Google Cloud

Leveraging GCP

Data Engineering on Google Cloud Platform

Agenda

Cloud Functions
BigQuery support + Lab
Customizing clusters + Lab

Agenda

BigQuery support + Lab
Customizing clusters + Lab
Workflow Orchestration

Extract data in BigQuery, pull in the data into Spark cluster for further analysis

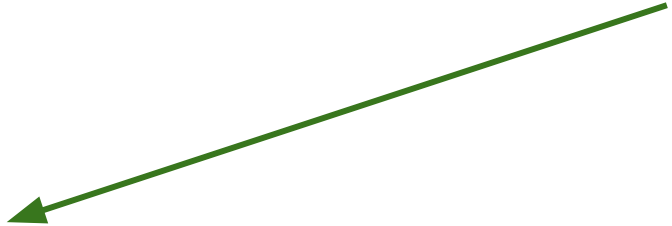
	year	month	day	weight_pounds
1	1969	10	2	9.37626000286
2	1969	7	30	6.8122838958
3	1969	7	1	7.68751907594
4	1969	10	8	8.062304921339999
5	1969	82	24	6.686620406459999

```
projectId = <your-project-id>

sql = "
SELECT
  n.year,
  n.month,
  n.day,
  n.weight_pounds
FROM
   `bigquery-public-data.samples.natality` AS n
ORDER BY
  n.year
LIMIT 50"


print "Running query..."
data = gbq.read_sql.gb1(sql,projectid=projectId)
data [:5]

Running query...
Requesting query… ok.
Query running...
Query done.
Processed 3.5Gb

Retrieving results.
Got 50 rows.
Time taken 1.14 s.
Finished at 2018-02-12 22:20:13
```

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

This example is about the PySpark BigQuery connector

Other connectors work differently

2. Load data using the BigQuery connector as an 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 …

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))
```

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))
```

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())
```

6. Clean up temporary files

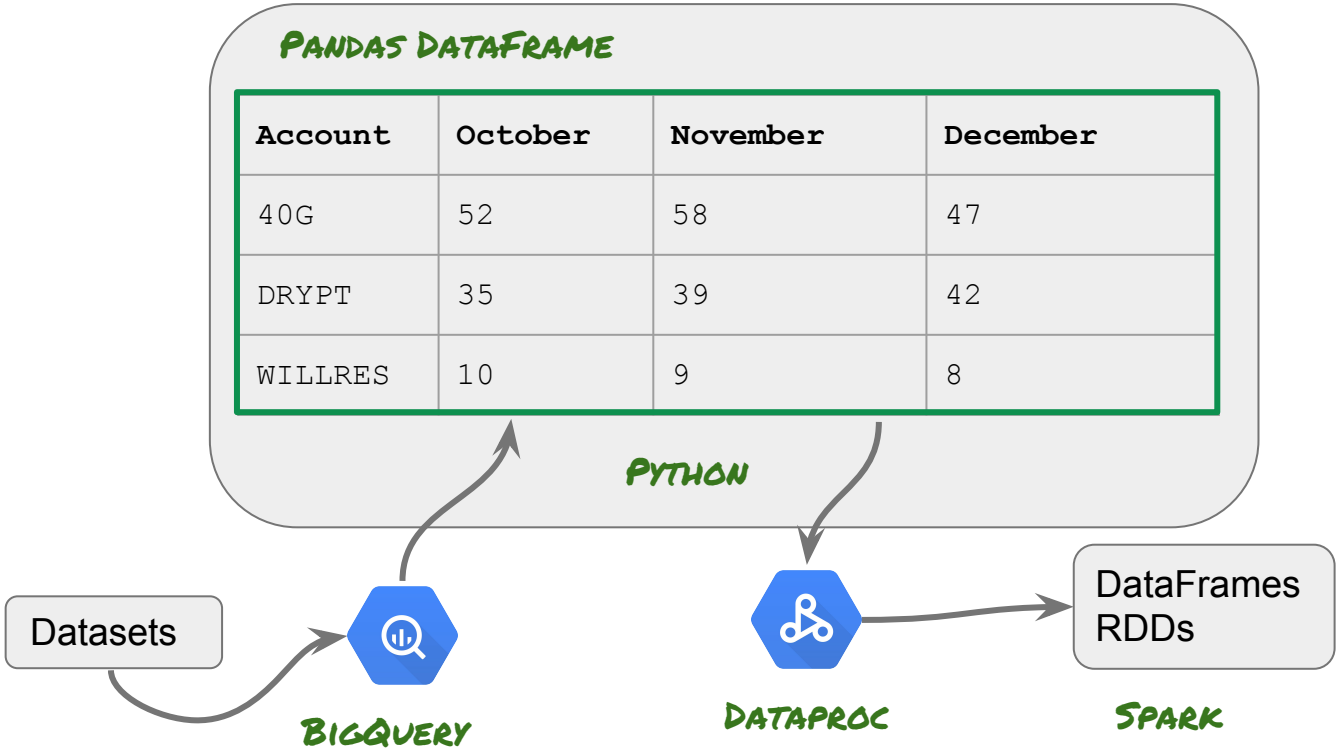
```
input_path = sc._jvm.org.apache.hadoop.fs.Path(input_directory)
input_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(input_path, True)
output_path = sc._jvm.org.apache.hadoop.fs.Path(output_directory)
output_path.getFileSystem(sc._jsc.hadoopConfiguration()).delete(
    output_path, True)
```

Extract data in BigQuery, pull in the data into Spark cluster for further analysis

	year	month	day	weight_pounds
1	1969	10	2	9.37626000286
2	1969	7	30	6.8122838958
3	1969	7	1	7.68751907594
4	1969	10	8	8.062304921339999
5	1969	82	24	6.686620406459999

