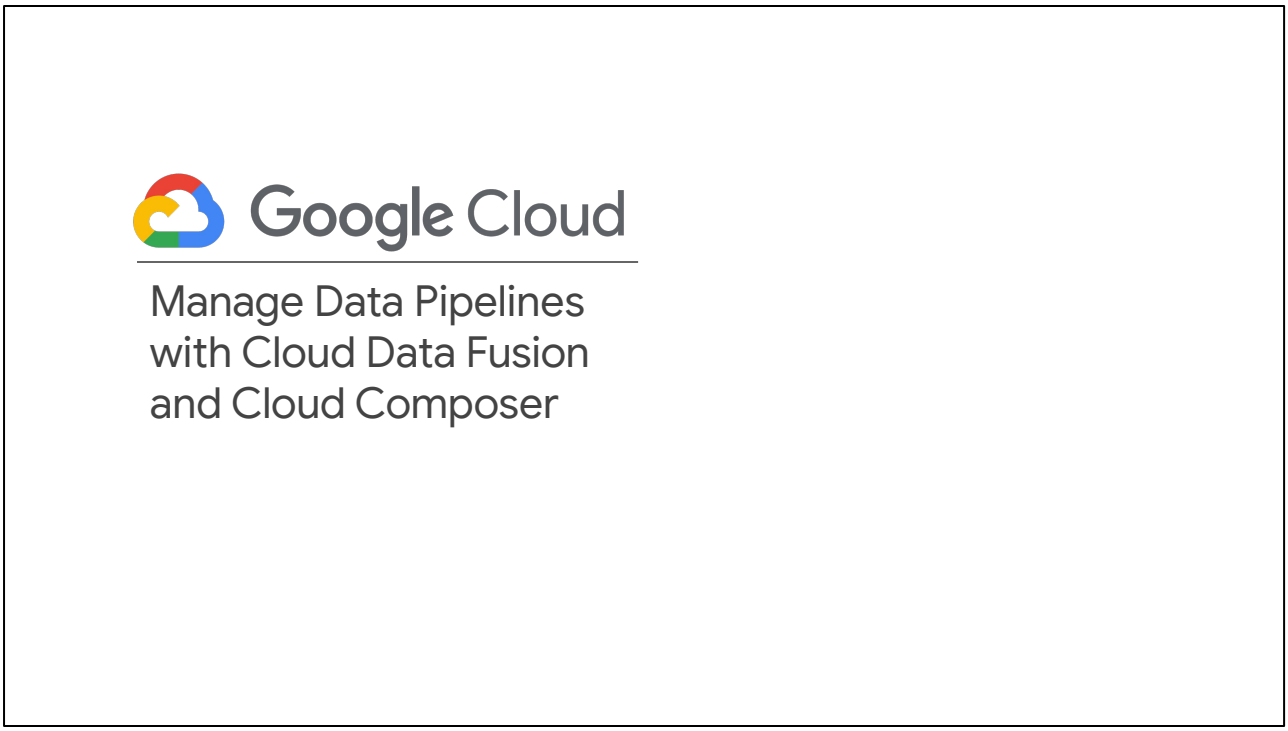
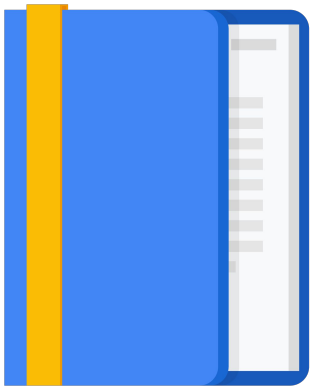


In this module, we will discuss how to manage end-to-end data pipelines using tools in Google Cloud.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



Google Cloud

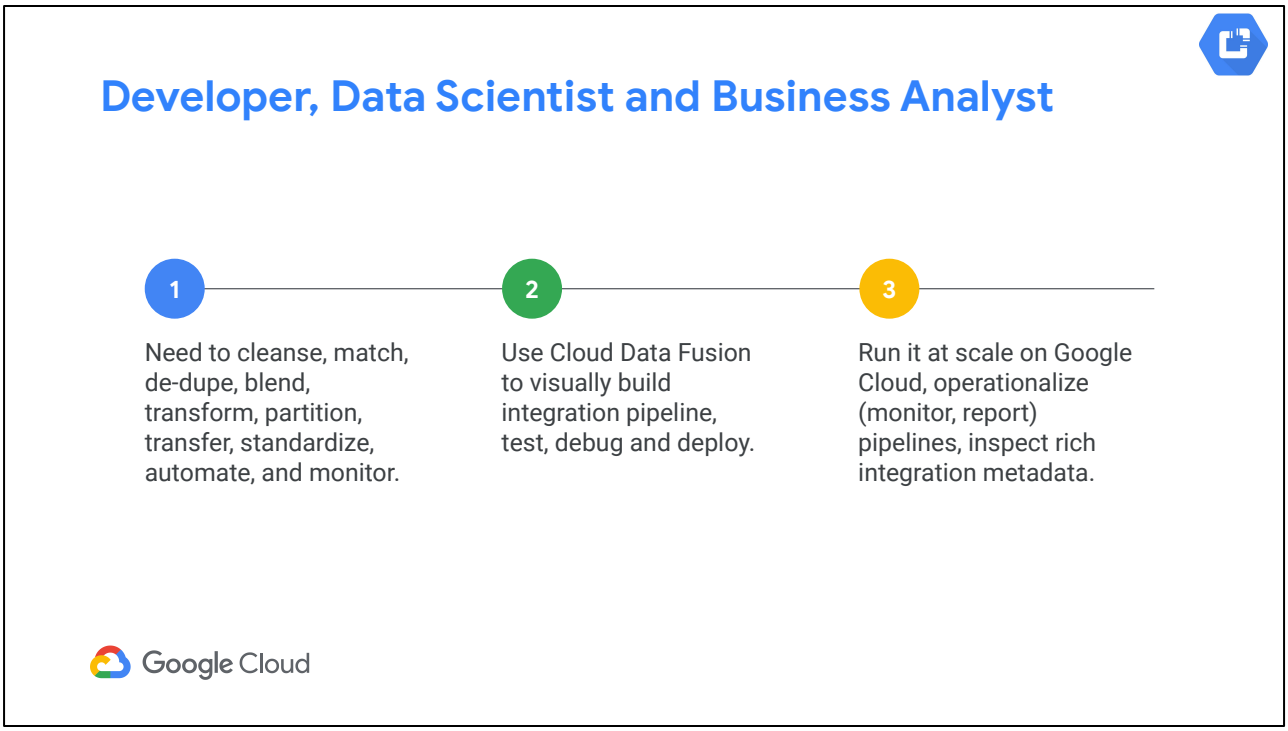
The two solutions we will discuss in this module are Cloud Data Fusion and Cloud Composer.

Cloud Data Fusion

Cloud Data Fusion is a fully-managed, cloud native, enterprise data integration service for quickly building and managing data pipelines.

Google Cloud

Cloud Data Fusion provides a graphical user interface and APIs that increase time efficiency and reduce complexity. It equips business users, developers and data scientists to quickly and easily build, deploy and manage data integration pipelines. Cloud Data Fusion is essentially a graphical no code tool to build data pipelines.



Cloud Data Fusion is used by developers, data scientists, and business analysts alike. For developers, Cloud Data Fusion allows you to cleanse, match, remove duplicates, blend, transform, partition, transfer, standardize, automate, and monitor data. Data scientists can use Cloud Data Fusion to visually build integration pipelines, test, debug, and deploy applications. Business analysts can run Cloud Data Fusion at scale on Google Cloud, operationalized pipelines, and inspect rich integration metadata.

