquery.googleapis.com/projects/patrick-haggerty/datasets/fun_sink_logs.

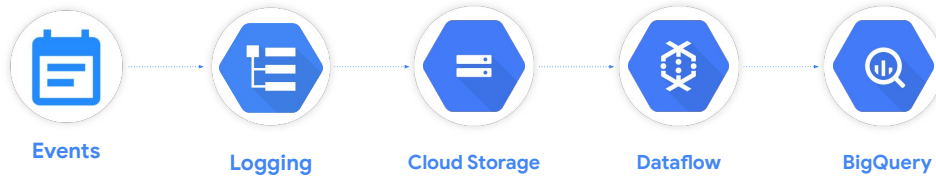
CLOSE



The process for creating log sinks mimics that of creating log exclusions. It involves writing a **query** that selects the log entries you want to export in the Logs Viewer, and choosing a **destination** of Cloud Storage, BigQuery, or Pub/Sub. The query and destination are held in an object called a **sink**. Sinks can be created in Google Cloud projects, organizations, folders, and billing accounts.

Log Archiving and Analysis

Example pipeline



Over the next several slides, we will investigate some possible log export processing options.

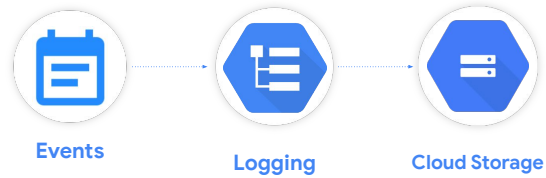
Here, for example, we are exporting through Pub/Sub, to Dataflow, to BigQuery. Dataflow is an excellent option if you're looking for real-time log processing at scale.

In this example, the Dataflow job could react to real-time issues, while streaming the logs into BigQuery for longer-term analysis.

We've already discussed one variation on this pattern: If the real-time Dataflow analysis wasn't needed, we could stream directly from logging to BigQuery.

Archive Logs for Long-Term Storage

Example pipeline

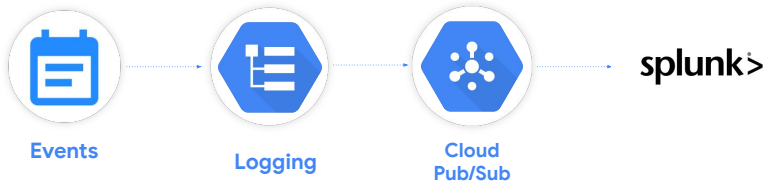


Sink pipelines targeting Cloud Storage tend to work best when your needs are in line with the Cloud Storage strengths, like long term retention, reduced storage costs, and configurable object lifecycles.

Cloud Storage features include automated storage class changes, auto-delete, and guaranteed retention.