```
projectId = <your-project-id>

sql = "
SELECT
  n.year,
  n.month,
  n.day,
  n.weight_pounds
FROM
   `bigquery-public-data.samples.natality` AS n
ORDER BY
  n.year
LIMIT 50"


print "Running query..."
data = gbq.read_sql.gb1(sql,projectid=projectId)
data [:5]

Running query...
Requesting query… ok.
Query running...
Query done.
Processed 3.5Gb

Retrieving results.
Got 50 rows.
Time taken 1.14 s.
Finished at 2018-02-12 22:20:13
```

BigQuery integration using Python Pandas



Pandas is a data analysis package for Python

Spark supports DataFrames
Python supports Pandas DataFrames

The Pandas package provides methods to read BigQuery queries into a Pandas DataFrame

Perform additional analysis in Spark or in Python as meets your needs

Lab 4: Leverage GCP

Leverage GCP

- Using Cloud Storage instead of HDFS
- Run a PySpark application from Cloud Storage

Agenda

Customizing clusters + Lab

Cloud Dataproc provides compelling reasons to run open-source tools on GCP

- Stateless clusters in <90 seconds — Module 1
- Supports Hadoop, Spark, Pig, Hive, etc.
- High-level APIs for job submission — Module 2
- Connectors to Bigtable, BigQuery, Cloud Storage

Use Dataproc to run OSS on GCP

Dataproc uses Apache Bigtop

- Conservative about which packages are installed by default

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?

Like … Cloud Datalab?

Can I run Cloud Datalab on the master?

With input as BigQuery?

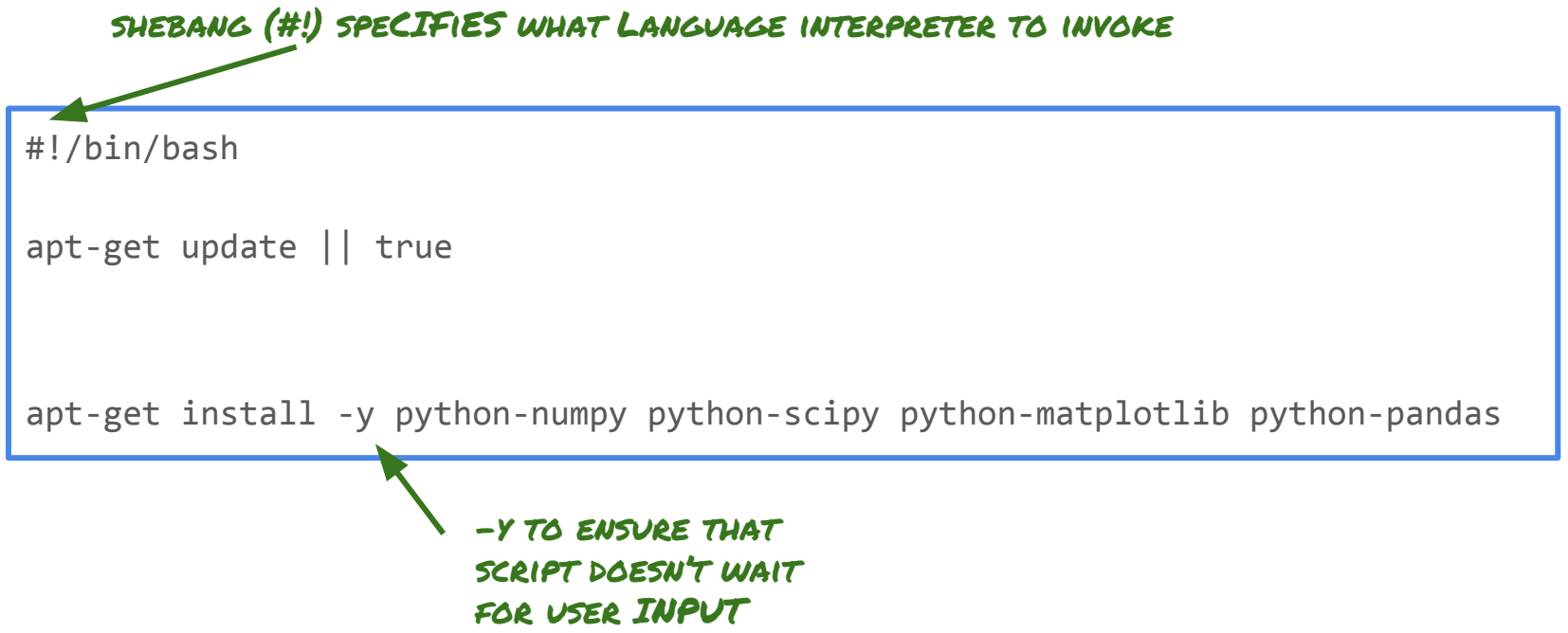
Preprocess data with Spark?

Train a TensorFlow model on Cloud ML?

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



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
```

