End Lab 01:23:27

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Console

Username
student-00-7529b873a8e

Password
TxLwEL6iv4pM

GCP Project ID
qwiklabs-gcp-03-f268b0

QL Region
us-central1

Streaming Data Processing: Streaming Data Pipelines

2 hours Free ★★★★½

Overview

In this lab, you will use Dataflow to collect traffic events from simulated traffic sensor data made available through Google Cloud PubSub, process them into an actionable average, and store the raw data in BigQuery for later analysis. You will learn how to start a Dataflow pipeline, monitor it, and, lastly, optimize it.

Note: at the time of this writing, streaming pipelines are not available in the DataFlow Python SDK. So the streaming labs are written in Java.

Objectives

In this lab, you will perform the following tasks:

- Launch Dataflow and run a Dataflow job
- Understand how data elements flow through the transformations of a Dataflow pipeline
- Connect Dataflow to Pub/Sub and BigQuery
- Observe and understand how Dataflow autoscaling adjusts compute resources to process input data optimally
- Learn where to find logging information created by Dataflow
- Explore metrics and create alerts and dashboards with Cloud Monitoring

Setup

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

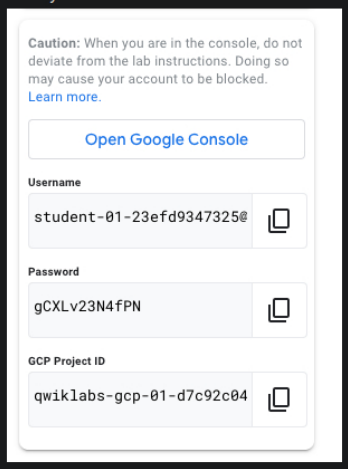
1. Make sure you signed into Qwiklabs using an **incognito window**.

2. Note the lab's access time (for example, 02:00:00 and make sure you can finish in that time block.

> 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. You will use them to sign in to Cloud Platform 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 get errors or **incur charges**.

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

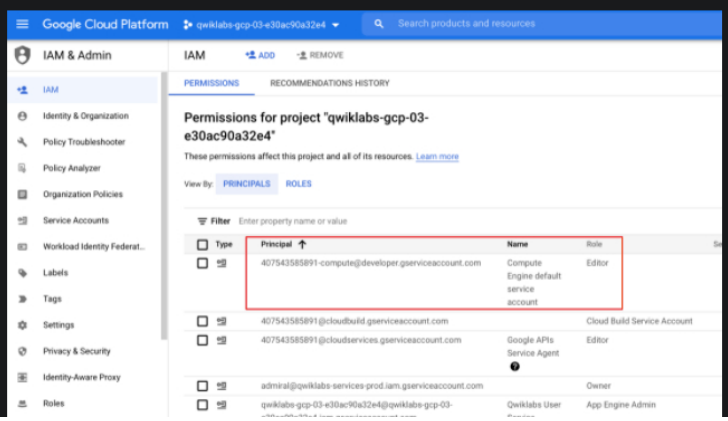
> Do not click **End Lab** unless you are finished with the lab or want to restart it. This clears your work and removes the project.

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** (☰), click **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** > **Home**.



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

- In the Google Cloud console, on the **Navigation menu**, click **Home**.
- Copy the project number (e.g. `729328892908`).
- On the **Navigation menu**, click **IAM & Admin** > **IAM**.
- At the top of the **IAM** page, click **Add**.
- For **New principals**, type:

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

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

- For **Role**, select **Project** (or Basic) > **Editor**. Click **Save**.

Task 1: Preparation

You will be running a sensor simulator from the training VM. In Lab 1 you manually setup the Pub/Sub components. In this lab several of those processes are automated.

Open the SSH terminal and connect to the training VM

1. In the Console, on the **Navigation menu** (☰), click **Compute Engine** > **VM instances**.
2. Locate the line with the instance called **training-vm**.
3. On the far right, under **Connect**, click on **SSH** to open a terminal window.
4. In this lab, you will enter CLI commands on the **training-vm**.

