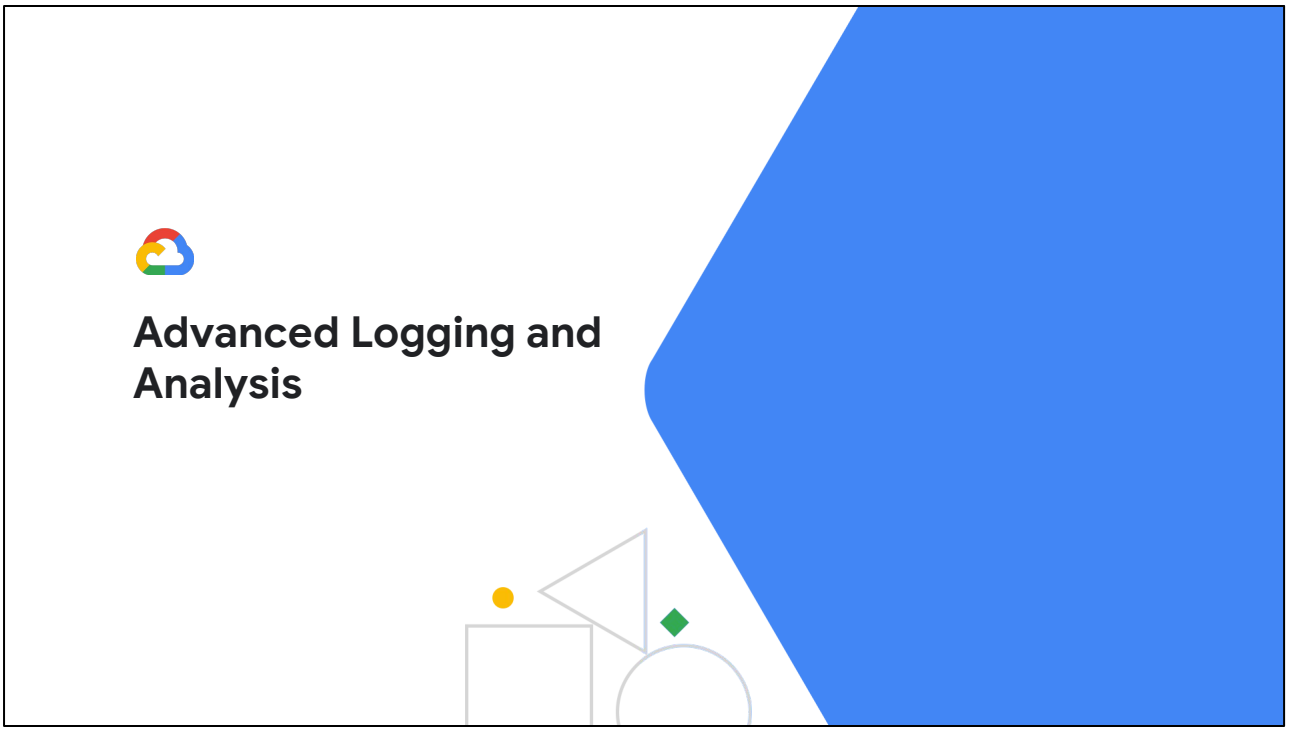


In this module, we will examine some of Google Cloud's advanced logging and analysis capabilities.

Agenda

Strategic Logging

- Labeling
- Working with Logs Explorer
- Using Logs-Based Metrics

Exporting and Analyzing Logs

Error Reporting

Specifically, in this module you will learn to:

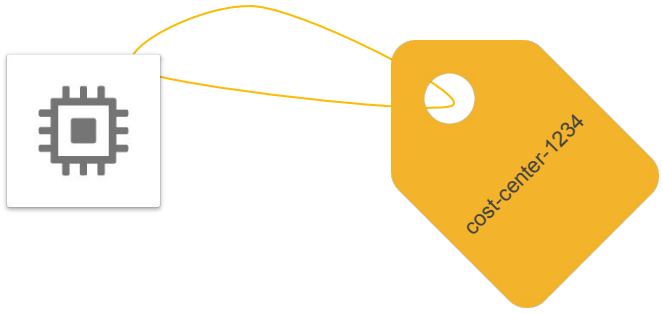
- Identify and choose among resource tagging approaches because tags can make locating and tracking resources easier.
- Define log sinks (inclusion filters) to include specific log entries, and exclusion filters to exclude others.
- Create monitoring metrics based on log entries. For example, "this NGINX 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

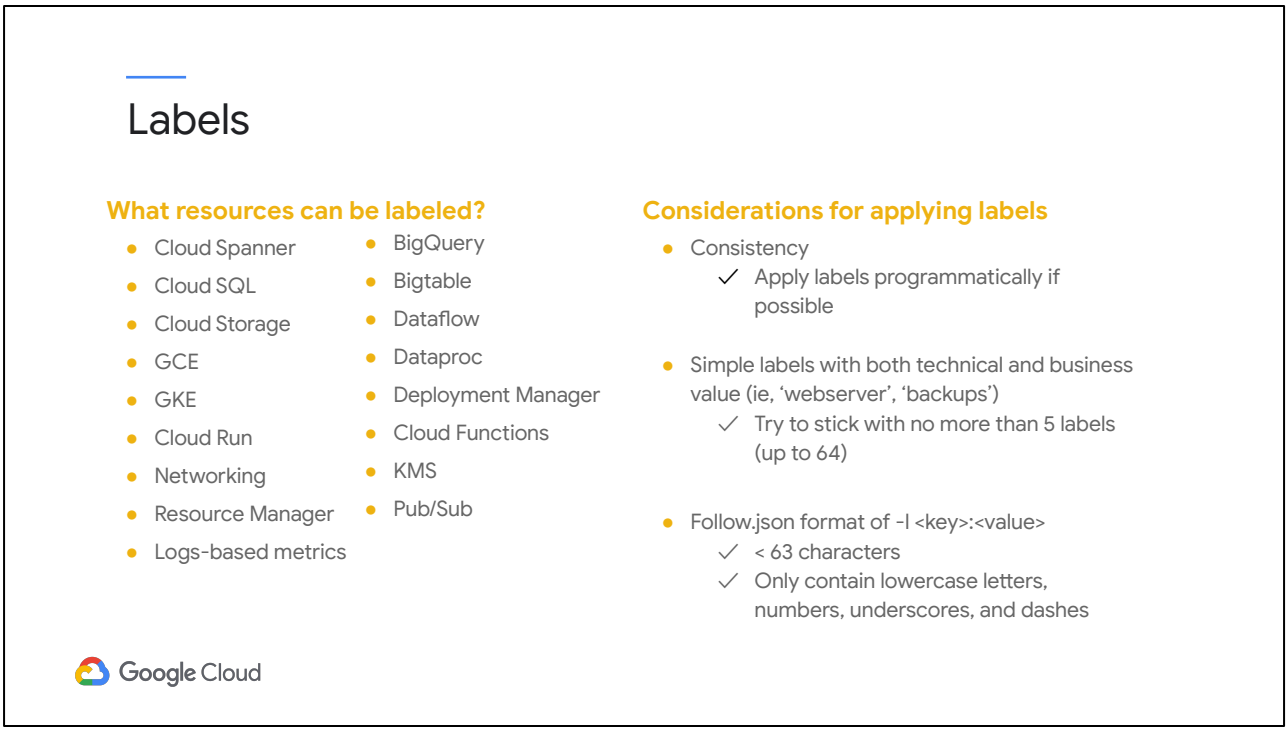
- Strategic Logging
  - Labeling
  - Working with Logs Explorer
  - Using Logs-Based Metrics
- Exporting and Analyzing Logs
- Error Reporting

Let’s start with labeling.

Labels Help Identify Resources



Labels are key-value pairs that help organize and to identity 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.



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

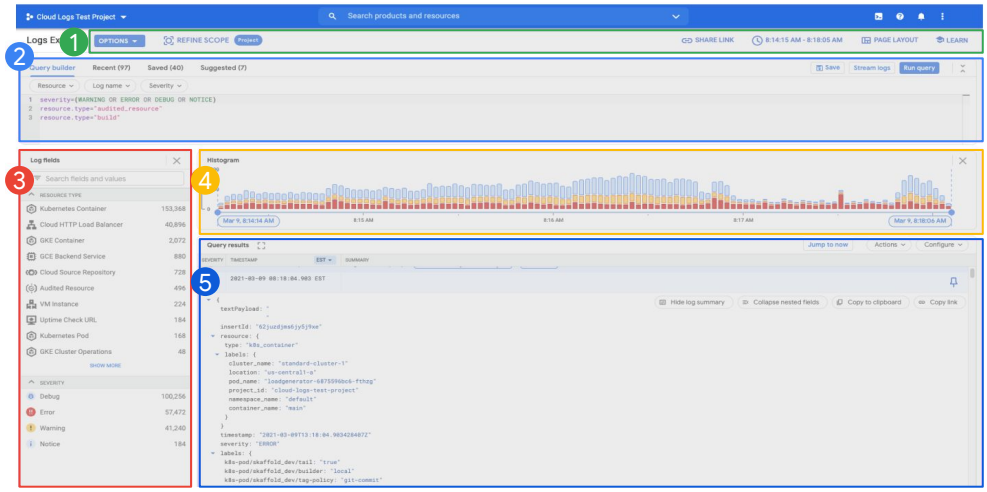
- Since labels are arbitrary, key-value pairs, consistency can be challenging. If one user sets a label city:ny and another sets a label as city:nyc, GCP considers these two different labels, and text-based searching could easily find one and miss the other. Consider applying labels programmatically 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 Logs Explorer
  - 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.

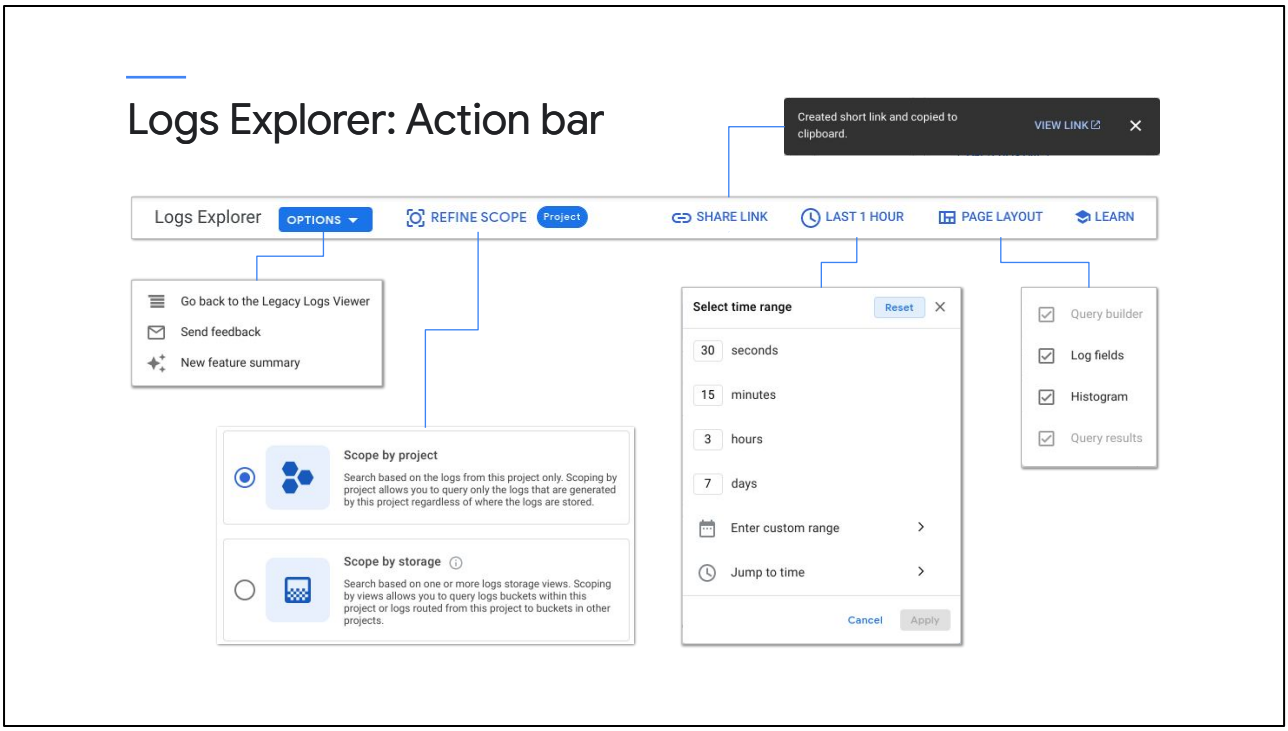
Logs Explorer interface



The Logs Explorer interface lets you retrieve logs, parse and analyze log data, and refine your query parameters. The Logs Explorer contains five different panes:

- Action bar
- Query builder
- Log fields
- Histogram, and
- Query results

Let’s look at each in more detail.



