Proprietary + Confidential

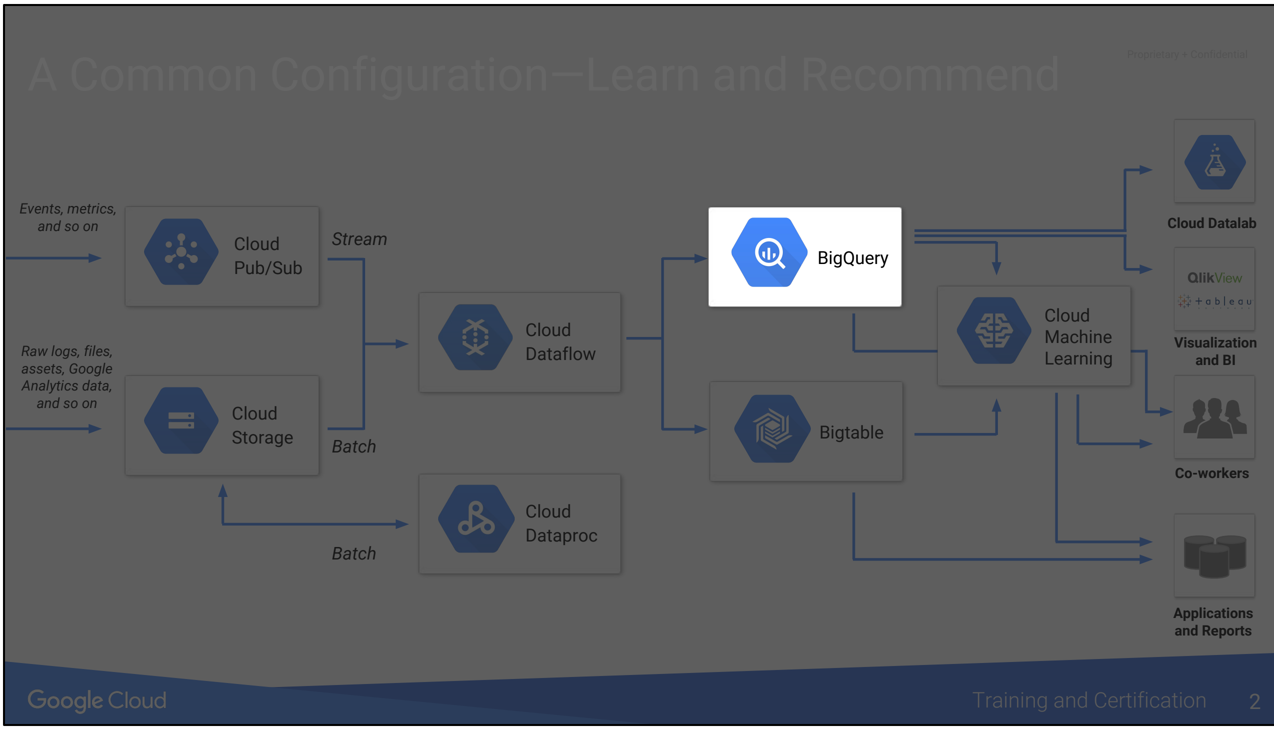
Google Cloud

Streaming analytics and dashboards

Data Engineering on Google Cloud Platform

Google Cloud

1 hour, including lab



Notes:

How our engagement model has changed:

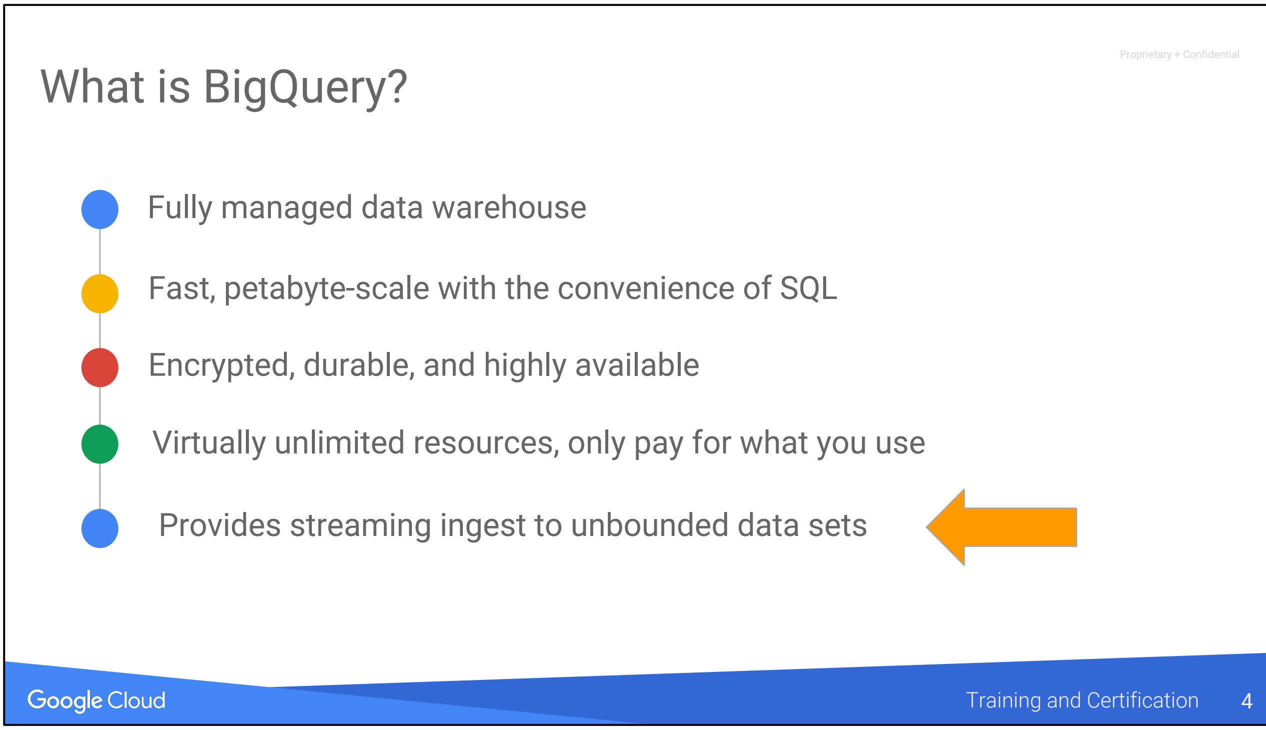
And now, in the new model, we publish the code in tandem with the paper. Case in point, for Dataflow we put the SDK out first (12/2014) and then published the paper (8/2015).

Graphically, the most obvious way to do this would be to use the timeline slide as starting point and do a build-up in 4 steps:

- Show our papers (GFS, MapReduce, Dremel, BigTable, FlumeJava, Millwheel)
- Show how they were followed by open source implementations (see list above)
- Show how for Dataflow we published the SDK ourselves, and it was followed by the paper