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:

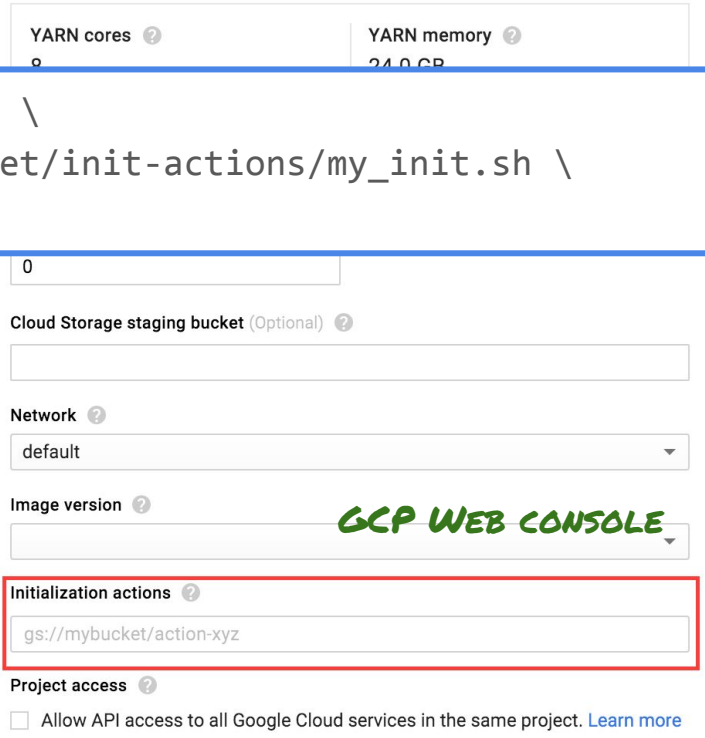
```
gs://dataproc-initialization-actions
```

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

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



Use initialization actions to install custom software and cluster properties to configure Hadoop

