Serverless Data Processing with Dataflow - Monitoring, Logging and Error Reporting for Dataflow Jobs

2 hours No cost

Overview

In this lab, you:

- Create an alerting policy.
- Perform simple troubleshooting for pipelines.
- See how the Diagnostic and BQ tabs work.
- Explore the Error Reporting page.

Prerequisites

Basic familiarity with Python.

Setup and requirements

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.

2. Note the lab's access time (for example, `1:15:00`), and make sure you can finish within that time.
   There is no pause feature. You can restart if needed, but you have to start at the beginning.

3. When ready, click **Start lab**.

4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.

5. Click **Open Google Console**.

6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
   If you use other credentials, you'll receive errors or **incur charges**.

7. Accept the terms and skip the recovery resource page.

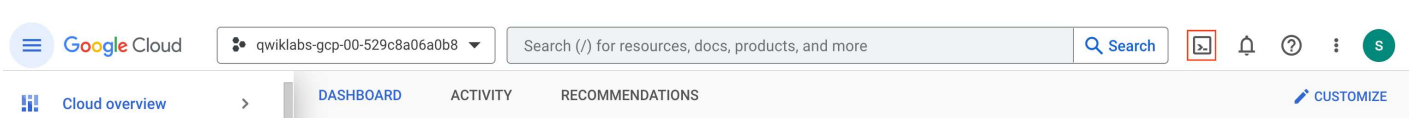
> **Note:** Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

Activate Google Cloud Shell

Google Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud.

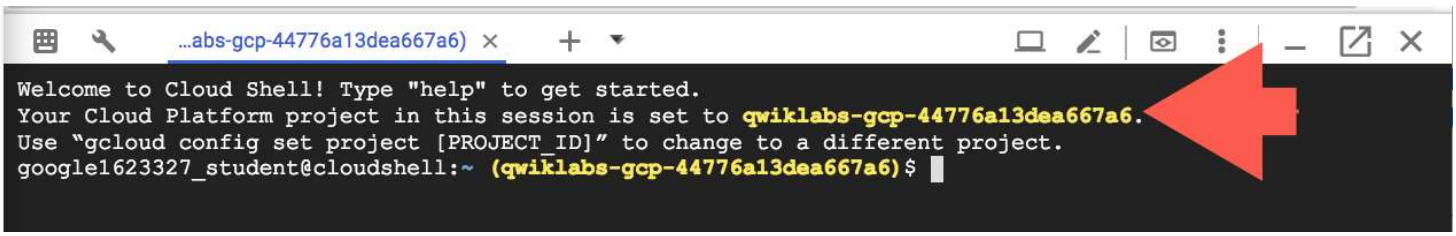
Google Cloud Shell provides command-line access to your Google Cloud resources.

1. In Cloud console, on the top right toolbar, click the Open Cloud Shell button.



2. Click **Continue**.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

- You can list the active account name with this command:

```
gcloud auth list
```

content_c

Output:

```
Credentialed accounts:
 - @.com (active)
```

Example output:

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

- You can list the project ID with this command:

```
gcloud config list project
```

content_c

Output:

```
[core]
project =
```

Example output:

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

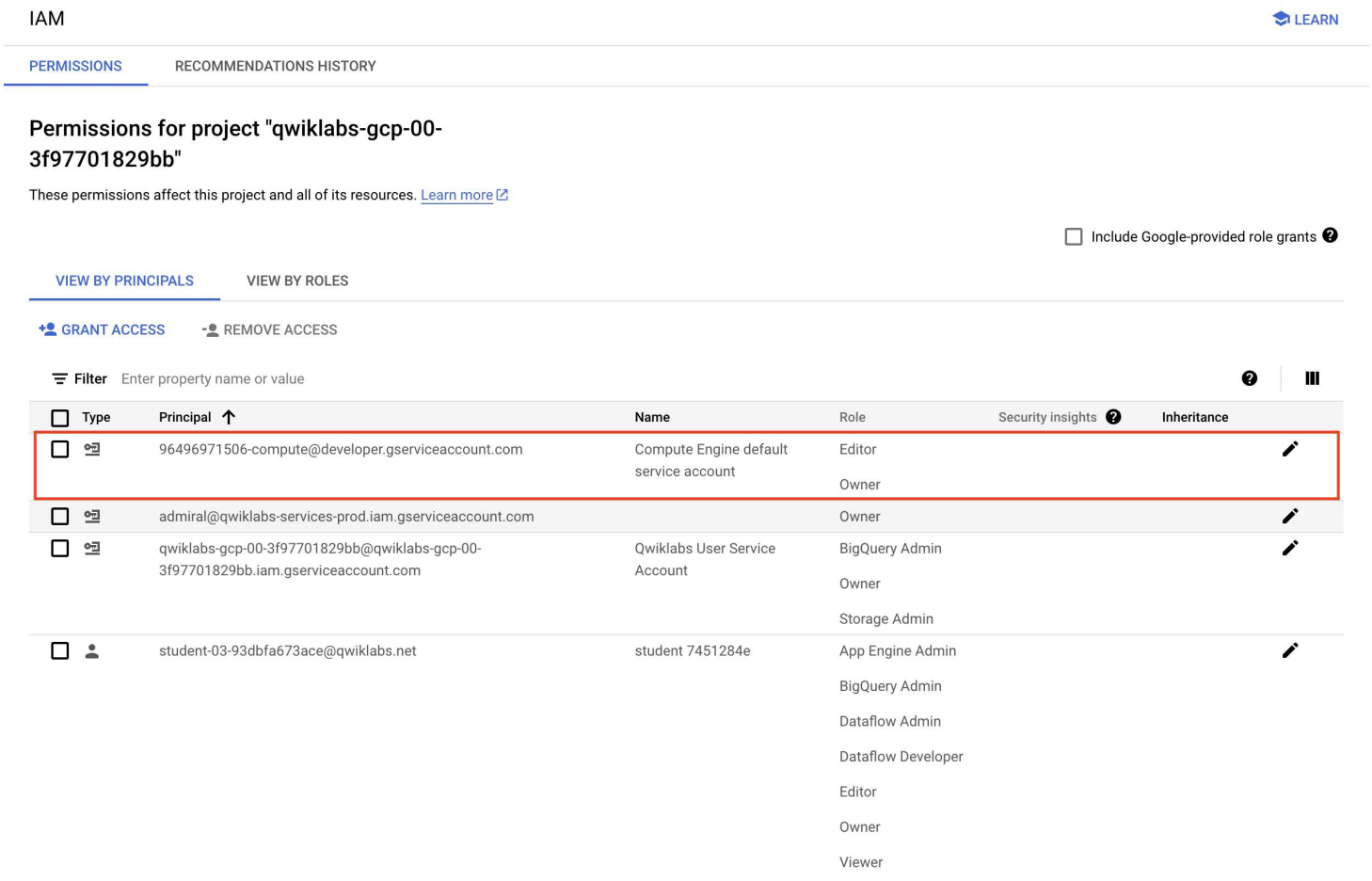
Note: Full documentation of **gcloud** is available in the gcloud CLI overview guide .

Check project permissions

Before you begin your work on Google Cloud, you need to ensure that your project has the correct permissions within Identity and Access Management (IAM).

1. In the Google Cloud console, on the **Navigation menu** (☰), select **IAM & Admin** > **IAM**.

2. Confirm that the default compute Service Account `{project-number}-compute@developer.gserviceaccount.com` is present and has the `editor` role assigned. The account prefix is the project number, which you can find on **Navigation menu** > **Cloud Overview** > **Dashboard**.



Note: If the account is not present in IAM or does not have the `editor` role, follow the steps below to assign the required role.