Verify initialization is complete

5. The **training-vm** is installing some software in the background. Verify that setup is complete by checking the contents of the new directory.

```
ls /training
```

The setup is complete when the result of your list (ls) command output appears as in the image below. If the full listing does not appear, wait a few minutes and try again.

> **Note:** it may take 2 to 3 minutes for all background actions to complete.

```
student-04-2324a1e71896@training-vm:~$ ls /training
bq_magic.sh  project_env.sh  sensor_magic.sh
student-04-2324a1e71896@training-vm:~$
```

Download Code Repository

6. Next you will download a code repository for use in this lab.

```
git clone https://github.com/GoogleCloudPlatform/training-data-analyst
```

Set environment variables

7. On the **training-vm** SSH terminal enter the following:

```
source /training/project_env.sh
```

This script sets the `DEVSHELL_PROJECT_ID` and `BUCKET` environment variables.

Task 2: Create a BigQuery Dataset and Cloud Storage Bucket

The Dataflow pipeline will be created later and will write into a table in the BigQuery dataset.

Open BigQuery Console

1. In the Google Cloud Console, select **Navigation menu** > **BigQuery**.

The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and lists UI updates.

2. Click **Done**.

Create a BigQuery Dataset

1. To create a dataset, click on the **View actions** icon next to your project ID and select **Create dataset**.

2. Next, name your Dataset ID **demos** and leave all other options at their default values, and then click **Create dataset**.

Verify the Cloud Storage Bucket

A bucket should already exist that has the same name as the Project ID.

3. In the Console, on the **Navigation menu** (☰), click **Cloud Storage** > **Browser**.

4. Observe the following values:

Property	**Value** (type value or select option as specified)
Name	**<Same as the Project ID>**
Default storage class	**Regional**
Location	**us-central1**

Click Check my progress to verify the objective.

Create a BigQuery Dataset

Check my progress

Task 3: Simulate traffic sensor data into Pub/Sub

1. In the **training-vm** SSH terminal, start the sensor simulator. The script reads sample data from a CSV file and publishes it to Pub/Sub.

```
/training/sensor_magic.sh
```

This command will send 1 hour of data in 1 minute. Let the script continue to run in the current terminal.

Open a second SSH terminal and connect to the training VM

2. In the upper right corner of the **training-vm** SSH terminal, click on the gear-shaped button (⚙) and select **New Connection to training-vm** from the drop-down menu. A new terminal window will open.

3. The new terminal session will not have the required environment variables. Run the following command to set them.

4. In the new **training-vm** SSH terminal enter the following:

```
source /training/project_env.sh
```

Click Check my progress to verify the objective.

Simulate traffic sensor data into Pub/Sub

Check my progress

Task 4: Launch Dataflow Pipeline

Verify that Google Cloud Dataflow API is enabled for this project

1. Return to the browser tab for Console.

2. In the top search bar, enter **Dataflow API**. This will take you to the page, **Navigation menu > APIs & Services > Dashboard > Dataflow API**.

3. It will either show status information or it will give you the option to **Enable** the API.

4. If necessary, **Enable** the API.

5. Return to the second **training-vm** SSH terminal. Change into the directory for this lab.

```
cd ~/training-data-analyst/courses/streaming/process/sandiego
```

6. Identify the script that creates and runs the Dataflow pipeline.

```
cat run_oncloud.sh
```

7. Copy-and-paste the following URL into a new browser tab to view the source code on Github.

```
https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/courses/streaming/process/sandiego/run_oncloud.
```

8. The script requires three arguments: **project id**, **bucket name**, **classname**

A 4th optional argument is **options**. The **options** argument discussed later in this lab.

project id	<your Project ID>
bucket name	<your Bucket Name>
classname	<java file that runs aggregations>
options	<options>

There are 4 java files that you can choose from for **classname**. Each reads the traffic data from Pub/Sub and runs different aggregations/computations.

