Solutions (https://cloud.google.com/solutions/) Solutions

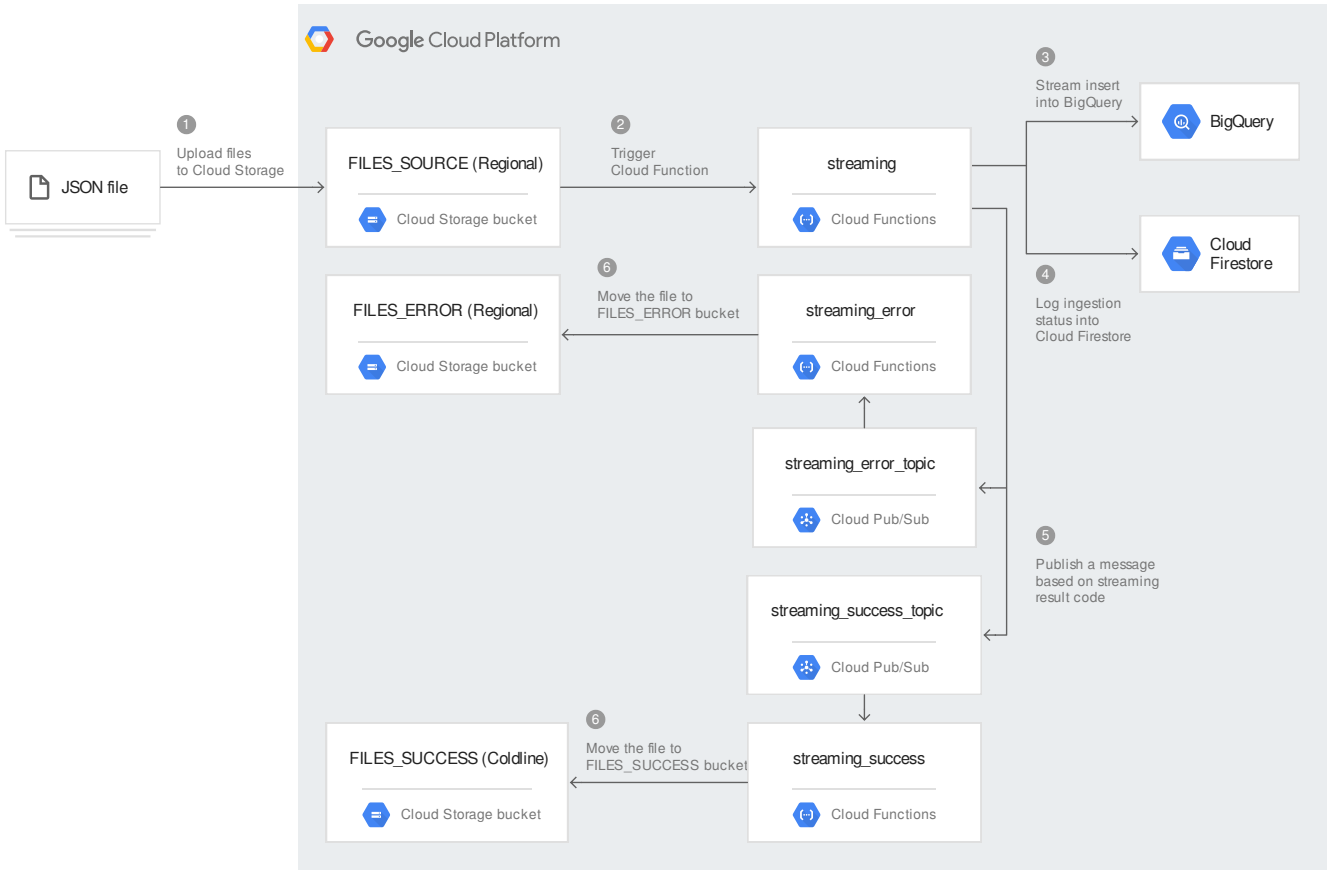
Streaming data from Cloud Storage into BigQuery using Cloud Functions

This tutorial demonstrates how to stream new objects from a Cloud Storage (https://cloud.google.com/products/storage/) bucket into BigQuery (https://cloud.google.com/bigquery/) by using Cloud Functions (https://cloud.google.com/functions/). Cloud Functions is a Google Cloud event-driven, serverless compute platform, which provides automatic scaling, high availability, and fault tolerance with no servers to provision, manage, update, or patch. Stream data through Cloud Functions to let you connect and extend other Google Cloud services while paying only when your app is running.

This article is for data analysts, developers, or operators, who need to run near real-time analysis on files added to Cloud Storage. The article assumes you are familiar with Linux, Cloud Storage, and BigQuery.