1. In the Google Cloud console, on the **Navigation menu**, click **Cloud Overview > Dashboard**.

2. Copy the project number (e.g. `729328892908`).

3. On the **Navigation menu**, select **IAM & Admin** > **IAM**.

4. At the top of the roles table, below **View by Principals**, click **Grant Access**.

5. For **New principals**, type:

```
{project-number}-compute@developer.gserviceaccount.com
```


6. Replace `{project-number}` with your project number.

7. For **Role**, select **Project** (or Basic) > **Editor**.

8. Click **Save**.

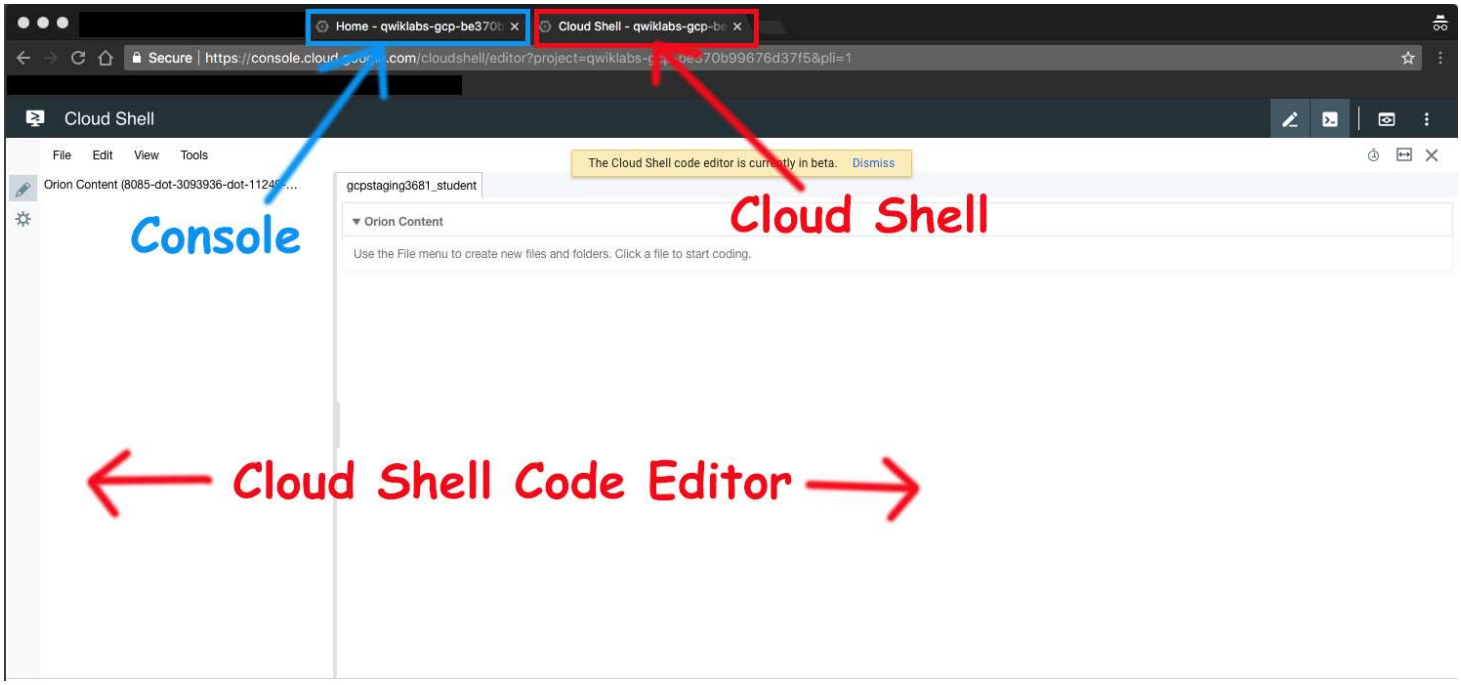
Launch Google Cloud Shell Code Editor

Use the Google Cloud Shell Code Editor to easily create and edit directories and files in the Cloud Shell instance.

- Once you activate the Google Cloud Shell, click **Open editor** to open the Cloud Shell Code Editor.

You now have three interfaces available:

- The Cloud Shell Code Editor
- Console (By clicking on the tab). You can switch back and forth between the Console and Cloud Shell by clicking on the tab.
- The Cloud Shell Command Line (By clicking on **Open Terminal** in the Console)



Task 1. Create virtual environment and install dependencies

1. Create a virtual environment for your work in this lab:

```
sudo apt-get install -y python3-venv
python3 -m venv df-env
source df-env/bin/activate
```

content_c

2. Next, install the packages you will need to execute your pipeline:

```
python3 -m pip install -q --upgrade pip setuptools wheel
python3 -m pip install apache-beam[gcp]
```

content_c

3. Finally, ensure that the Dataflow API is enabled:

```
gcloud services enable dataflow.googleapis.com
```

content_c

Task 2. Create alerting policy

In this section, you will create an alert policy that gets triggered when the condition is met.

1. In the Cloud Console, from the **Navigation menu**, select **Monitoring** > **Alerting**.

2. On the Alerting page, click on **+ CREATE POLICY**.

3. Click on **Select a metric** dropdown and disable the **Show only active resources & metrics**.

a. Type **Dataflow Job** in filter by resource and metric name and click on **Dataflow Job > Job**. Select `Failed` and click **Apply**.

b. Set **Rolling windows function** to `Sum`.

c. Click **Next**. Set **0** as your **Threshold value**.

4. Click on the **NEXT** button to go to the **Who should be notified?** step.

5. On the form that loads, click on the **Notification Channels** drop-down menu, and click on **MANAGE NOTIFICATION CHANNELS**.

This opens a new window that lists the supported notification channel types.

6. Scroll down to the row that says **Email** and click on **ADD NEW** on the far right.

   a. For **Email Address**, enter your personal email address.

   > **Note:** The Qwiklabs student account email will not work as it is a temporary account.

   b. For **Display Name**, enter `Qwiklabs Student`.

   c. Click on **Save**. You can now close this window and go back to the previous **Create alerting policy** window.