9. Go into the java directory. Identify the source file **AverageSpeeds.java**.

```
cd ~/training-data-analyst/courses/streaming/process/sandiego/src/main/java/com/google
cat AverageSpeeds.java
```

What does the script do?

Close the file to continue. You will want to refer to this source code while running the application. So for easy access you will open a new browser tab and view the file **AverageSpeeds.java** on Github.

10. Copy-and-paste the following URL into a browser tab to view the source code on Github.

```
https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/courses/streaming/process/sandiego/src/main/jav
```

Leave this browser tab open. You will be referring back to the source code in a later step in this lab.

11. Return to the **training-vm** SSH terminal. Run the Dataflow pipeline to read from PubSub and write into BigQuery.

```
cd ~/training-data-analyst/courses/streaming/process/sandiego
./run_oncloud.sh $DEVSHELL_PROJECT_ID $BUCKET AverageSpeeds
```

This script uses maven to build a Dataflow streaming pipeline in Java.

Example successful completion:

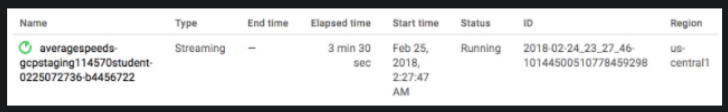
```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 45.542 s
[INFO] Finished at: 2018-06-08T16:51:30+00:00
[INFO] Final Memory: 56M/216M
[INFO] ------------------------------------------------------------------------
```

Task 5: Explore the pipeline

This Dataflow pipeline reads messages from a Pub/Sub topic, parses the JSON of the input message, produces one main output and writes to BigQuery.

1. Return to the browser tab for Console. On the **Navigation menu** (☰), click **Dataflow** and click on your job to monitor progress.

Example:



Name	Type	End time	Elapsed time	Start time	Status	ID	Region
averagespeeds-gcpstaging114570student-0225072736-b4456722	Streaming	–	3 min 30 sec	Feb 25, 2018, 2:27:47 AM	Running	2018-02-24_23_27_46-10144500510778459298	us-central1

> **Note:** If Dataflow Job failed, run the command `./run_oncloud.sh $DEVSHELL_PROJECT_ID $BUCKET AverageSpeeds` again.

2. After the pipeline is running, click on the **Navigation menu** (☰), click **Pub/Sub** > **Topics**.

3. Examine the line for **Topic name** for the topic **sandiego**.

4. Return to the **Navigation menu** (☰), click **Dataflow** and click on your job.

5. Compare the code in the Github browser tab, **AverageSpeeds.java** and the pipeline graph on the page for your Dataflow job.

6. Find the **GetMessages** pipeline step in the graph, and then find the corresponding code in the **AverageSpeeds.java** file. This is the pipeline step that reads from the Pub/Sub topic. It creates a collection of Strings - which corresponds to Pub/Sub messages that have been read.

- Do you see a subscription created?

- How does the code pull messages from Pub/Sub?

7. Find the **Time Window** pipeline step in the graph and in code. In this pipeline step we create a window of a duration specified in the pipeline parameters (sliding window in this case). This window will accumulate the traffic data from the previous step until end of window, and pass it to the next steps for further transforms.

- What is the window interval?

- How often is a new window created?

8. Find the **BySensor** and **AvgBySensor** pipeline steps in the graph, and then find the corresponding code snippet in the AverageSpeeds.java file. This **BySensor** does a grouping of all events in the window by sensor id, while **AvgBySensor** will then compute the mean speed for each grouping.

9. Find the **ToBQRow** pipeline step in the graph and in code. This step simply creates a "row" with the average computed from previous step together with the lane information.

> **Note:** in practice, other actions could be taken in the **ToBQRow** step. For example, it could compare the calculated mean against a predefined threshold and log the results of the comparison in Cloud Logging.