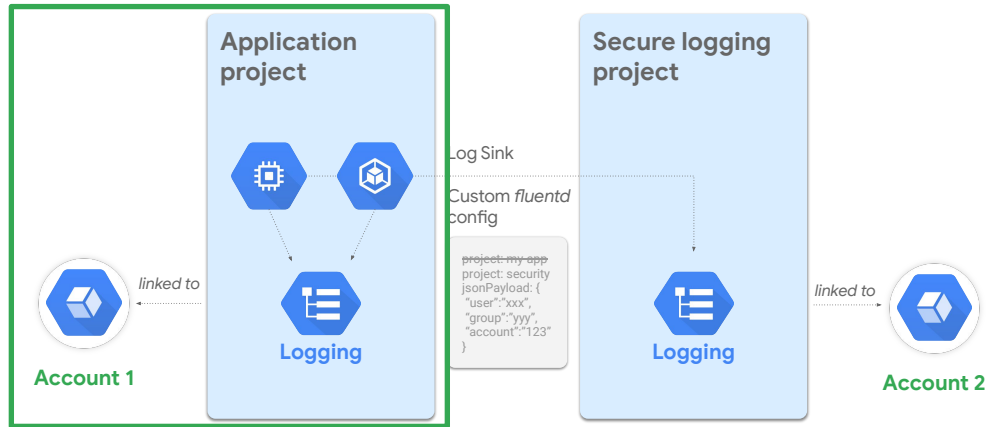
Exporting Back to Splunk

Example pipeline



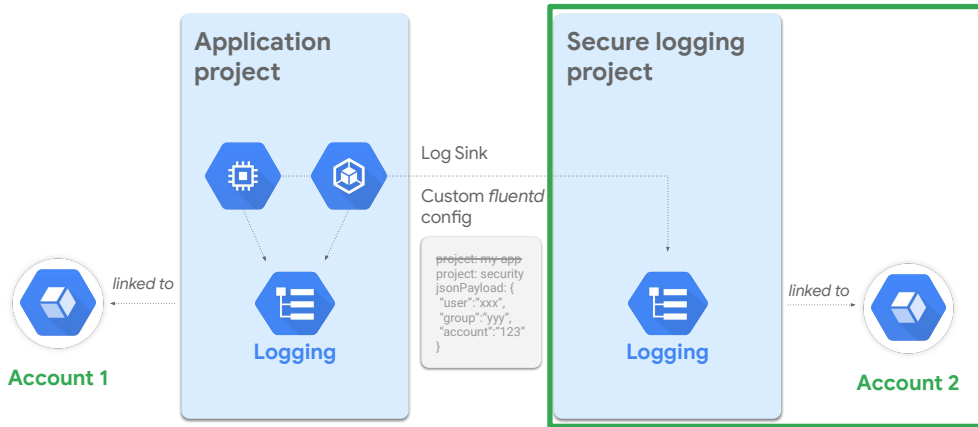
Here we have an example organization that wants to integrate the logging data from Google Cloud, back into an on-prem Splunk instance. Pub/Sub is one of the options available for exporting to Splunk, or to other third party System Information and Event Management (SIEM) software packages.

Security Logging



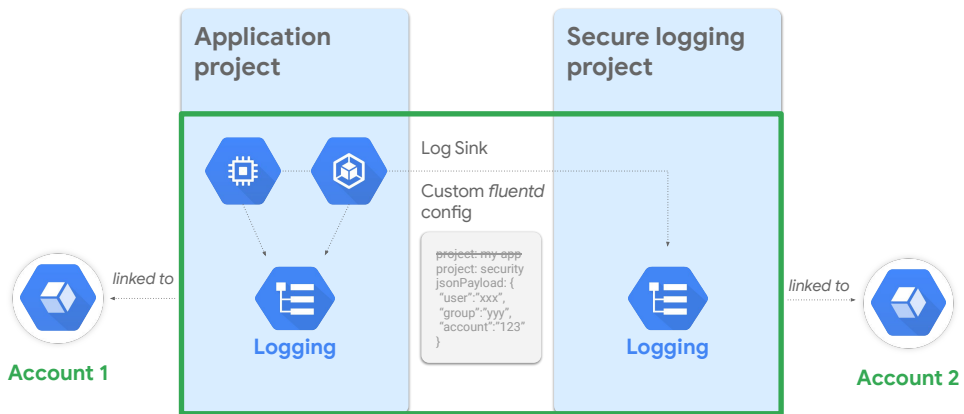
In our final example, we have Google Cloud resources creating logging data in an application project linked to Account1.

Security Logging



We have a separate secure logging project designed to provide access to Account 2 - but only to some of the logging data.

Security Logging



In the appropriate VMs in the application project, we've customized the fluentd configuration file to send specific log messages in a specific format. We've then sent those particular messages to the secure logging project using a log sink.

Aggregation Levels



Project

A **project-level log sink** exports all the logs for a **specific project**.

A **log filter** can be specified in the sink definition to include/exclude certain log types.



Folder

A **folder-level log sink** aggregates logs on the folder level.

You can also include logs from children resources (subfolders, projects).



Organization

An **organization-level log sink** aggregates logs on the organization level.

You can also include logs from children resources (subfolders, projects).