From the **Action bar** pane, you can access the following:

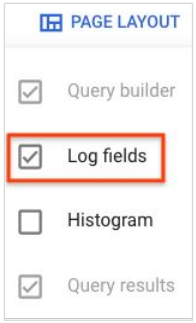
1. **Options**: Lets you go to the Legacy Logs Viewer, send feedback, and view a summary of new Logging features.
2. **Refine scope**: Lets you scope your search by logs in your current Cloud project only or by one or more storage views. For more information, see Refining scope.
3. **Share link**: Lets you create a shortened URL of the current query and copies it to your clipboard, making it easier to share a query.
4. **Time-range selector**: Lets you restrict query results by time range. The default time range is one hour.
5. **Page layout**: Lets you enable and disable the **Histogram** and **Logs field explorer** panes.
6. **Learn**: Lets you view links to relevant documentation.

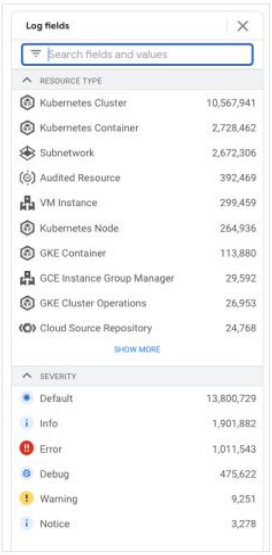
Logs Explorer: Query builder

From the **Query builder** pane,you can access the following:

1. **Query-builder field**: Lets you build queries using the Logging query language.
2. **Query builder drop-down menus**: Lets you add query expressions based on **Resource**, **Log name**, and **Severity**. For more information, see Query builder drop-down menus.
3. **Recent**: Lets you view your recent queries. For more information, see Recent queries.
4. **Saved**: Lets you view your saved queries and queries that other users of the Cloud project have shared with you. For more information, see Saved queries and Shared queries.
5. **Suggested**: Lets you view suggested queries based on the resources in your Cloud project. For more information, see Suggested queries.
6. **Save**: Lets you save queries that can be viewed and run from the **Saved** tab.
7. **Stream logs**: Lets you view log entries as Logging ingests them. For more information, see Streaming logs.
8. **Run query**: Lets you run your queries after you have built them in the query-builder field.

Logs Explorer: Log fields



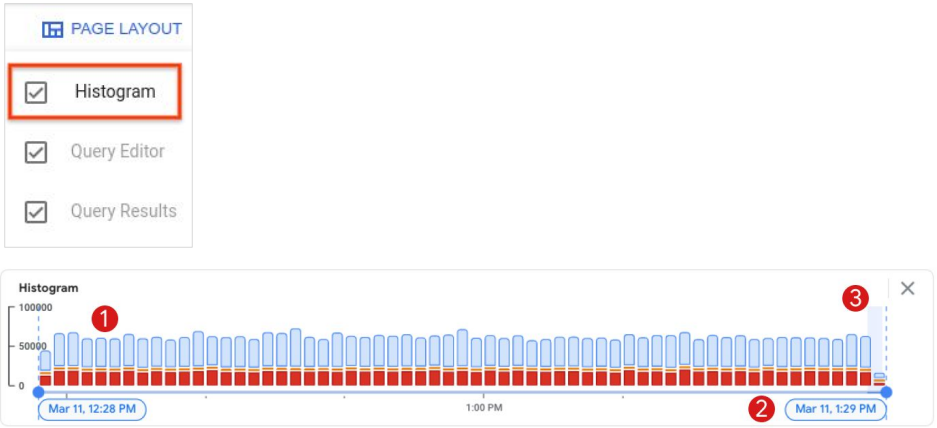


The Log fields pane is enabled and disabled in the **Page Layout** dropdown located in the action bar.

The Log fields pane offers a high-level summary of logs data and provides a more efficient way to refine a query. It shows the count of log entries, sorted by decreasing count, for the given log field, and provides aggregation-based results for the resource.type, resource.labels, logName, and severity fields. The log-field counts correspond to the time range used by the Histogram pane.

The Log fields pane is populated and updated based on an executed query. When there is an empty query, the Log fields pane displays the counts of log entries by the resource type and log severity fields.

Logs Explorer: Histogram



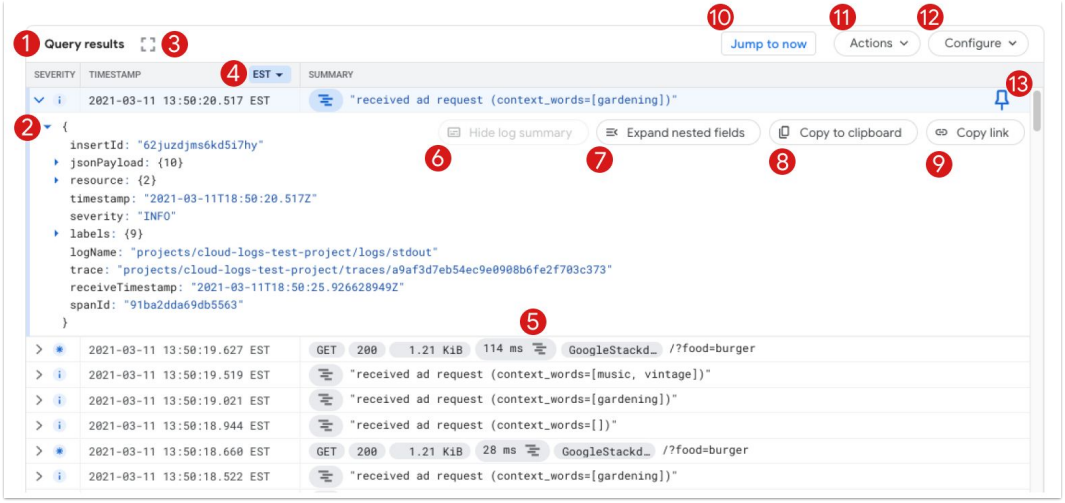
The Histogram pane is also enabled and disabled in the **Page Layout** dropdown.

The Histogram pane lets you visualize the distribution of logs over time. The histogram is generated when you run a query, making it easier to see trends in your logs data and troubleshoot problems.

1. Each histogram bar contains a three-color breakdown for the log-severity levels captured in each histogram's time range. The colors represent the following log severities:
    - Blue (Low severities) : Default, Debug, Info, Notice
    - Yellow (Medium severities): Warning
    - Red (High severities): Error, Critical, Alert, Emergency

2. You can change the time range used for queries by adjusting the handles. After adjusting the handles, a Run button is displayed to update the time range used in the query.
3. The histogram viewport lets you see the time range of the logs that are currently displayed within the Query results pane. The viewport helps to orient you to the logs you're currently viewing within the larger time range of your query.

Logs Explorer: Query results

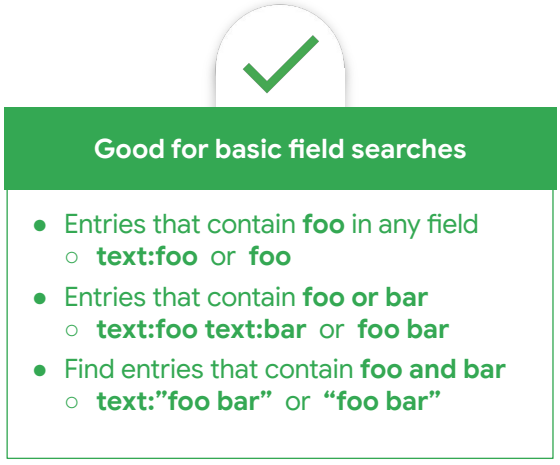


The **Query results** pane lets you explore the log entries that match your query expressions.

1. **Query results**: Lets you view the retrieved logs from your query.
2. **Log entries**: Lets you view log entries in the structured JSON format.
3. **Expand and collapse query results**: Lets you expand the query-results pane to view more log entries.
4. **Time zone**: Lets you change the time zone that logs are displayed in.
5. **Trace data**: Lets you view trace details and refine your query based on the trace. For more information, see Viewing trace data.
6. **Hide log summary**: Lets you hide the log summary line from the query results.
7. **Expand or collapse nested log fields**: Lets you expand or collapse nested fields.
8. **Copy to clipboard**: Lets you copy the log entry in its JSON format.
9. **Copy link to a log entry**: Lets you share a link to a log entry. For more information, see Copying a link to a log entry.
10. **Jump to now**: Lets you perform a forced refresh to include the current time. If the time-range selector uses a custom range and an end time is set, it runs the query with a default time range of one hour. Otherwise, it refreshes with the current start date or duration, and runs the query.
11. **Actions**: Lets you set up a logs-based metric, create a sink destination, or download your logs. For more information on downloading logs, see Downloading logs.
12. **Configure**: Lets you add the value of a log field to the summary line at the

1. beginning or end of the log entry. It also lets you choose to show newest logs either first or last. For more information on adding a summary field, see Adding summary fields.
2. **Pin log entry**: Lets you pin a log entry to the **Query results** and **Histogram** panes. Depending on how your **Query results** pane is configured, Logging pins the log either to the top or to the bottom of the **Query results** pane. For more information, see Pinning logs.

Basic Filters



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

Basic features won't support:

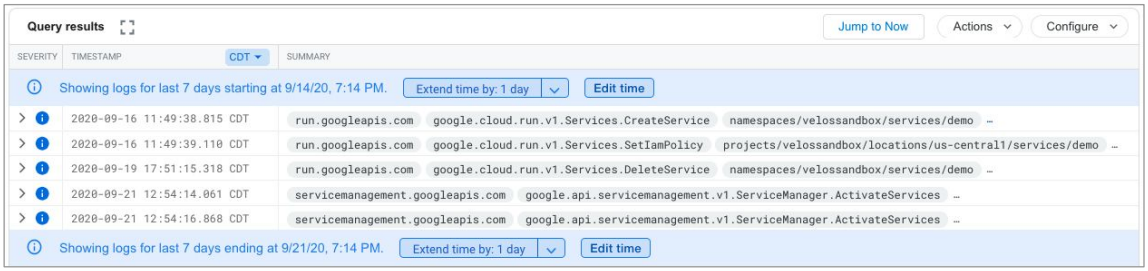
Wildcard characters like in **foo***

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

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

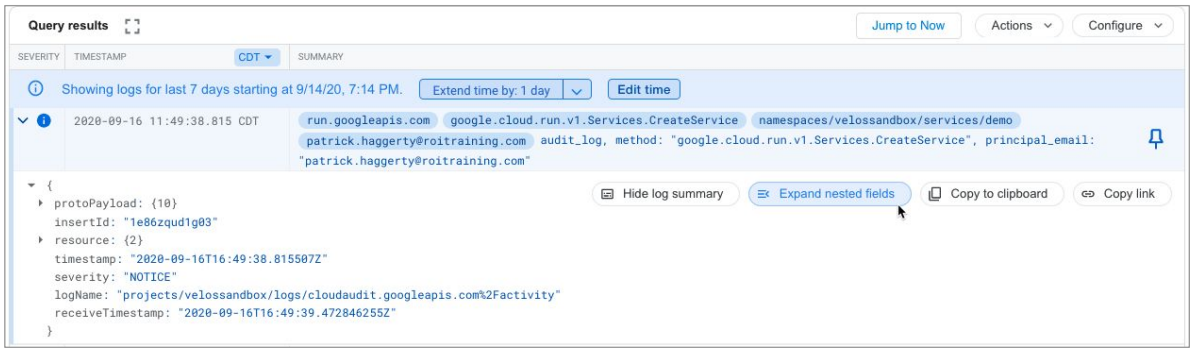
or using boolean operators in strings like in **"foo NOT bar"**

Log entries



The log-entry table displays an entry line for each log entry. In the line, you see the entry severity, timestamp, and any values for fields that have been promoted to the summary.