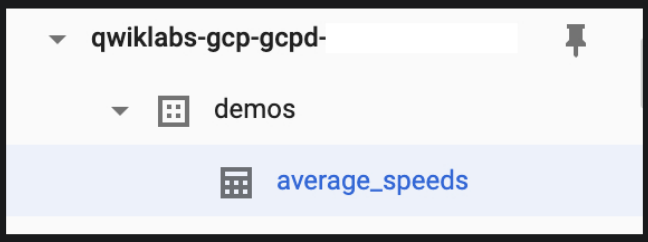
10. Find the **BigQueryIO.Write** in both the pipeline graph and in the source code. This step writes the row out of the pipeline into a BigQuery table. Because we chose the **WriteDisposition.WRITE_APPEND** write disposition, new records will be appended to the table.

11. Return to the BigQuery web UI tab. Refresh your browser.

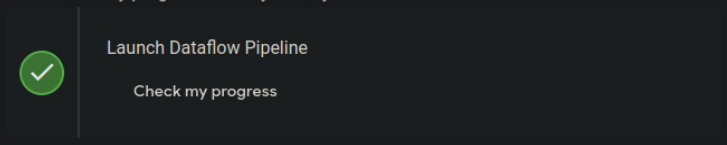
12. Find your project name and the demos dataset you created. The small arrow to the left of the dataset name **demos** should now be active and clicking on it will reveal the **average_speeds** table.

13. It will take several minutes before the **average_speeds** table appears in BigQuery.

Example:



Click Check my progress to verify the objective.



Task 6: Determine throughput rates

One common activity when monitoring and improving Dataflow pipelines is figuring out how many elements the pipeline processes per second, what the system lag is, and how many data elements have been processed so far. In this activity you will learn where in the Cloud Console one can find information about processed elements and time.

1. Return to the browser tab for Console. On the **Navigation menu** (☰), click **Dataflow** and click on your job to monitor progress (it will have your username in the pipeline name).

2. Select the **GetMessages** pipeline node in the graph and look at the step metrics on the right.

- **System Lag** is an important metric for streaming pipelines. It represents the amount of time data elements are waiting to be processed since they "arrived" in the input of the transformation step.

- **Elements Added** metric under output collections tells you how many data elements exited this step (for the **Read PubSub Msg** step of the pipeline it also represents the number of Pub/Sub messages read from the topic by the Pub/Sub IO connector).

3. Select the **Time Window** node in the graph. Observe how the Elements Added metric under the Input Collections of the **Time Window** step matches the Elements Added metric under the Output Collections of the previous step **GetMessages**.

Task 7: Review BigQuery output

1. Return to the BigQuery web UI.

Note: streaming data and tables may not show up immediately, and the Preview feature may not be available for data that is still in the streaming buffer.

If you click on **Preview** you will see the message "This table has records in the streaming buffer that may not be visible in the preview." You can still run queries to view the data.

2. In the **Query editor** window, type (or copy-and-paste) the following query. Use the following query to observe the output from the Dataflow job. Click **Run**.

```
SELECT *
FROM `demos.average_speeds`
ORDER BY timestamp DESC
LIMIT 100
```

3. Find the last update to the table by running the following SQL.

```
SELECT
MAX(timestamp)
FROM
`demos.average_speeds`
```

4. Next use the time travel capability of BigQuery to reference the state of the table at a previous point in time.

The query below will return a subset of rows from the **average_speeds** table that existed at 10 minutes ago.

If your query requests rows but the table did not exist at the reference point in time, you will receive the following error message:

```
Invalid snapshot time 1633691170651 for Table PROJECT:DATASET.TABLE__
```

If you encounter this error please reduce the scope of your time travel by lowering the minute value.

```
SELECT *
FROM `demos.average_speeds`
FOR SYSTEM_TIME AS OF TIMESTAMP_SUB(CURRENT_TIMESTAMP, INTERVAL
10 MINUTE)
ORDER BY timestamp DESC
LIMIT 100
```

Task 8: Observe and understand autoscaling

Observe how Dataflow scales the number of workers to process the backlog of incoming Pub/Sub messages.

1. Return to the browser tab for Console. On the **Navigation menu** (☰), click **Dataflow** and click on your pipeline job.

