From S3 to BigQuery - How A First-Time Airflow User Successfully Implemented a Data Pipeline

Leah Cole (with huge thanks to Emily Darrow!)
Intro to Leah
Today's Story

- Prologue
- Chapter 1: BigQuery Public Datasets
- Chapter 2: Growing Pains
- Chapter 3: The Goal
- Chapter 4: The DAG
- Epilogue
- Q&A
Prologue
Intro to Composer

Note: not all Composer components are depicted in the diagram (others were highlighted in a presentation last week from Rafał Biegacz)
Google BigQuery

- Google Cloud’s enterprise data warehouse for analytics
- Gigabyte to petabyte scale storage and SQL queries
- Encrypted, durable, and highly available
- Fully managed and serverless for maximum agility and scale
- Real-time insights from streaming data
- Built-in ML for out-of-the-box predictive insights
- High-speed, in-memory BI Engine for faster reporting and analysis

Built-in ML for out-of-the-box predictive insights
BigQuery: architecture

Serverless. Decoupled storage and compute for maximum flexibility.

- Streaming ingest
- Free bulk loading
- Replicated, distributed storage (99.9999999999% durability)
- Distributed memory shuffle tier
- Petabit network
- High-available cluster compute (Dremel)

- SQL:2011 Compliant
- REST API
- Web UI, CLI
- Client libraries in 7 languages

@leahecole
Chapter 1: BigQuery Public Datasets
The "Data Science" method

You need to...

**Discover** the dataset and where to access it.

**Negotiate** access to the dataset.

**Understand** the dataset, how it can be joined with your data, and its changes.

**Load** the data into your systems.

**Update, maintain, and secure** your data and database.

**Manage access** and keep the data updated.

**Link** public data with private data.

**Analyze, Visualize and communicate** your results.
What if you only did this?

You need to...

Discover the dataset and where to access it.

Negotiate access to the dataset.

Understand the dataset, how it can be joined with your data, and its changes.

Load the data into your systems.

Update, maintain, and secure your data and database.

Manage access and keep the data updated.

Link public data with private data.

Analyze, Visualize and communicate your results.
Current catalog
>180 datasets

Onboarded and maintained by Googler(s) with data provider input/guidance
Chapter 2: Growing Pains
Growing Pains in the Public Datasets Program

**Understand** the dataset, how it can be joined with your data, and its changes.

**Load** the data into your systems.
Late 2019: Onboarding a New Dataset

- New dataset comes in
- Temporarily stored
- Perform transformations
- Ends up in BQ
Late 2019: Problems with Current Process

- Disparate data sources + formats
- Internal/external resource communication
- Access control inconsistent
- Tooling
- Transformations
- Manual
Chapter 3: The Goal
The Goals

• Unified, repeatable process
• Utilize GCP products designed for this
• Hopefully open source process
• See process through eyes of first-time Airflow user (Leah + Emily)
Early 2020-Present: Proposed solution

1. Clone repo, make branch
2. Add config + transformations
   - YAML config
   - Custom transformations
3. Generate DAG + .tf config
4. Create a PR
5. Presubmit checks
6. Human review
7. Deploy
Chapter 4: The DAG
The DAG Development Process

- Shared repo
- Shared GCP project
  - Leah + Emily both owners
- Shared notes
- Meetings
  - Pairing as needed
  - Regular team meetings
DAG version 0.0

move_file_from_s3 = s3_to_gcs_operator.S3ToGoogleCloudStorageOperator(
    task_id='move_file_from_s3',
    bucket=config['source_bucket'],
    prefix='new_dataset/hourly/2019/10/2019-10-01.parquet',
    aws_conn_id='aws_default',
    dest_gcs_conn_id='google_cloud_default',
    dest_gcs='gs://us-central1-leah-emily-bucket/dags/datasets/',
    replace=False,
    gzip=True
)

make bq dataset for this = bash_operator.BashOperator(
    task_id='make bq dataset for this',
    bash_command='bq mk ' + config['target_dataset']
)