Architecture

The following architecture diagram illustrates all components and the entire flow of this tutorial's streaming pipeline. Although this pipeline expects you to upload JSON files into Cloud Storage, minor code changes are required to support other file formats. The ingestion of other file formats isn't covered in this article.



In the preceding diagram, the pipeline consists of the following steps:

1. JSON files are uploaded to the `FILES_SOURCE` Cloud Storage bucket.
2. This event triggers the `streaming` Cloud Function.
3. Data is parsed and inserted into BigQuery.
4. The ingestion status is logged into Firestore (https://cloud.google.com/firestore/) and Stackdriver Logging (https://cloud.google.com/logging/).
5. A message is published in one of the following Pub/Sub (https://cloud.google.com/pubsub/) topics:
   - `streaming_success_topic`
   - `streaming_error_topic`
6. Depending on the results, Cloud Functions moves the JSON file from the `FILES_SOURCE` bucket to one of the following buckets:
   - `FILES_ERROR`
   - `FILES_SUCCESS`

Objectives

- Create a Cloud Storage bucket to store your JSON files.
- Create a BigQuery dataset and table to stream your data in to.
- Configure a Cloud Function to trigger whenever files are added to your bucket.
- Set up Pub/Sub topics.
- Configure additional functions to handle function output.
- Test your streaming pipeline.
- Configure Stackdriver Monitoring to alert on any unexpected behaviours.

Costs

This tutorial uses the following billable components of Google Cloud:

- Cloud Storage
- Cloud Functions
- Firestore
- BigQuery
- Logging
- Monitoring

To generate a cost estimate based on your projected usage, use the pricing calculator (https://cloud.google.com/products/calculator). New Google Cloud users might be eligible for a free trial (https://cloud.google.com/free-trial).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).
2. In the Cloud Console, on the project selector page, select or create a Cloud project.

> **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).
4. Enable the Cloud Functions API.

   ENABLE THE API (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=CLOUDFUN
5. In the Cloud Console, go to Monitoring.

   GO TO STACKDRIVER MONITORING (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/MONITORING)

   A Workspace is created automatically for you, if you don't have any existing Workspaces. Otherwise, you have the option to create a new Workspace, or add your project to an existing Workspace.

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

Setting up your environment

In this tutorial, you use Cloud Shell (https://cloud.google.com/shell/docs/overview) to enter commands. Cloud Shell gives you access to the command line in the Cloud Console, and includes the Cloud SDK and other tools that you need to develop in Google Cloud. Cloud Shell appears as a window at the bottom of the Cloud Console. It can take several minutes to initialize, but the window appears immediately.

To use Cloud Shell to set up your environment and to clone the git repository used in this tutorial:

1. In the Cloud Console, open Cloud Shell.

   OPEN CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)
2. Make sure you are working in the project you just created. Replace `[YOUR_PROJECT_ID]` with your newly created Google Cloud project.

```
gcloud config set project [YOUR_PROJECT_ID]
```

3. Set the default compute zone. For the purposes of this tutorial, it is `us-east1`. If you are deploying to a production environment, deploy to a region of your choice (https://cloud.google.com/about/locations/#products-available-by-location).

```
REGION=us-east1
```

4. Clone the repository containing the functions used in this tutorial.

```
git clone https://github.com/GoogleCloudPlatform/solutions-gcs-bq-streaming-fun
cd solutions-gcs-bq-streaming-functions-python
```

Creating streaming source and destination sinks

To stream content into BigQuery, you need to have a `FILES_SOURCE` Cloud Storage bucket and a destination table in BigQuery.

Create the Cloud Storage bucket

You create a Cloud Storage bucket that represents the source of the streaming pipeline presented in this tutorial. The main goal of this bucket is to temporarily store JSON files that are streamed into BigQuery.

- Create your `FILES_SOURCE` Cloud Storage bucket, where `FILES_SOURCE` is set up as an environment variable with a unique name.

```
FILES_SOURCE=${DEVSHELL_PROJECT_ID}-files-source-$(date +%s)
gsutil mb -c regional -l ${REGION} gs://${FILES_SOURCE}
```

Create the BigQuery table

This section creates a BigQuery table which is used as the content destination for your files. BigQuery lets you specify the table's schema when you load data into the table or when you create a new table. In this section, you create the table and specify its schema at the same time.

1. Create a BigQuery dataset and table. The schema defined in the `schema.json` file must match the schema of the files coming from the `FILES_SOURCE`bucket.

```
bq mk mydataset
bq mk mydataset.mytable schema.json
```

2. Verify that the table was created.

```
bq ls --format=pretty mydataset
```

The output is:

```
+---------+-------+--------+-------------------+
| tableId | Type  | Labels | Time Partitioning |
+---------+-------+--------+-------------------+
| mytable | TABLE |        |                   |
+---------+-------+--------+-------------------+
```

Streaming data into BigQuery

Now that you created the source and destination sinks, you create the Cloud Function to stream data from Cloud Storage into BigQuery.