One of the most common needs regarding logs is centralizing logging data in a single location for auditing, retention, and non-repudiation purposes.

There are three available Google Cloud Logging aggregation levels.

A project-level log sink we've discussed; it exports all the logs for a specific project and a log filter can be specified in the sink definition to include/exclude certain log types.

A folder-level log sink aggregates logs on the folder level and can include logs from children's resources (subfolders, projects).

And for a global view, an organization-level log sink can aggregate logs on the organization level and can also include logs from children resources (subfolders, projects).

Aggregated Sinks

- Export log entries for multiple projects, folders, up to the organization or billing account level

```
gcloud logging sinks create [SINK_NAME] \  
storage.googleapis.com/[BUCKET_NAME] --include-children \  
--folder=[FOLDER_ID] --log-filter="logName:activity"
```

- `--folder` could also be `--organization` and `--billing-account`
- Need *Logs Configuration Writer* IAM role for parent



An aggregated sink can export log entries from all the projects, folders, and billing accounts of a Google Cloud organization. For instance, you might aggregate and export audit log entries from all an organization's projects to a central location.

The destination for log sinks has to be created before the sink. Once again, the supported destinations are a Cloud Storage bucket, Pub/Sub topic, or BigQuery table.

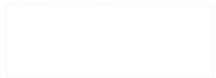
To create an aggregated sink in Google Cloud folders, billing accounts, or organizations, you can use either the [Cloud Logging API](#) or [gcloud command-line tool](#).

Here we see an example using gcloud. You would need to supply the sink name, sink destination bucket name, logs query, and the ID of the folder, billing account, or organization. Here we are filtering for the logName activity.

Other valid options besides folder include organization and billing accounts.

You would need the **Logs Configuration Writer** Cloud IAM role for the parent to create the sink.

BigQuery



Stream load logs



Make insights accessible



Build a foundation for AI



Provide real-time insights



Secure storage and access



Simplify data operations



We've mentioned several times that a common export destination for logs is BigQuery.

BigQuery has many features that can be of use when processing exported logging data.

It supports stream loading logs for to support easy access to real-time insights, while serving as a good foundation for making those insights easily accessible.

BigQuery can store data both securely and inexpensively.

It can form an excellent foundation for AI system training data and simplify data operations through its easy-to-use, ANSI 2011 compliant, SQL interface.

BQ query results can also be visualized using tools such as Data Studio and Looker.

Table Schema Based on LogEntry

Query history	Query editor
Saved queries	winston_log_20200207
Job history	Schema Details Preview
Transfers	
Scheduled queries	
Reservations	
BI Engine	
Resources + ADD DATA	
Search for your tables and datasets	
patrick-haggerty	
fun_sinks_logs	
winston_log_20200207	

Field name	Type	Mode	Description
logName	STRING	NULLABLE	
resource	RECORD	NULLABLE	
resource.type	STRING	NULLABLE	
resource.labels	RECORD	NULLABLE	
resource.labels.project_id	STRING	NULLABLE	
textPayload	STRING	NULLABLE	
jsonPayload	RECORD	NULLABLE	
jsonPayload.message	STRING	NULLABLE	
jsonPayload.metadata	RECORD	NULLABLE	
jsonPayload.metadata.score	STRING	NULLABLE	
jsonPayload.metadata.containerid	STRING	NULLABLE	
jsonPayload.metadata.funfactor	STRING	NULLABLE	
timestamp	TIMESTAMP	NULLABLE	
receiveTimestamp	TIMESTAMP	NULLABLE	
severity	STRING	NULLABLE	
insertId	STRING	NULLABLE	
httpRequest	RECORD	NULLABLE	
httpRequest.requestMethod	STRING	NULLABLE	
httpRequest.requestUri	STRING	NULLABLE	
httpRequest.requestSize	INTEGER	NULLABLE	



BigQuery table schemas for exported logs are based on the structure of the [LogEntry](#) type and the contents of the log payloads.

Cloud Logging also applies some special rules to shorten BigQuery schema field names for [audit logs](#).