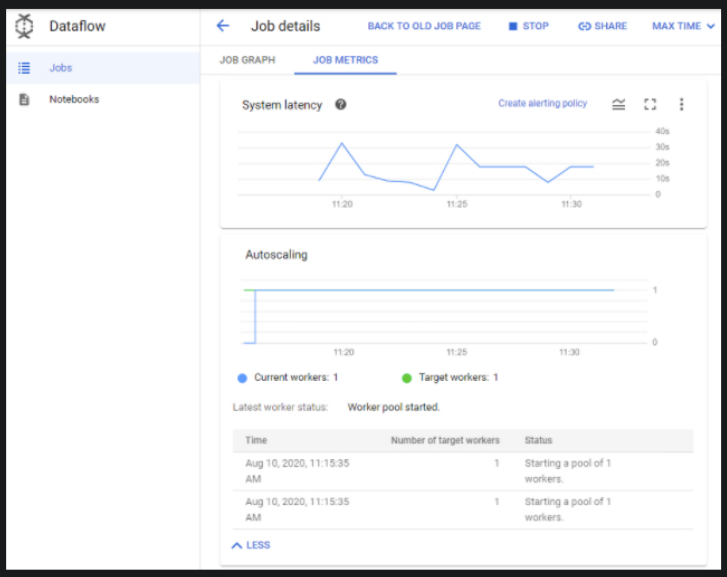
2. Examine the **JOB METRICS** panel on the right, and review the **Autoscaling** section. How many workers are currently being used to process messages in the Pub/Sub topic?

3. Click on **More history** and review how many workers were used at different points in time during pipeline execution.

4. The data from a traffic sensor simulator started at the beginning of the lab creates hundreds of messages per second in the Pub/Sub topic. This will cause Dataflow to increase the number of workers to keep the system lag of the pipeline at optimal levels.

5. Click on **More history**. In the **Worker pool**, you can see how Dataflow changed the

number of workers. Notice the **Status** column that explains the reason for the change.



Task 9: Refresh the sensor data simulation script

Note: the training lab environment has quota limits. If the sensor data simulation script runs too long it will pass a quota limit, causing the session credentials to be suspended.

1. Return to the **training-vm** SSH terminal where the sensor data script is running.

2. If you see messages that say **INFO: Publishing** then the script is still running. Press **CRTL+C** to stop it. Then issue the command to start the script again.

```
cd ~/training-data-analyst/courses/streaming/publish
./send_sensor_data.py --speedFactor=60 --project
$DEVSHELL_PROJECT_ID
```

3. If the script has passed the quota limit, you will see repeating error messages that "credentials could not be refreshed" and you may not be able to use **CTRL+C** to stop the script. Simply close the SSH terminal. Open a new SSH terminal. The new session will have a fresh quota.

4. In the Console, on the **Navigation menu** (☰), click **Compute Engine** > **VM instances**.

5. Locate the line with the instance called **training-vm**.

6. On the far right, under **Connect**, click on **SSH** to open a third terminal window.

7. In the **training-vm** SSH terminal, enter the following to create environment variables.

```
source /training/project_env.sh
```

8. Use the following commands to start a new sensor simulator.

```
cd ~/training-data-analyst/courses/streaming/publish
./send_sensor_data.py --speedFactor=60 --project
$DEVSHELL_PROJECT_ID
```

Task 10: Cloud Monitoring integration

Cloud Monitoring integration with Dataflow allows users to access Dataflow job metrics such as System Lag (for streaming jobs), Job Status (Failed, Successful), Element Counts, and User Counters from within Cloud Monitoring.

Integration features of Cloud Monitoring

- **Explore Dataflow Metrics**: Browse through available Dataflow pipeline metrics and visualize them in charts.

Some common Dataflow metrics.

Metrics	Features
Job status	Job status (Failed, Successful), reported as an enum every 30 secs and on update.
Elapsed time	Job elapsed time (measured in seconds), reported every 30 secs.
System lag	Max lag across the entire pipeline, reported in seconds.
Current vCPU count	Current # of virtual CPUs used by job and updated on value change.
Estimated byte count	Number of bytes processed per PCollection.