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

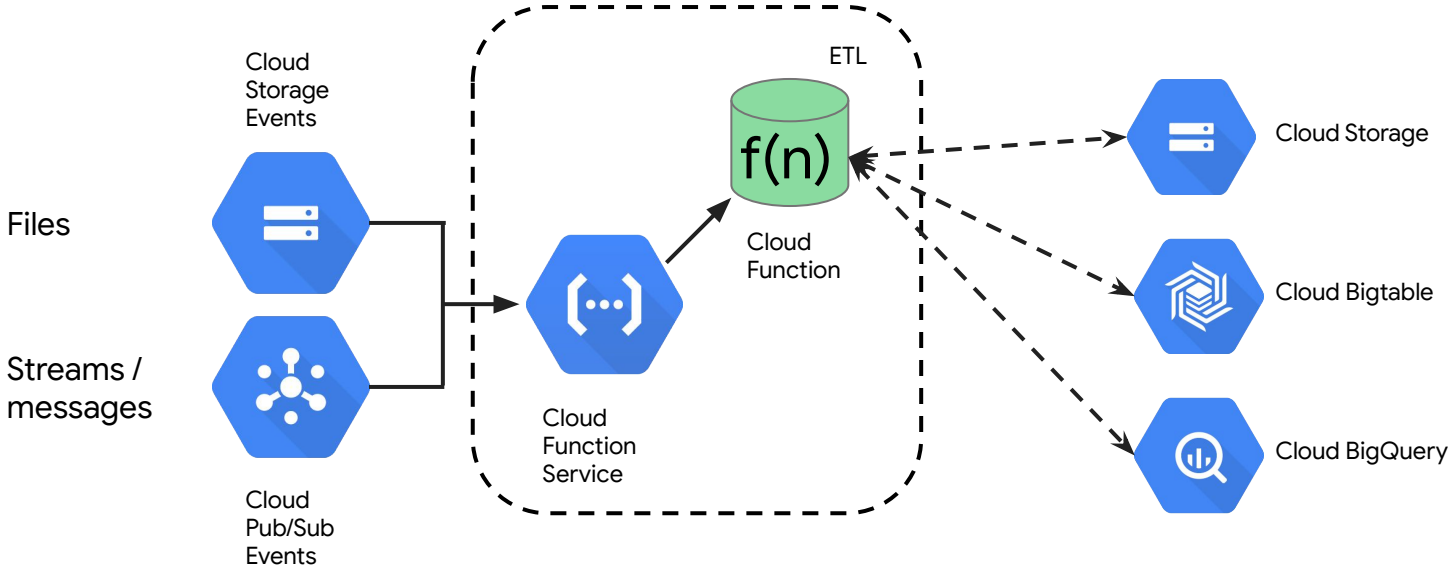
Lab 5: Cluster automation using CLI commands

- Leverage GCP
- Create a customized Dataproc cluster using Cloud Shell CLI commands
- Explore workflow automation