Set up the streaming Cloud Function

The streaming function listens for new files added to the `FILES_SOURCE` bucket and then triggers a process which does the following:

- Parses and validates the file.
- Checks for duplications.
- Inserts the file's content into BigQuery.
- Logs the ingestion status in Firestore and Logging.
- Publishes a message to either an error or success topic in Pub/Sub.

To deploy the function:

1. Create a Cloud Storage bucket to stage your functions during deployment where `FUNCTIONS_BUCKET` is set up as an environment variable with a unique name.

```
FUNCTIONS_BUCKET=${DEVSHELL_PROJECT_ID}-functions-$(date +%s)
gsutil mb -c regional -l ${REGION} gs://${FUNCTIONS_BUCKET}
```

2. Deploy your `streaming` function. The implementation code is in the `./functions/streaming` folder. It might take a few minutes to finish.

```
gcloud functions deploy streaming --region=${REGION} \
    --source=./functions/streaming --runtime=python37 \
    --stage-bucket=${FUNCTIONS_BUCKET} \
    --trigger-bucket=${FILES_SOURCE}
```

   This code deploys a Cloud Function written in Python, which is named `streaming`. It is triggered whenever a file is added to your `FILES_SOURCE` bucket.

3. Verify that the function was deployed.

```
gcloud functions describe streaming  --region=${REGION} \
    --format="table[box](entryPoint, status, eventTrigger.eventType)"
```

   The output is:

```
┌─────────────┬────────┬─────────────────────────────────────┐
│ ENTRY_POINT | STATUS |           EVENT_TYPE                |
├─────────────┼────────┼─────────────────────────────────────┤
| streaming   | ACTIVE | google.storage.object.finalize      |
└─────────────┴────────┴─────────────────────────────────────┘
```

4. Provision a Pub/Sub topic, called `streaming_error_topic`, to handle the error path.

```
STREAMING_ERROR_TOPIC=streaming_error_topic
gcloud pubsub topics create ${STREAMING_ERROR_TOPIC}
```

5. Provision a Pub/Sub topic, called `streaming_success_topic`, to handle the success path.

```
STREAMING_SUCCESS_TOPIC=streaming_success_topic
gcloud pubsub topics create ${STREAMING_SUCCESS_TOPIC}
```

Set up your Firestore database

While data is streamed into BigQuery it is important to understand what is happening with each file ingestion. For example, suppose you have files that were improperly imported. In this case, you need to figure out the root cause of the problem and fix it to avoid generating corrupted data and inaccurate reports at the end of your pipeline. The `streaming` function, deployed in the previous section, stores the file ingestion status in Firestore documents so you can query recent errors to troubleshoot any issues.

To create your Firestore instance, follow these steps:

1. In the Google Cloud console, go to Firestore.

   GO TO THE FIRESTORE PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/WELCOME)

2. In the **Choose a Cloud Firestore mode** window, click **Select Native Mode**.
3. In the **Select a location** list, select **nam5 (United States)**, and then click **Create Database**. Wait for the Firestore initialization to finish. It usually takes a few minutes.

Handle streaming errors

To provision a path to handle error files, you deploy another Cloud Function, which listens for messages published to `streaming_error_topic`. Your business needs determine how you handle such errors in a production environment. For the purpose of this tutorial, problematic files are moved to another Cloud Storage bucket to facilitate troubleshooting.

1. Create your Cloud Storage bucket to store problematic files. `FILES_ERROR` is set up as an environment variable with a unique name for the bucket that stores error files.

```
FILES_ERROR=${DEVSHELL_PROJECT_ID}-files-error-$(date +%s)
gsutil mb -c regional -l ${REGION} gs://${FILES_ERROR}
```

2. Deploy `streaming_error` function to handle errors. It might take a few minutes.

```
gcloud functions deploy streaming_error --region=${REGION} \
    --source=./functions/move_file \
    --entry-point=move_file --runtime=python37 \
    --stage-bucket=${FUNCTIONS_BUCKET} \
    --trigger-topic=${STREAMING_ERROR_TOPIC} \
    --set-env-vars SOURCE_BUCKET=${FILES_SOURCE},DESTINATION_BUCKET=${FILES_ERR
```

This command is similar to the command to deploy the `streaming` function. The main difference is that in this command the function is triggered by a message published to a topic, and it receives two environment variables: the `SOURCE_BUCKET` variable, where files are copied from, and the `DESTINATION_BUCKET` variable, where files are copied to.