- **Chart Dataflow metrics in Monitoring Dashboards**: Create Dashboards and chart time series of Dataflow metrics.

- **Configure Alerts**: Define thresholds on job or resource group-level metrics and alert when these metrics reach specified values. Monitoring alert can notify on a variety of conditions such as long streaming system lag or failed jobs.

- **Monitor User-Defined Metrics**: In addition to Dataflow metrics, Dataflow exposes user-defined metrics (SDK Aggregators) as Monitoring custom counters in the Monitoring UI, available for charting and alerting. Any Aggregator defined in a Dataflow pipeline will be reported to Monitoring as a custom metric. Dataflow will define a new custom metric on behalf of the user and report incremental updates to Monitoring approximately every 30 seconds.

Task 11: Explore metrics

Cloud monitoring is a separate service in Google Cloud Platform. So you will need to go through some setup steps to initialize the service for your lab account.

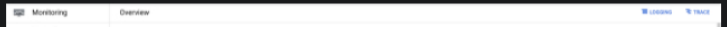
Create a Monitoring workspace

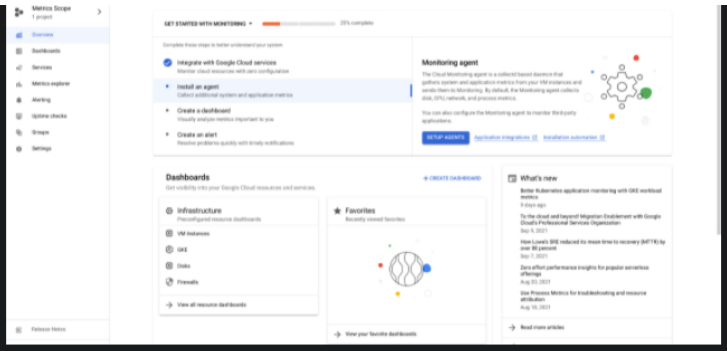
You will now setup a Monitoring workspace that's tied to your Qwiklabs GCP Project. The following steps create a new account that has a free trial of Monitoring.

1. In the Google Cloud Platform Console, click on **Navigation menu** > **Monitoring**.

2. Wait for your workspace to be provisioned.

When the Monitoring dashboard opens, your workspace is ready.





3. In the panel to the left click on **Metrics Explorer**.

4. In the Metrics Explorer, under **Resource & Metric** click on **SELECT A METRIC**.

5. Select `Dataflow Job > Job` You should see a list of available Dataflow-related metrics. Select **Data watermark lag** and click **Apply**.

6. Cloud monitoring will draw a graph on the right side of the page.

7. Under metric, click on the **Reset** to remove the **Data watermark lag** metric. Select a new dataflow metric **System lag**.

> **Note:** the metrics that Dataflow provides to Monitoring are listed here. You can search on the page for Dataflow. The metrics you have viewed are useful indicators of pipeline performance.
>
> **Data watermark lag:** The age (time since event timestamp) of the most recent item of data that has been fully processed by the pipeline.
>
> **System lag:** The current maximum duration that an item of data has been awaiting processing, in seconds.

Task 12: Create alerts

If you want to be notified when a certain metric crosses a specified threshold (for example, when System Lag of our lab streaming pipeline increases above a predefined value), you could use the Alerting mechanisms of Monitoring to accomplish that.

Create an alert

1. On the Cloud Monitoring, click **Alerting**.

2. Click **+ Create Policy**.

3. Click **Add Condition**.

4. In the **Target** section, set the **RESOURCE TYPE** to **Dataflow Job**.

5. Set the **Metric** to **System Lag**.

6. Under **Configuration** set **CONDITION** to **is above**.

7. Set **THRESHOLD** to **5**.

8. Set **FOR** to **1 minute**.

9. Click **Add**.

10. Click **Next**.

Add a notification

11. Click on drop down arrow next to **Notification Channels**, then click on **Manage Notification Channels**.

A **Notification channels** page will open in new tab.