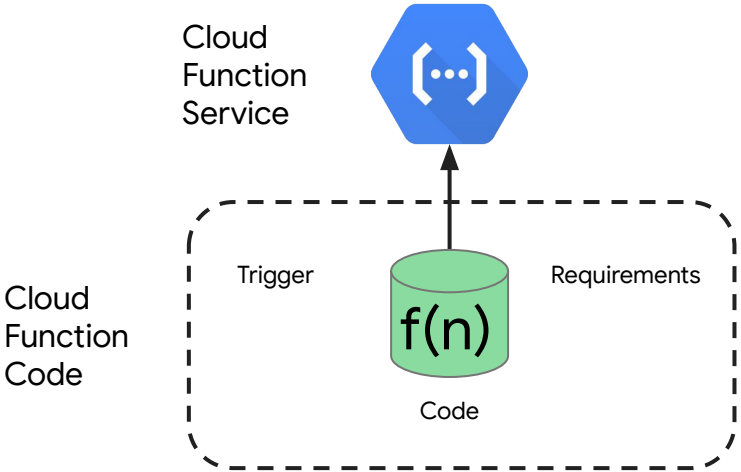
Agenda

Workflow Orchestration

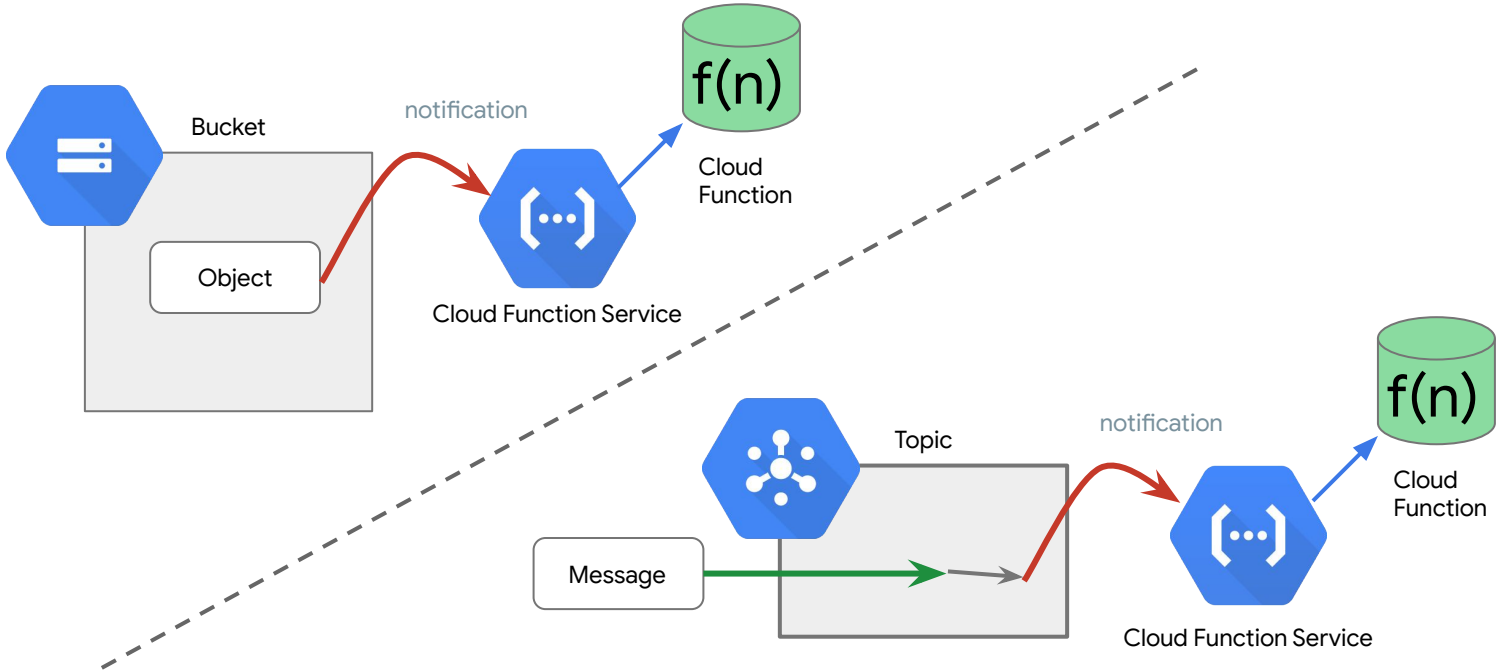
Using Cloud Functions for Data Processing