Log entry details



To see the full details for one log entry, click the expander arrow (▸) at the front of the summary line, and then click **Expand nested fields**. The log entry is displayed using JSON format.

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 modify the log query appropriately.

The query selects the entries displayed by Logs Explorer

Ultimately, it's the query that selects the entries displayed by Logs Explorer. Queries may be created directly with the Logging Query Language (LQL), using the drop-down menus, the logs field explorer, or by clicking fields in the results themselves.

Start with what you know about the entry you're trying to find. If it belongs to a resource, a particular log file, or has a known severity, use the query builder drop-down menus.

The query selects the entries displayed by Logs Explorer

Resource lets you specify resource.type. You can select a single resource at a time to add to the **Query builder**. Entries use the logical operator AND.

The query selects the entries displayed by Logs Explorer

Log name lets you specify logName. You can select multiple log names at once to add to the **Query builder**. When selecting multiple entries, the logical operator OR is used.

The query selects the entries displayed by Logs Explorer

And **Severity** lets you specify severity. You can select multiple severity levels at once to add to the **Query builder**. When selecting multiple entries, the logical operator OR is used.

Advanced Filter Comparison Operators

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

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 for to lessen the amount of 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 Logs Explorer
  - Using Logs-Based Metrics
- Exporting and Analyzing Logs
- Error Reporting

Now that we've seen how Logs Explorer 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

Let's review some of the key IAM roles that relate to logging and monitoring.

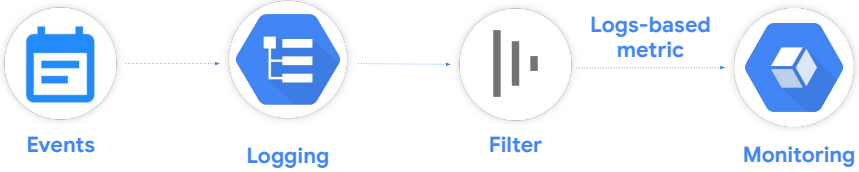
First, on the logging side:

- **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 time series 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



Google Cloud

Logs-based metrics are Cloud Monitoring metrics that are based on the content of log entries.

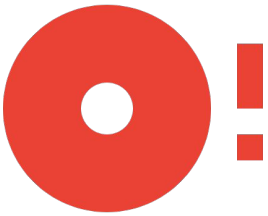
Resources generate logging events that are streamed into Google Cloud Logging.

Logs-based metrics 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.

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.