You can view the table schema by selecting a table with exported log entries in the BigQuery web UI as seen on this slide.

Field Naming

Log entry field	LogEntry type mapping	BigQuery field name
insertId	insertId	insertId
textPayload	textPayload	textPayload
httpRequest.status	httpRequest.status	httpRequest.status
httpRequest. requestMethod.GET	httpRequest. requestMethod.[ABC]	httpRequest. requestMethod.get
resource.labels.moduleId	resource.labels.[ABC]	resource.labels.moduleId
jsonPayload.MESSAGE	jsonPayload.[ABC]	jsonPayload.message
jsonPayload.myField. mySubfield	jsonPayload.[ABC].[XYZ]	jsonPayload.myfield. mysubfield



There are a few naming conventions that apply to log entry fields:

- For log entry fields that are part of the [LogEntry](#) type, the corresponding BigQuery field names are precisely the same as the log entry fields.
- For any user-supplied fields, the letter case is normalized to the lower case, but the naming is otherwise preserved.
- For fields in structured payloads, as long as the `@type` specifier is not present, the letter case is normalized to the lower case, but naming is otherwise preserved. For information on structured payloads where the `@type` specifier is present, see the [Payload fields with @type](#) documentation.

You can see some examples on the current slide.

Last Three Days from *syslog* and *apache_access* for a Particular *gce_instance*

```
SELECT
  timestamp AS Time, logName as Log, textPayload AS Message
FROM
  (TABLE_DATE_RANGE(my_bq_dataset.syslog_,
    DATE_ADD(CURRENT_TIMESTAMP(), -2, 'DAY'), CURRENT_TIMESTAMP())),
  (TABLE_DATE_RANGE(my_bq_dataset.apache_access_,
    DATE_ADD(CURRENT_TIMESTAMP(), -2, 'DAY'), CURRENT_TIMESTAMP()))
WHERE
  resource.type == 'gce_instance'
  AND resource.labels.instance_id == '1554300700000000000'
ORDER BY time;
```



Here's a sample query over the Compute Engine logs. It retrieves log entries for multiple log types over multiple days,
The query searches the last three days (today -2) of the syslog and apache-access logs.
The query retrieves results for the single Compute Engine instance id seen in the where clause.

Failed App Engine Requests for the Last Month

```
SELECT
  timestamp AS Time,
  protoPayload.host AS Host,
  protoPayload.status AS Status,
  protoPayload.resource AS Path
FROM
  (TABLE_DATE_RANGE(my_bq_dataset.appengine_googleapis_com_request_log_,
    DATE_ADD(CURRENT_TIMESTAMP(), -1, 'MONTH'), CURRENT_TIMESTAMP()))
WHERE
  protoPayload.status != 200
ORDER BY time
```



In this BigQuery example, we are looking for unsuccessful App Engine requests from the last month.

Notice how the *from* clause is constructing the table data range.

The status not equal to 200 is examining the HTTP status for anything that isn't 200 - that is to say, anything that isn't a successful response.

Agenda

Strategic Logging

Exporting and Analyzing Logs

Error Reporting



Now that we've spent some time learning about core logging, let's see how Google moves past simply reporting error logs, by taking a look at Error Reporting.

Error Reporting

- Find and analyze code crashes in your cloud based services
- Centralized error management interface
- Views of error details, time charts, occurrences, and stack trace
- Support for many popular languages
 - Node.js, Python, Java, .NET, PHP, Ruby, GO
 - API available



Error Reporting counts, analyzes, and aggregates the crashes in your running cloud services.

It provides centralized error management interface displays the results with sorting and filtering capabilities.

And a dedicated view shows the error details: time chart, occurrences, affected user count, first- and last-seen dates, and a cleaned exception stack trace. Opt in to receive email and mobile alerts on new errors.