7. Click on the refresh icon to the left of **MANAGE NOTIFICATION CHANNELS**.

8. Next, click on the **Notification Channels** drop-down menu again. This time you should see the display name of the student account you just added.

9. Check the checkbox to the left of the name **Qwiklabs Student** and click **OK**.

10. Enter `Failed Dataflow Job` as the **Alert Name** in the textbox.

11. Click **Next** again.

12. Review the alert and click **Create Policy**.

Click *Check my progress* to verify the objective.

Create Alerting Policy

Check my progress

Assessment Completed!

Task 3. Failure starting the pipeline on Dataflow

1. Review the code for your pipeline below:

content_c

```
import argparse
import logging
import argparse, logging, os
import apache_beam as beam
from apache_beam.io import WriteToText
from apache_beam.options.pipeline_options import PipelineOptions

class ReadGBK(beam.DoFn):

 def process(self, e):
   k, elems = e
   for v in elems:
     logging.info(f"the element is {v}")
     yield v


def run(argv=None):
   parser = argparse.ArgumentParser()
   parser.add_argument(
     '--output', dest='output', help='Output file to write results
to.')
   known_args, pipeline_args = parser.parse_known_args(argv)
   read_query = """(
```

```
                  SELECT
                    version,
                    block_hash,
                    block_number
                  FROM
                    `bugquery-public-data.crypto_bitcoin.transactions`
                  WHERE
                    version = 1
                  LIMIT
                    1000000 )
                UNION ALL (
                  SELECT
                    version,
                    block_hash,
                    block_number
                  FROM
                    `bigquery-public-data.crypto_bitcoin.transactions`
                  WHERE
                    version = 2
                  LIMIT
                    1000 ) ;"""
   p = beam.Pipeline(options=PipelineOptions(pipeline_args))
   (p
   | 'Read from BigQuery' >>
beam.io.ReadFromBigQuery(query=read_query, use_standard_sql=True)
   | "Add Hotkey" >> beam.Map(lambda elem: (elem["version"], elem))
   | "Groupby" >> beam.GroupByKey()
   | 'Print' >>  beam.ParDo(ReadGBK())
   | 'Sink' >>  WriteToText(known_args.output))

   result = p.run()

if __name__ == '__main__':
 logger = logging.getLogger().setLevel(logging.INFO)
 run()
```

2. In **Cloud Shell**, open a new file named `my_pipeline.py` . Copy and paste the code from above into this file and save it, using your preferred text editor (the example below refers to Vim):

```
vi my_pipeline.py
```

content_c

3. After you paste the code into the file, be sure to save it.

4. Run the command below to create a storage bucket:

```
export PROJECT_ID=$(gcloud config get-value project)
gsutil mb -l US gs://$PROJECT_ID
```

content_c

5. Next, you will attempt to launch the pipeline using the command below (note that the command will fail):

```
# Attempt to launch the pipeline
python3 my_pipeline.py \
  --project=${PROJECT_ID} \
  --region=us-central1 \
  --tempLocation=gs://$PROJECT_ID/temp/ \
  --runner=DataflowRunner
```

content_c

The above command to launch the pipeline will fail with a stacktrace similar to the one shown in the screenshot below:

```
Traceback (most recent call last):
  File "my_pipeline.py", line 56, in <module>
    run()
  File "my_pipeline.py", line 50, in run
    | 'Sink' >>  WriteToText(known_args.output))
  File "/home/student_00_380e9c1f411a/df-env/lib/python3.7/site-packages/apache_beam/io/textio.py", line 653, in __init__
    footer)
  File "/home/student_00_380e9c1f411a/df-env/lib/python3.7/site-packages/apache_beam/io/textio.py", line 399, in __init__
    compression_type=compression_type)
  File "/home/student_00_380e9c1f411a/df-env/lib/python3.7/site-packages/apache_beam/io/filebasedsink.py", line 90, in __init__
    'got %r instead' % file_path_prefix)
```

This failure is on the Beam end. In the code, we specify `WriteToText(known_args.output)` . Since we did not pass in the --output flag, the code did not pass Beam validation, and was not able to launch in Dataflow. As this did not reach Dataflow, no job ID is associated with this launch operation. This means you should not get any alert email in your inbox.

Click *Check my progress* to verify the objective.

Failure starting the pipeline on Dataflow

Check my progress

Assessment Completed!

Task 4. Invalid BigQuery table

In this section, you try to launch the pipeline again with the required **--output** flag. While this does succeed in launching the Dataflow pipeline, the job will fail after a few minutes because we intentionally add an invalid BigQuery table. This should trigger the alerting policy and result in an alert email being sent.