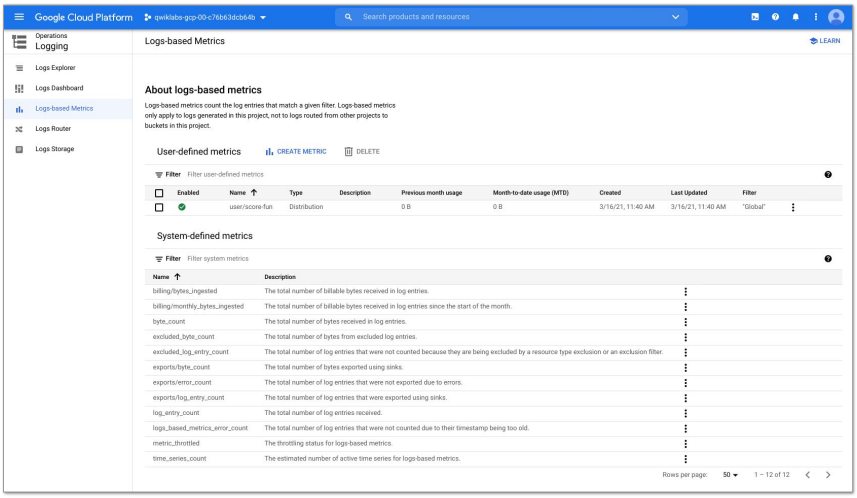
Basic test code

```
//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 Google Cloud, by default the entries are made into the /projects/<project-id>/logs/winston_log, which can be found in the Global category.

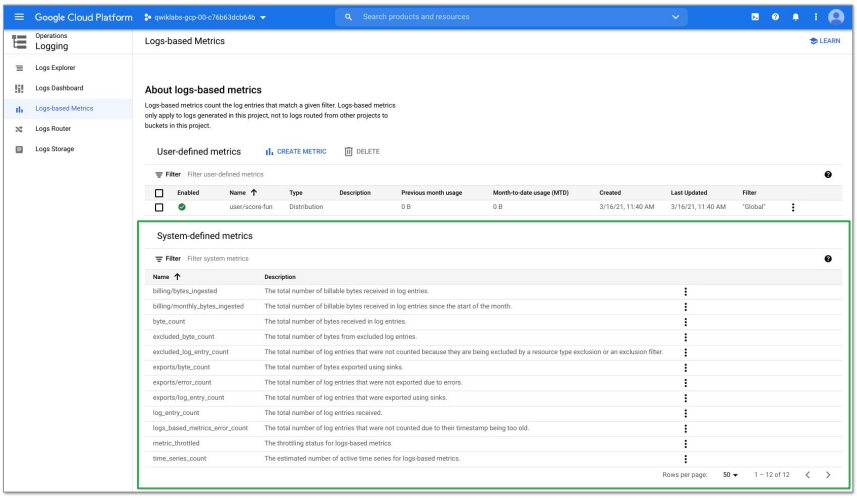
Logs-based metrics



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.

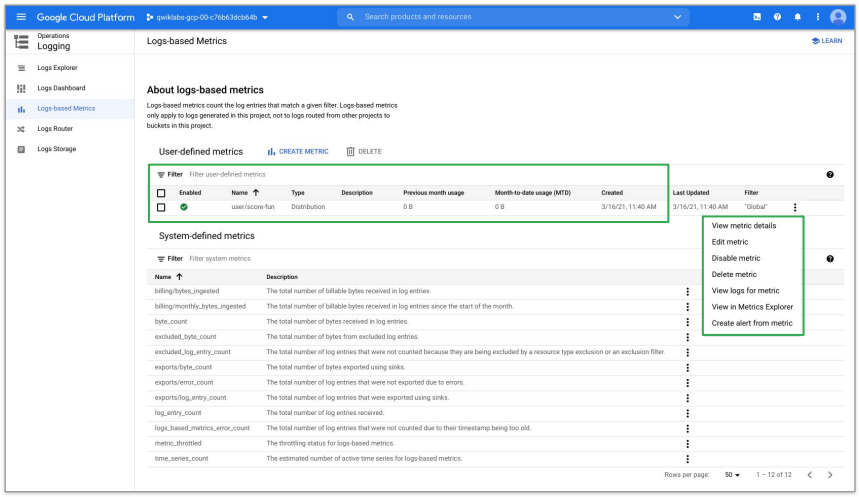
There are two fundamental logs-based metric types:

Logs-based metrics



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

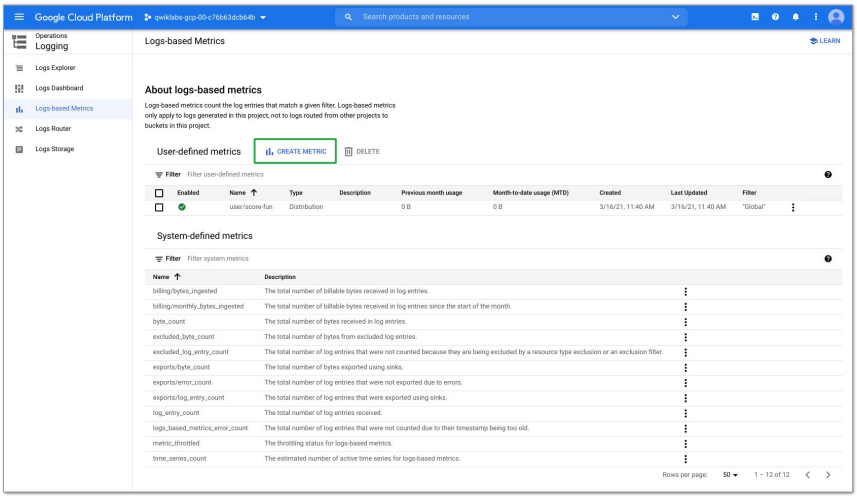
Logs-based metrics



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

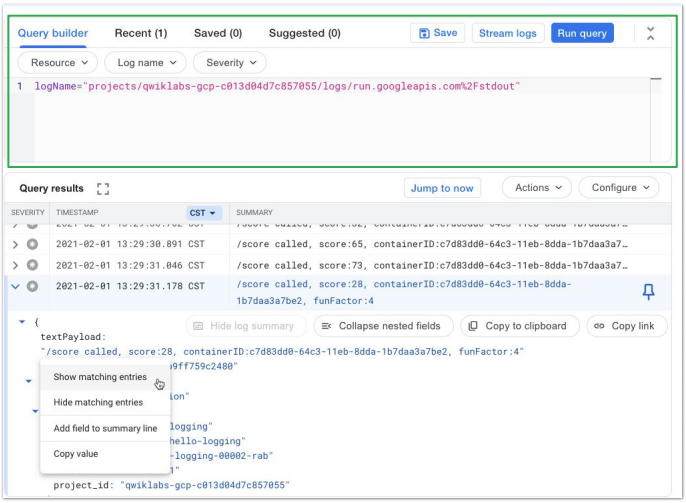
In Logs Explorer

1. Find the log with the requisite data
2. Filter to the required entries
3. Actions | Create Metric
4. Pick a metric type (Counter or Distribution)
5. If Distribution, set configurations
6. Optional: Add labels

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

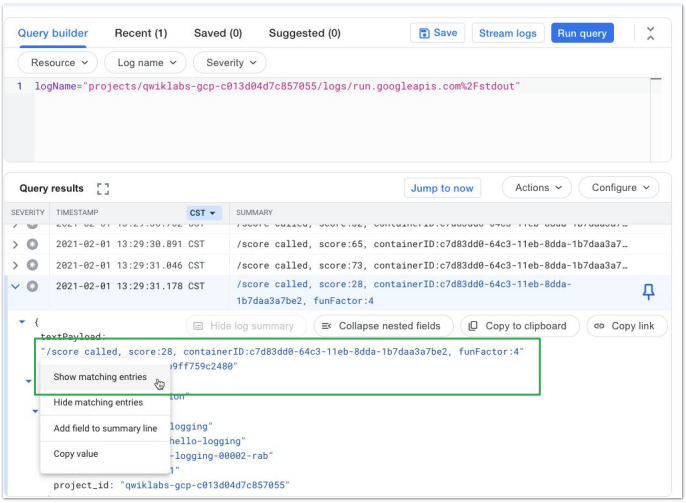
1. You start by finding the log with the requisite data.
2. Then you filter it to the required entries.
3. Create a metric.
4. Pick your metric type (Counter or Distribution).
5. If Distribution, then set configurations.
6. And finally, add labels as needed.

Filtering entries



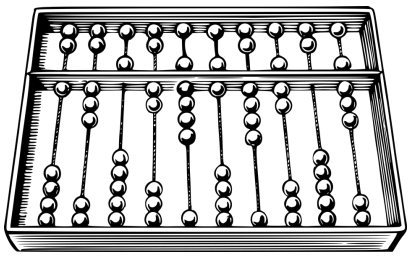
Use the Query builder to access project logs.

Filtering entries

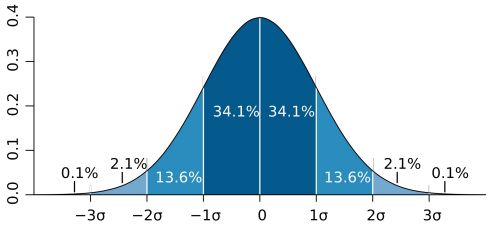


In the list of entries, we've located one of the "/score called" entries, and now we can filter to select those by clicking on "/score called" and selecting **Show matching entries**.

Logs-based metric types



Counter



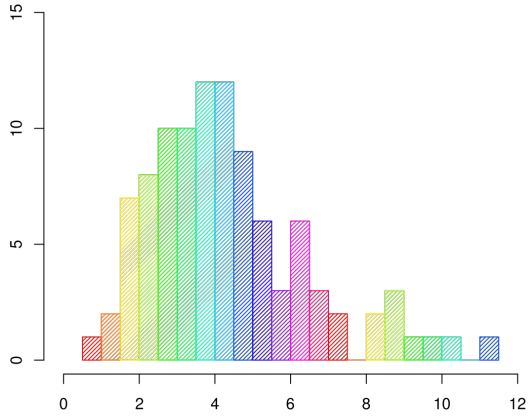
Distribution

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



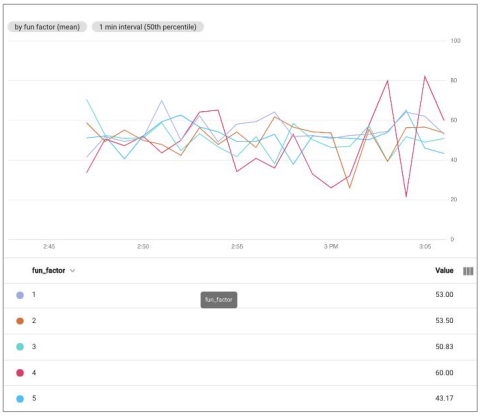
- Linear: buckets of fixed width
- Exponential:
  - N+2 buckets
  - Upper: scale * (growthFactor ^ i)
  - Lower:scale * (growthFactor ^ (i-1))
- Explicit: Array of bucket boundaries
- (Up to 200 buckets)

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.

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 logs-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).

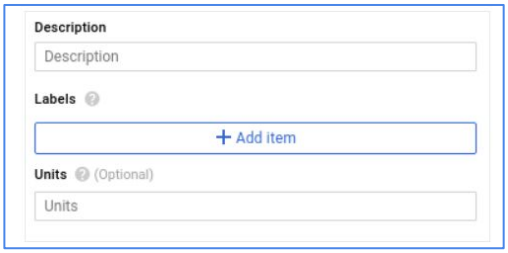
You can extract labels from the LogEntry built-in fields, such as httpRequest.status, or from one of the payload fields, textPayload, jsonPayload, or protoPayload.

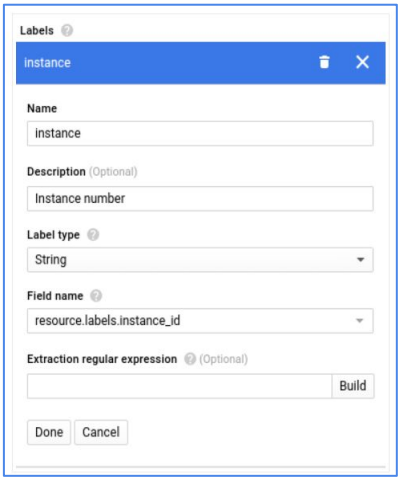
Label with care. A metric can support up to 10 user-defined labels, and once created, a metric cannot be removed. Also, each logs-based metric is limited to about 30,000 active time series.