Agenda

Streaming analytics & dashboards



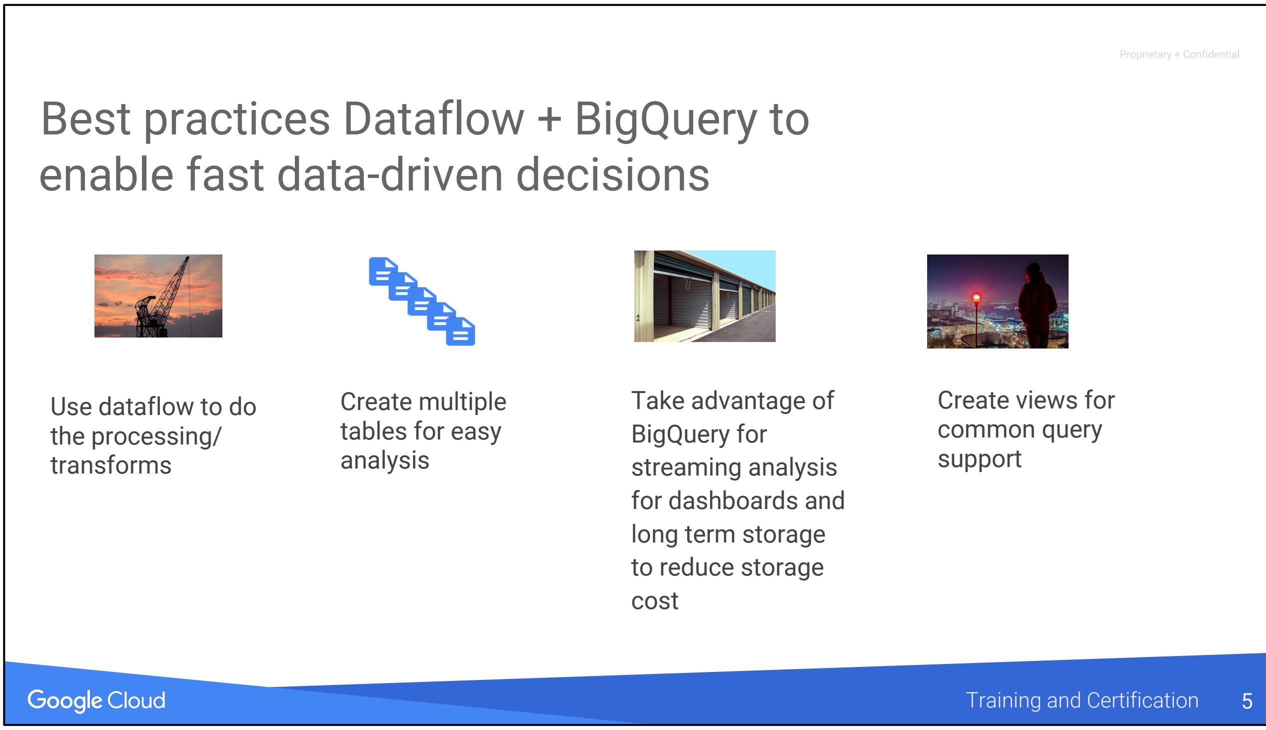
Notes:

We typically think and and seen BigQuery as place to store data and run queries….pretty much on data at rest or not changing live.

With the previous lab, we saw how we can run contiguous queries as data comes in real time.

Managed services help reduce the cost and complexity of building systems at internet scale.

Datastore is essentially Bigtable….but lets you query on attribute keys. It adds semantics.



Notes:

Use Cloud Dataflow to produce aggregations to support common queries

- Process all of the data : Cloud Dataflow is an excellent ETL solution for BigQuery
- Dataflow SDKs have built-in transforms that read/write into BigQuery very easily

Ingest (stream || batch) all non/slowly-changing data into BigQuery

- Take advantage of BigQuery long term storage to reduce storage cost

Create views for common query support

Templates allow automatic table creation

- Split table into smaller tables without adding client-side code

Crane: https://pixabay.com/en/harbour-crane-sunset-sky-clouds-1643476/ (cc0)
Storage unit:
https://pixabay.com/en/storage-warehouse-storage-bins-2089775/(cc0)

View: https://pixabay.com/en/man-view-from-the-roof-618344/(cc0)

Streaming data into BigQuery

- BigQuery provides streaming ingestion at a rate of 100,000 rows/table/second
  - Provided by the REST APIs `tabledata().insertAll()` method
  - Works for partitioned and standard tables
- Streaming data can be queried as it arrives
  - Data available within seconds
- For data consistency, enter `insertId` for each inserted row
  - De-duplication is based on a best-effort basis, and can be affected by network errors
  - Can be done manually

Notes:

You might have to retry an insert because there's no way to determine the state of a streaming insert under certain error conditions, such as network errors between your system and BigQuery or internal errors within BigQuery. This means you may end up with duplicates. If you retry an insert, use the same insertID for the same set of rows so that BigQuery can attempt to de-duplicate your data.

To help ensure data consistency, you can supply insertId for each inserted row. BigQuery remembers this ID for at least one minute. If you try to stream the same set of rows within that time period and the insertId property is set, BigQuery uses the insertId property to de-duplicate your data on a best effort basis.
De-duplication is based on a best-effort basis, and can be affected by network errors.

You can use the following manual process to ensure that no duplicate rows exist after you are done streaming.