1. In the terminal, execute the following command to launch the pipeline:

```
# Launch the pipeline
python3 my_pipeline.py \
  --project=${PROJECT_ID} \
  --region=us-central1 \
  --output gs://$PROJECT_ID/results/prefix \
  --tempLocation=gs://$PROJECT_ID/temp/ \
  --max_num_workers=5 \
  --runner=DataflowRunner
```

content_c

2. In the Cloud Console, and from the **Navigation menu**, select **Dataflow** > **Jobs**.

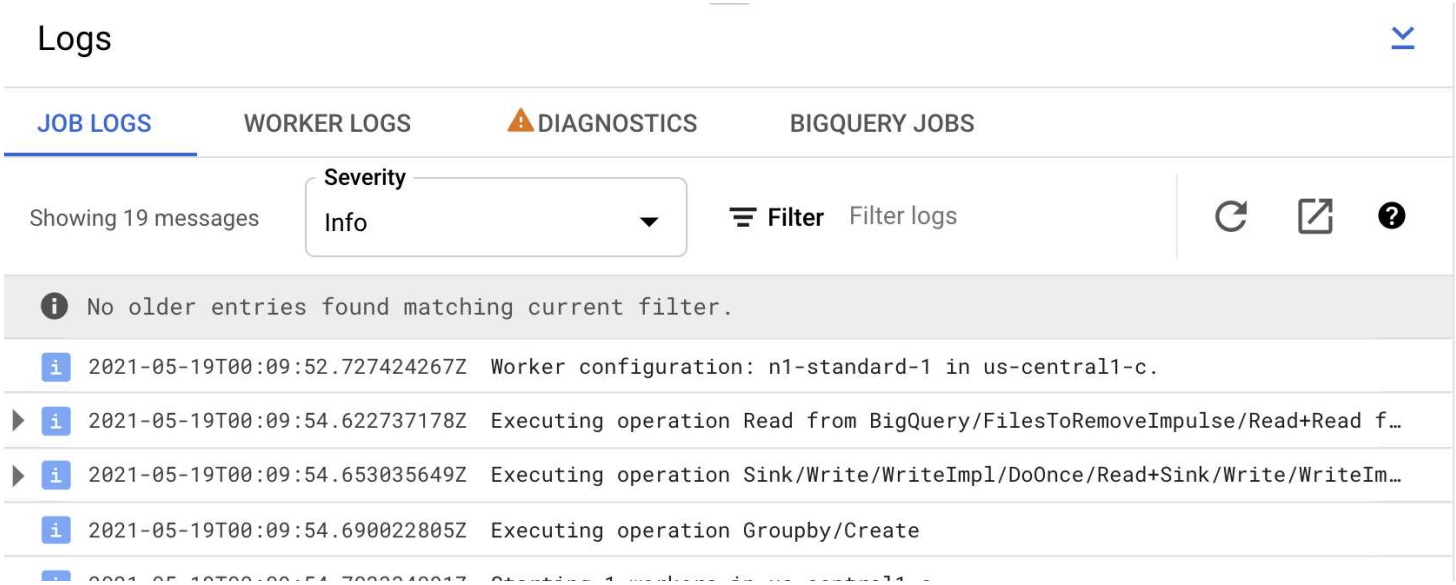
3. Click on the Dataflow job listed on that page.

If you wait for about four minutes, you will see that the job fails. This triggers the alerting policy and you can look in your email inbox to view the alert email.

4. Below the Dataflow pipeline job graph, you will see the **Logs** panel. Expand the panel by clicking on the expansion icon to the far right of the word "Logs". This will expose the following tabs: **Jobs logs**, **Worker logs**, **Diagnostics**, and **BigQuery jobs**.