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



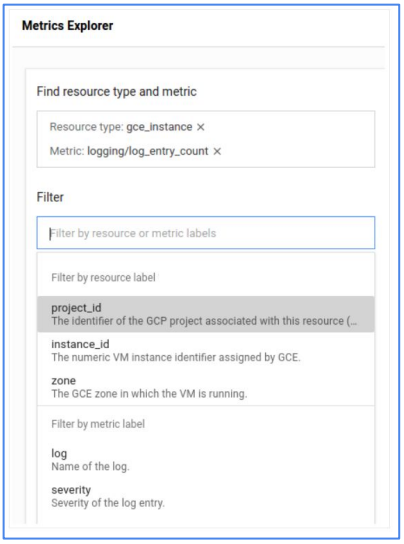


User-defined labels can be created when creating a log-based metric. The label form requires:

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 (regexp) that extracts the label value from the field value.

Using labels

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



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

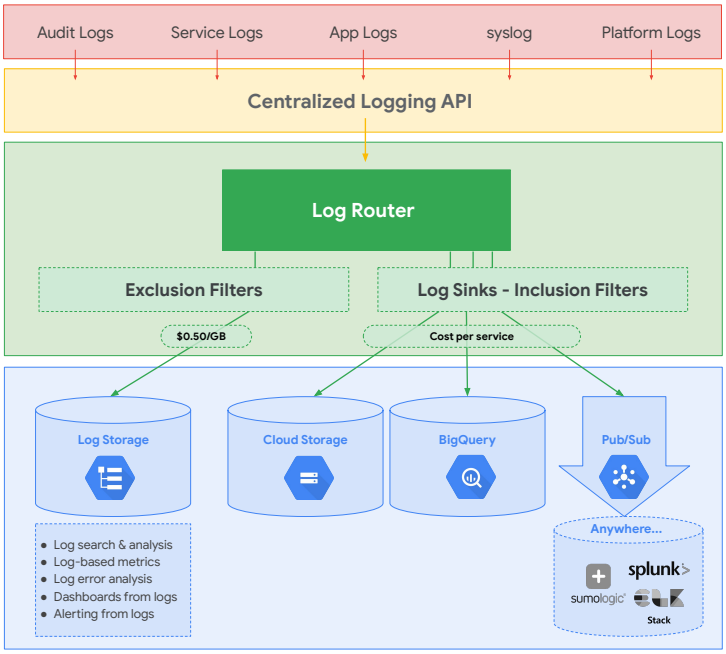
- Labeling
- Working with Logs Explorer
- Using Logs-Based Metrics

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.

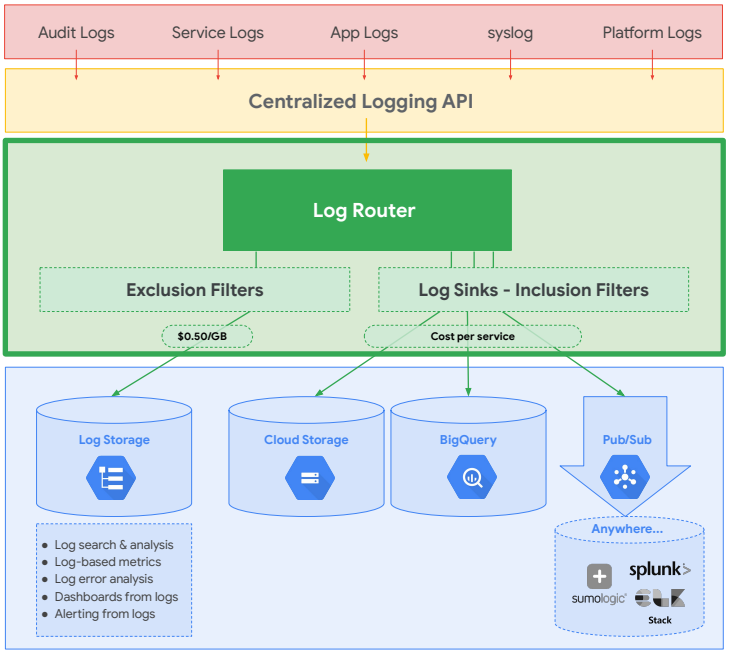
Logging Architecture



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.

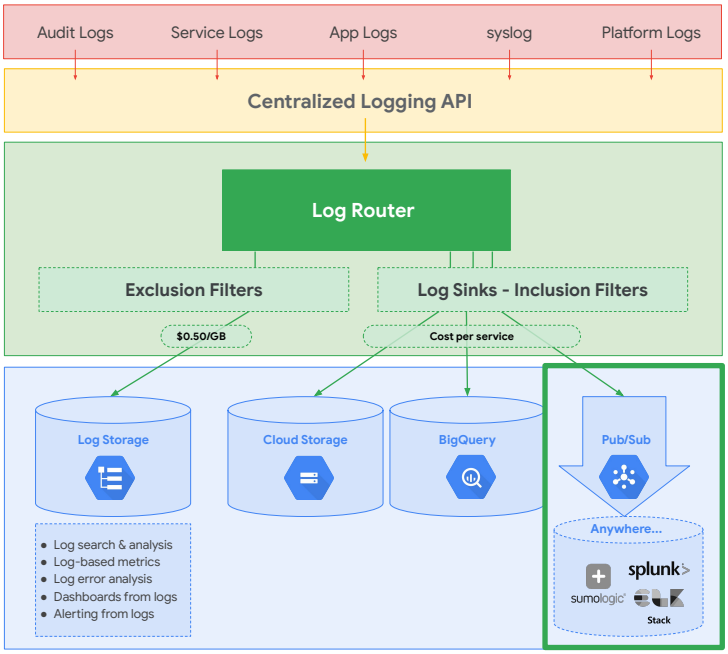
Logging Architecture



To give users a choice about where their logs go, Google created the Log Router.

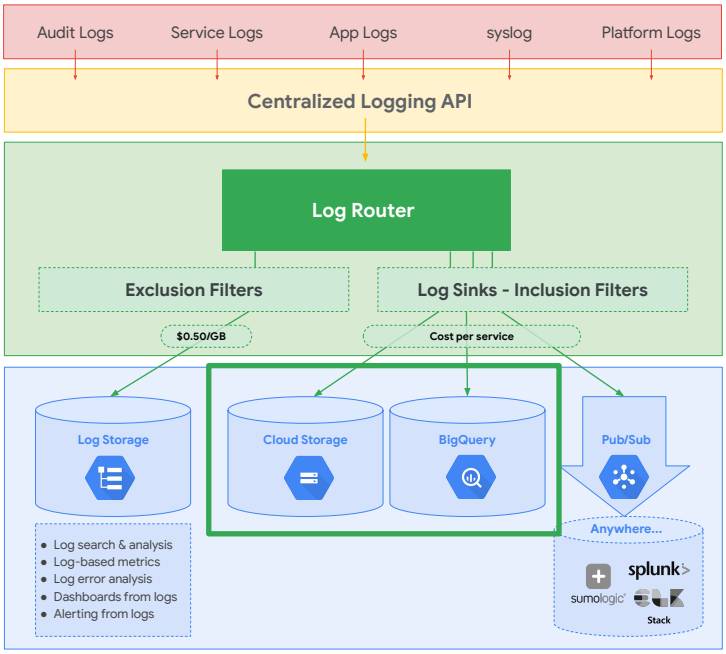
Log Router is optimized for processing streaming data, reliably buffering it, and sending it to any combination of BigQuery, Cloud Storage, Pub/Sub, and of course, Log management.

Logging Architecture



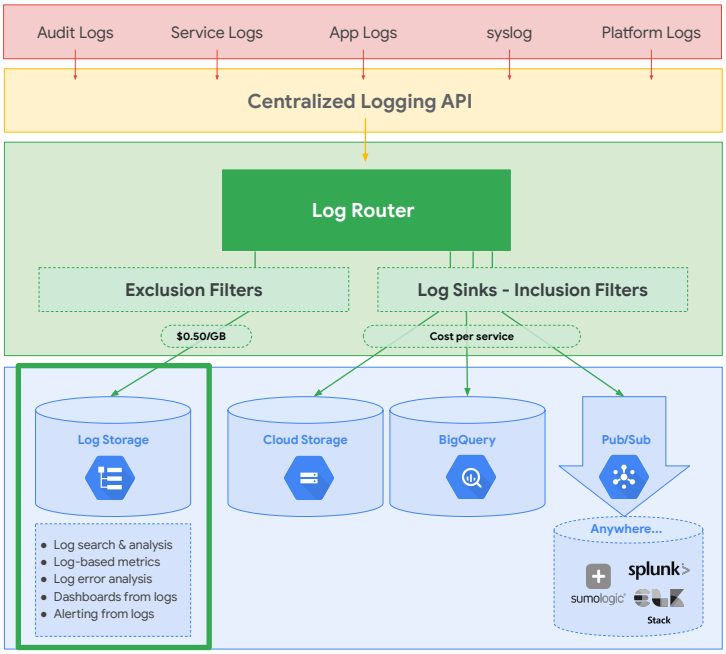
If users are already happily using Splunk, or some other external system or codebase, routing through Pub/Sub can easily make logging events available.

Logging Architecture



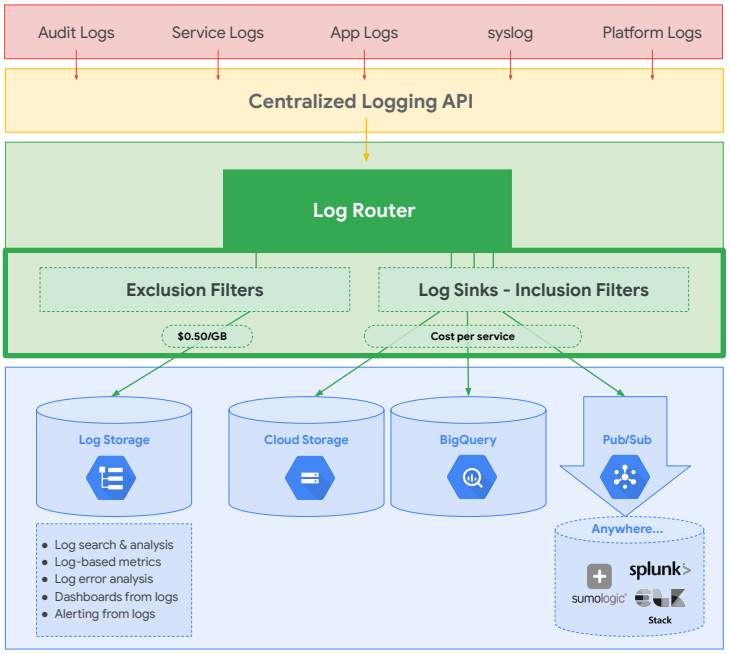
Log Router can also sink to BigQuery or Cloud Storage for long term storage, with BigQuery having the added benefit of being able to run massively scalable queries over those logs.

Logging Architecture



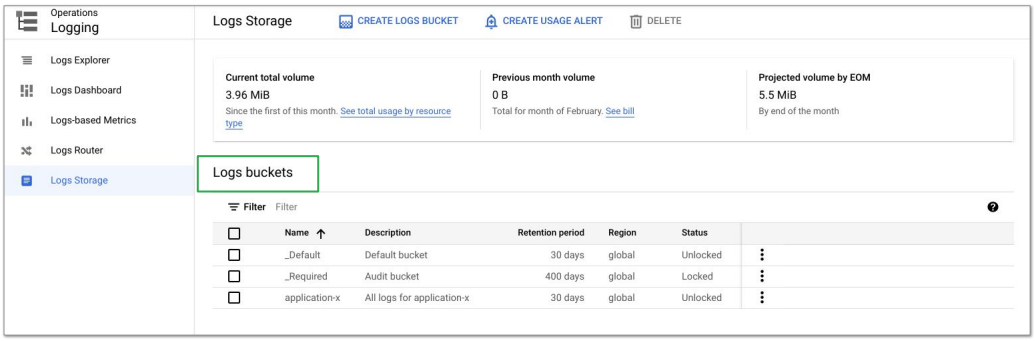
And finally, if users want to see their logs for operational or troubleshooting purposes, we have the cloud-based storage and interface abilities native to Google Cloud and covered earlier in this module.

Logging Architecture



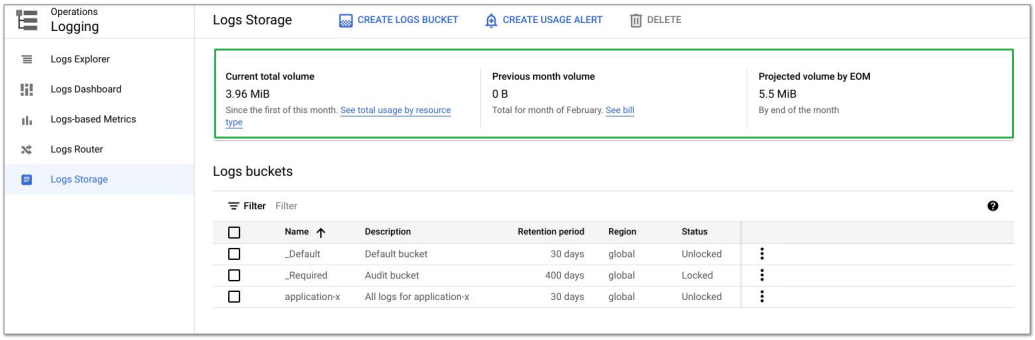
Inclusion and exclusion filters can control exactly which logging entries end up at a particular destination, and which are ignored completely.

Logs buckets



Logs buckets are containers in your Google Cloud projects that hold your logs data. You can create logs sinks to route all, or just a subset, of your logs to any logs bucket. This flexibility allows you to choose which Google Cloud project your logs are stored in and what other logs are stored with them. Log buckets may also be placed in specific regions for regulatory compliance. Using the gcloud command-line tool and the Google Cloud Console, you can create, update, and delete your custom logs buckets.