3. Verify that the `streaming_error` function was created.

```
gcloud functions describe streaming_error --region=${REGION} \
    --format="table[box](entryPoint, status, eventTrigger.eventType)"
```

The output is:

```
┌─────────────┬────────┬─────────────────────────────────────┐
│ ENTRY_POINT | STATUS |          EVENT_TYPE         |
├─────────────┼────────┼─────────────────────────────────────┤
| move_file   | ACTIVE | google.pubsub.topic.publish |
└─────────────┴────────┴─────────────────────────────────────┘
```

Handle successful streaming

To provision a path to handle success files, you deploy a third Cloud Function, which listens for published messages to the `streaming_success_topic`. For the purposes of this tutorial, successfully ingested files are archived in a Coldline Cloud Storage bucket.

1. Create your Coldline Cloud Storage bucket. `FILES_SUCCESS`is set up as an environment variable with a unique name for the bucket that stores success files.

```
FILES_SUCCESS=${DEVSHELL_PROJECT_ID}-files-success-$(date +%s)
gsutil mb -c coldline -l ${REGION} gs://${FILES_SUCCESS}
```

2. Deploy `streaming_success` function to handle success. It might take a few minutes.

```
gcloud functions deploy streaming_success --region=${REGION} \
    --source=./functions/move_file \
    --entry-point=move_file --runtime=python37 \
    --stage-bucket=${FUNCTIONS_BUCKET} \
    --trigger-topic=${STREAMING_SUCCESS_TOPIC} \
    --set-env-vars SOURCE_BUCKET=${FILES_SOURCE},DESTINATION_BUCKET=${FILES_SUC
```

3. Verify that the function was created.

```
gcloud functions describe streaming_success  --region=${REGION} \
    --format="table[box](entryPoint, status, eventTrigger.eventType)"
```

The output is:

```
┌──────────────┬────────┬─────────────────────────────────────┐
│ ENTRY_POINT  │ STATUS │             EVENT_TYPE              │
├──────────────┼────────┼─────────────────────────────────────┤
│ move_file    │ ACTIVE │ google.pubsub.topic.publish         │
└──────────────┴────────┴─────────────────────────────────────┘
```

Testing your streaming pipeline

At this point, you have finished creating your streaming pipeline. Now it is time to test different paths. First, you test the ingestion of new files, then the ingestion of duplication files, and finally, the ingestion of problematic files.

Ingest new files

To test the ingestion of new files, you upload a file which must successfully pass through the entire pipeline. To make sure everything is behaving correctly, you need to check all storage pieces: BigQuery, Firestore, and Cloud Storage buckets.

1. Upload the `data.json` file to the `FILES_SOURCE` bucket.

   ```
   gsutil cp test_files/data.json gs://${FILES_SOURCE}
   ```

   The output:

   ```
   Operation completed over 1 objects/312.0 B.
   ```

2. Query your data in BigQuery.

   ```
   bq query 'select first_name, last_name, dob from mydataset.mytable'
   ```

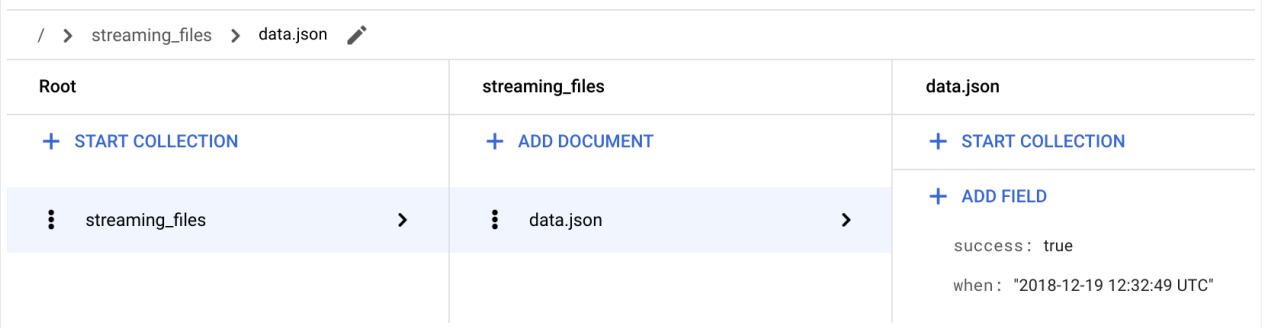
   This command outputs the contents of the `data.json` file:

```
+------------+-----------+------------+
| first_name | last_name |    dob     |
+------------+-----------+------------+
| John       | Doe       | 1968-01-22 |
+------------+-----------+------------+
```

3. In the Cloud Console, go to the Firestore page.

   GO TO THE FIRESTORE PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/DATA)

4. Go to the **/** > **streaming_files** > **data.json** document to verify that the **success: true** field is there. The `streaming` function is storing the file's status in a collection called **streaming_files** and uses the file name as the document ID.