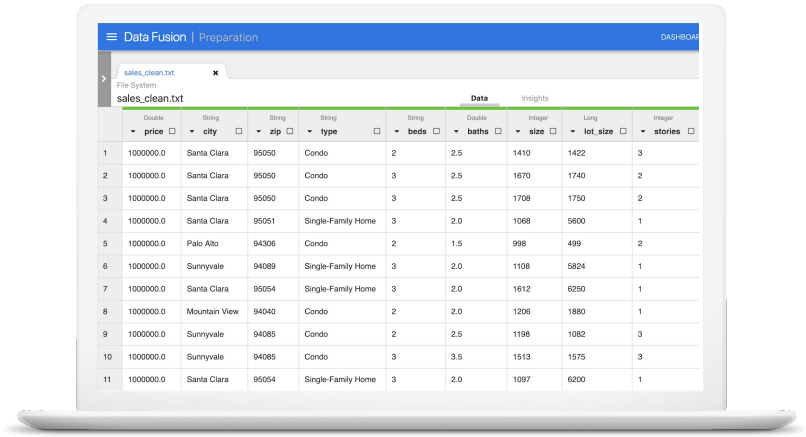
Benefits

Integrate with any data

Increase productivity

Reduce complexity

Increase flexibility



	price	city	zip	type	beds	baths	size	lot_size	stories
1	1000000.0	Santa Clara	95050	Condo	2	2.5	1410	1422	3
2	1000000.0	Santa Clara	95050	Condo	3	2.5	1670	1740	2
3	1000000.0	Santa Clara	95050	Condo	3	2.5	1708	1750	2
4	1000000.0	Santa Clara	95051	Single-Family Home	3	2.0	1068	5600	1
5	1000000.0	Palo Alto	94306	Condo	2	1.5	998	499	2
6	1000000.0	Sunnyvale	94089	Single-Family Home	3	2.0	1108	5824	1
7	1000000.0	Santa Clara	95054	Single-Family Home	3	2.0	1612	6250	1
8	1000000.0	Mountain View	94040	Condo	2	2.0	1206	1880	1
9	1000000.0	Sunnyvale	94085	Condo	2	2.5	1198	1082	3
10	1000000.0	Sunnyvale	94085	Condo	3	3.5	1513	1575	3
11	1000000.0	Santa Clara	95054	Single-Family Home	3	2.0	1097	6200	1

Google Cloud

Integrate with any data - through a rich ecosystem of connectors for a variety of legacy and modern systems, relational databases, file systems, cloud services, object stores, NoSQL, EBCDIC, and more.

Increase productivity - If you have to constantly move between numerous systems to gather insight, your productivity is significantly reduced. With Cloud Data Fusion, your data from all the different sources can be pooled into a view like in BigQuery, Spanner or any other Google Cloud technologies, allowing you to be more productive faster.

Reduce Complexity - through a visual interface for building data pipelines, code free transformations, and reusable pipeline templates.

Increase flexibility - through support for on-prem and cloud environments, interoperability with OSS CDAP

Build data pipelines with a friendly UI

Rich graphical interface

100+ plugins - connectors, transforms & actions

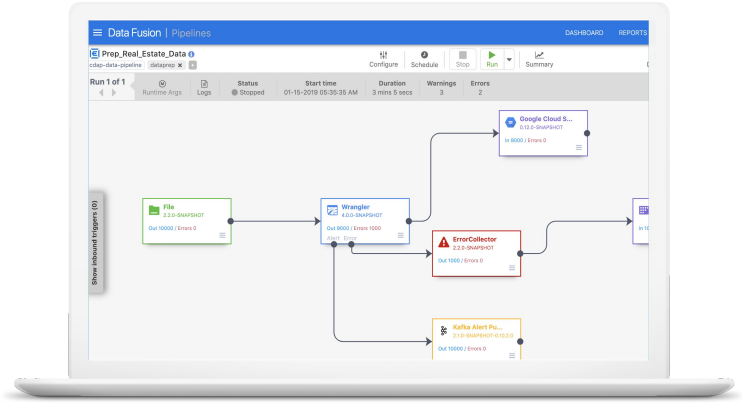
Code free visual transformations

1000+ transforms, data quality

Test and debug pipeline

Pre-built pipelines

Developer SDK



Google Cloud

At a high level, Cloud Data Fusion provides you with a graphical user interface to build data pipelines with no code. You can use existing templates, connectors to Google Cloud, and other Cloud services providers and an entire library of transformations to help you get your data in the format and quality you want. Also, you can test and debug the pipeline and follow along with each node as it receives and processes data.

Integration Metadata

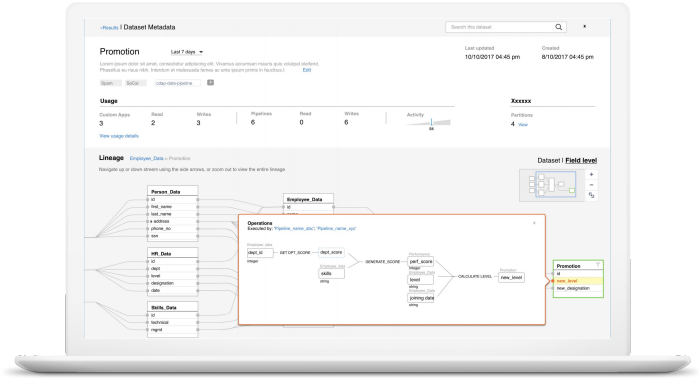
Tags and Properties support

- Pipeline
- Dataset
- Schema

Search integrated entities by

- Keyword
- Schema name and type

Dataset level and Field level Lineage

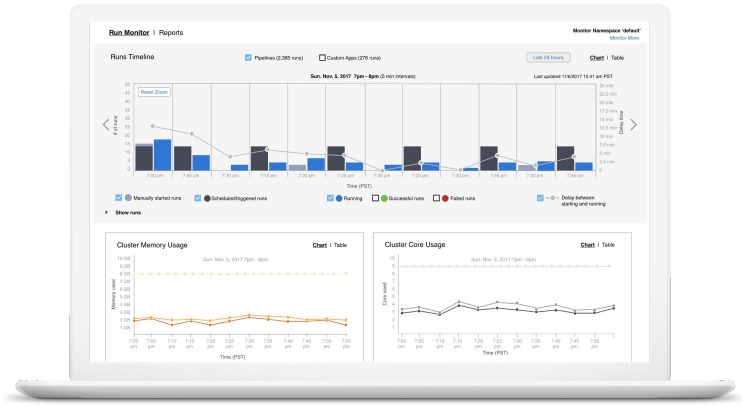


Google Cloud

As you will see in the next section, you can tag pipelines to help organize them more efficiently for your team, and you can use the unified search functionality to quickly find field values or other keywords across your pipelines and schemas. Lastly, we will talk about how Cloud Data Fusion tracks the lineage of transformations that happen before and after any given field on your dataset.

Extensible

- Pipeline templatization
- Conditional Pipeline Triggers
- Plugin Management
- Plugin templatization
- Plugin UI Widget
- Custom Provisioners
- Custom Compute Profiles
- Hub Integration



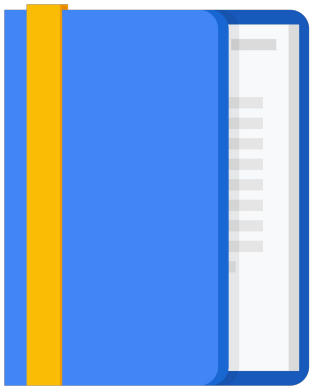
Google Cloud

One of the advantages of Cloud Data Fusion is that it's extensible. This includes the ability to templatize pipelines, create conditional triggers, and manage and templatize plugins. There is a UI widget plug-in as well as custom provisioners, custom compute profiles, and the ability to integrate to hub.