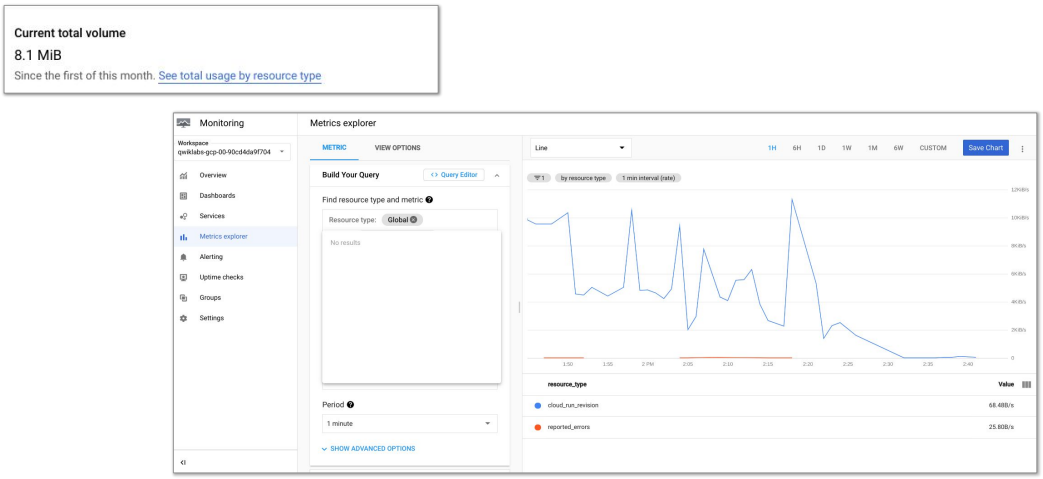
Resource usage



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

- **Current total volume**: The amount of logs your project has received since the first date of the current month.
- **Previous month volume**: The amount of logs your project received in the last calendar month.
- **Projected volume by EOM**: The estimated amount of logs your project will receive by the end of the current month, based on current usage.

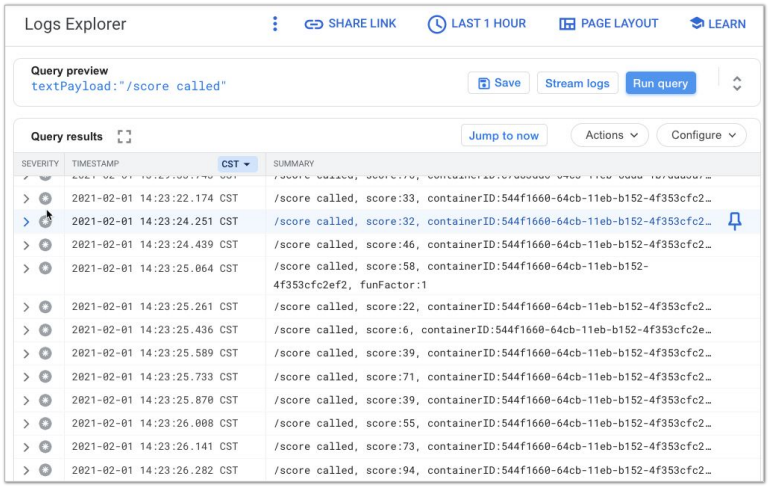
Resource usage



You can view the total usage by resource type for the current total volume. The link opens Metrics Explorer, which allows you to build charts for any metric collected by your project.

[Metrics Explorer: https://cloud.google.com/monitoring/charts/metrics-explorer]

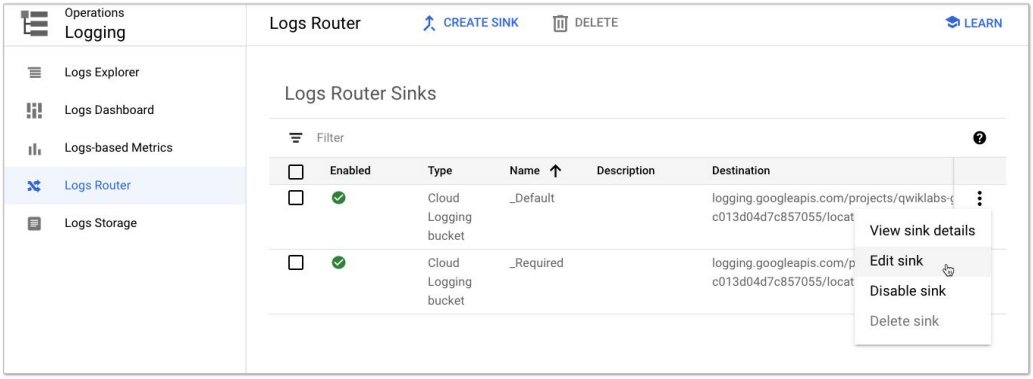
Exclusions: Identify log entries



Use **Logs Explorer** to build a query that selects the logs you want to exclude.

Save the query to use when building the exclusion.

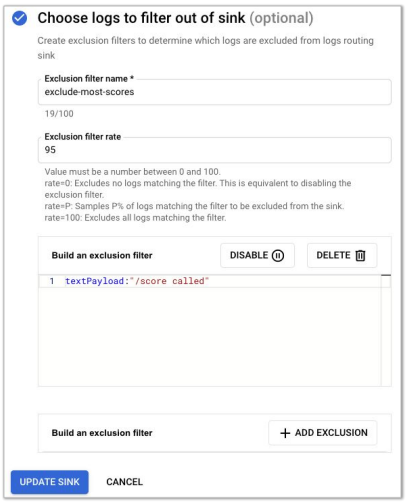
Exclusions: Edit the target log sink



Use the "hamburger menu" to the right of the target log sink to initiate editing of that entity

Take care here, because excluded log events will be lost forever.

Exclusions: Build the exclusion

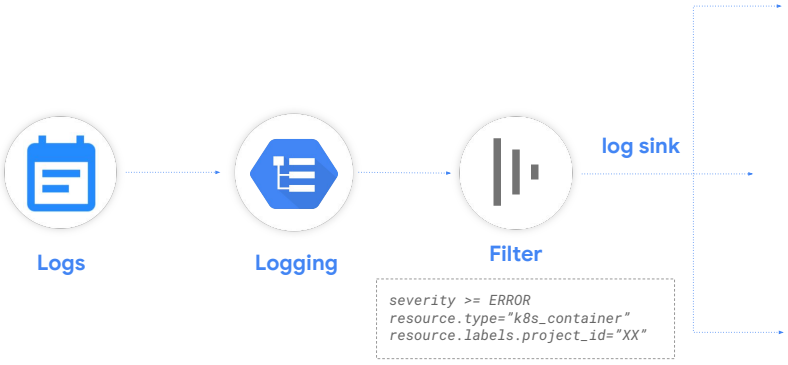


Use the **Log Explorer** query to create an exclusion filter that filters the unwanted entries out of the sink. Give the exclusion a name and description and decide the percentage of log entries to exclude.

It might be helpful to leave some representative events, depending on the exclusion.

Create the exclusion and it will go into effect immediately.

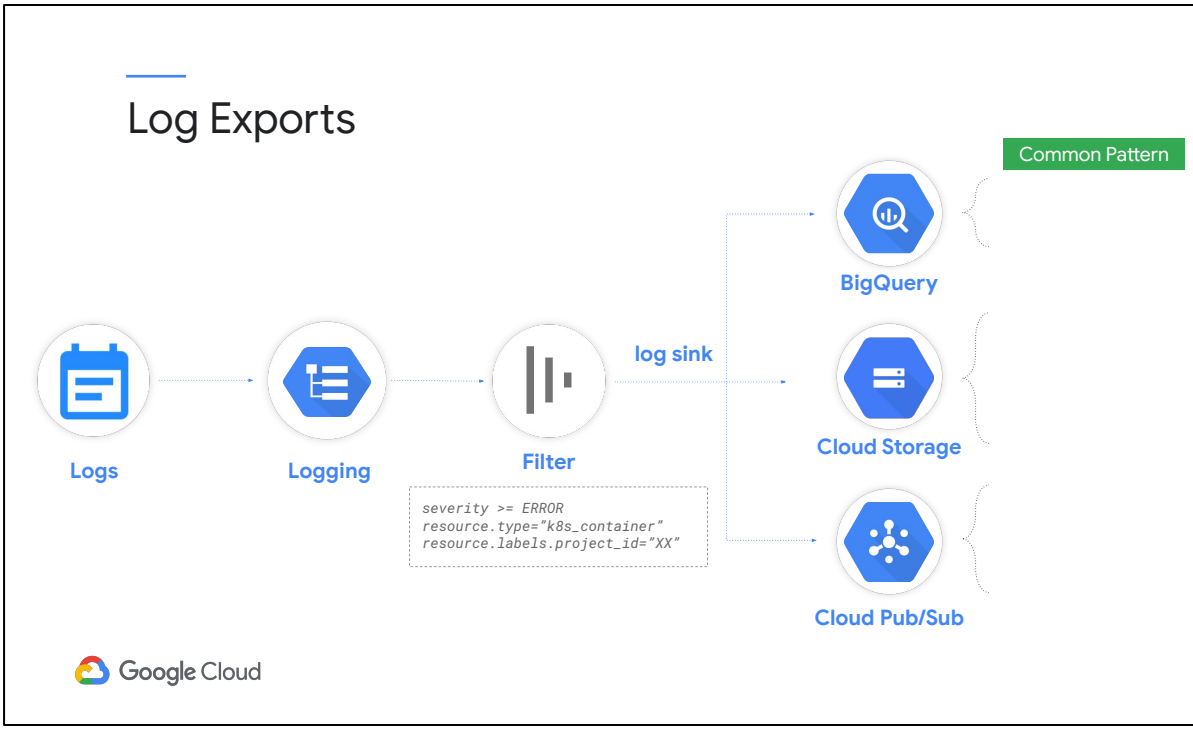
Log Exports



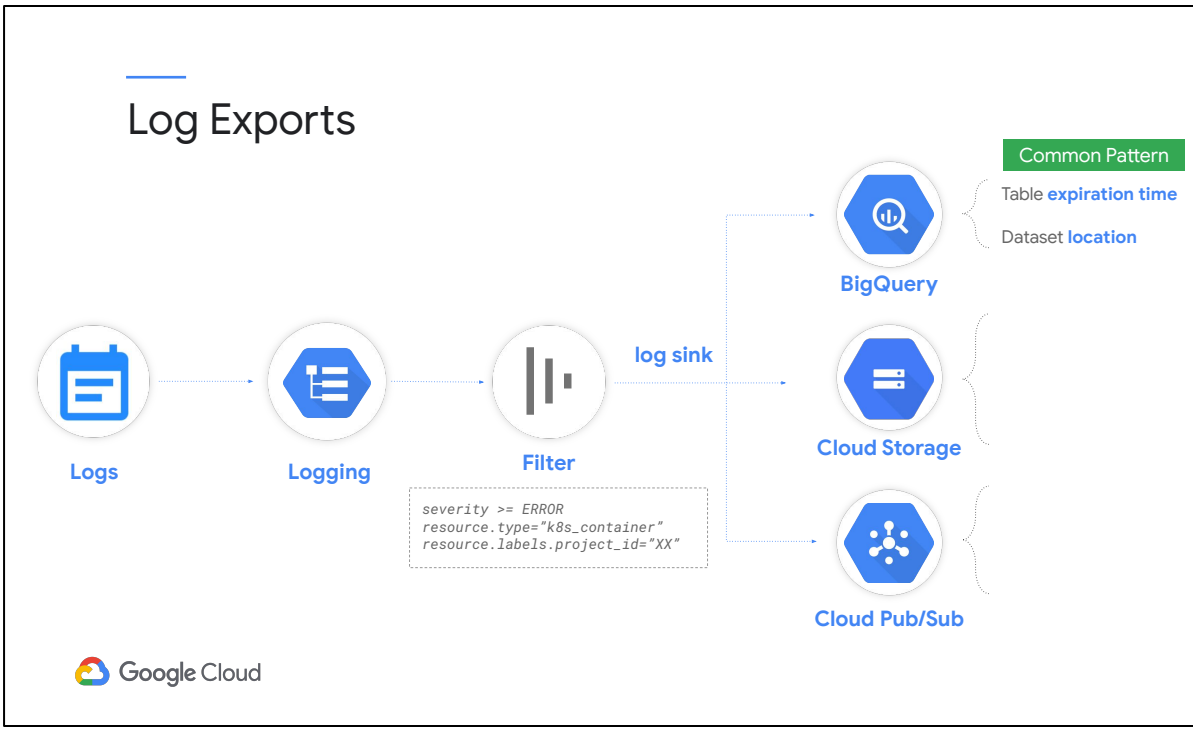
Google Cloud

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:

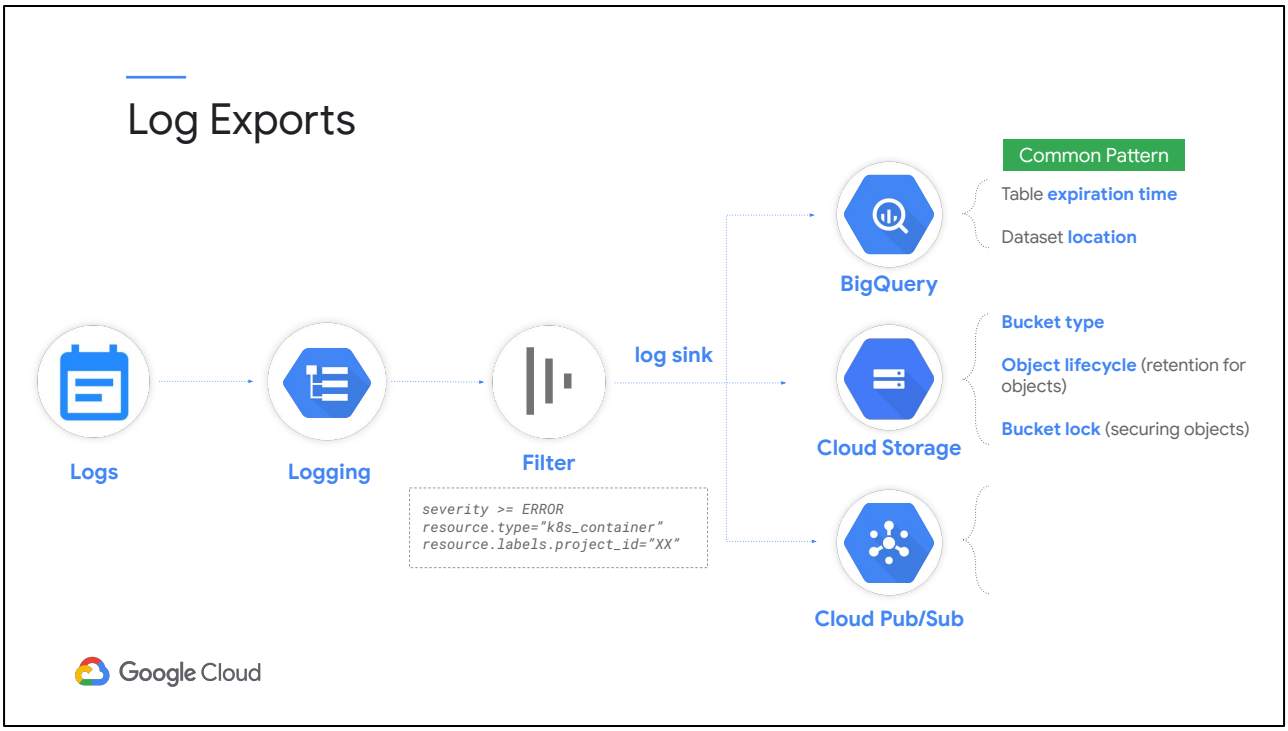
- Storing logs for extended periods.
- The use of big-data analysis tools on your logs, and
- Streaming your logs to other applications, other repositories, or to third parties.



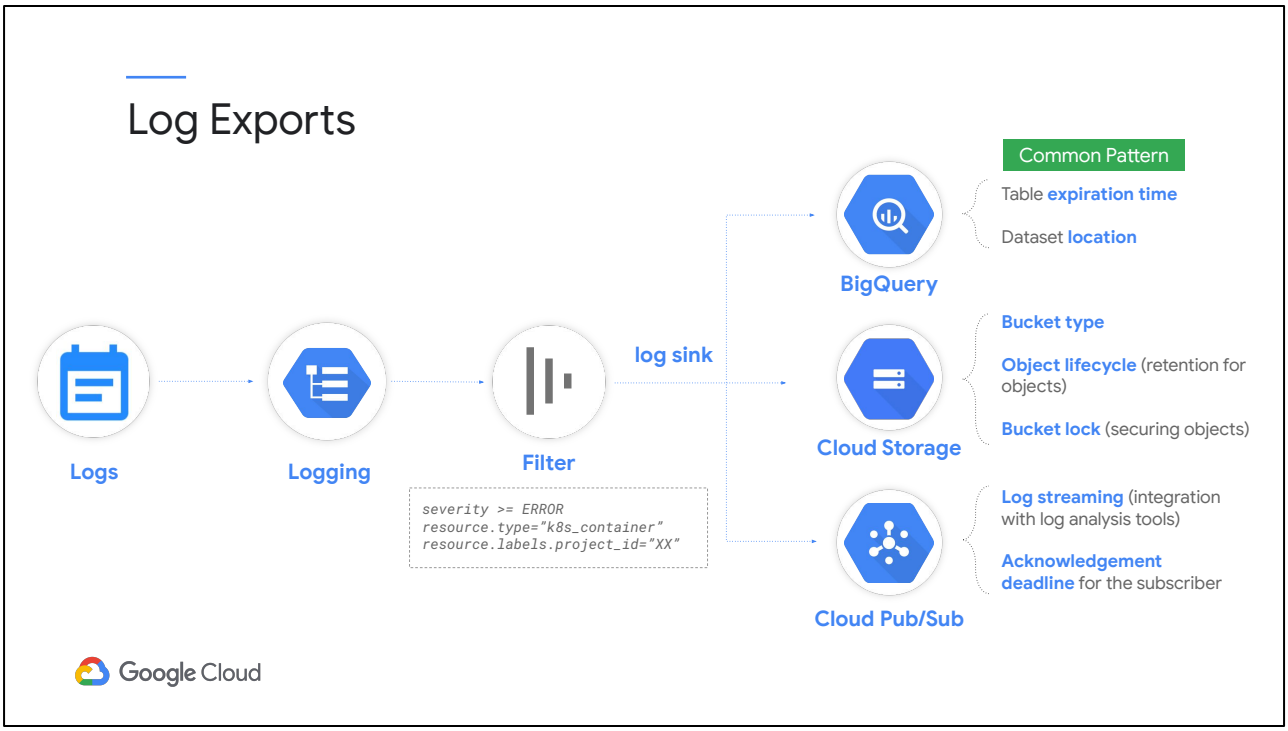
Log Sinks currently support three main targets, BigQuery, Cloud Storage, and messaging through Cloud Pub/Sub.



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



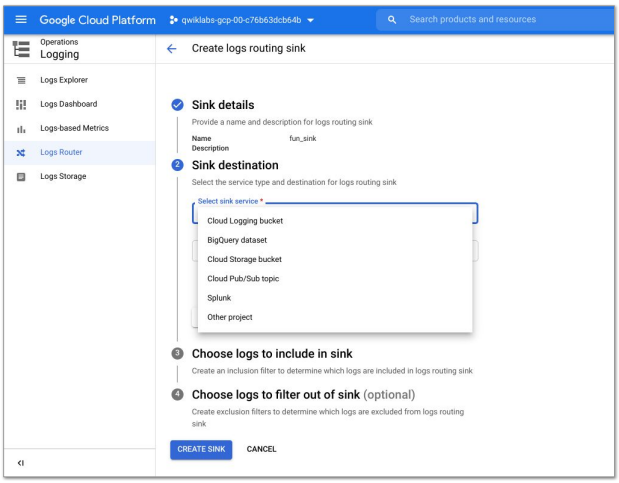
Cloud Storage supports inexpensive, unlimited retention periods, lifecycle control, and locks.



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



Google Cloud