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

Google Cloud Product Support

Error Reporting can aggregate and display errors for:

- App Engine standard and Flex
- Cloud Functions
- Apps Script
- Cloud Run
- Compute Engine
- Kubernetes Engine



Error Reporting can aggregate and display errors for:

App Engine standard and Flex

Cloud Functions

Apps Script

Cloud Run

Compute Engine

and

Kubernetes Engine

Error Reporting Setup

To report errors, code will need the Error Reporting Writer IAM role

Error Reporting is available in many Google compute environments:

- App Engine, Cloud Functions, and Cloud Run: configured automatically
- GKE and GCE: use the Error Reporting API or the logging API

Note: A log message from any language or environment using [ReportedErrorEvent](#) formatting will be reported to Error Reporting



First, to report errors, code will need the [Error Reporting Writer](#) IAM role

Error Reporting is then available in many Google compute environments:

App Engine, Cloud Functions, and Cloud Run are configured to use Error Reporting automatically

In Google's Kubernetes and Compute Engine, you can either use the Error Reporting API to report errors, or you can use logging.

A side note: A log message from any language or environment using ReportedErrorEvent formatting will be reported to Error Reporting automatically

Uncaught Exceptions Automatically Reported

Let's look at an example written in Node.js and run on Cloud Run

```
//Uncaught exception, auto reported
app.get('/uncaught', (req, res) => {
  doesNotExist();
  res.send("Broken now, come back later.")
});
```



Let's look at an error-catching example written in Node.js (JavaScript), and run on Google's Cloud Run.

The full source for this example can be [found on GitHub](#) and you'll also see it in the upcoming lab.

Since Cloud Run is an environment that has pre-integrated support for Error Reporting, we don't have to do anything special to get Error Reporting working.

In languages like Node, exceptions are used to wrap error messages.

Exceptions are a language way of saying, "Hey, something bad happened, and here are some details."

Exceptions can be caught and handled by code, or they can be bubbled out to the environment.

Uncaught Exceptions Automatically Reported

Let's look at an example written in Node.js and run on Cloud Run

```
//Uncaught exception, auto reported
app.get('/uncaught', (req, res) => {
  doesNotExist();
  res.send("Broken now, come back later.")
});
```



In this case, the `doesNotExist()` function literally isn't defined by the code. As a result, calling it will generate an unhandled exception, which in turn, will generate a report in Error Reporting.

Setup to use the API

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

// Get ready to talk to the Error Reporting GCP Service
const errors = new ErrorReporting({
  reportMode: 'always' //as opposed to only while in production
});
```



To manually log errors to Error Reporting in Node, the easiest way is to import the Error Reporting library.

Setup

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

// Get ready to talk to the Error Reporting GCP Service
const errors = new ErrorReporting({
  reportMode: 'always' //as opposed to only while in production
});
```



When creating the new ErrorReporting object, the **reportMode** configuration option is used to specify when errors are reported to the Error Reporting Console.

It can have one of three values:

- **'production'**: (default): Only report errors if the NODE_ENV environment variable is set to "production".
- **'always'**: Always report errors regardless of the value of NODE_ENV.
- **'never'**: Never report errors regardless of the value of NODE_ENV.

Manually Log and Report to Error Reporting

```
app.get('/error', (req, res) => {  
  try{  
    doesNotExist();  
  }  
  catch(e) {  
    //This is a log, will not show in Error Reporter  
    logger.error("Error processing /error " + e);  
    //Let's manually pass it to Error Reporter  
    errors.report("Error processing /error " + e);  
  }  
  res.send("Broken now, come back later.")  
});
```



Now, let's handle actually an error.

This example method also calls `doesNotExist`, but it handles the error itself by catching it, instead of just letting it bubble out to the environment.

Manually Log and Report to Error Reporting

```
app.get('/error', (req, res) => {  
  try{  
    doesNotExist();  
  }  
  catch(e){  
    //This is a log, will not show in Error Reporter  
    logger.error("Error processing /error " + e);  
    //Let's manually pass it to Error Reporter  
    errors.report("Error processing /error " + e);  
  }  
  res.send("Broken now, come back later.")  
});
```



The catch first logs the error through the Winston logging library integration for Cloud Logging. That generates an entry in the `projects/YOUR_PROJECT_ID/logs/winston_log` file, but does nothing in Error Reporting.