5. Go back to Cloud Shell.

   GO TO CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

6. Verify that the ingested file was removed from the `FILES_SOURCE` bucket by the `streaming_success` function.

```
gsutil ls -l gs://${FILES_SOURCE}/data.json
```

   The output is a `CommandException` because the file doesn't exist in the `FILES_SOURCE` bucket anymore.

7. Verify that the ingested file is now in `FILES_SUCCESS` bucket.

```
gsutil ls -l gs://${FILES_SUCCESS}/data.json
```

   The output is:

```
TOTAL: 1 objects, 312 bytes.
```

Ingest already processed files

The file name is used as document ID in Firestore. This makes it easy for the `streaming` function to query if a given file was processed or not. If a file was previously successfully ingested, any new attempts to add the file are ignored because it would duplicate information in BigQuery, and result in inaccurate reports.

Note: To mitigate duplications, we recommend providing an `insertId` for each inserted row when streaming data into BigQuery. (https://cloud.google.com/bigquery/streaming-data-into-bigquery#dataconsistency) BigQuery remembers this ID for at least one minute, which works well for retries. If your system might produce a file with the same name in a larger than one minute interval, you must have another mechanism to ensure deduplication, such as Firestore.

In this section you verify that the pipeline is working as expected when duplicate files are uploaded to the `FILES_SOURCE` bucket.

1. Upload the same `data.json` file to the `FILES_SOURCE` bucket again.

   ```
   gsutil cp test_files/data.json gs://${FILES_SOURCE}
   ```

   The output is:

   ```
   Operation completed over 1 objects/312.0 B.
   ```

2. Querying BigQuery returns the same result as before. Meaning that the pipeline processed the file, but it didn't insert its content into BigQuery because it was ingested before.

   ```
   bq query 'select first_name, last_name, dob from mydataset.mytable'
   ```

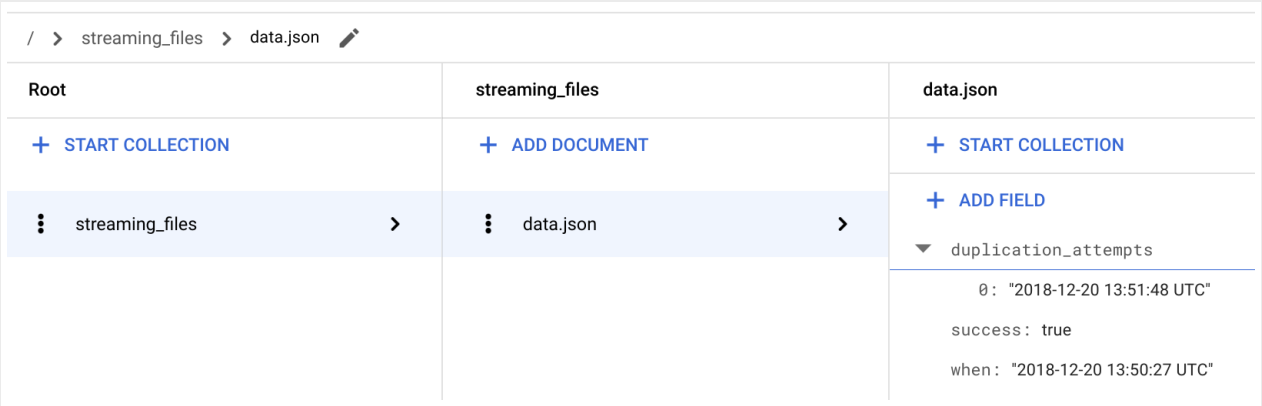
   The output is:

   ```
   +------------+-----------+------------+
   | first_name | last_name |    dob     |
   +------------+-----------+------------+
   | John       | Doe       | 1968-01-22 |
   +------------+-----------+------------+
   ```

3. In the Cloud Console, go to thee Firestore page.

GO TO THE FIRESTORE PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/DATA)

4. In the **/** > **streaming_files** > **data.json** document, verify that the new `**duplication_attempts**` field is added.



   Each time a file is added to the `FILES_SOURCE` bucket with the same name as one previously successfully processed, the content of the file is ignored and a new duplication attempt is appended to the `**duplication_attempts**` field in Firestore.

5. Go back to Cloud Shell.

   GO TO CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

6. Verify that the duplicate file is still in the `FILES_SOURCE` bucket.

```
gsutil ls -l gs://${FILES_SOURCE}/data.json
```

   The output is:

```
TOTAL: 1 objects, 312 bytes.
```

   In the duplication scenario, the `streaming` function logs the unexpected behaviour in Logging, ignores the ingestion, and leaves the file in the `FILES_SOURCE` bucket for later analysis.

Ingest files with errors

Now that you have confirmed that your streaming pipeline is working and that duplications aren't ingested into BigQuery, it's time to check the error path.

