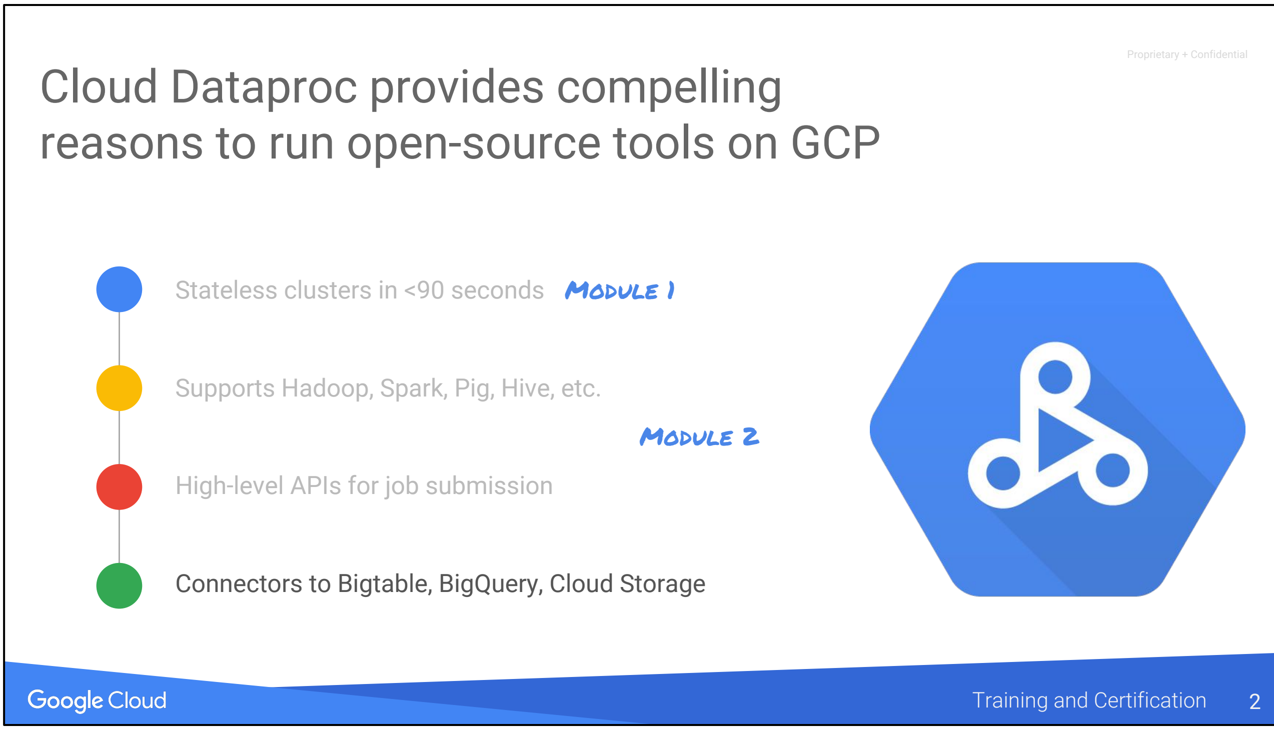
Leveraging GCP

Data Engineering on Google Cloud Platform

Notes:

25 slides + 1 lab: 1 hour



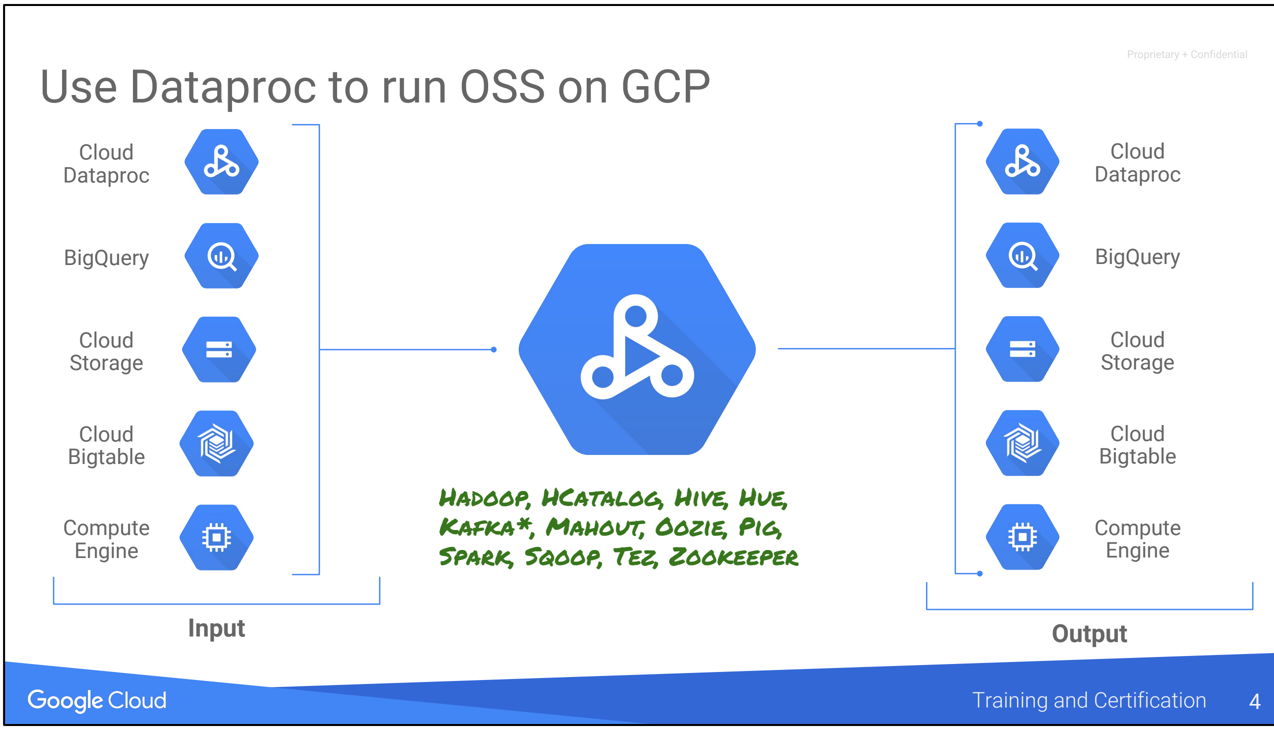
Notes:

We have already looked at #1 to #3. Let's look at #3 here.

Agenda

Customizing clusters + Lab

BigQuery support



Notes:

You can read from GCP sources and write to GCP sources, and use Dataproc as the intermingling glue.

Kafka support is experimental at present.

Dataproc uses Apache Bigtop

- Conservative about which packages are installed by default

Notes:

To ensure that clusters are performant and resources are not squandered on un-needed stuff.
https://dataproc-bigtop-repo.storage.googleapis.com

But about OSS that's not already installed?

WOULDN'T IT BE NICE IF WE
COULD CREATE DATAPROC
CLUSTERS WITH SPECIFIC
SOFTWARE PRE-INSTALLED?

MASTER?
WORKERS?
BOTH MASTER + WORKERS?

Notes:

https://pixabay.com/en/boy-idea-sad-eyes-school-thinking-1867332/ (cc0)

Not as simple as a deployment manager because we need to know whether to install it on the master-only or workers-only.

Like … Cloud Datalab?

CAN I RUN CLOUD DATALAB ON THE MASTER?

WITH INPUT AS BIGQUERY?

PREPROCESS DATA WITH SPARK?

TRAIN A TENSORFLOW MODEL ON CLOUD ML?

Notes:

https://pixabay.com/en/boy-idea-sad-eyes-school-thinking-1867332/ (cc0)

BigQuery & Cloud ML are serverless, so that’s easy. You can do it from anywhere. But datalab & spark do need a machine to run on.

To install software on Dataproc cluster...