The process for creating log sinks mimics that of creating log exclusions.

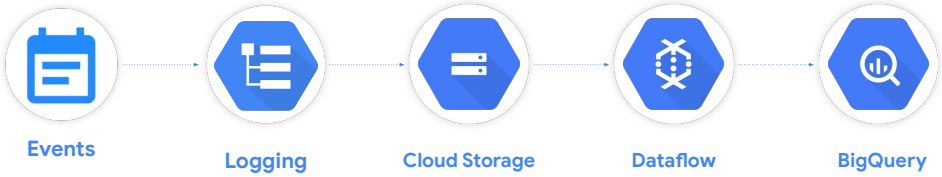
It involves writing a **query** that selects the log entries you want to export in the Logs Explorer, and choosing a **destination** of Cloud Storage, BigQuery, or Pub/Sub.

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



Google Cloud

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

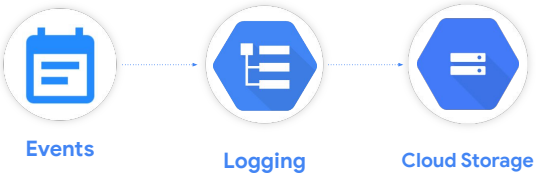
Here, for example, we are exporting through Pub/Sub, to Dataflow, to BigQuery. Dataflow is an excellent option if you're looking for real-time log processing at scale.

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



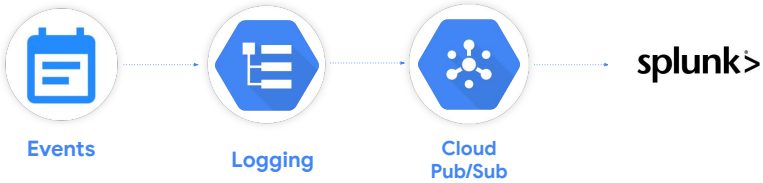
Google Cloud

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

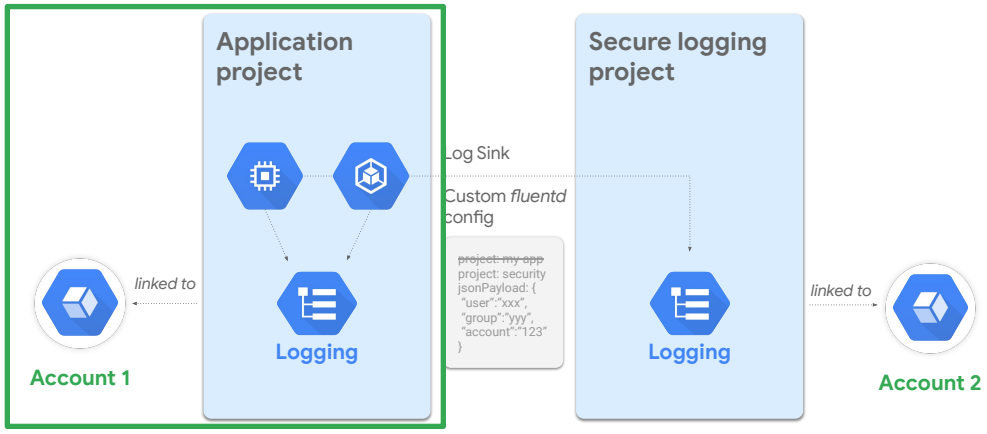


Google Cloud

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.

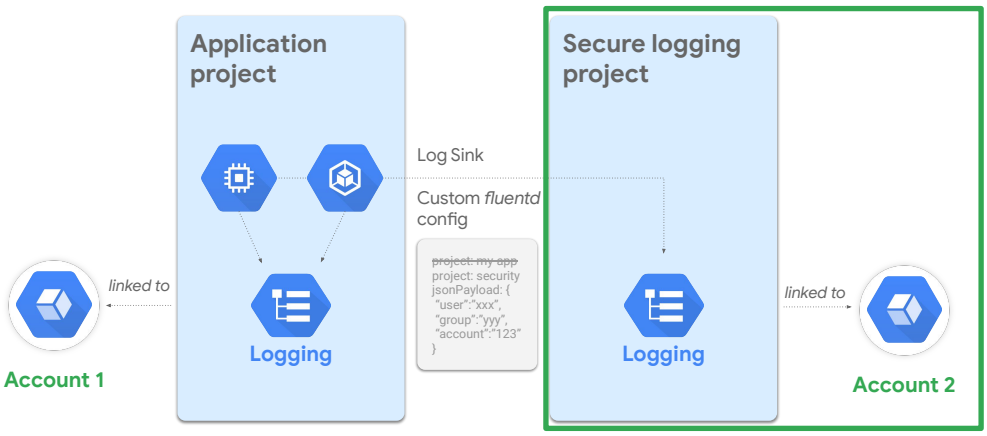
Security Logging



Google Cloud

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

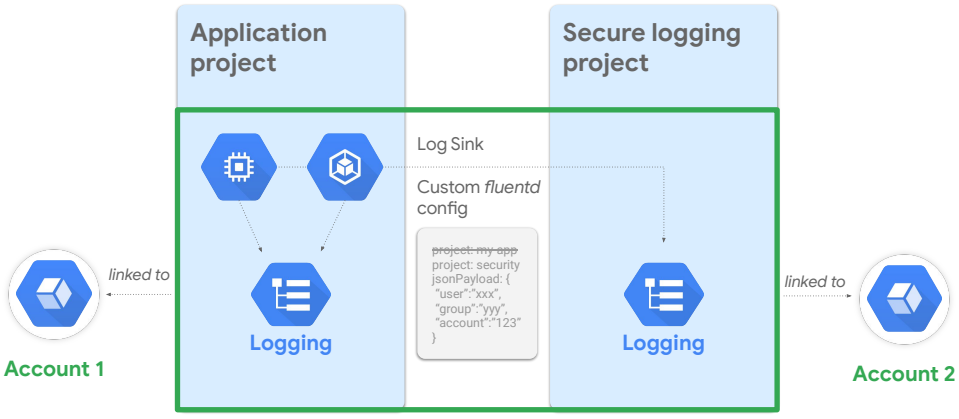
Security Logging



Google Cloud

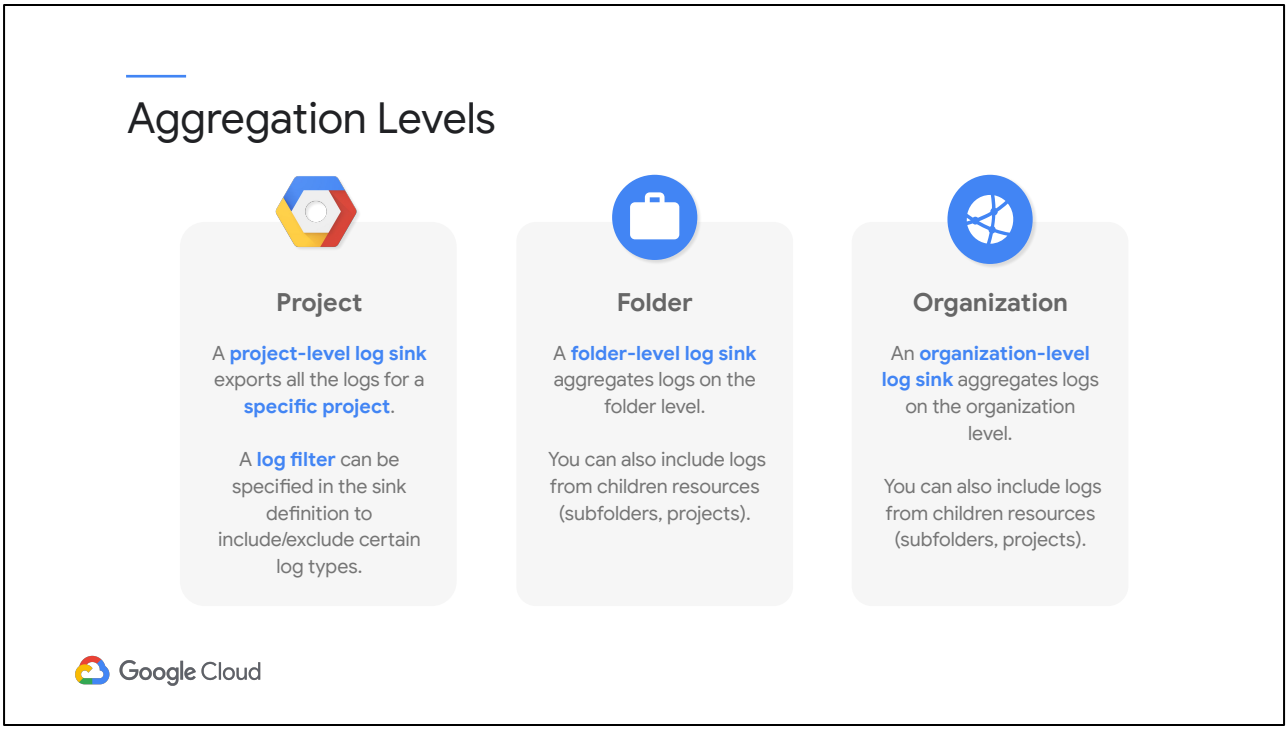
We have a separate secure logging project designed to provide access to Account 2 - but only to some of the logging data.

Security Logging



Google Cloud

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.



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.

1. The first is a project-level log sink, which we've discussed. This option 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.
2. A folder-level log sink aggregates logs on the folder level and can include logs from child resources (subfolders, projects).
3. And for a global view, an organization-level log sink can aggregate logs on the organization level and can also include logs from child 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.

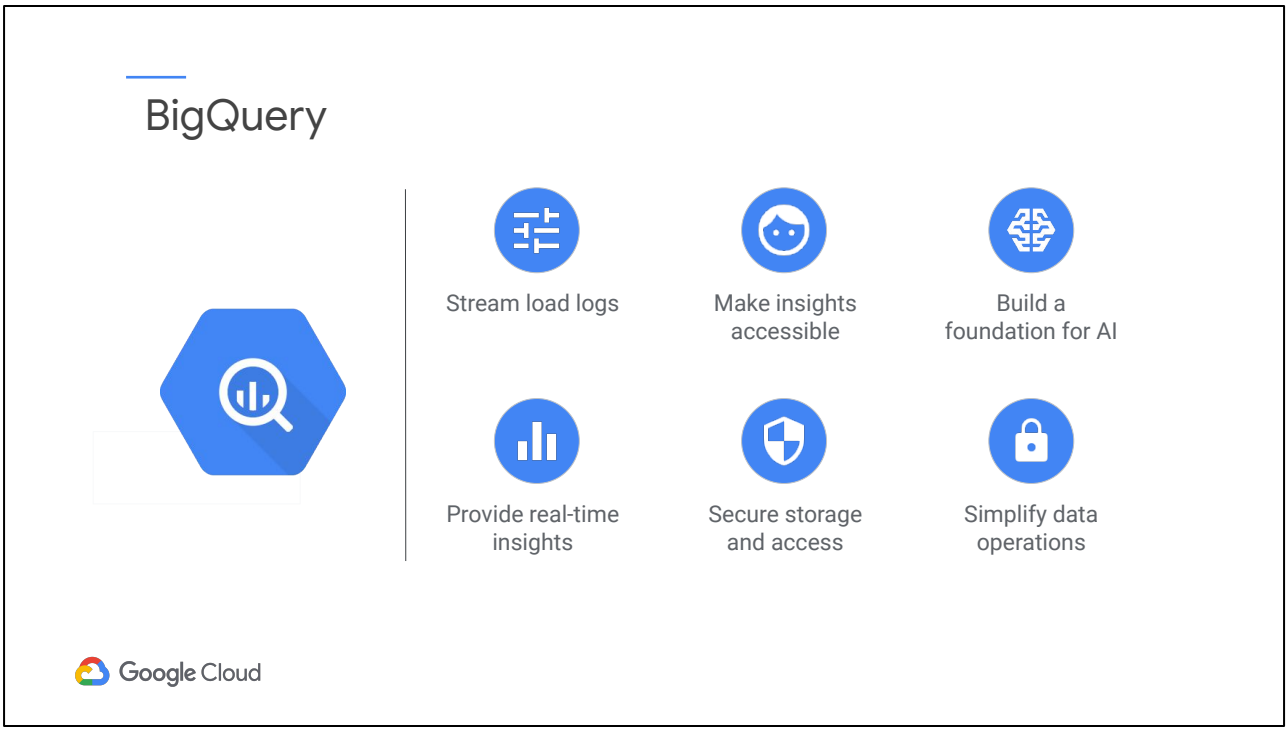
The destination for log sinks has to be created before the sink. Once again, the supported destinations are a Cloud Storage bucket, Pub/Sub topic, or BigQuery table.

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.



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.

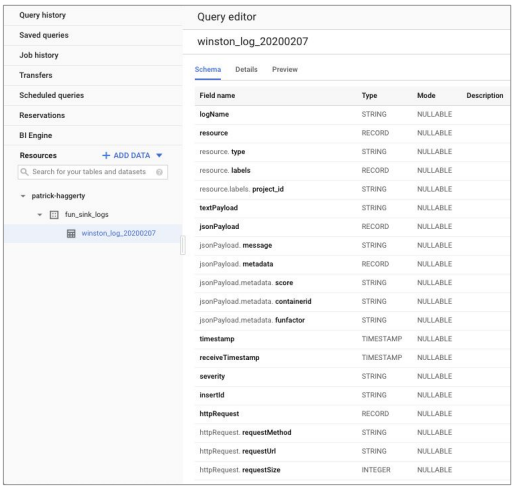
It supports stream loading logs for to support easy access to real-time insights, while serving as a good foundation for making those insights easily accessible.

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.

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

Table Schema Based on LogEntry



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;
```