Manually Log and Report to Error Reporting

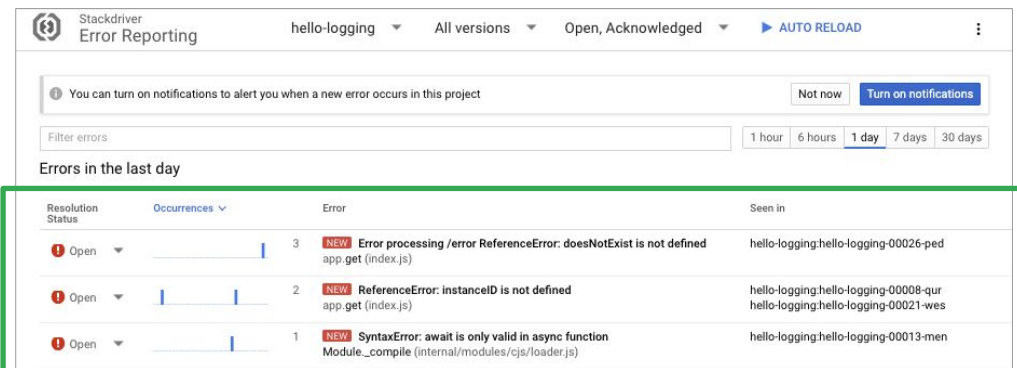
```
app.get('/error', (req, res) => {  
  try{  
    doesNotExist();  
  }  
  catch(e) {  
    //This is a log, will not show in Error Reporter  
    logger.error("Error processing /error " + e);  
    //Let's manually pass it to Error Reporter  
    errors.report("Error processing /error " + e);  
  }  
  res.send("Broken now, come back later.")  
});
```



Then we manually report the error to Error Reporting using **errors.report**.

Make sure to check the [GitHub repository for Node's Error Reporting library for syntax details](#).

Grouped List of Errors (Filtered)



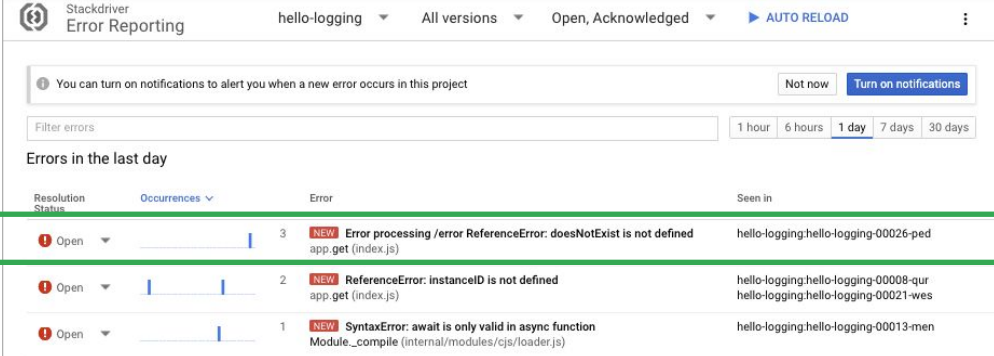
The screenshot displays the Stackdriver Error Reporting interface for a project named 'hello-logging'. The interface includes a navigation bar with 'All versions', 'Open, Acknowledged', and an 'AUTO RELOAD' button. A notification banner at the top suggests turning on notifications. Below this, a filter bar allows selecting a time range, with '1 day' currently selected. The main section, titled 'Errors in the last day', contains a table of error groups. Each group is represented by a bar chart showing the frequency of occurrences. The table lists three error groups, all with a status of 'Open'.

