View logs routed to BigQuery

This document explains how you can find log entries that you routed from Cloud Logging to BigQuery tables (/bigquery/docs). Logging sinks stream logging data into BigQuery in small batches, which lets you query data without running a load job. To help you create queries and understand the format of your BigQuery table, this document also describes the BigQuery schema for routed logs (/logging/docs/export/bigquery#bigquery-schema).

Cloud Logging uses the legacy streaming API (/bigquery/docs/streaming-data-into-bigquery) to stream your log entries to BigQuery. Typically, log entries are visible in BigQuery within one minute. However, when a new table is created, it might take several minutes before the first log entries are available.

Note: Although you can stream your log entries to BigQuery, we recommend that you store your log entries in a log bucket, upgrade the bucket to use Log Analytics (/logging/docs/log-analytics#analytics), and then create a linked dataset. You can use BigQuery to query the linked dataset.

Before you begin

For a conceptual discussion of sinks, see Overview of routing and storage models: Sinks (/logging/docs/routing/overview#sinks).

For instructions about how to route your logs, see Route logs to supported destinations (/logging/docs/export/configure_export_v2).

To learn how the routed log entry fields are named, see BigQuery schema for routed logs (#bigquery-schema).

View logs

To view the logs routed to BigQuery, do the following:

1. In the Google Cloud console, go to the **BigQuery** page:

Go to **BigQuery Studio** (https://console.cloud.google.com/bigquery)

You can also find this page by using the search bar.

2. In the **Explorer** panel, expand your project and select a dataset.

   The log entries are visible on the **Details** tab, or you can query the table to return your data.

Sample queries

For information about BigQuery query syntax, see Query reference (/bigquery/docs/reference/legacy-sql). Especially useful are table wildcard functions (/bigquery/docs/querying-wildcard-tables), which let you query across multiple tables, and the flatten operator (/bigquery/query-reference#flatten), which lets you display data from repeated fields.

Sample Compute Engine query

The following BigQuery query retrieves log entries from multiple days and multiple log types:

- The query searches the last three days of the logs `syslog` and `apache-access`. The query was made on 23-Feb-2020 and it covers all log entries received on 21-Feb and 22-Feb, plus log entries received on 23-Feb up to the time the query was issued.
- The query retrieves results for a single Compute Engine instance, `1554300700000000000`.

```
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 are some example output rows:

```
Row | Time                    | Log                                          | Me
--- | ----------------------- | -------------------------------------------- | --
 5  | 2020-02-21 03:40:14 UTC | projects/project-id/logs/syslog              | Fe
 6  | 2020-02-21 04:17:01 UTC | projects/project-id/logs/syslog              | Fe
 7  | 2020-02-21 04:49:58 UTC | projects/project-id/logs/apache-access       | 12
 8  | 2020-02-21 05:17:01 UTC | projects/project-id/logs/syslog              | Fe
 9  | 2020-02-21 05:30:50 UTC | projects/project-id/log/syslogapache-access | 92
```

Sample App Engine query

The following BigQuery query retrieves unsuccessful App Engine requests from 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
```

Here are some of the results:

```
Row | Time                    | Host                                    | Status |
--- | ----------------------- | --------------------------------------- | ------ |
 6  | 2020-02-12 19:35:02 UTC | default.my-gcp-project-id.appspot.com |    404 |
 7  | 2020-02-12 19:35:21 UTC | default.my-gcp-project-id.appspot.com |    404 |
```

```
 8  | 2020-02-16 20:17:19 UTC | my-gcp-project-id.appspot.com          |    404 |
 9  | 2020-02-16 20:17:34 UTC | my-gcp-project-id.appspot.com          |    404 |
```

BigQuery schema for routed logs

BigQuery table schemas for routed logs are based on the structure of the `LogEntry` (/logging/docs/reference/v2/rest/v2/LogEntry) type and the contents of the log payloads. Cloud Logging also applies rules to shorten BigQuery schema field names for audit logs (/logging/docs/audit) and for certain structured payload fields. You can view the table schema by selecting a table with routed log entries in the BigQuery interface (/bigquery/docs/bigquery-web-ui).