5. Click on the **Job logs** tab and explore the log entries.

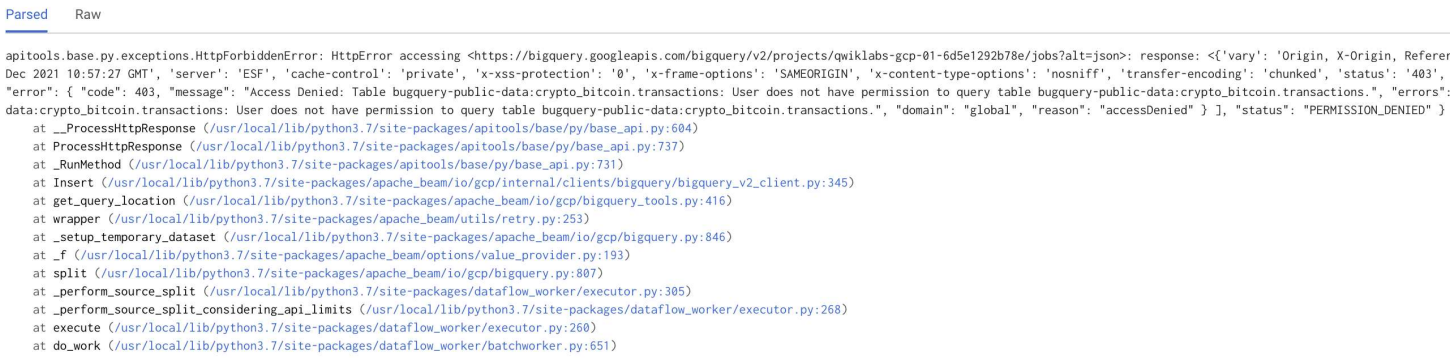
6. Next, click on the **Worker logs** tab and explore those logs as well.



Note: You will notice that the logs can be expanded when you click on them. Logs that get repeated can appear in the **Diagnostics** tab.

7. When you click on the **Diagnostics** tab, you will see a clickable link.

8. Click on it and this takes you to the Error Reporting page. This shows us an expanded view of the error. Looking at the stack trace, you will see the issue: a typo in the pipeline code!



In the code, **bigquery** is intentionally misspelled as **bugquery**, so this fails the Dataflow job and triggers the alerting policy to send an email. In the next section, you will fix the code and relaunch the pipeline.

Click *Check my progress* to verify the objective.

Invalid BigQuery table

Check my progress

Assessment Completed!

Task 5. Too much logging

1. Using a text editor (such as Vim or your preferred editor), open the `my_pipeline.py` file and fix the code by replacing `bugquery` with `bigquery`.

> **Note:** In the code, you will find the misspelled **bugquery** in the **SELECT** statement as `bugquery-public-data.crypto_bitcoin.transactions`. Change it to `bigquery-public-data.crypto_bitcoin.transactions`.

2. In the terminal, execute the following command to launch the pipeline:

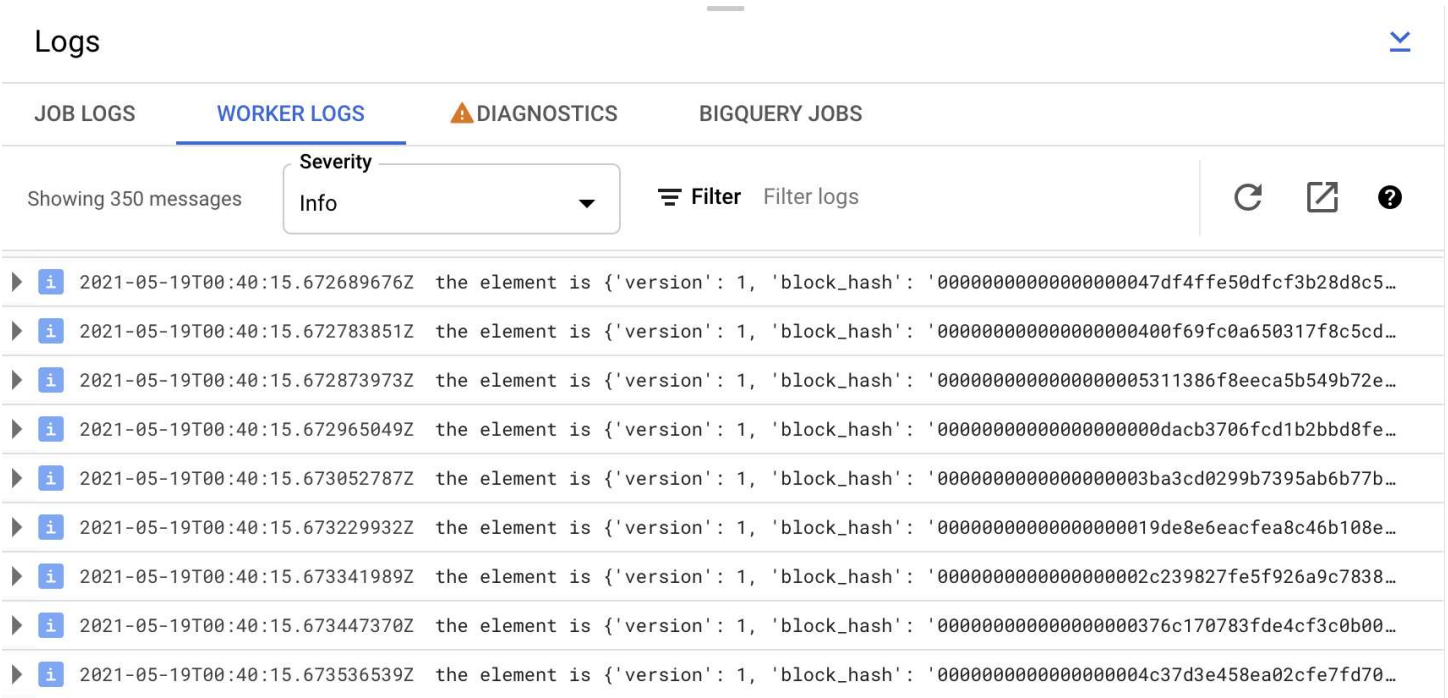
```
# Launch the pipeline
python3 my_pipeline.py \
  --project=${PROJECT_ID} \
  --region=us-central1 \
  --output gs://$PROJECT_ID/results/prefix \
  --tempLocation=gs://$PROJECT_ID/temp/ \
  --max_num_workers=5 \
  --runner=DataflowRunner
```