Agenda

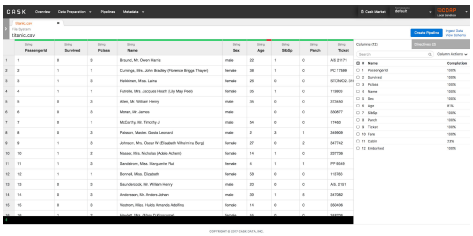
Building Batch Data Pipelines visually with Cloud Data Fusion
- **Components**
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



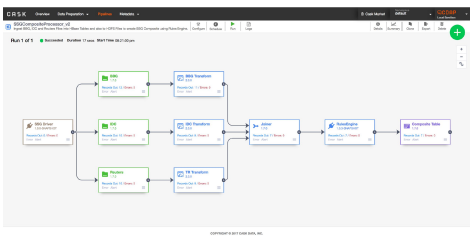
Google Cloud

Components of Cloud Data Fusion



Wrangler — Framework

- Data Preparation for on-boarding new sources and datasets.
- Perform Data Transformations, Data Quality checks with visual feedback.
- Extend the Wrangler by building new user defined directives.
- Integrates with Data Pipeline for operationalizing transformations

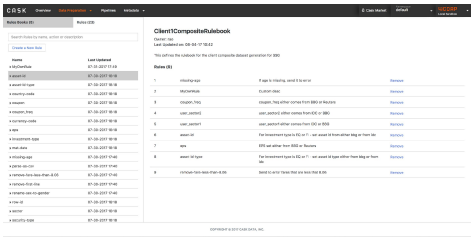


Data Pipeline — Framework

- User interface for building complex data workflows
- Join, Lookup, Aggregate, Filtering data in-flight
- Building complex workflows with 100s of connectors
- Extend Data Pipeline using simple APIs
- Integrates with Dataprep, Rule Engine and Metadata Aggregator

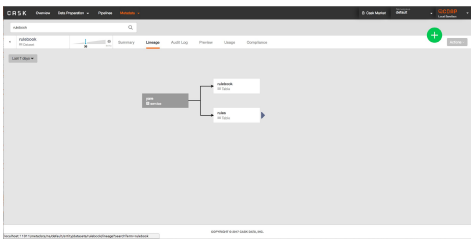
The two major user interface components we will focus our attention on in this course, are the Wrangler UI for exploring data sets visually, and building pipelines with no code, and the Data Pipeline UI for drawing pipelines right on to a canvas. You can choose from existing templates for common data processing tasks like Cloud Storage, to BigQuery.

Components of Cloud Data Fusion



Rules Engine — Tool*

- Business Data Transformations and checks codified for business users
- Define Complex rules using intuitive and simple to use user interface
- Logically group Rules in Rulebook and trigger or schedule processing.
- Integrates with Data Pipeline for operationalizing Rules.

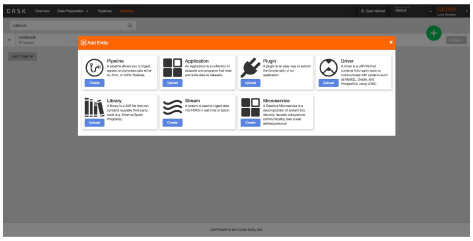


Metadata Aggregator — Tool

- Aggregate Business, Technical and Operational Metadata
- Track the flow of data (Lineage) for richer data needed for governance
- Create Data Dictionary and Metadata Repository
- Integrate with enterprise MDM solutions.
- Integrates with Data Pipeline, Rules Engine

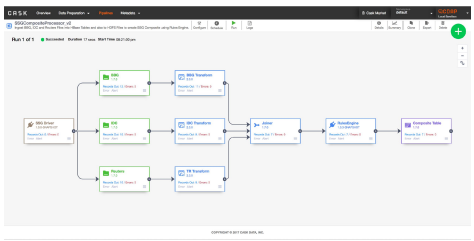
There are other features of Cloud Data Fusion that you should be aware of too. There's an integrated rules engine where business users can program in their pre-defined checks and transformations, and store them in a single place. Then data engineers can call these rules as part of a rule book or pipeline later. We mentioned data lineage as part of field metadata earlier. You can use the metadata aggregator to access the lineage of each field in a single UI and analyze other rich metadata about your pipelines and schemas as well. For example, you can create and share a data dictionary for your schemas directly within the tool.

Components of Cloud Data Fusion



Microservice — Framework*

- Build specialized logic for processing data
- Create loosely coupled network for processing events
- Bind processing to varied set of queues



Event Condition Action (ECA) — Application*

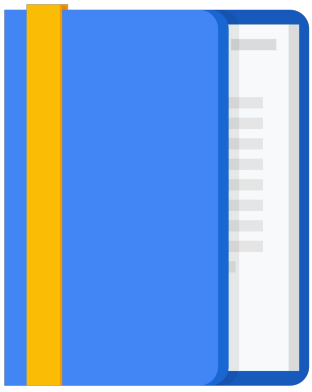
- Delivers a specialized solutions for IoT event processing
- Parses any events, triggers conditions and executes Action.
- Real-time notification system, with easy-to-use user interface for configuring event parsing, condition and actions

Other features such as the microservice framework allow you to build specialized logic for processing data. You can also use the Event Condition Action (ECA) Application to parse any event, trigger conditions, and execute an action based on those conditions.

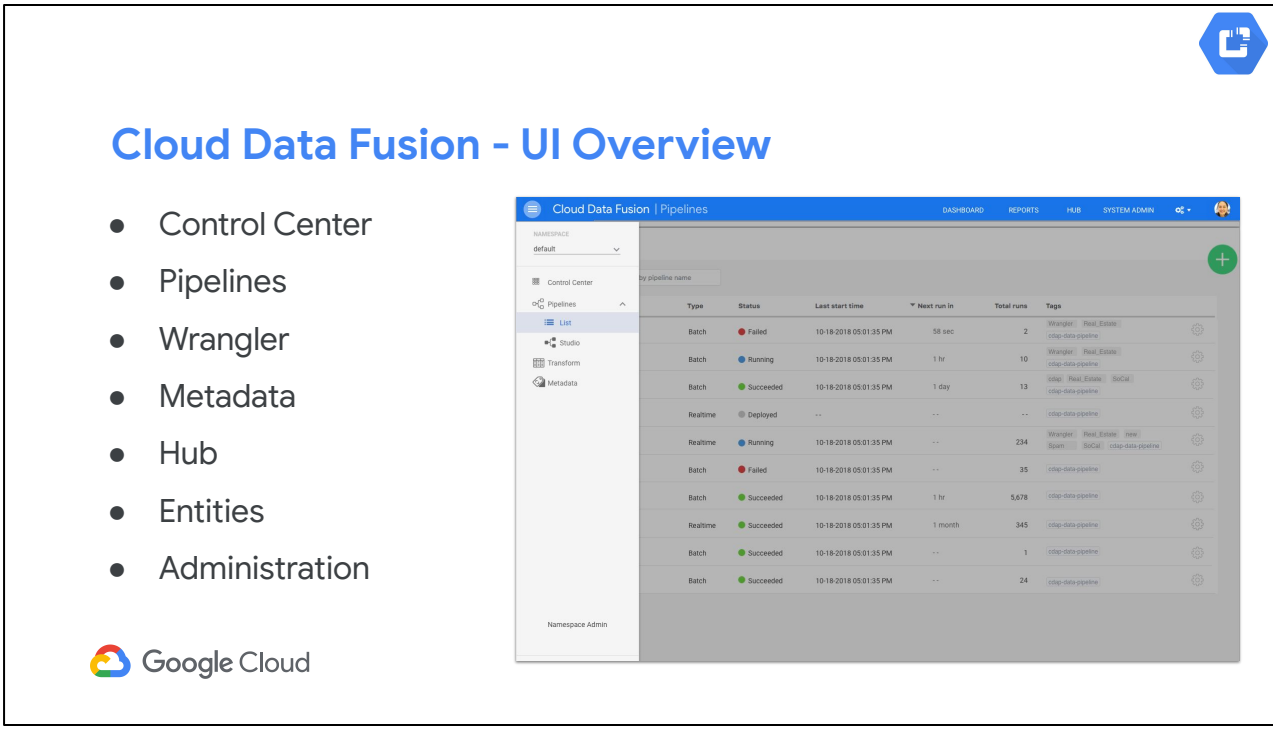
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- **UI Overview**
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