1. Upload `data_error.json` to the `FILES_SOURCE` bucket.

```
gsutil cp test_files/data_error.json gs://${FILES_SOURCE}
```

The output is:

```
Operation completed over 1 objects/311.0 B.
```

2. Querying BigQuery returns the same result as before. This means that the pipeline processed the file, but it didn't insert the content into BigQuery because it doesn't comply with the expected schema.

```
bq query 'select first_name, last_name, dob from mydataset.mytable'
```

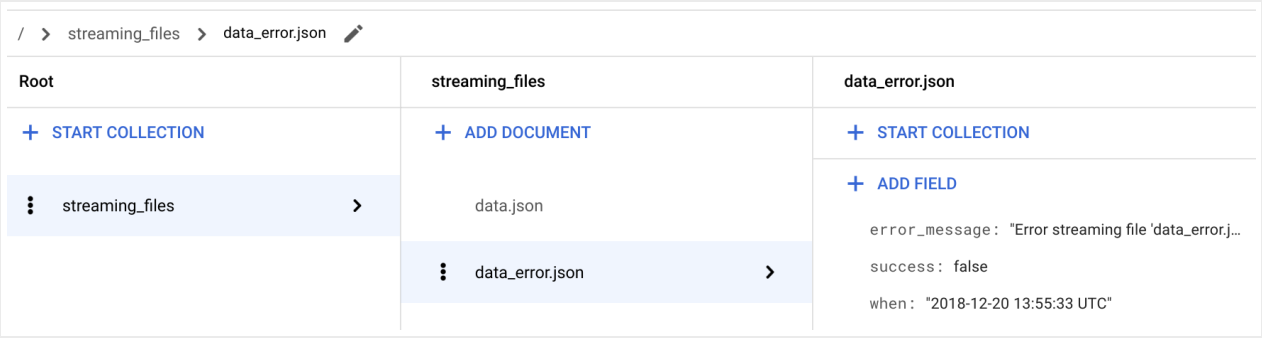
The output is:

```
+------------+-----------+------------+
| first_name | last_name |    dob     |
+------------+-----------+------------+
| John       | Doe       | 1968-01-22 |
+------------+-----------+------------+
```

3. In the Cloud Console, go to the Firestore page.

GO TO THE FIRESTORE PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FIRESTORE/DATA)

4. In the **/** > **streaming_files** > **data_error.json** document, verify that the **success: false** field is added.



For files with errors, the `streaming` function also stores an `error_message` field, which gives you detailed information about why the file wasn't ingested.

5. Go back to Cloud Shell.

GO TO CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/?CLOUDSHELL=TRUE)

6. Verify that the file was removed from the `FILES_SOURCE` bucket by the `streaming_error` function.

```
gsutil ls -l gs://${FILES_SOURCE}/data_error.json
```

The output is a `CommandException` because the file doesn't exist in the `FILES_SOURCE` bucket anymore.

7. Verify that the file is now in the `FILES_ERROR` bucket, as expected.

```
gsutil ls -l gs://${FILES_ERROR}/data_error.json
```

The output is:

```
TOTAL: 1 objects, 311 bytes.
```

Find and fix data ingestion issues

Running queries against the `streaming_files` collection in Firestore lets you quickly diagnose and fix issues. In this section, you filter all error files by using the standard Python API for Firestore (https://pypi.org/project/google-cloud-firestore/).

firestore/show_streaming_errors.py (https://github.com/GoogleCloudPlatform/solutions-gcs-bq-streaming-functions-python/blob/master/firestore/show_streaming_errors.py)

GCS-BQ-STREAMING-FUNCTIONS-PYTHON/BLOB/MASTER/FIRESTORE/SHOW_STREAMING_ERRORS.PY)

```
db = firestore.Client()
docs = db.collection(u'streaming_files')\
    .where(u'success', u'==', False)\
    .get()
```

To see the results of the query in your environment:

1. Create a virtual environment in your `firestore` folder.

```
pip install virtualenv
virtualenv firestore
source firestore/bin/activate
```

2. Install the Python Firestore module in your virtual environment.

```
pip install google-cloud-firestore
```

3. Visualize the existing pipeline issues.

```
python firestore/show_streaming_errors.py
```

The `show_streaming_errors.py` file contains the Firestore query and other boilerplate for looping the result and formatting the output. After you run the preceding command, the output is similar to:

```
+-----------------+-------------------------+---------------------------------
| File Name       | When                    | Error Message
+-----------------+-------------------------+---------------------------------
| data_error.json | 2019-01-22 11:31:58 UTC | Error streaming file 'data_error.
+-----------------+-------------------------+---------------------------------
```

4. Deactivate your virtual environment when you finish your analysis.

```
deactivate
```

After you find and fix the problematic files, upload them to the `FILES_SOURCE` bucket again with the same filename. This process makes them pass through the entire streaming pipeline to insert their content into BigQuery.