move_parquet_from_gcs_to_bq = gcs_to_bq.GoogleCloudStorageToBigQueryOperator(
    task_id='move_parquet_from_gcs_to_bq',
    bucket='us-central1-leah-emily-bucket',
    source_objects=['dags/datasets/new_dataset/hourly/2019/10/2019-10-01.parquet'],
    autodetect=True,
    source_format=config['source_format'],
    destination_project_dataset_table=config['target_dataset_table'],
    write_disposition='WRITE_TRUNCATE',
    trigger_rule='all_done'
)

delete bq dataset for this = bash_operator.BashOperator(
    task_id='delete bq dataset for this',
    bash_command='bq rm -rf ' + config['target_dataset'],
    trigger_rule='all_done'
)

print dag finished message = bash_operator.BashOperator(
    task_id='print dag finished message',
    bash_command='echo "Operation Complete"',
    trigger_rule='all_done'
)

Problem:

- Get data from S3, store in GCS
- Make target dataset
- Put data into BigQuery

Problem:

- Leftover GCS bucket
Get data from S3, store in GCS
- Make target dataset
- Put data into BigQuery
- Delete staging bucket

from gcs_delete_operator import GCSDeleteObjectsOperator

define_delete_parquet_from_gcs():
    task_id = "delete_parquet_from_gcs",
    bucket_name = GCS_BUCKET,
    objects = [DEST_FOLDER + SOURCE_PREFIX_DATED]
)
DAG version 1.x - Schema Definition, Resource Creation

```
14   bq mk --table \
15    --schema hourly_downloads_schema.json \  
16    --time_partitioning_field time \  
17    --clustering_fields name \  
18    --description "New Public Dataset" \  
19    new_public_dataset.hourly_downloads
```
DAG version 1.x - YAML config

```yaml
# s3 bucket template
source_bucket: 'new_public_dataset-package-data'
source_prefix: 'new_public_dataset/hourly/{year}/{month}/{year}-{month}-{day}.parquet'
source_format: 'PARQUET'

# BigQuery table
target_dataset: 'new_public_dataset'
target_dataset_table: 'new_public_dataset.hourly_downloads'

# GCS Config
gcs_bucket: 'us-central1-leah-emily-bucket'
gcs_dest_folder: 'dags/datasets/'

# General config
lag: 45

# Column renames, defined as a key-value mapping - Must be exhaustive for now,
# including all columns and column names desired.
# This is a mask as well as a dictionary.
columns_with_aliases:
  data_source: data_source
time: time
pkg_name: name
pkg_version: version
pkg_platform: platform
pkg_python: python_version
counts: total_downloads
target_time_field_name: 'time'
source_time_field_name: 'time'
```
DAG version 1.x - Verify

```python
# Make sure the configuration is set up correctly
verify_configuration = python_operator.PythonOperator(
    task_id='verify_configuration',
    python_callable=check_config,
    op_kwargs={'config_string': config_string}
)
```

```
# Make sure we've got configuration variables for everything we need, or else don't run.
def check_config(config_string):
    required_variables = {
        'gcs_bucket',
        'gcs_dest_folder',
        'source_format',
        'source_bucket',
        'target_dataset',
        'target_dataset_table',
        'source_prefix',
        'columns_with_aliases',
        'lag'
    }
    config_dict = json.loads(config_string)
    for config_key in required_variables:
        assert config_key in config_dict.keys() and config_dict[config_key], f'{{key}} is undefined'
```
DAG version 1.x - Verify

```
# We are assuming that the dataset has already been created. This is an ingestion job, not a setup job.
# This will fail if the dataset doesn't exist, and should echo a message to that effect.
verify_dataset_requirement = bash_operator.BashOperator(
    task_id='verify_dataset_requirement', bash_command='if bq show ' + TARGET_DATASET + '; then ' +
    'echo "Dataset ' + TARGET_DATASET + ' is required - failing now."; exit 1; fi'
)

# As above, verify that the dataset contains the master table we're planning to insert into.
# This will fail if the master table doesn't exist.
# We do not verify that the master table's schema matches any expectation at this time.
verify_target_table_requirement = bash_operator.BashOperator(
    task_id='verify_target_table_requirement', bash_command='if bq show --schema ' + TARGET_DATASET_TABLE + '; then ' +
    'echo "Target table exists"; else echo "Target table ' + TARGET_DATASET_TABLE + ' is required - failing now."; exit 1; fi'
)
```