1. Write an executable program (bash, python, etc.)
2. Upload it to Cloud Storage
3. Specify GCS location in Dataproc creation command

1. Write executable program that runs as root

SHEBANG (#!) SPECIFIES WHAT LANGUAGE INTERPRETER TO INVOKE

```
#!/bin/bash

apt-get update || true


apt-get install -y python-numpy python-scipy python-matplotlib python-pandas
```

-Y TO ENSURE THAT
SCRIPT DOESN'T WAIT
FOR USER INPUT

Notes:

Because the script is run as root, there is no need to use “sudo”.

This installs a set of python packages on all nodes.

If you don’t have the -y, the installer will wait (default timeout = 10 minutes) before failing.

Can carry out tasks only on the master node, or only on the worker nodes


```
#!/bin/bash
apt-get update || true

ROLE=$(/usr/share/google/get_metadata_value attributes/dataproc-role)
if [[ "${ROLE}" == 'Master' ]]; then
   apt-get install -y vim
else
   # something that goes only on worker
Fi

# things that go on both

apt-get install -y python-numpy python-scipy python-matplotlib python-pandas
```


Notes:

/usr/share/google is present on all Dataproc nodes

In this case, we are installing the editor “vim” only on the master node.

Proprietary + Confidential

2. Upload it to Google Cloud Storage (GCS)

```
gsutil cp my_init.sh gs://mybucket/init-actions/my_init.sh
```

A library of pre-built initialization actions are hosted in this publicly-accessible bucket:

```
gs://dataproc-initialization-actions
```

See the GitHub repository at https://github.com/GoogleCloudPlatform/dataproc-initialization-actions

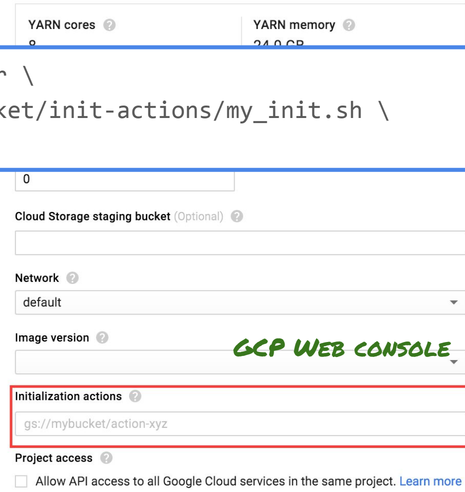
Notes:

Click on the link to browse the publicly hosted ones.

3. Specify GCS location when creating cluster

```
gcloud dataproc clusters create mycluster \
    --initialization-actions  gs://mybucket/init-actions/my_init.sh \
    --initialization-action-timeout 3m
```

GCLOUD SDK



GCP WEB CONSOLE

Notes:

Here, we are changing the timeout to be 3 minutes. Changing the timeout could be necessary for things like establishing database replicas etc. which might take time.

Separate multiple initialization actions by commas.

You can do it on the web console also.

Set cluster properties

Initialization actions

Optional executable scripts (Shell, Python, etc.) which run when your cluster starts

Allows you to install additional components, stage files, or change the node

We provide a set of common initialization actions on GitHub

Cluster properties

Allows you to modify properties in common configuration files, like `core-site.xml`

Removes the need to manually change property files by hand or initialization action

Specified by `file_prefix:property=value` in gcloud SDK

Notes:

Cluster properties not currently available on web-UI.

Proprietary + Confidential

Lab - Leveraging Unstructured Data : Part 4

- Create a Dataproc cluster with an Initialization Action that installs Google Cloud Datalab
- Run Jupyter Notebooks on the Dataproc cluster using Google Cloud Datalab
- Create Python and PySpark jobs that utilize Google Cloud Storage, BigQuery, and Spark

Notes:

Datalab, BigQuery & Spark.