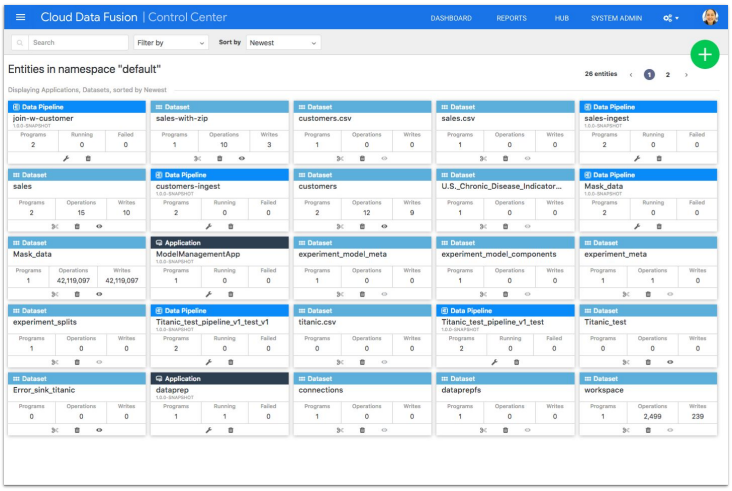
Google Cloud



Here are some of the key user interface elements that you will encounter when using Cloud Data Fusion. Let's look at each of them in turn under Control Center.

Control Center

- Application
- Artifact
- Dataset



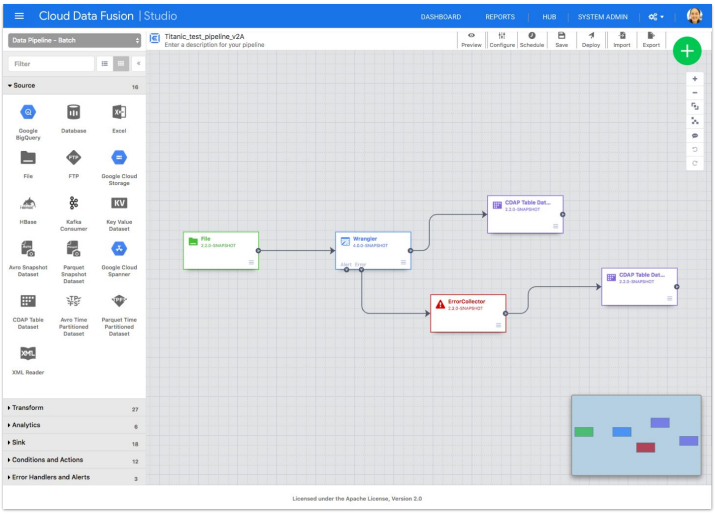
Google Cloud

There is a section for applications, artifacts and a dataset. Here you could have multiple pipelines associated with a particular application.

The Control Center gives you the ability to see everything at a glance and search for what you need, whether it's particular dataset, pipeline or other artifact like a data dictionary for example.

Pipelines

- Developer Studio
- Preview
- Export
- Schedule
- Connector and function palette
- Navigation

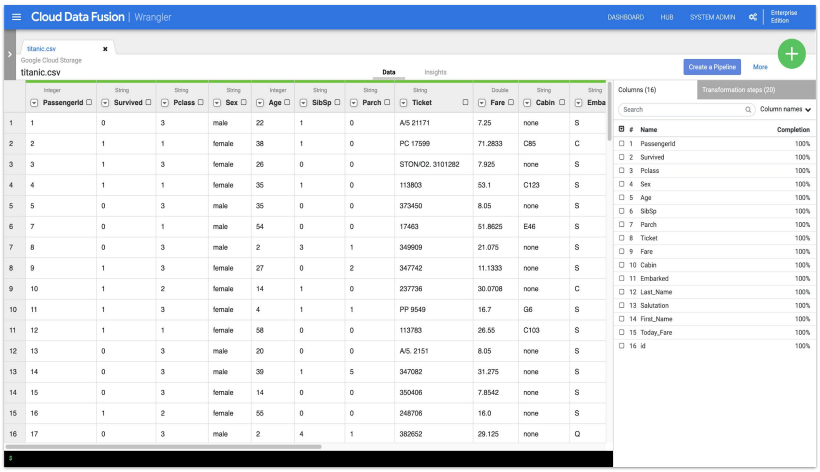


Google Cloud

Under the Pipelines section you have a Developer Studio. You can preview, export, schedule a job or a project. You also have a connector and function palette and a navigation section.

Wrangler

- Connections
- Transforms
- Data Quality
- Insights
- Functions

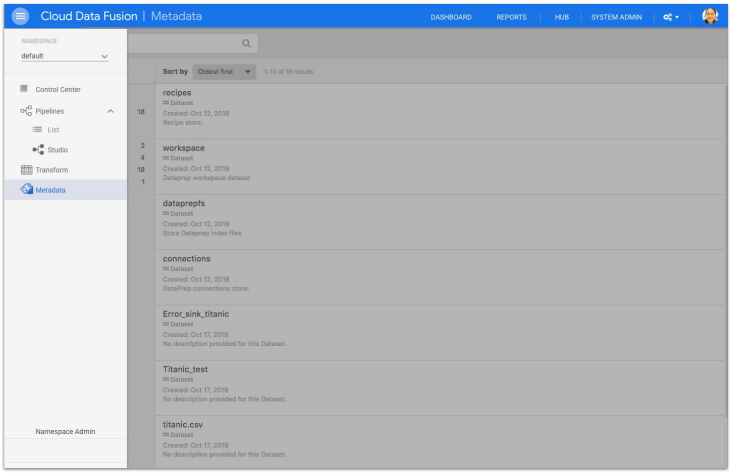


Google Cloud

Under the Wrangler section you have connections, transforms, data quality, insights, and functions.

Integration Metadata

- Search
- Tags & Properties
- Lineage - Field and Data

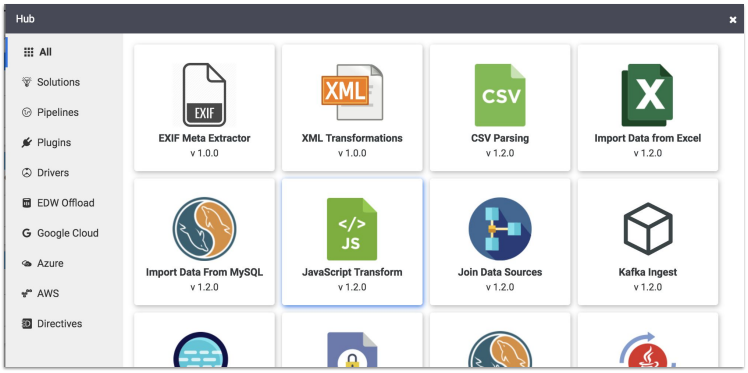


Google Cloud

Under the Integration metadata section you can search, add tags and properties, and see the data lineage for field and data.