Note: The BigQuery table schema used to represent complex log entry payloads can be confusing and, in the case of routed audit logs, some special naming rules are used. For more information, see the Audit logs fields (#audit-logs) section on this page.

Field naming conventions

There are a few naming conventions that apply to the log entry fields when sending logs to BigQuery:

- Log entry field names can't exceed 128 characters.
- Log entry field names can be composed only of alphanumeric characters. Any unsupported characters are removed from field names and replaced with underscore characters. For example, `jsonPayload.foo%%` would be transformed to `jsonPayload.foo__`.

  Log entry field names must begin with an alphanumeric character, even after transformation; any leading underscores are removed.
- For log entry fields that are part of the `LogEntry` (/logging/docs/reference/v2/rest/v2/LogEntry) type, the corresponding BigQuery field names are exactly the same as the log entry fields.

- For any user-supplied log entry fields, the corresponding BigQuery field names are normalized to the lowercase, but naming is otherwise preserved.
- For fields in structured payloads, as long as the `@type` specifier isn't present, the corresponding BigQuery field names are normalized to the lowercase, but naming is otherwise preserved.

  For information about structured payloads where the `@type` specifier is present, see Payload fields with `@type` (#type-specifier) on this page.

The following examples show how these naming conventions are applied:

Log entry field	LogEntry (/logging/docs/reference/v2/rest/v2/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`

Payload fields with `@type`

This section discusses special BigQuery schema field names for log entries whose payloads contain the specifier `@type`. This includes audit log entries routed to BigQuery.

Payloads in log entries can contain structured data. Any structured field can include an optional type specifier in the following format:

```
@type: type.googleapis.com/[TYPE]
```

Naming rules explain why an audit log entry's **protoPayload** field might be mapped to the BigQuery schema field **protopayload_auditlog**.

Naming rules for @type

Structured fields that have type specifiers are customarily given BigQuery field names that have a **[TYPE]** appended to their field name. The value of **[TYPE]** can be any string.

The naming rules for **@type** apply only to the top level of **jsonPayload** or **protoPayload**; nested fields are ignored. When treating top-level structured payload fields, Logging removes the prefix **type.googleapis.com**.

For example, the following table shows the mapping of the top-level structured payload fields to BigQuery field names:

Payload	Payload @type	Payload field	BigQuery field name
jsonPayload	(none)	statusCode	jsonPayload.statusCode
jsonPayload	type.googleapis.com/abc.Xyz	statusCode	jsonpayload_abc_xyz.statuscode
protoPayload	(none)	statusCode	protoPayload.statuscode
protoPayload	type.googleapis.com/abc.Xyz	statusCode	protopayload_abc_xyz.statuscode

A few exceptions apply to the preceding rules for fields with type specifiers:

- In App Engine request logs, the payload's name in logs routed to BigQuery is **protoPayload**, even though the payload includes a type specifier.
- Cloud Logging applies some special rules to shorten BigQuery schema field names for audit logs. This is discussed in the Audit logs fields (#audit-logs) section on this page.

Example

This example shows how structured payload fields are named and used when received by BigQuery.

Assume that a log entry's payload is structured like the following:

```
jsonPayload: {
  @type: "type.googleapis.com/google.cloud.v1.CustomType"
    name_a: {
      sub_a: "A value"
    }
    name_b: {
      sub_b: 22
    }
  }
```

The mapping to BigQuery fields is as follows:

- The top-level structured field **`jsonPayload`** contains a **`@type`** specifier. Its BigQuery name is **`jsonpayload_v1_customtype`**.
- The nested fields are treated with the standard BigQuery naming rules (#field-naming), as type-specifier rules don't apply to nested fields.

Thus, the following BigQuery names are defined for the log entry's payload:

```
jsonpayload_v1_customtype
jsonpayload_v1_customtype._type
jsonpayload_v1_customtype.name_b
jsonpayload_v1_customtype.name_b.sub_b
jsonpayload_v1_customtype.name_a
jsonpayload_v1_customtype.name_a.sub_a
```

Audit logs fields

If you aren't working with audit logs that have been routed to BigQuery, then you can skip this section.