@leahecole
# The S3 to GCS operator copies files from s3 to GCS, but it can copy multiple files.
# We will be using macros to specify prefixes and elements.
# It is worth noting that this operator copies the full folder structure.
#
# Requirements:
# - Boto3 package in python environment
# - S3 Credentials with access to the bucket defined in the airflow configuration.
move_file_from_s3 = s3_to_gcs_operator.S3ToGoogleCloudStorageOperator(
    task_id='move_file_from_s3',
    bucket=SOURCE_BUCKET,
    prefix=SOURCE_PREFIX_DATED,
    aws_conn_id='aws_default',
    dest_gcs_conn_id='google_cloud_default',
    dest_gcs='gcs://' + GCS_BUCKET + '/'+ DEST_FOLDER,
    replace=False,
    gzip=True
)

# This is where we actually put things into BigQuery. Since this is a parquet file, we can skip the
# schema parameter. Parquet files are self-describing. If this were a csv file, we would need to
# describe the expected schema.
#
# It is worth noting that the autodetect parameter is required for parquet files to work.
move_parquet_from_gcs_to bq = gcs_to_bq.GoogleCloudStorageToBigQueryOperator(
    task_id='move_parquet_from_gcs_to bq',
    bucket=GCS_BUCKET,
    source_objects=[DEST_FOLDER + SOURCE_PREFIX_DATED],
    autodetect='true',
    source_format=SOURCE_FORMAT,
    destination_project_dataset_table=TEMP_TABLE_DATED,
    write_disposition='WRITE_TRUNCATE',
)
DAG version 1.x - Transform + Load

# Here we transfer the data from temp to Master, with the column remapping. The actual operator.
transfer_data = bigquery_operator.BigQueryOperator(
    task_id='transfer_data',
    sql=MERGE_TRANSFORM_STATEMENT,  # Append to existing tables
    write_disposition='WRITE_TRUNCATE',  # Don't create tables.
    create_disposition='CREATE_NEVER',  # Don't create tables.
    use_legacy_sql=False,
    allow_large_results=True
)

# Since we have imported the file, we no longer need it. Lets delete it.
delete_parquet_from_gcs = GCSDelOperator(
    task_id="delete_parquet_from_gcs",
    bucket_name=GCS_BUCKET,
    objects=[DEST_FOLDER + SOURCE_PREFIX_DATED]
)

# We no longer need the temporary table, so we'll wipe it out.
drop_temp_table = bash_operator.BashOperator(
    task_id='drop_temp_table',
    bash_command='bq rm -f -t ' + TEMP_TABLE_DATED + '; echo "Deleted the temp table"'
)
Epilogue
Lessons Learned

• Double check your Composer and Airflow versions
• Documentation is extremely important
• Changelogs and release notes are extremely important
• Transferring data between cloud providers is REALLY easy with Airflow
Call to Action

- Contribute
- Automate
- Collaborate
Thank you to

• Emily Darrow - for their technical legwork with this project
• Tim Swast - technical advice + vision, moral support
• Shane Glass - technical advice, vision, and presentation content
• Rafał Biegacz + the Composer Team - presentation content + tireless engineering work
• Seth Hollyman and my Data Analytics DevRel colleagues - presentation content, moral support, and constant inspiration
• Moderators, sponsors, and attendees!
Q&A with Leah, Tim, and Shane