Resolution Status	Occurrences	Error	Seen in
Open	3	NEW Error processing /error ReferenceError: doesNotExist is not defined app.get (index.js)	hello-logging:hello-logging-00026-ped
Open	2	NEW ReferenceError: instanceID is not defined app.get (index.js)	hello-logging:hello-logging-00008-qur hello-logging:hello-logging-00021-wes
Open	1	NEW SyntaxError: await is only valid in async function Module_compile (internal/modules/cjs/loader.js)	hello-logging:hello-logging-00013-men



To see your errors, open the [Error Reporting](#) page in the Google Cloud Console. By default, Error Reporting will show you a list of recently occurring open and acknowledged errors, in order of frequency. Errors are grouped and de-duplicated by analyzing their stack traces. Error Reporting recognizes the common frameworks used for your language and groups errors accordingly.

Grouped List of Errors (Filtered)



The screenshot shows the Stackdriver Error Reporting interface for a project named 'hello-logging'. The interface includes a header with the project name, version ('All versions'), and status ('Open, Acknowledged'). There is a 'Turn on notifications' button and a 'Filter errors' input field. Below the filter, there are tabs for time ranges: '1 hour', '6 hours', '1 day' (selected), '7 days', and '30 days'. The main section is titled 'Errors in the last day' and displays a table of errors. The table has columns for 'Resolution Status', 'Occurrences', 'Error', and 'Seen in'. The first error is highlighted with a green box: 'Error processing /error ReferenceError: doesNotExist is not defined app.get (index.js)' with 3 occurrences, status 'Open', and seen in 'hello-logging:hello-logging-00026-ped'. The second error is 'ReferenceError: instanceID is not defined app.get (index.js)' with 2 occurrences, status 'Open', and seen in 'hello-logging:hello-logging-00008-qur' and 'hello-logging:hello-logging-00021-wes'. The third error is 'SyntaxError: await is only valid in async function Module_compile (internal/modules/cjs/loader.js)' with 1 occurrence, status 'Open', and seen in 'hello-logging:hello-logging-00013-men'.

Resolution Status	Occurrences	Error	Seen in
Open	3	NEW Error processing /error ReferenceError: doesNotExist is not defined app.get (index.js)	hello-logging:hello-logging-00026-ped
Open	2	NEW ReferenceError: instanceID is not defined app.get (index.js)	hello-logging:hello-logging-00008-qur hello-logging:hello-logging-00021-wes
Open	1	NEW SyntaxError: await is only valid in async function Module_compile (internal/modules/cjs/loader.js)	hello-logging:hello-logging-00013-men



Here, if you look at the top error, you'll see that 3 errors have been generated by calls to **/error**.

The error states that *doesNotExist* is not defined, and that the error came out of one of the hello-logging Cloud Run services.

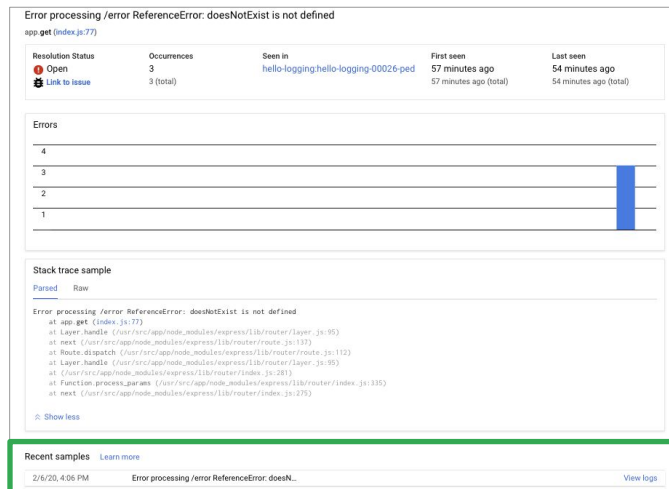
Error Reporting samples up to 1,000 errors per hour.

When this limit is reached, the displayed counts are estimated.

If too many events are received for the whole day, Error Reporting can sample up to 100 errors per hour and continue to extrapolate the counts.

You can filter, sort, and view additional details about errors, as well as restrict the errors that appear in the list to a specific time range.