1. Add the insertID as a column in your table schema and include the insertID value in the data for each row.
2. After streaming has stopped, perform the following query to check for

1. duplicates:
2. `#standardSQL`
3. `SELECT MAX(count) FROM(`
4. `  SELECT[ID_COLUMN],count(*) as count`
5. `  FROM `[TABLE_NAME]``
6. `  GROUP BY`
7. `    [ID_COLUMN])`
8. If the result is greater than 1, duplicates exist.
9. To remove duplicates, perform the following query. You should specify a destination table, allow large results, and disable result flattening.
10. `#standardSQL`
11. `SELECT`
12. `  * EXCEPT(row_number)`
13. `FROM (`
14. `  SELECT`
15. `    *,`
16. `    ROW_NUMBER()`
17. `          OVER (PARTITION BY [ID_COLUMN]) row_number`
18. `  FROM`
19. `    `[TABLE_NAME]`)`
20. `WHERE`
21. `  row_number = 1`

Notes about the duplicate removal query:

- The safer strategy for the duplicate removal query is to target a new table. Alternatively, you can target the source table with write disposition `WRITE_TRUNCATE`.
- The duplicate removal query adds a `row_number` column with the value `1` to the end of the table schema. The query uses a SELECT * EXCEPT statement from standard SQL to exclude the `row_number` column from the destination table. The `#standardSQL` prefix enables standard SQL for this query. Alternatively, you can can select by specific column names to omit this column.
- For querying live data with duplicates removed, you can also create a

- view over your table using the duplicate removal query. Be aware that query costs against the view will be calculated based on the columns selected in your view, which can result in large bytes scanned sizes.

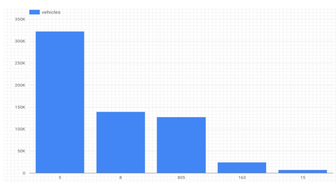
The stuff that powers your near real-time dashboards

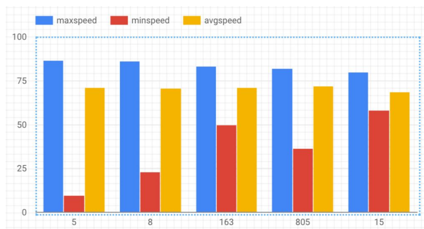
```
SELECT
  s.*
FROM
  demos.current_conditions AS s
JOIN (
  SELECT
    sensorId,
    MAX(timestamp) AS timestamp
  FROM
    demos.current_conditions
  GROUP BY
    sensorId) AS c
ON
  s.sensorId = c.sensorId
  AND s.timestamp = c.timestamp
```

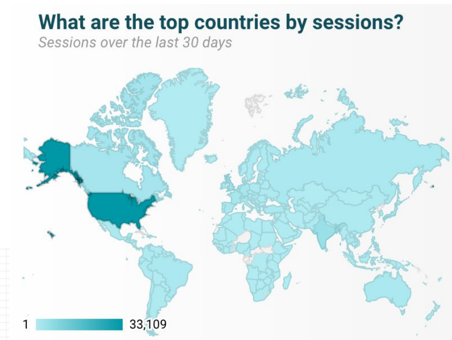
1	2008-11-01 02:10:00 UTC	32.67696	-117.109463	5	S	4	65.0	32.67696,-117.109463,5,S,4
2	2008-11-01 02:10:00 UTC	32.817541	-117.16012	805	S	5	65.4	32.817541,-117.16012,805,S,5
3	2008-11-01 02:10:00 UTC	32.743125	-117.184141	5	S	5	65.3	32.743125,-117.184141,5,S,5
4	2008-11-01 02:10:00 UTC	32.759802	-117.187519	8	W	2	69.2	32.759802,-117.187519,8,W,2
5	2008-11-01 02:10:00 UTC	32.67696	-117.109463	5	S	4	65.0	32.67696,-117.109463,5,S,4
6	2008-11-01 02:10:00 UTC	32.793785	-117.149874	805	S	3	81.7	32.793785,-117.149874,805,S,3
7	2008-11-01 02:10:00 UTC	32.762484	-117.163806	8	W	3	64.5	32.762484,-117.163806,8,W,3

Data Studio lets you build dashboards and reports

- Easy to read, share, and fully customizable
- Handles authentication, access rights, and structuring of data