Hub

- Plugins
- Use cases
- Pre-built pipelines

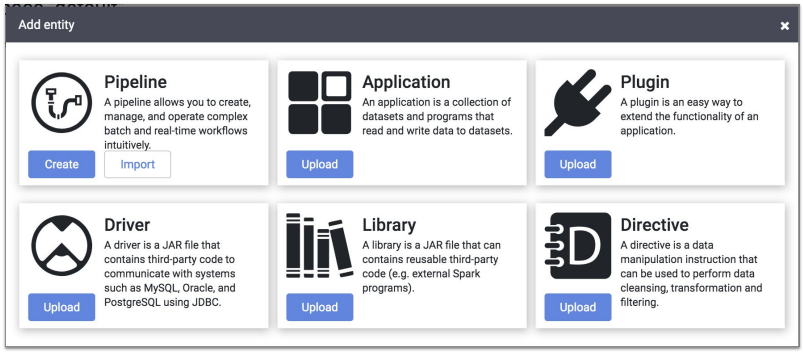


Google Cloud

The Hub allows you to see all the available plugins, sample use cases, and pre-built pipelines.

Entities

- Pipeline
- Application
- Plugin
- Driver
- Library
- Directive

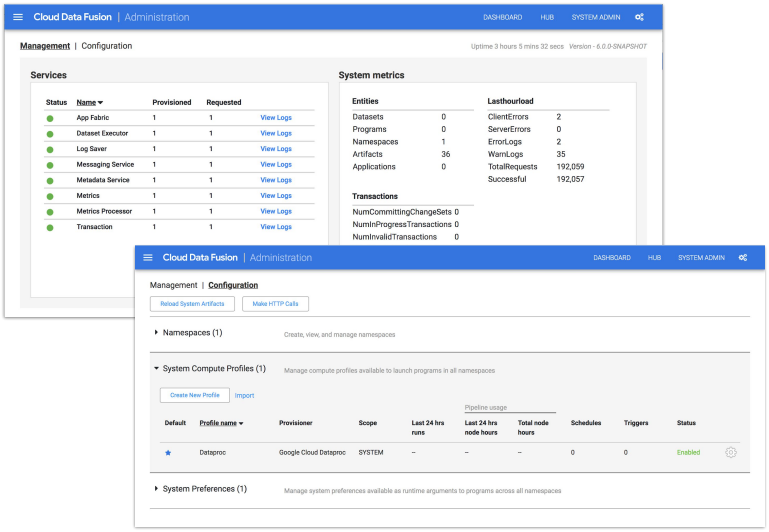


Google Cloud

Entities include, the ability to create pipelines, upload an application, plug-in, driver, library, and directives.

Administration

- Management
  - Services
  - Metrics
- Configuration
  - Namespace*
  - Compute Profiles
  - Preferences
  - System Artifacts
  - REST Client



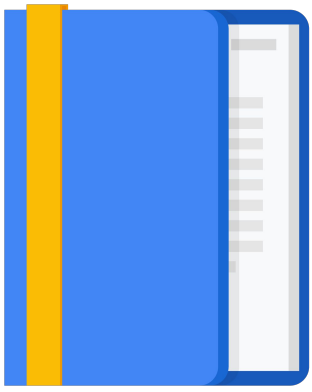
Google Cloud

There are two components in Administration: Management and Configuration. Under Management you have services and metrics. Under Configuration you have namespace, compute profiles, preferences, system artifacts, and the REST client.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- **Building a Pipeline**
- Exploring Data using Wrangler

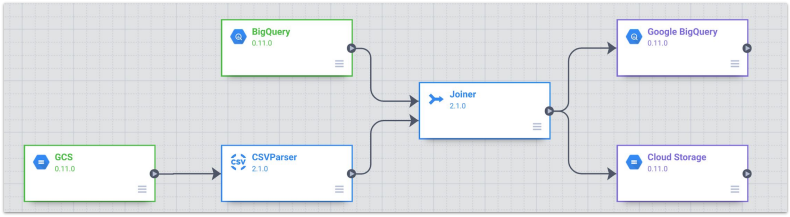
Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



Google Cloud

Data Pipeline | Directed Acyclic Graph (DAG)

- Represented by a series of stages arranged in a DAG. This forms a **one-way** pipeline.
- Stages, which are the "nodes" in the pipeline graph, can be of different types



Google Cloud

A pipeline is represented visually as a series of stages arranged in a graph. These graphs are called DAGs or directed acyclic graphs because they flow from one direction to another and they can not feed into themselves. Acyclic simply means not a circle. Each stage is a node, and as you can see here, it can be of a different type. You may start with a node that pulls data from Google Cloud Storage, then passes it on to a node that parses a CSV. The next node takes multiple nodes, has an input, and joins them together before passing the join data to two separate data sync nodes.

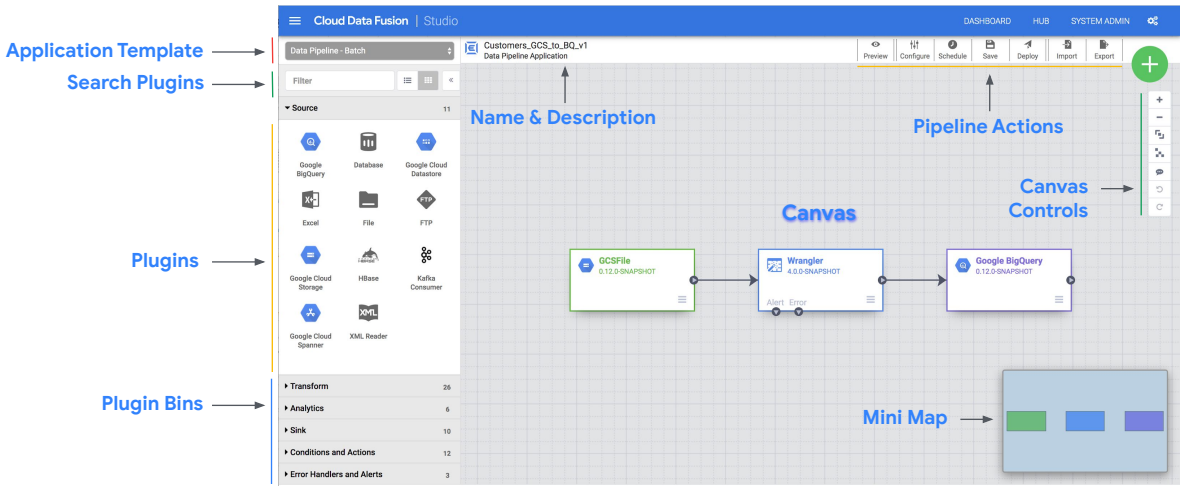
Data Pipeline

- Allows non-linear pipelines.
- Can fork from a node, where output from a node can be sent to two or more stages.
- Two or more forked nodes can merge at a transform or a sink node.

Google Cloud

As you saw in our previous example, you can have multiple nodes fork out from a single parent node. This is useful because you may want to kick off another data processing workstream that should not be blocked by any processing on a separate series of nodes. You can combine data from two or more nodes into a single output in a sync.