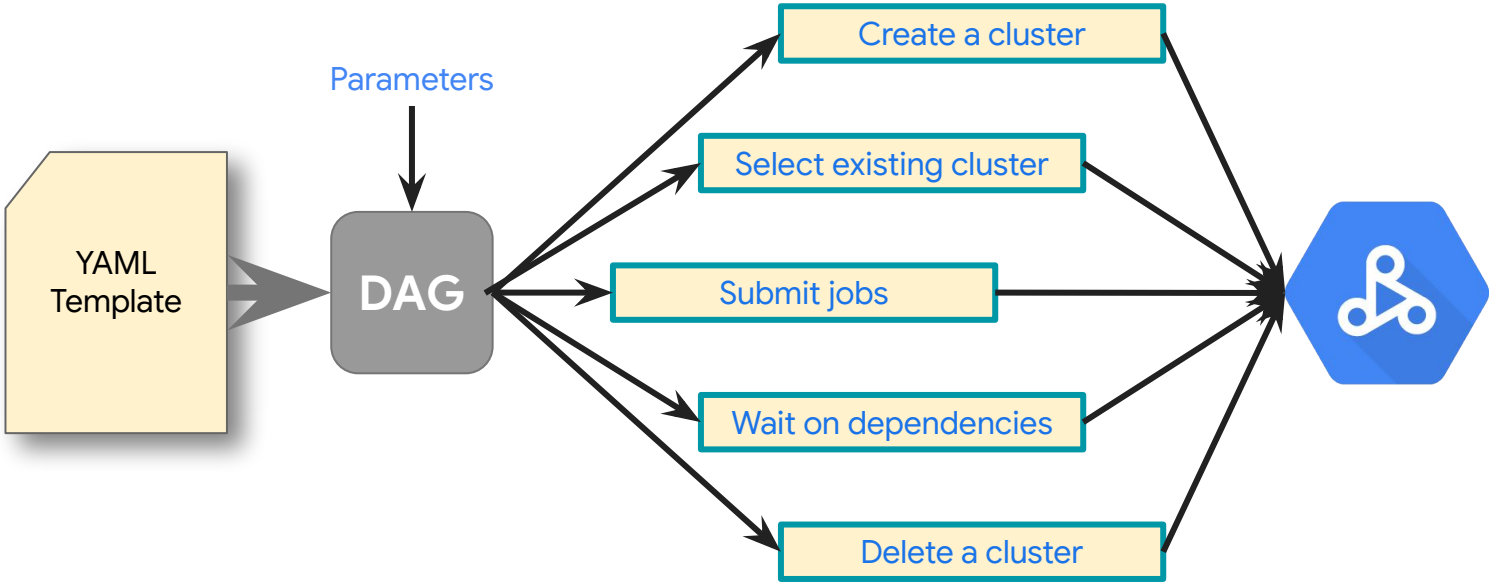
Deploying code to Cloud Functions



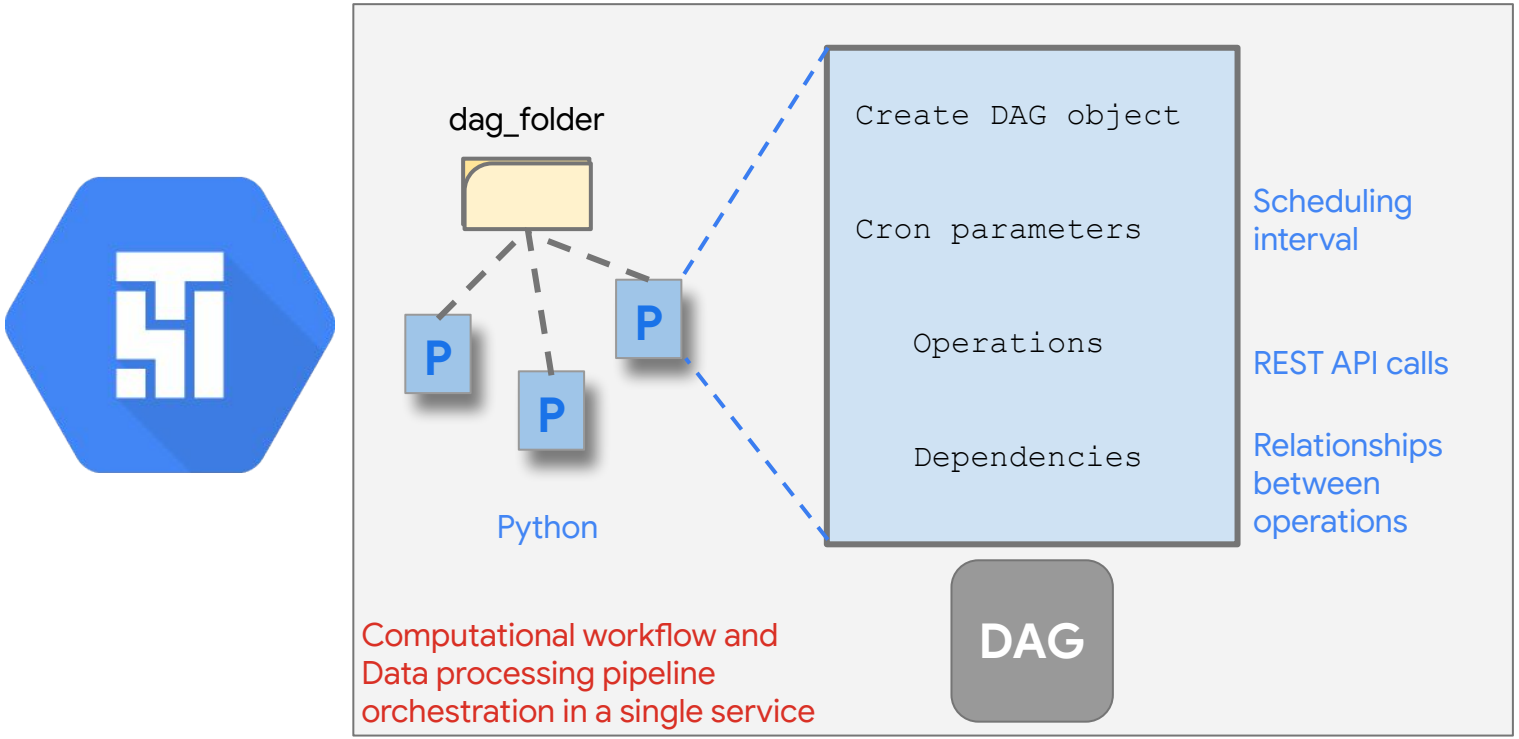
Triggering a Cloud Function from an object or message



Cloud Dataproc Workflow Template



Cloud Composer: Extensible dependency management




cloud.google.com
Images by Connie Zhou