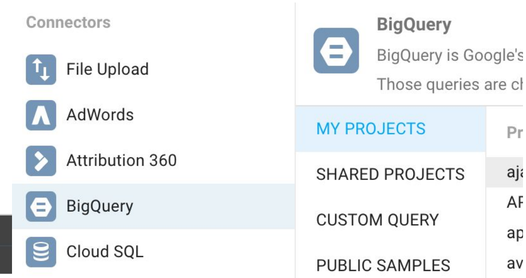


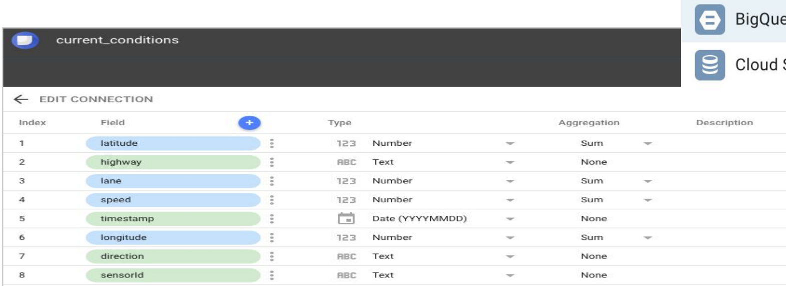




Data Studio connects to various GCP data sources

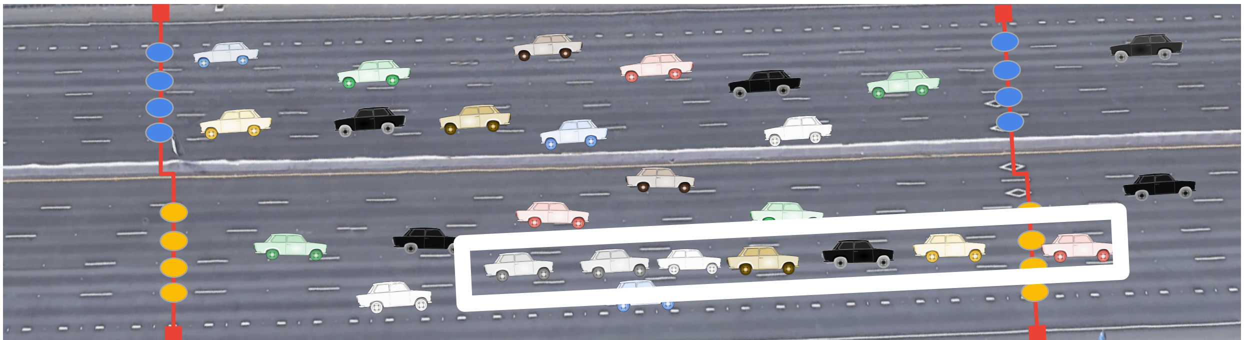
- Offers a BigQuery connector
- Read from table or run a custom query
- Build charts, graphs or lay it on a map





Index	Field	Type		Aggregation	Description
1	latitude	123	Number	Sum	
2	highway	ABC	Text	None	
3	lane	123	Number	Sum	
4	speed	123	Number	Sum	
5	timestamp		Date (YYYYMMDD)	None	
6	longitude	123	Number	Sum	
7	direction	ABC	Text	None	
8	sensorId	ABC	Text	None	

If a lane is significantly slower than other lanes at the same location, issue an accident alert



Notes:

Car image: https://pixabay.com/en/trabant-car-transport-white-drive-782799/ (cc0)
The traffic sensors report the speed of the traffic in the lane by computing # of cars that cross in some time-period (it is not clear how long that time-period is) -- let's assume 30s, which is the frequency at which the data are reported.