Studio is the UI where you create new pipelines

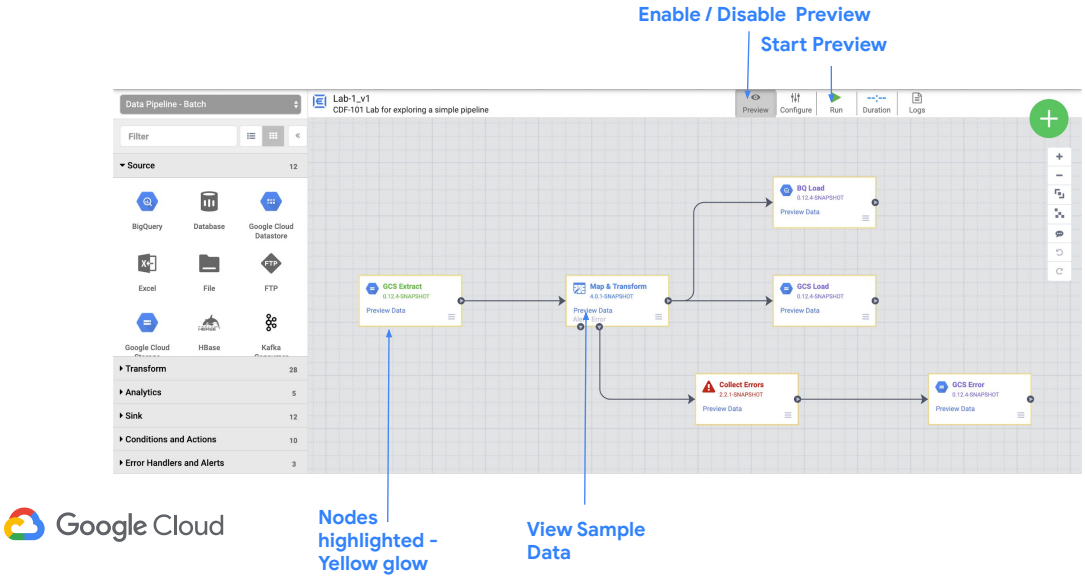


Google Cloud

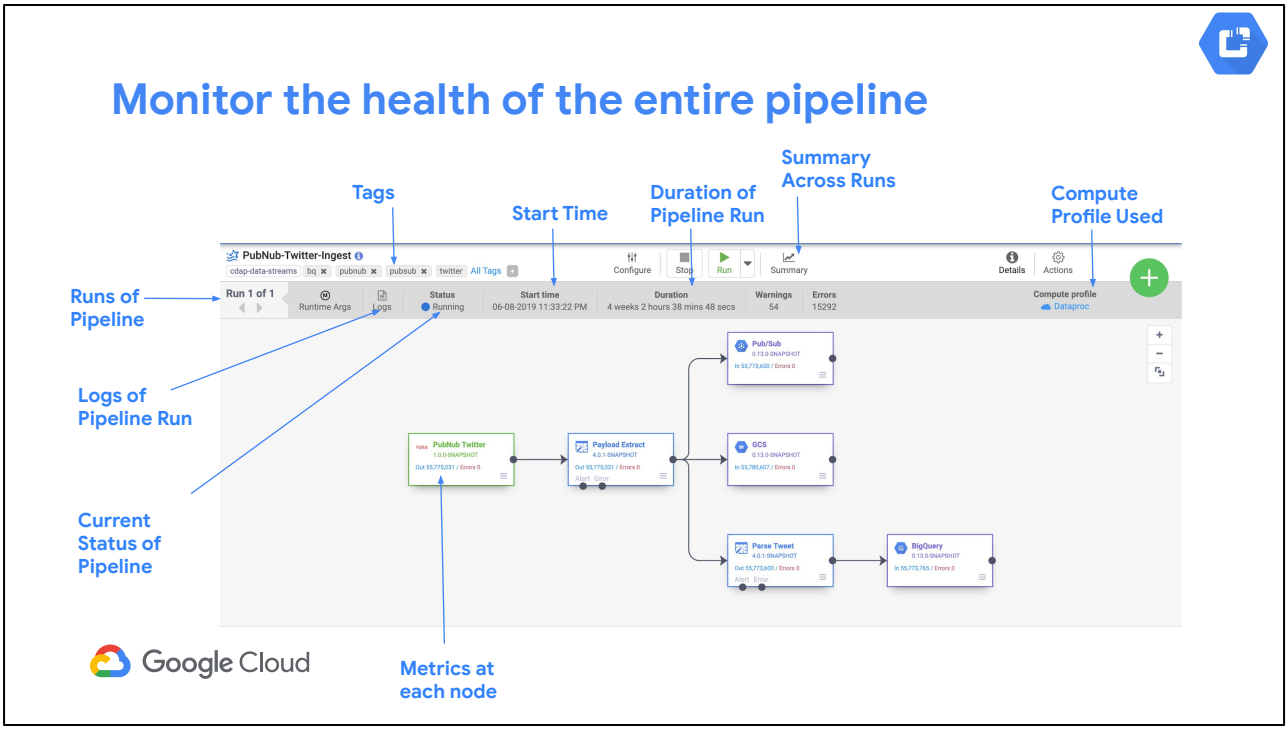
In Cloud Data Fusion, the studio is the user interface where you author and create new pipelines.

The area where you create nodes and chain them together in your pipeline is your canvas. If you have many nodes in a pipeline, the canvas can get visually cluttered, so use the mini map to help navigate around a huge pipeline quickly. You can interact with the canvas and add objects by using the Canvas Control Panel. When you're ready to save and run the entire pipeline, you can do so with the pipeline actions toolbar at the top. Don't forget to give your pipeline a name and description, as well as make use of the many preexisting templates and plugins, so you don't have to write your pipeline from scratch. Here, we've used a template or data pipeline batch which gives us the three nodes you see here to move data from a Cloud Storage file, process it in a wrangler, and output it to BigQuery.

Use Preview Mode to see how the pipeline will run

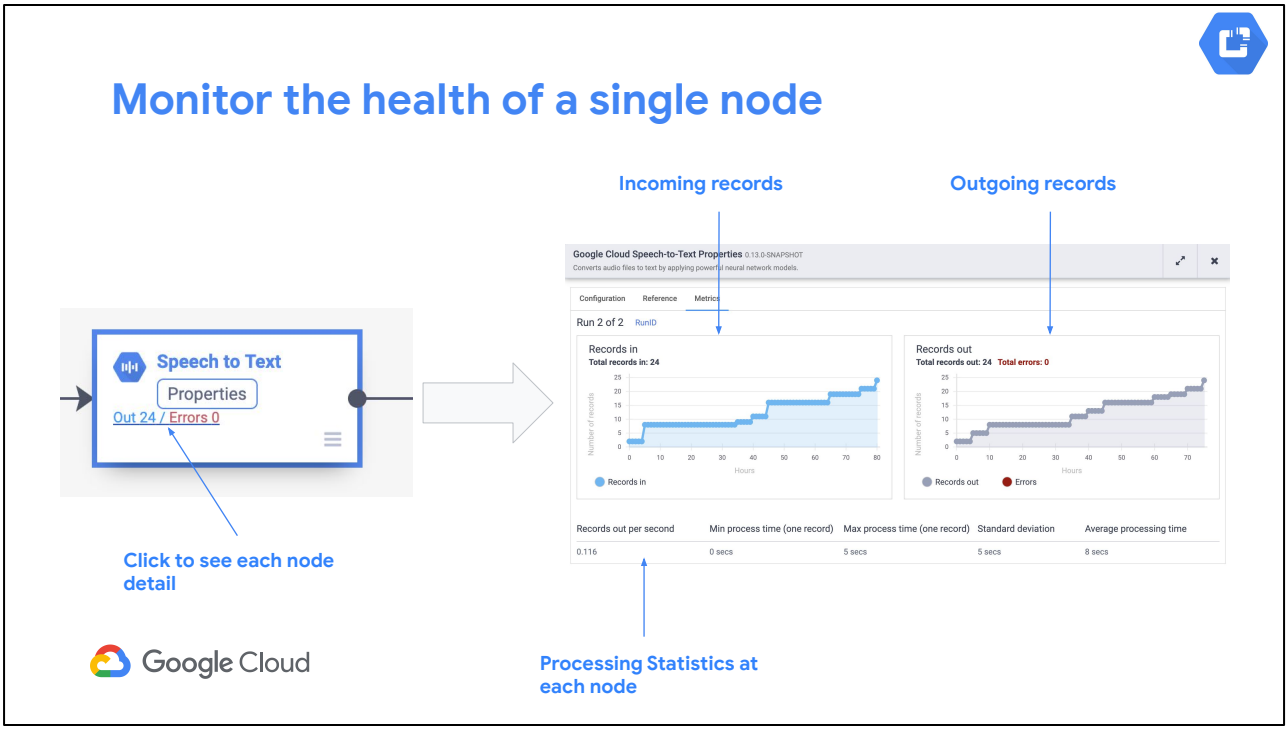


You should make use of preview mode before you deploy and run your pipeline in production to ensure everything you run will run properly. While a pipeline is in preview, you can click on each node and see any sample data or errors that you will need to correct before deploying.