Alert on unexpected behaviours

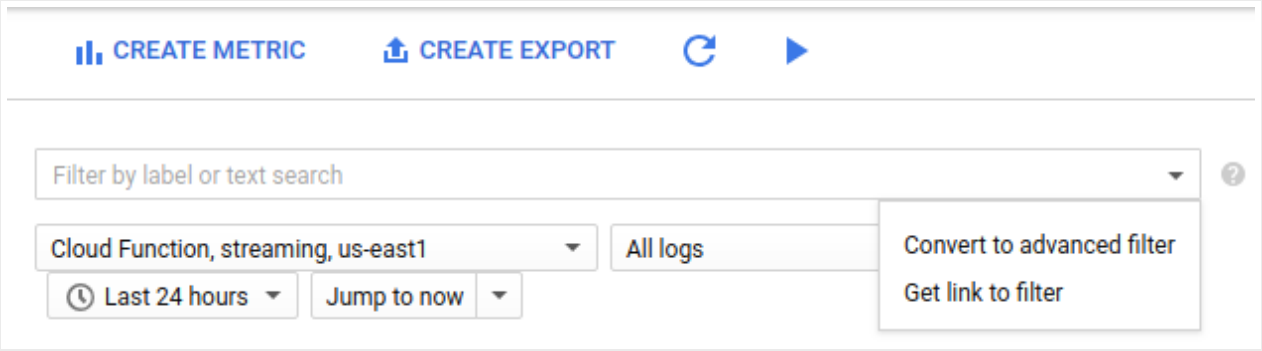
In production environments, it's important to monitor and alert whenever something unexpected happens. One of the many Logging (https://cloud.google.com/logging/) features are custom metrics. Custom metrics let you create alerting policies to notify you and your team when the metric meets specified criteria.

In this section, you configure Monitoring to send email alerts whenever a file ingestion fails. To identify a failing ingestion, the following configuration uses the default Python `logging.error(..)` messages.

1. In the Cloud Console, go to the Logs-based metrics page.

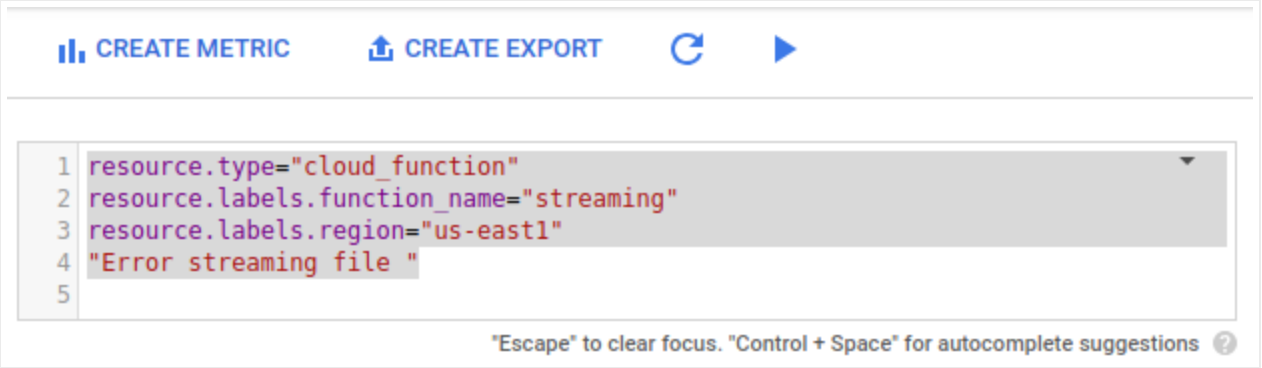
GO TO THE LOGS-BASED METRICS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/LOGS/METRICS