The pipeline will take about seven minutes to complete.

3. In the Cloud Console, from the **Navigation menu**, select **Dataflow** > **Jobs**. When you click on this new job, in the job graph page on the far right you should see **Job status** as `succeeded`. If the job is not complete, please wait for it to complete and succeed.

4. Expand on the **Logs** at the bottom and review the logs in the **Worker logs** tab.

In the **Worker logs** tab, you will see log entries with the format `the element is {'version' : ..... }`, as shown in the screenshot below:

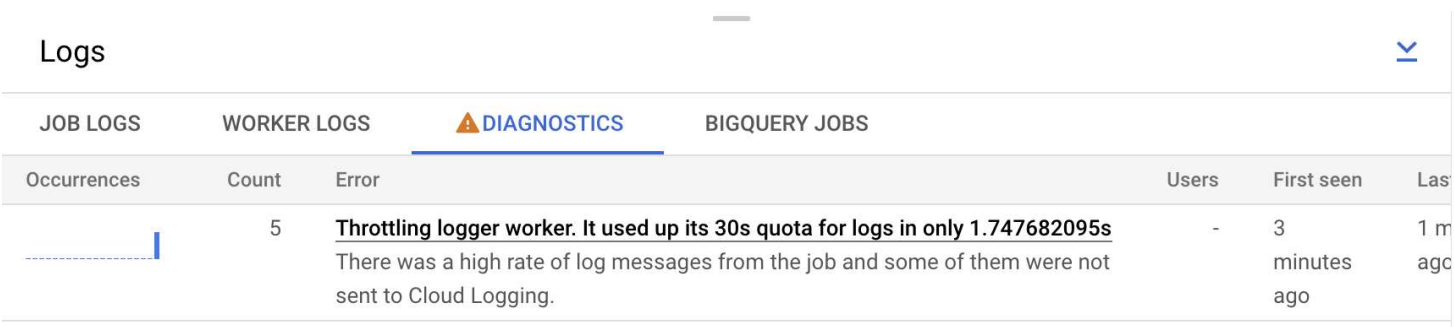


These entries are being logged because of the following line in the pipeline code:

```
logging.info(f"the element is {v}")
```