Agenda

BigQuery support

Proprietary + Confidential

Where was the computational work being done?

```
projectId = "YOUR-PROJECT-ID-HERE"
sql = "SELECT year, month, day, weight_pounds FROM [publicdata:samples.natality] LIMIT 50"

print 'Running query...'
data = gbq.read_gbq(sql, project_id=projectId)

data[:5]
```

```
Running query...
Requesting query... ok.
Query running...
Query done.
Processed: 3.5 Gb

Retrieving results...
Got 50 rows.

Total time taken 1.14 s.
Finished at 2017-02-12 22:20:13.
```

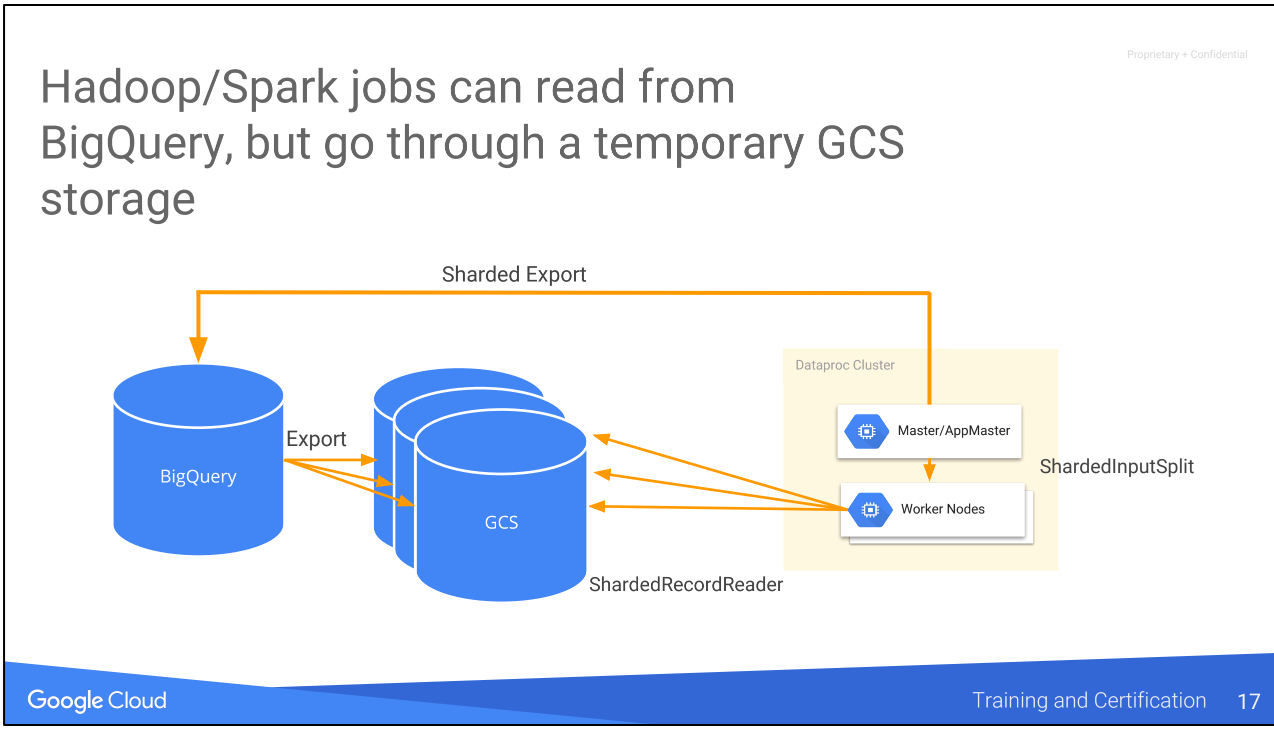
	year	month	day	weight_pounds
0	1969	3	9.0	8.875811
1	1969	3	25.0	7.874912
2	1969	1	6.0	7.063611
3	1970	6	8.0	5.813590
4	1971	8	22.0	5.749656

Notes:

In the lab, most of the work was done in BigQuery. Notice that what comes back is only 50 rows.