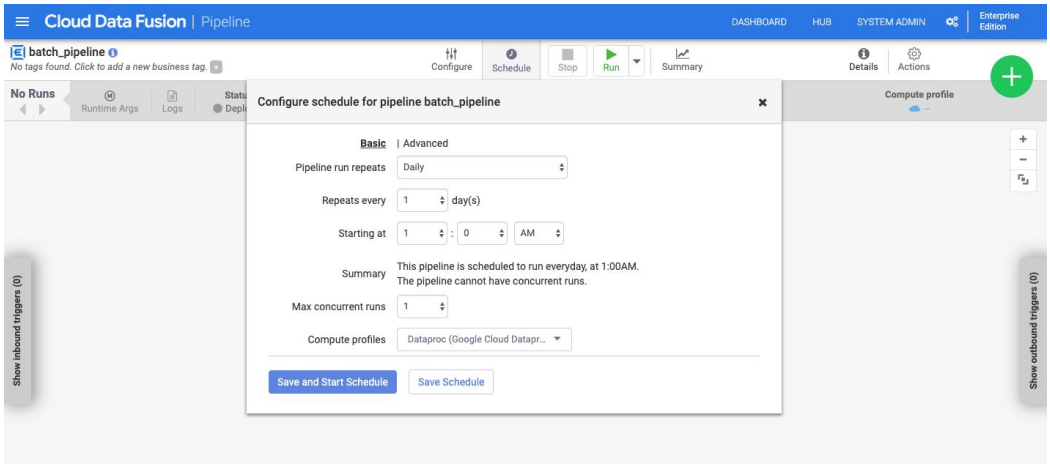
After deployment, you can monitor the health of your pipeline and collect key summary stats of each execution. Here, we are ingesting data from Twitter and Google Cloud, and parsing each tweet before loading them into a variety of data syncs.

If you have multiple pipelines, it is recommended that you make liberal use of the tags feature to help you quickly find and organize each pipeline for your organization. You can view the start time, the duration of the pipeline run, and the overall summary across runs for each pipeline. You can quickly see the data throughput at each node in the pipeline simply by interacting with the node. Note the Compute profile used in the Cloud. Currently, Cloud Data Fusion supports running on Dataproc, but Cloud Data Source support is on the road map.



Clicking on a node gives you detail on the inputs, outputs, and errors for that given node. Here, we are integrating with the Speech-to-Text API to process audio files into searchable text. You can track the individual health of each node and get useful metrics like records out per second, average processing time, and max processing time, which can alert you to any anomalies in your pipeline.

You can schedule batch pipelines

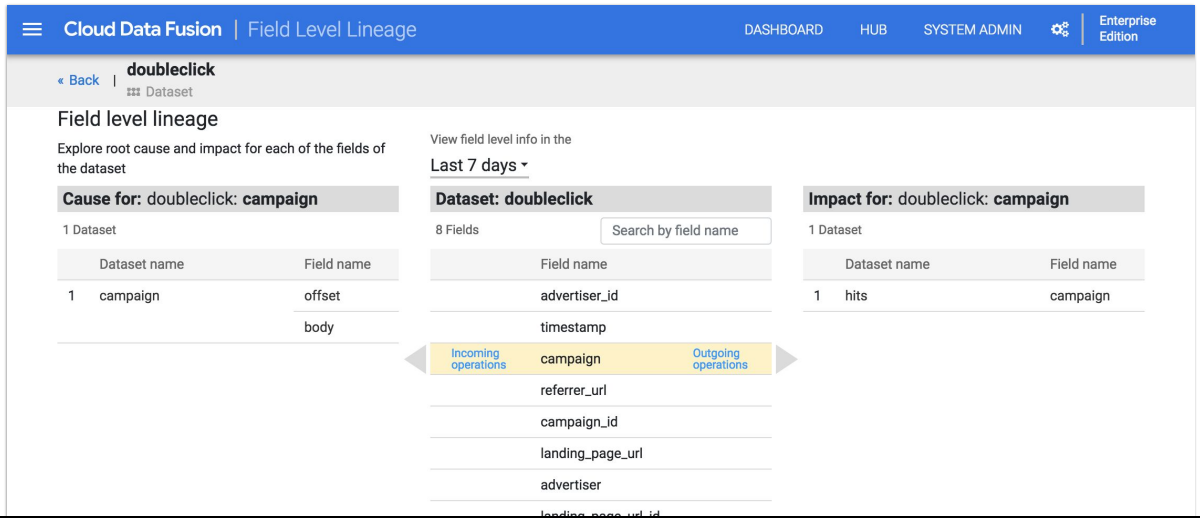


Google Cloud

You can set your pipelines to run automatically at certain intervals. If your pipeline normally takes a long time to process the entire dataset, you can also specify a maximum number of concurrent runs to help avoid processing data unnecessarily. Keep in mind that Cloud Data Fusion is designed for batch data pipelines. We will dive into streaming data pipelines in future modules.

After data is transformed, you can track field-level lineage

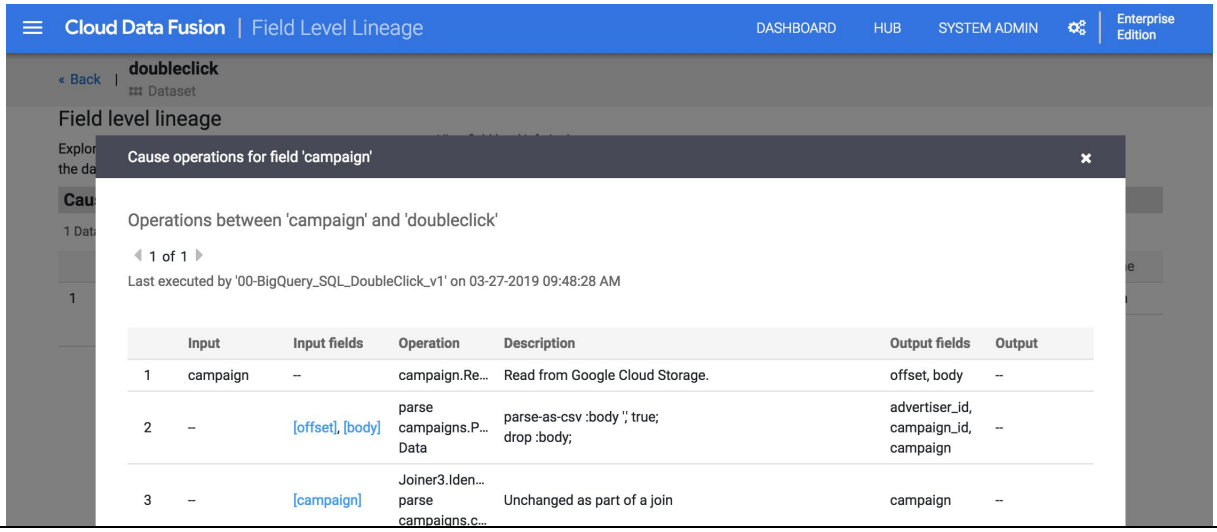
Relationship between fields of datasets: Provenance, Impact



One of the big features of Cloud Data Fusion is the ability to attract the lineage of a given field value. Let's take this example of a campaign field for double-click dataset and track every transform operation that happened before and after this field.