Google Cloud

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
```

Google Cloud

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
  - Labeling
  - Working with Logs Explorer
  - Using Logs-Based Metrics
- 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 that occur in your running cloud services.

It provides a centralized error management interface and 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 opt to receive email and mobile alerts for new errors.

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

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, your 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 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, of you can use logging.

A side note: A log message from any language or environment using ReportedErrorEvent formatting will also 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.

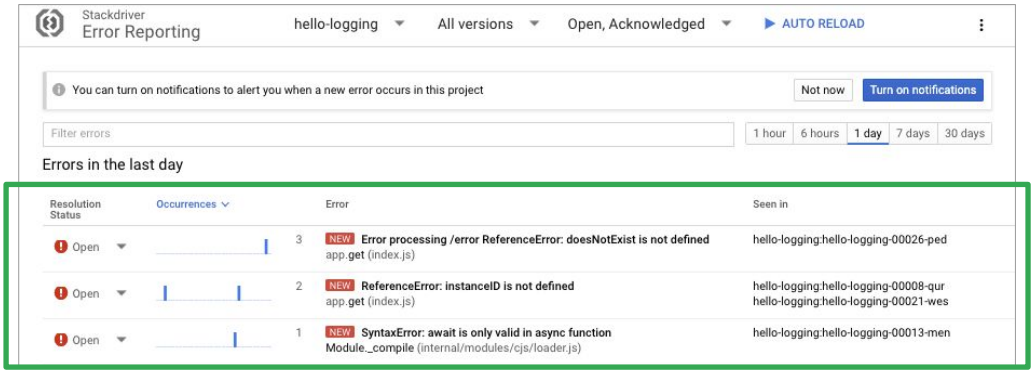
Manually Log and Report to Error Reporting

```
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)



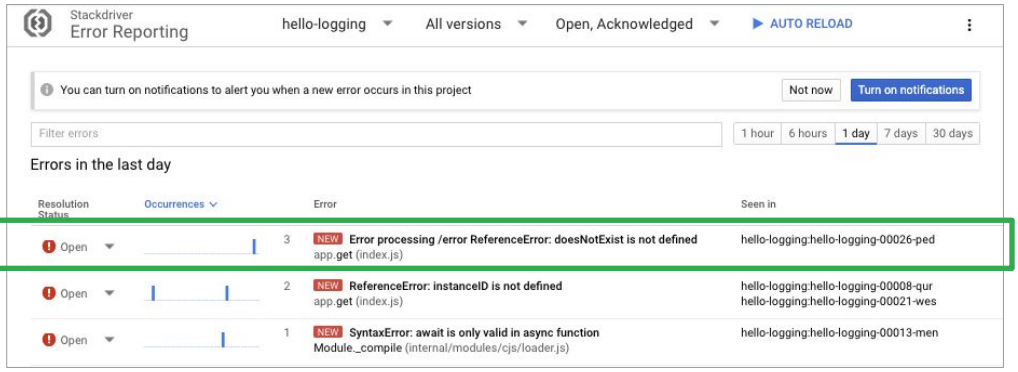
To see your errors, open the Error Reporting page in the Google Cloud Console.

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)

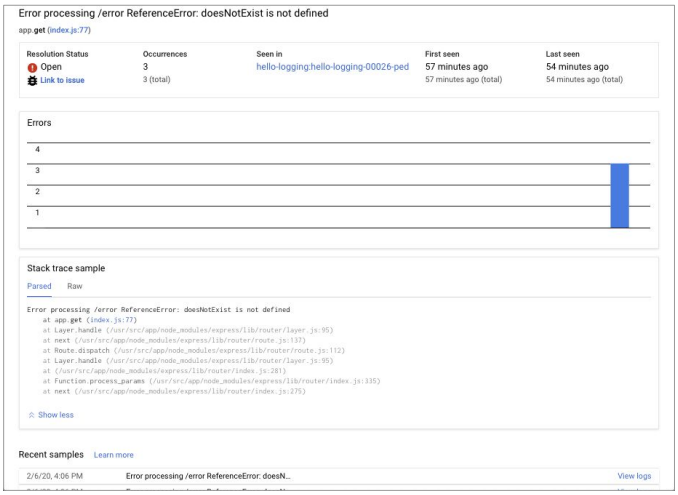


Here, if you look at the top error, you'll see that 3 errors have been generated by calls to ***/*error**. The error states that *doesNotExist* is not defined, and that the error came out of one of the hello-logging Cloud Run services.

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.

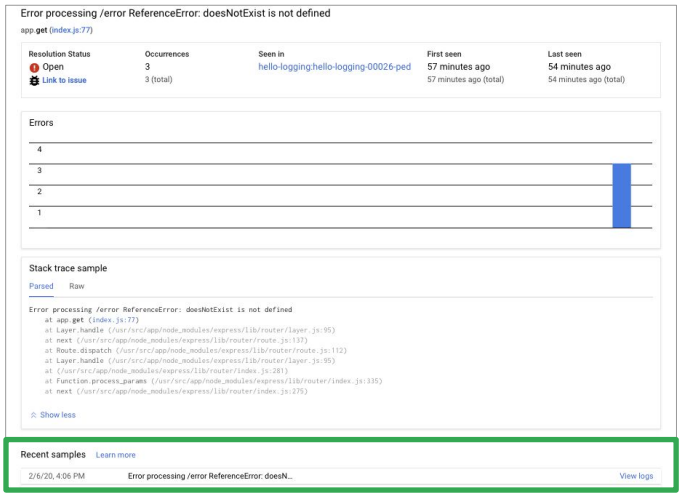
Select an Error for Details



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

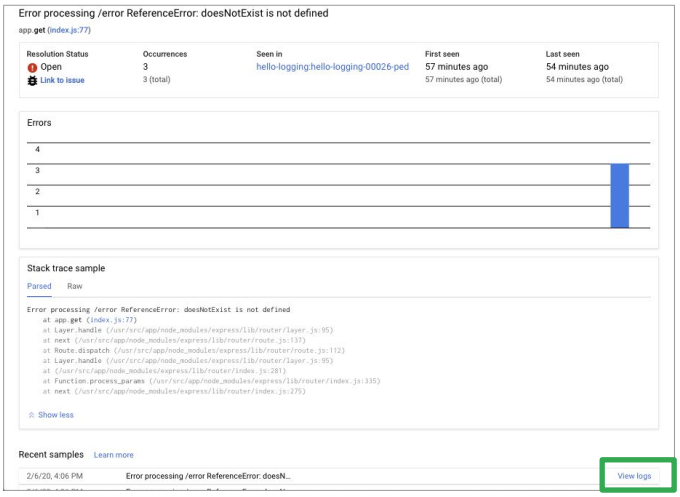
On this page, you can examine information about the error group, including the history of a specific error, specific error instances, and diagnostic information contained in samples for the error.

Select an Error for Details



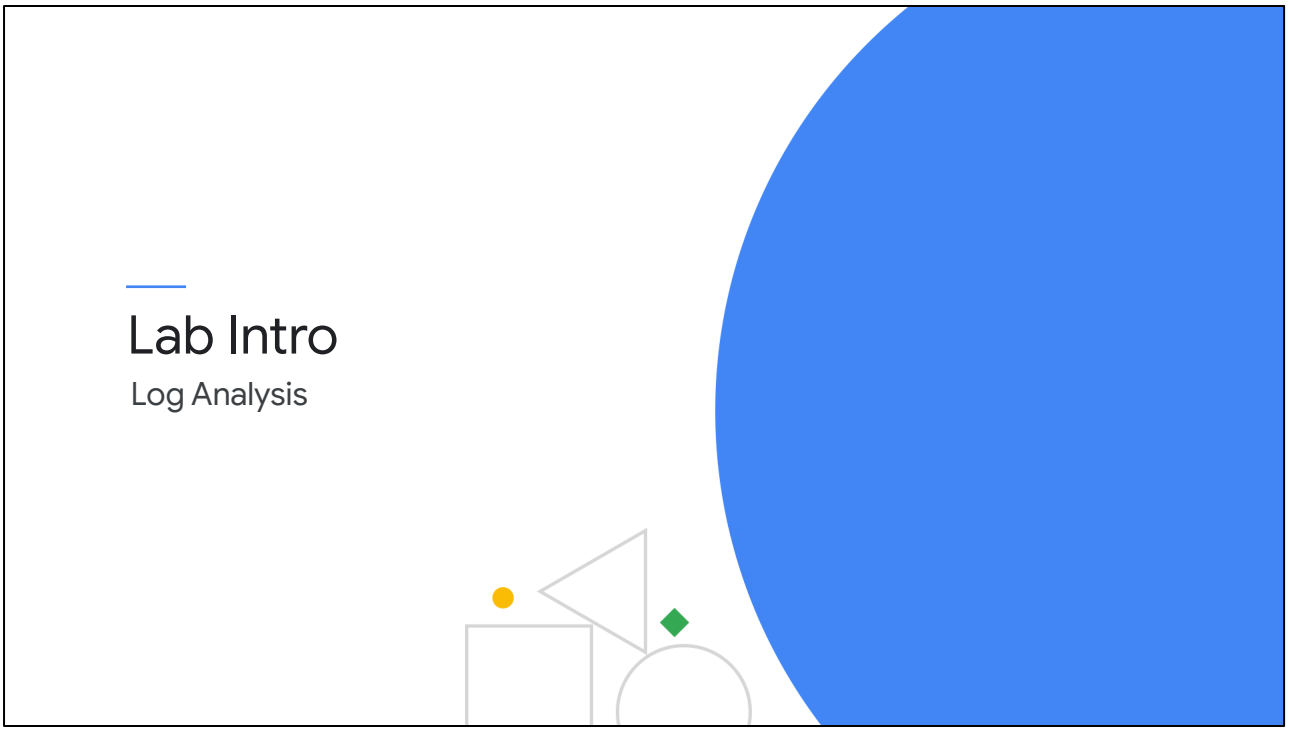
Sample errors are found in the **Recent samples** panel. Each sample represents one occurrence of the error and includes a parsed stack trace.

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.



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.