We then read the results from BQ directly into a *pandas* dataframe. But what if you want to process the dataset in your Dataproc cluster? You need to read into a RDD or Spark Dataframe in order to do that …. The Pandas dataframe is in-memory and won't support it.

You can do this, but it involves import/export to GCS ..



Notes:

See
https://cloud.google.com/hadoop/examples/bigquery-connector-spark-example
Hadoop/Spark job begins immediately, reading export results as they come

If the job fails, you may need to manually remove any remaining temporary Google Cloud Storage files, BigQuery datasets, and BigQuery tables. Typically, you'll find temporary BigQuery exports used by InputFormat in gs://bucket/hadoop/tmp/bigquery/ and temporary datasets named after your specified output dataset with a hadoop_temporary_job_[jobid] suffix.

1. Set up connector to read from BQ

```
sc = pyspark.SparkContext()
bucket = sc._jsc.hadoopConfiguration().get('fs.gs.system.bucket')
project = sc._jsc.hadoopConfiguration().get('fs.gs.project.id')
input_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_input'.format(bucket)
conf = {
    # Input Parameters
    'mapred.bq.project.id': project,
    'mapred.bq.gcs.bucket': bucket,
    'mapred.bq.temp.gcs.path': input_directory,
    'mapred.bq.input.project.id': 'publicdata',
    'mapred.bq.input.dataset.id': 'samples',
    'mapred.bq.input.table.id': 'shakespeare',
}
```

PULL PARAMS FROM GCS CONNECTOR TO SPECIFY THE TEMPORARY GCS DIRECTORY

SPECIFY PARAMETERS FOR BIGQUERY INPUT

Notes:

Essentially dump the BQ table to GCS, so that you can read it from Spark.

The GCS path is the input_directory for pyspark.

2. Load data using the BigQuery connector as a RDD

```
# Load data in from BigQuery.
table_data = sc.newAPIHadoopRDD(
    'com.google.cloud.hadoop.io.bigquery.JsonTextBigQueryInputFormat',
    'org.apache.hadoop.io.LongWritable',
    'com.google.gson.JsonObject',
    conf=conf)
```

EXPORTS THE BQ TABLE AS JSON INTO GCS, THEN READS IT ...

Notes:

No DataFrame support at this time, only RDD.

Also, you can only read a table, not a query. To read the results of a query, first run query in BQ, and export it as a table.

3. The Spark code is as normal


```
# Perform word count.
word_counts = (
    table_data
    .map(lambda (_, record): json.loads(record))
    .map(lambda x: (x['word'].lower(), int(x['word_count'])))
    .reduceByKey(lambda x, y: x + y))

# Display 10 results.
pprint.pprint(word_counts.take(10))
```


Notes:

Datalab, BigQuery & Spark.

4. Output to sharded files in GCS


```
# Stage data formatted as newline-delimited JSON in Google Cloud Storage.
output_directory = 'gs://{}/hadoop/tmp/bigquery/pyspark_output'.format(bucket)
partitions = range(word_counts.getNumPartitions())
output_files = [output_directory + '/part-{:05}'.format(i) for i in partitions]

(word_counts
 .map(lambda (w, c): json.dumps({'word': w, 'word_count': c}))
 .saveAsTextFile(output_directory))
```


Notes:

Output to GCS. You can then call “bq load” if you want the otuput in BQ.