Select an Error for Details

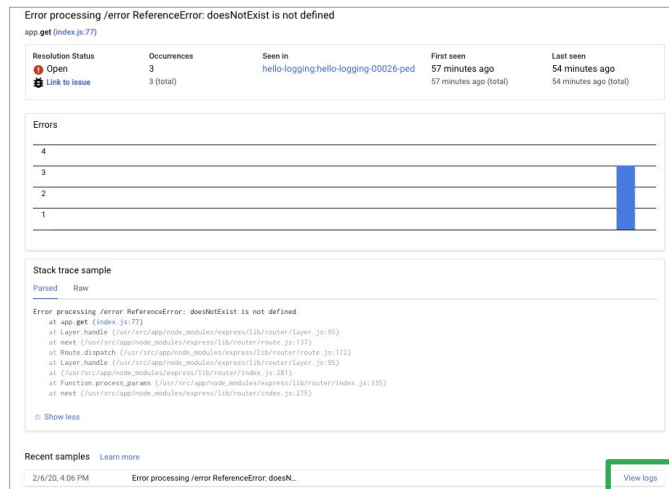


Selecting an error entry will allow you to drill down into the Error Details page.

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

Sample errors are found in the **Recent samples** panel. Each sample represents one occurrence of the error and includes a parsed stack trace.

Select an Error for Details



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

This takes you to the Logs Viewer in the Cloud Logging console.

Lab Intro

Log Analysis



In this lab, you will generate logging entries from an application, filter and analyze logs, work with Error Reporting, and export logs to a BigQuery sink.

Quiz

You want to be able to compare resource utilization for VMs used for production, development, and testing?

- A. Add a label called “state” to your VMs with the values “dev”, “test”, and “prod” and group by that label in your monitoring chart
- B. Put those resources in different projects and use dataflow to create an aggregation of log values for each
- C. Name the VMs with a prefix like “dev-”, “test-”, and “prod-” and filter on the name property when reporting
- D. Export all machine logs to Cloud Storage and use Cloud Functions to build reports based on the VM tags

Quiz

You want to be able to compare resource utilization for VMs used for production, development, and testing?

- A. Add a label called “state” to your VMs with the values “dev”, “test”, and “prod” and group by that label in your monitoring chart
- B. Put those resources in different projects and use dataflow to create an aggregation of log values for each
- C. Name the VMs with a prefix like “dev-”, “test-”, and “prod-” and filter on the name property when reporting
- D. Export all machine logs to Cloud Storage and use Cloud Functions to build reports based on the VM tags

Quiz

Your governance team has mandated you save your log data for 5 years for compliance reasons. Where would the best place to export it be?

- A. Cloud Storage in a multi-region bucket
- B. Cloud Storage in a single-region bucket using the archival storage class
- C. BigQuery
- D. You don't need to export it, just set the retention policy in Logging to 5 years

Quiz

Your governance team has mandated you save your log data for 5 years for compliance reasons. Where would the best place to export it be?

- A. Cloud Storage in a multi-region bucket
- B. Cloud Storage in a single region bucket using the archival storage class
- C. BigQuery
- D. You don't need to export it, just set the retention policy in Logging to 5 years

Quiz

You want to use the logs to monitor application usage in real time.
Where would the best export sink be?

- A. Cloud Storage
- B. Pub/Sub
- C. BigQuery
- D. Spanner

Quiz

You want to use the logs to monitor application usage in real time.
Where would the best export sink be?

A. Cloud Storage

B. Pub/Sub

C. BigQuery

D. Spanner

Quiz

You manager wants a daily report of resource utilization by application. Where would the best export sink be?

- A. Cloud Storage
- B. Pub/Sub
- C. BigQuery
- D. Spanner

Quiz

You manager wants a daily report of resource utilization by application. Where would the best export sink be?

- A. Cloud Storage
- B. Pub/Sub
- C. BigQuery
- D. Spanner

Learned how to...

Identify and choose among resource tagging approaches

Define log sinks (inclusion filters) and exclusion filters

Create metrics based on logs

Connect application errors to Logging using Error Reporting

Export logs to BigQuery