content_c

5. Click on the **Diagnostics** tab and you will see a message about throttling logs.



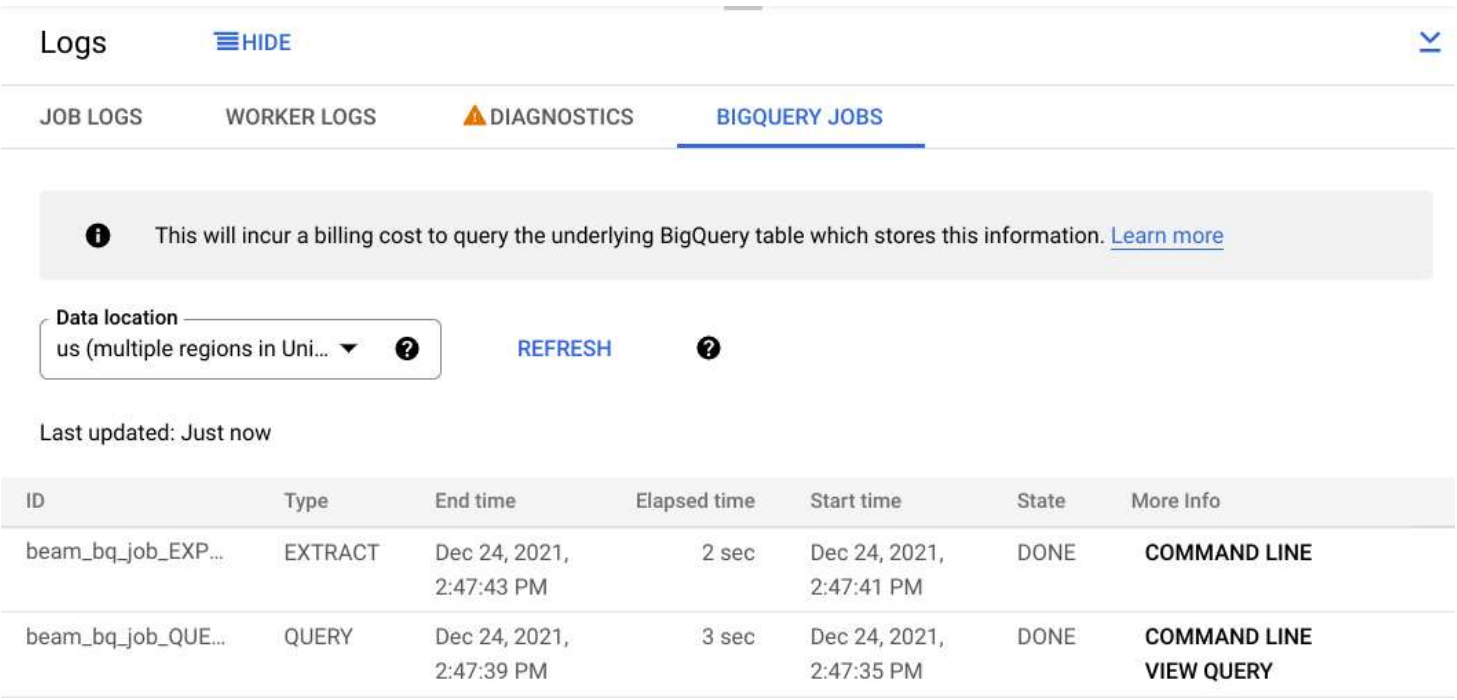
6. Click on it and navigate to the **Error Reporting** page. Excessive logging is raised as a job insight, with a public doc link showing what the issue is.

The simple fix to this issue would be to remove the logging line of code from your pipeline code and rerun the dataflow pipeline. Upon completion, you should no longer see the same message in the **Diagnostics** tab.

> **Note:** You are not required to rerun this pipeline in order to complete this lab.

7. Go back to the Dataflow Jobs page by clicking the back button on your browser.

8. Under the expanded logs panel, click on the **BigQuery jobs** tab. From the **Data Location** drop-down menu, select us (multiple regions in United States) and click on **Load BigQuery jobs**. Then click **Confirm**.



You will see two jobs listed that were part of the pipeline.

9. Click on **Command line** on the far right of the first job, and, on the pop-up, click on **Run in Cloud Shell**.

This will paste the command in your Cloud Shell. Make sure to run the command.

10. The output of the command shows the file locations used to read/write to BigQuery, their size, and how long the job took. If a BigQuery job has failed, this would be the place to look for the cause of failure.

```
  extract:
    compression: NONE
    destinationFormat: AVRO
    destinationUri: gs://dataflow-staging-us-central1-c1a9299613c47de1a561cb7e8d52f29e/574ccccf-7/bigquery-table-dump-*.json
    destinationUris:
    - gs://dataflow-staging-us-central1-c1a9299613c47de1a561cb7e8d52f29e/574ccccf-7/bigquery-table-dump-*.json
    printHeader: false
    sourceTable:
      datasetId: temp_dataset_40cd59091f064c75a753b9e4377c0e1b
      projectId: qwiklabs-gcp-02-f8c6a26b2f43
      tableId: temp_table_40cd59091f064c75a753b9e4377c0e1b
    useAvroLogicalTypes: true
  jobType: EXTRACT
  labels:
    beam_job_id: 2021-05-18_17_33_54-12699155996613753449
    step_name: readfrombigquery
etag: FpH0x+VAWS9qKinZMsg2HQ==
id: qwiklabs-gcp-02-f8c6a26b2f43:US.beam_bq_job_EXPORT_BQ_EXPORT_JOB_574ccccf-7_1621384641_564
```

Click *Check my progress* to verify the objective.

Too much logging

Check my progress

Please fix "my_pipeline.py" file code by replacing bugquery with bigquery, rerun your pipeline and wait until job status gets succeed.

End your lab