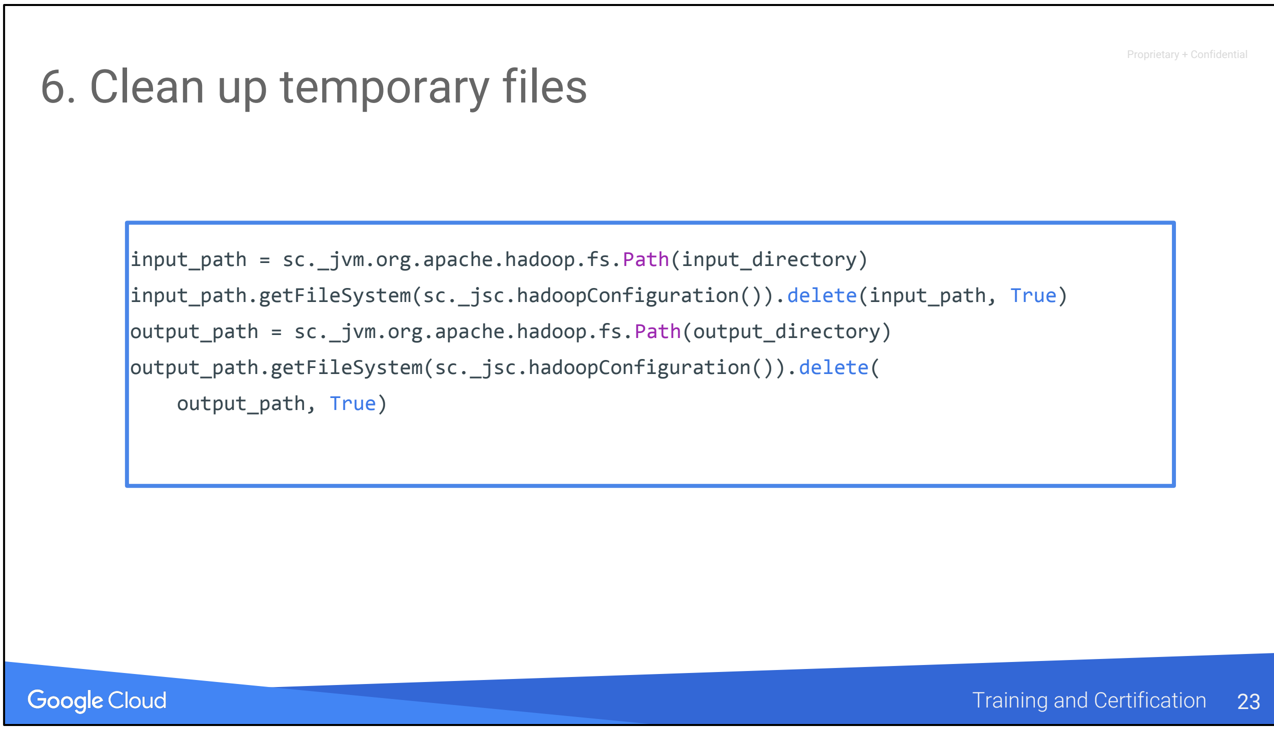
5. Call bq load to ingest GCS files


```
# Output Parameters
output_dataset = 'wordcount_dataset'
output_table = 'wordcount_table'

subprocess.check_call(
    'bq load --source_format NEWLINE_DELIMITED_JSON '
    '--schema word:STRING,word_count:INTEGER '
    '{dataset}.{table} {files}'.format(
        dataset=output_dataset, table=output_table, files=','.join(output_files)
    ).split())
```


Notes:

Calling bq load to go from gCS -> BQ.



Notes:

Clean up the temporary input/output files.

Easier to extract data in BigQuery, pull in the data into Spark cluster for further analysis

```
projectId = "YOUR-PROJECT-ID-HERE"
sql = "SELECT year, month, day, weight_pounds FROM [publicdata:samples.natality] LIMIT 50"

print 'Running query...'
data = gbq.read_gbq(sql, project_id=projectId)

data[:5]
```

```
Running query...
Requesting query... ok.
Query running...
Query done.
Processed: 3.5 Gb

Retrieving results...
Got 50 rows.

Total time taken 1.14 s.
Finished at 2017-02-12 22:20:13.
```

	year	month	day	weight_pounds
0	1969	3	9.0	8.875811
1	1969	3	25.0	7.874912
2	1969	1	6.0	7.063611
3	1970	6	8.0	5.813590
4	1971	8	22.0	5.749656

Notes:

So, this is what we recommend.


cloud.google.com
Images by Connie Zhou