The audit log payload fields `protoPayload.request`, `protoPayload.response`, and `protoPayload.metadata` have `@type` specifiers but are treated as JSON data. That is, their BigQuery schema names are their field names with `Json` appended to them, and they contain string data in JSON format.

The two sets of audit log payload field names are listed in the following table:

Log entry field	BigQuery field name
`protoPayload`	`protopayload_auditlog`
`protopayload.metadata`	`protopayload_auditlog.metadataJson`
`protoPayload.serviceData`	`protopayload_auditlog.servicedata_v1_bigquery` Example: `protopayload_auditlog.servicedata_v1_bigquery.tableInsertRequest`
`protoPayload.request`	`protopayload_auditlog.requestJson`
`protoPayload.response`	`protopayload_auditlog.responseJson`

Note that the `serviceData` naming convention is specific to audit logs that are generated by BigQuery and that are then routed from Cloud Logging to BigQuery. Those audit log entries contain a `serviceData` field that has a @type specifier of `type.googleapis.com/google.cloud.bigquery.logging.v1.auditdata`.

Example

An audit log entry generated by BigQuery has a field with the following name:

```
protoPayload.serviceData.tableInsertRequest
```

If this log entry were then routed to BigQuery, how would the `tableInsertRequest` field be referenced? Before the name shortening, the corresponding field name in BigQuery would be:

```
protopayload_google_cloud_audit_auditlog.servicedata_google_cloud_bigquery_loggi
```

After the name shortening, the same field is referenced in BigQuery tables like this:

```
protopayload_auditlog.servicedata_v1_bigquery.tableInsertRequest
```

Table organization

This section provides an overview of partitioned tables (/bigquery/docs/partitioned-tables) for logs that are routed to BigQuery.

When you route logs to a BigQuery dataset, Logging creates tables to hold the log entries. The first log entry received by BigQuery determines the schema for the destination BigQuery table. BigQuery creates a table whose columns are based on the first log entry's fields and their types. Subsequent log entries might cause a schema mismatch. For information about when these occur and how they are handled, see Mismatches in schema (#mismatch).

There are two table types by which Logging organizes the data it routes: **date-sharded tables** and **partitioned tables**. Both table types partition the logs data based on log entries' `timestamp` fields. However, there are two key differences between the table types as follows:

- Performance: A partitioned table divides a large table into smaller partitions, so that you can improve query performance and, thus, better control your BigQuery costs by reducing the number of bytes read by a query.
- Table nomenclature: The table types use different naming conventions, as discussed in the section below.

Table organization

Log entries are sharded into BigQuery tables whose organization and names are based on the entries' log names and timestamps.

The table names are suffixed with the calendar date of log entry's UTC timestamp, using the ISO 8601 basic format (YYYYMMDD).

The following table shows examples of how log names and sample timestamps are mapped to table names in BigQuery:

Log name	Log entry `timestamp`[1]	BigQuery table name (date-sharded)	BigQuery table name (partitioned)
`syslog`	`2017-05-23T18:19:22.135Z`	`syslog_20170523`	`syslog`
`apache-access`	`2017-01-01T00:00:00.000Z`	`apache_access_20170101`	`apache_access`
`compute.googleapis.com/activity_log`	`2017-12-31T23:59:59.999Z`	`compute_googleapis_com_activity_log_20171231`	`compute_googleapis_com_activity_log`

[1] The log entry timestamps are expressed in UTC (Coordinated Universal Time).

Creating partitioned tables

When creating a sink to route your logs to BigQuery, you can use either date-sharded tables or partitioned tables. The default selection is a date-sharded table:

For instructions about how to create sinks, see the following resources:

- Google Cloud console: Route logs to supported destinations (/logging/docs/export/configure_export_v2).
- Google Cloud CLI: **`gcloud logging sinks create`** (/sdk/gcloud/reference/logging/sinks/create).

Mismatches in schema

The first log entry received by BigQuery determines the schema for the destination BigQuery table. BigQuery creates a table whose columns are based on the first log entry's fields and their types.

A schema mismatch occurs when log entries are written to the destination table and either of the following errors occurs:

- A later log entry changes the field type for an existing field in the table.

  For example, if the initial log entry's `jsonPayload.user_id` field is a `string`, then that log entry generates a table with a string type for that field. If you later start logging `jsonPayload.user_id` as an `array`, then that causes a schema mismatch.