12. Scroll down the page and click on **ADD NEW** for **Email**.

13. In **Create Email Channel** dialog box, enter the Qwiklabs username as the **Email Address** field and a **Display name**.

> **Note:** if you enter your own email address, you might get alerts until all the resources in the project have been deleted.

14. Click **Save**.

15. Go back to the previous **Create alerting policy** tab.

16. Click on **Notification Channels** again, then click on the **Refresh icon** to get the display name you mentioned in the previous step.

17. Now, select your **Display name** and click **OK**.

18. Click **Next**.

19. Set **Alert name** as **MyAlertPolicy**.

20. Skip the Documentation step.

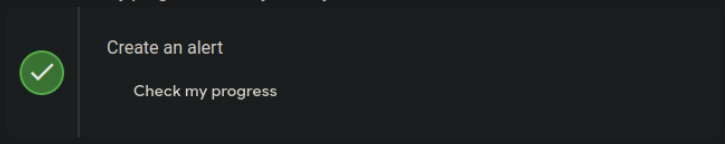
21. Click **Save**.

View events

22. On the Cloud Monitoring tab, click on **Alerting > Policies**.

23. Every time an alert is triggered by a Metric Threshold condition, an Incident and a corresponding Event are created in Monitoring. If you specified a notification mechanism in the alert (email, SMS, pager, etc), you will also receive a notification.

Click Check my progress to verify the objective.



Task 13: Set up dashboards

You can easily build dashboards with the most relevant Dataflow-related charts with Cloud Monitoring Dashboards.

1. In the left pane, click **Dashboards**.

2. Click **+Create Dashboard**.

3. For **New Dashboard Name**, type **My Dashboard**.

4. Click **Line Chart**.

5. Click on the `dropdown` box under **Resource & Metric**.

6. Select `Dataflow > Job > System Lag` and click **Apply**.

7. In the **Filters** panel, click **+ Add Filter**.

8. Select **project_id** in `Label` field, then select or type your *GCP project ID* in the `Value` field.

9. Click **Done**.

Example:

You can add more charts to the dashboard, if you would like, for example, Pub/Sub publish rates on the topic, or subscription backlog (which is a signal to the Dataflow auto-scaler).

Task 14: Launch another streaming pipeline

1. Return to **training-vm** SSH terminal, if you see messages that say **INFO: Publishing** then the script is still running. Press **CRTL+C** to stop it. Then issue the command to start the script again.

2. Examine the **CurrentConditions.java** application. **Do not make any changes to the code.**

```
cd ~/training-data-analyst/courses/streaming/process/sandiego/src/main/java/com/google
cat CurrentConditions.java
```

3. Copy-and-paste the following URL into a browser tab to view the source code on Github.

```
https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/courses/streaming/process/sandiego/src/main/jav
```

What does the script do?

4. Run the **CurrentConditions.java** code in a new Dataflow pipeline; this script is simpler in the sense that it does not do many transforms like **AverageSpeeds**. The results will be used in the next lab to build dashboards and run some transforms (functions) while retrieving data from BigQuery.

5. In the other **training-vm** SSH terminal, enter the following:

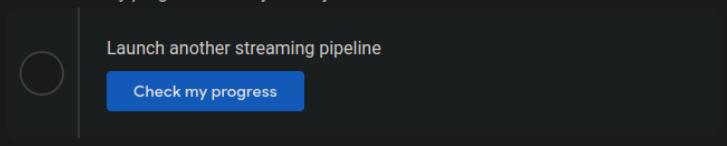
```
cd ~/training-data-analyst/courses/streaming/process/sandiego
./run_oncloud.sh $DEVSHELL_PROJECT_ID $BUCKET CurrentConditions
```

6. Return to the browser tab for Console. On the **Navigation menu** (☰), click **Dataflow** and click on the new pipeline job. Confirm that the pipeline job is listed and verify that it is running without errors.

running without errors.

7. It will take several minutes before the **current_conditions** table appears in BigQuery.

Click Check my progress to verify the objective.



End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.