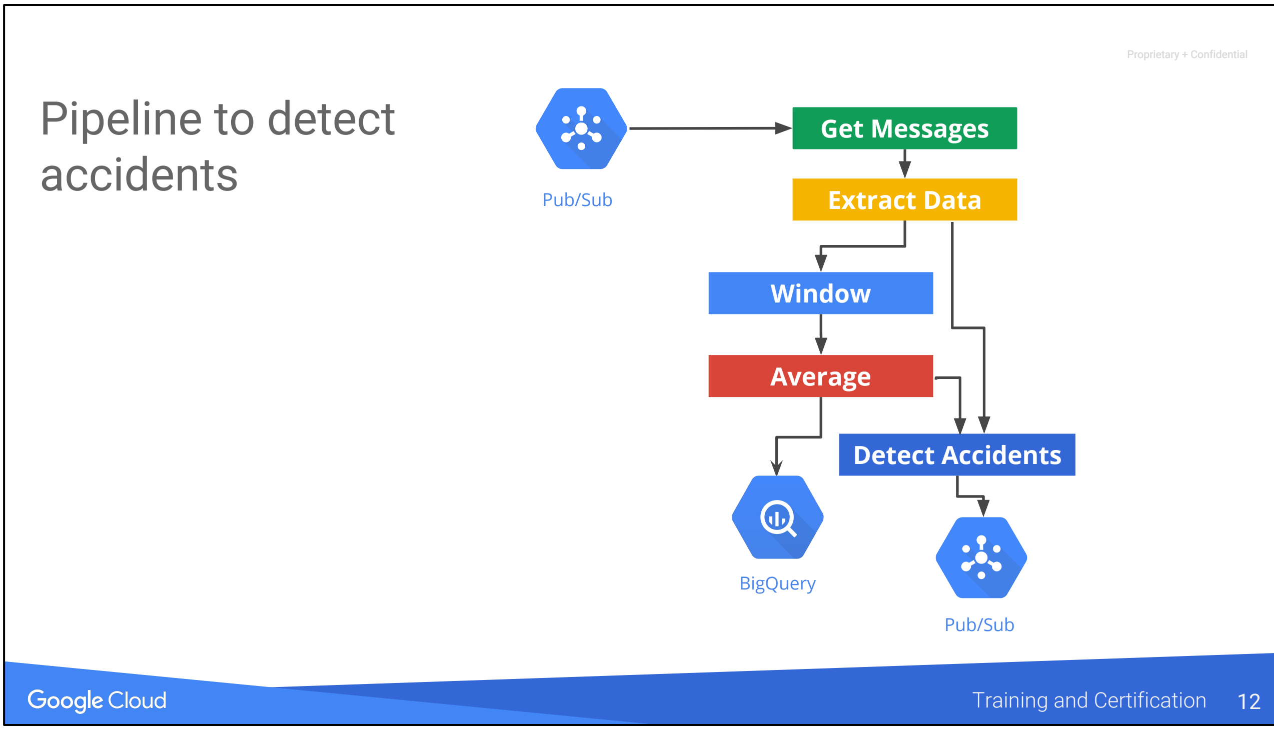
Running transformations to detect anomaly in pattern

For each window, we calculate the average over all lanes at sensor location, and then compare the lane speed with this average.

```
p.apply(PubsubIO.readStrings().fromTopic(t))
 .apply("TimeWindow",
        Window.into(SlidingWindows
        .of(Duration.standardSeconds(300))
    .every(Duration.standardSeconds(60)))
.apply(ParDo.of(new ExtractData()))
.apply(ParDo.of(new AvgByLocation()))
.apply(ParDo.of(new FindSlowDowns()))
.apply(BigQueryIO.writeTableRows().to(tbl))
```

https://github.com/GoogleCloudPlatform/training-data-analyst/blob/master/courses/streaming/process/sandiego/src/main/java/com/google/cloud/training/dataanalyst/sandiego/AccidentAlert.java

Pseudocode. Actual code in github



Notes:

Note that DetectAccidents uses the average speed at each location as a side-input ...

Writing into BigQuery

Record averages for each interval in storage for further/future analysis

```
p.apply(PubsubIO.readStrings().fromTopic(t))
 .apply("TimeWindow",
        Window.into(SlidingWindows
        .of(Duration.standardSeconds(300))
    .every(Duration.standardSeconds(60)))
.apply(ParDo.of(new ExtractData()))
.apply(ParDo.of(new AvgByLocation()))
.apply(ParDo.of(new FindSlowDowns()))
.apply(BigQueryIO.writeTableRows().to(tbl))
```

Agenda

Lab: Streaming Data Processing

Part 3: Streaming analytics & dashboards

Lab: Streaming data analytics and dashboards

In this lab, you will

- Create a streaming pipeline to process traffic events from Pub/Sub
- Perform transformations on the data (e.g., averages) and detect anomalies (compare averages and detect which lane is slower than rest
- Store results in BigQuery for future analysis


cloud.google.com
Images by Connie Zhou