- A new log entry contains a field that isn't in the current schema and inserting that field into the destination table would exceed the BigQuery column limit (/bigquery/quotas).

  The destination table can accept the new field if it doesn't cause the column limit to be exceeded.

When BigQuery identifies a schema mismatch, it creates a table within the corresponding dataset to store the error information. A table's type determines the table name. For date-sharded tables, the naming format is `export_errors_YYYYMMDD`. For partitioned tables, the naming format is `export_errors`. For more information, see Table organization (/logging/docs/export/bigquery#partition-table-organization).

When routing log entries, Logging sends messages as a batch to BigQuery. BigQuery uses the following rules to determine to which table the log entries in the current batch of messages are written:

- When a field type change occurs, only those log entries that caused a schema mismatch are written to the error table. Log entries in the current batch of messages that don't cause a schema mismatch are written to the original destination table.
- When the column limit is exceeded, all log entries in the current batch of messages are written to the error table.

★ **Note:** BigQuery supports loading nested data (/bigquery/docs/nested-repeated) into its tables. However, when loading data from Logging, the maximum nested depth limit for a column is 13 levels.

Error table schema

The error table contains data from the `LogEntry` (/logging/docs/reference/v2/rest/v2/LogEntry) and information about the mismatch:

- `logEntry`: Contains the complete log entry; however, the log entry is converted from JSON into a string.

- `schemaErrorDetail`: Contains the complete error message returned by BigQuery.
- `sink`: Contains the full resource path for the log sink.
- `logName`: Extracted from the `LogEntry`.
- `timestamp`: Extracted from the `LogEntry`.
- `receiveTimestamp`: Extracted from the `LogEntry`.
- `severity`: Extracted from the `LogEntry`.
- `insertId`: Extracted from the `LogEntry`.
- `trace`: Extracted from the `LogEntry`.
- `resourceType`: Extracted from the `LogEntry`.

Logging communicates schema mismatches to the Google Cloud project that contains the routing sink in the following ways:

- Project Owners (/logging/docs/access-control) receive an email. Details include: Google Cloud project ID, sink name, and destination.
- The Google Cloud console Activity page displays an error, `Stackdriver Config error`. Details include the sink name and destination, and a link to an example of a log entry that caused the error.

Prevent future field-type mismatches

To correct field-type mismatches for later log entries, fix the field type so that it matches the current schema. For information about how to fix a field type, see Change a column's data type (/bigquery/docs/managing-table-schemas#change_a_columns_data_type).

Sometimes the field type can't be changed, for example, you can't change a field type for logs that are automatically generated by Google Cloud services. To prevent schema mismatches when you can't change a field type, rename the table or change the sink's parameters, so that Logging recreates the table in a different dataset. For instructions, see Manage sinks (/logging/docs/export/configure_export_v2#managing_sinks).

Troubleshooting

If logs seem to be missing from your sink's destination or you otherwise suspect that your sink isn't properly routing logs, then see Troubleshoot routing logs (/logging/docs/export/troubleshoot).

Pricing

Cloud Logging doesn't charge to route logs to a supported destination; however, the destination might apply charges. With the exception of the `_Required` log bucket, Cloud Logging charges to stream logs into log buckets and for storage longer than the default retention period of the log bucket.

Cloud Logging doesn't charge for copying logs, for creating log scopes (/logging/docs/log-scope/create-and-manage) or analytics views (/logging/docs/analyze/about-analytics-views), or for queries issued through the **Logs Explorer** or **Log Analytics** pages.

For more information, see the following documents:

- Cloud Logging pricing summary (/stackdriver/pricing#logs-pricing-summary)
- Destination costs:
  - Cloud Storage pricing (/storage/pricing)
  - BigQuery pricing (/bigquery/pricing#data_ingestion_pricing)
  - Pub/Sub pricing (/pubsub/pricing)
  - Cloud Logging pricing (/stackdriver/pricing#logging-cost)
- VPC flow log generation charges (/vpc/network-pricing#network-telemetry) apply when you send and then exclude your Virtual Private Cloud flow logs from Cloud Logging.

Last updated 2025-04-30 UTC.