2. Click **Create Metric**.
3. In the **Filter** list, select **Convert to advanced filter**.

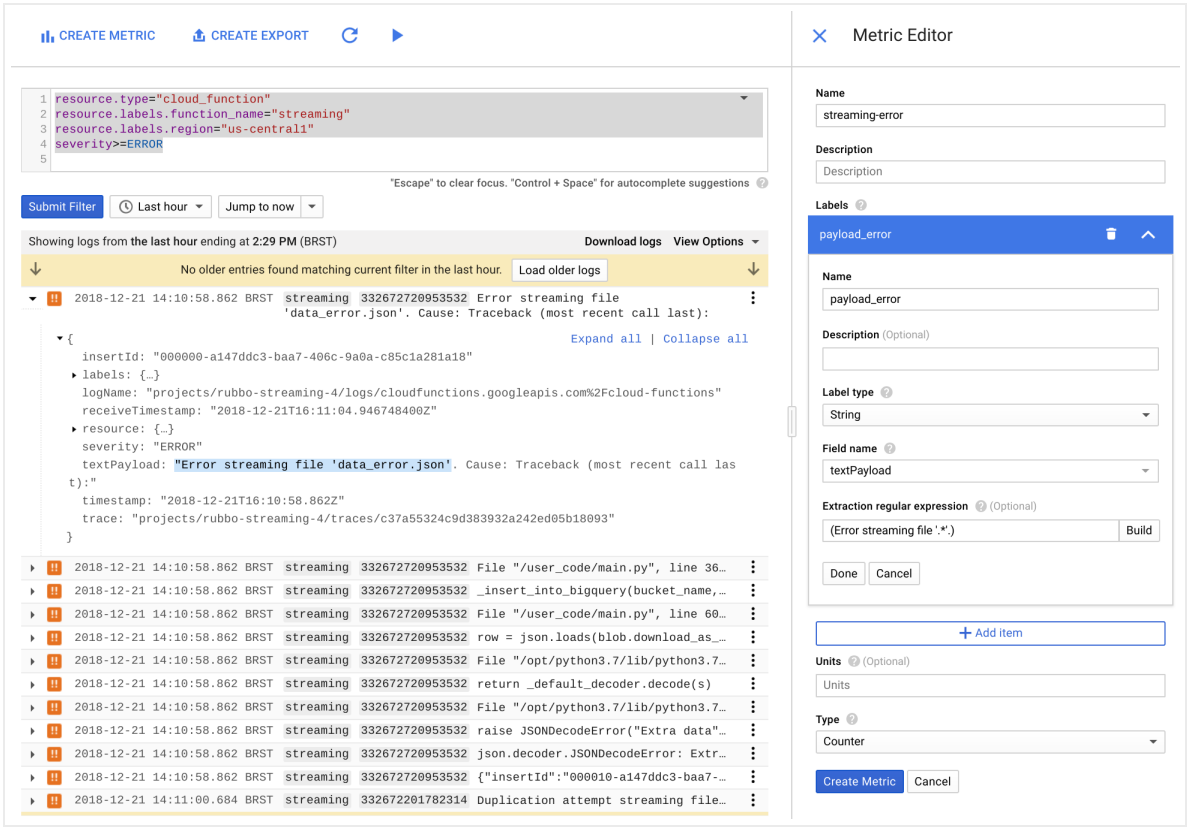


4. In the advanced filter, paste the following configuration.

```
resource.type="cloud_function"
resource.labels.function_name="streaming"
resource.labels.region="us-east1"
"Error streaming file "
```



5. In the **Metric Editor**, fill in the following fields and then click **Create Metric**.
   - In the **Name** field, enter `streaming-error`.
   - In the **Label** section, enter `payload_error` in the **Name** field.
   - In the **Label type** list, select **String**.
   - In the **Field name** list, select **textPayload**.
   - In the **Extraction regular expression** field, enter `(Error streaming file '.*'.)`.
   - In the **Type** list, select **Counter**.



6. In the Google Cloud Console, go to **Monitoring** or use the following button:

   GO TO MONITORING (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/MONITORING)

7. In the Monitoring navigation pane, select **Alerting** and then select **Create Policy**.
8. In the **Name this policy** field, enter `streaming-error-alert`.
9. Click **Add Condition**:
   - In the **Title** field, enter `streaming-error-condition`.
   - In the **Metric** field, enter `logging/user/streaming-error`.
   - In the **Condition trigger If** list, select **Any time series violates**.
   - In the **Condition** list, select **is above**.
   - In the **Threshold** field, enter `0`.
   - In the **For** list, select **1 minute**.
10. In the **Notification Channel Type** list, select **Email**, enter your email address, and then click **Add Notification Channel**.
11. (Optional) Click **Documentation** and add any information that you want included in a notification message.

12. Click **Save**.

After saving the alerting policy, Monitoring monitors the `streaming` function error logs and sends an email alert every time there are streaming errors during a one minute interval.

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Delete the project

> **Caution**: Deleting a project has the following effects:
>
> - **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
> - **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRC

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Review Events and triggers (https://cloud.google.com/functions/docs/concepts/events-triggers) to learn other ways to trigger a serverless function in Google Cloud.
- Visit the alerting (https://cloud.google.com/monitoring/alerts/) page to learn how to improve the alerting policy defined in this tutorial.

- Visit the Firestore documentation (https://cloud.google.com/firestore/) to learn more about this global scale, NoSQL database.
- Visit the BigQuery Quota and limits (https://cloud.google.com/bigquery/quotas#streaming_inserts) page to understand streaming insert limits while implementing this solution in a production environment.
- Visit the Cloud Functions quota and limits (https://cloud.google.com/functions/quotas) page to understand the maximum size a deployed function can handle.
- Try out other Google Cloud features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

Last updated January 16, 2020.