See every operation that is made on a field

Operations applied to a field

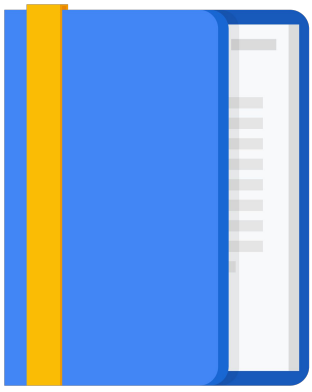


Here, you can see the lineage of operations that are applied to the campaign field between the campaign dataset and the double-click dataset. Note the time this field was last changed by a pipeline run and each of the input fields and descriptions that interacted with the field as part of processing it between datasets. Imagine the use cases if you have inherited a set of analytical reports and you want to walk back upstream all of the logic that went into a certain field. Well now, you can.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- Building a Pipeline
- **Exploring Data using Wrangler**

Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging

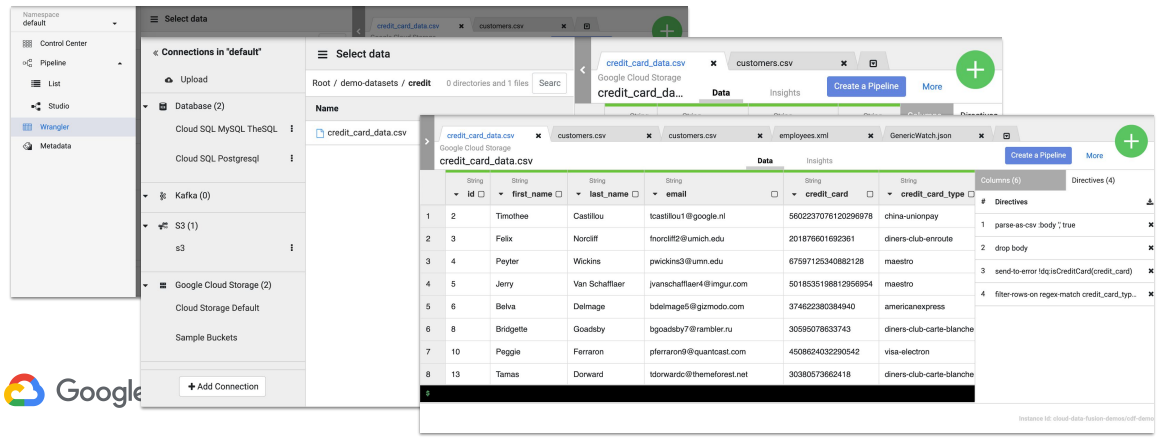


Google Cloud

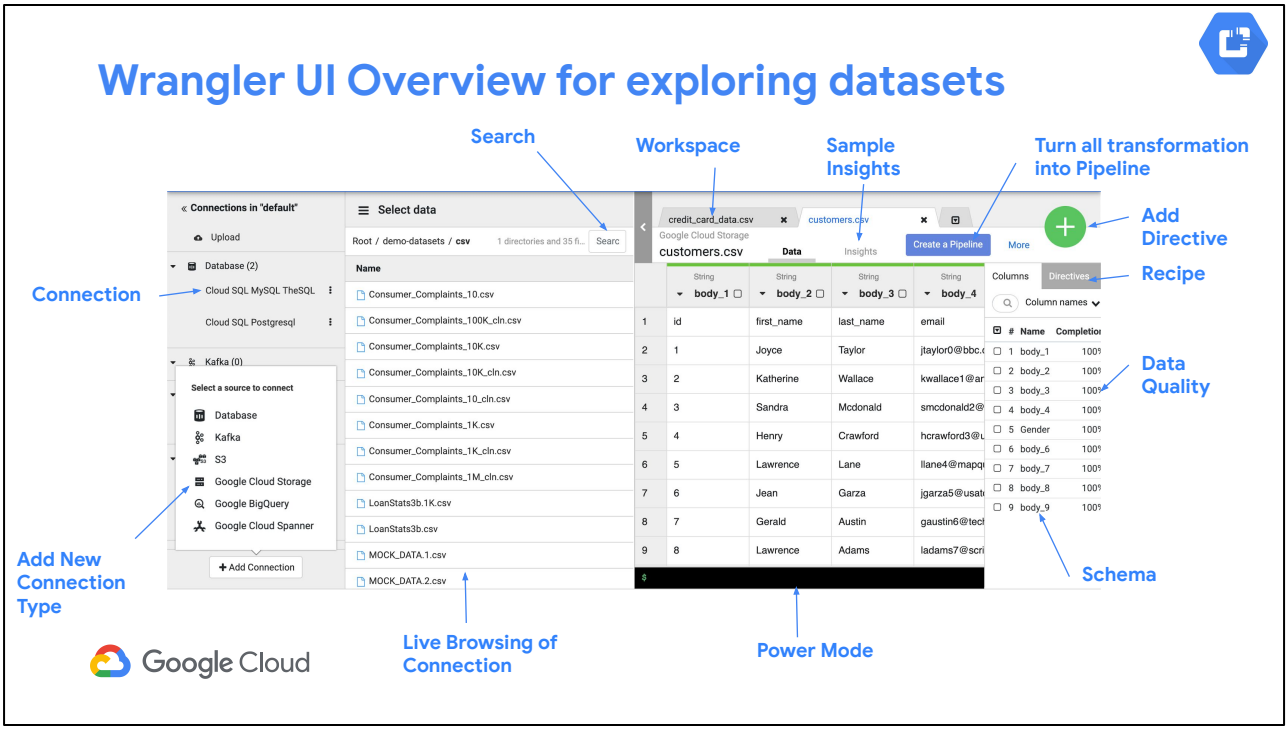
We've discussed the core components, tools, and processes of building data pipelines. Now, we'll look at using Wrangler to explore the dataset.

Data analysts can explore datasets in the Wrangler

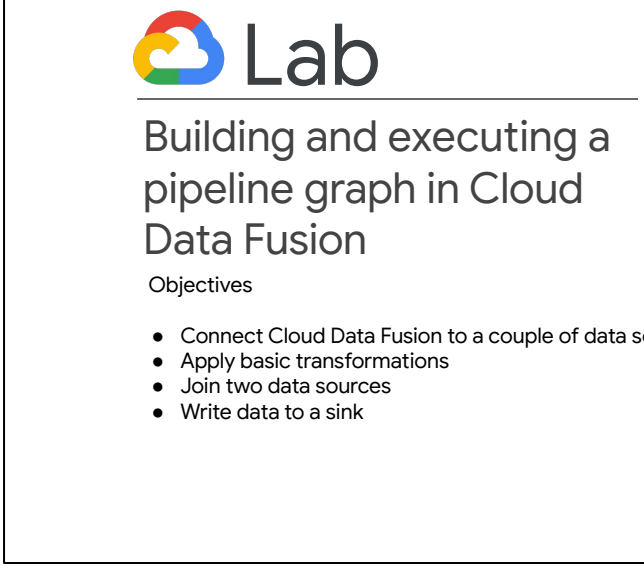
Wrangler is a code-free, visual environment for transforming data in data pipelines



So far in the course, we have focused on building new pipelines for our datasets. That presumes we know what the data is and what transformations need to be made already. Oftentimes, a new dataset still needs to be explored and analyzed for insights. The Wrangler UI is the Cloud Data Fusion environment for exploring new datasets visually for insights. Here, you can inspect the dataset and build a series of transformation steps called directives to stitch together a pipeline.



Here's what the Wrangler UI looks like. Starting from the left, you have your connections to existing datasets. Here, you can add new connections to a variety of data sources like Google Cloud Storage, BigQuery, or even other cloud providers. Once you specify your connection, you can browse all of the files or tables in that source. Here, you see a Cloud Storage bucket of demo data sets and all the CSV files of customer complaints. Once you've found an example dataset like customers.csv here, you can explore the rows and columns visually and view sample insights. As you explore the data, you might want to create new calculated fields, drop columns, filter rows, or otherwise wrangle the data. You can do so using the Wrangler UI by adding new directives to form a data transformation recipe. When you're happy with your transformations, you can create a pipeline that you can then run at regular intervals.



Building and Executing a Pipeline Graph with Data Fusion
https://gcpstaging.qwiklabs.com/labs/24798/edit

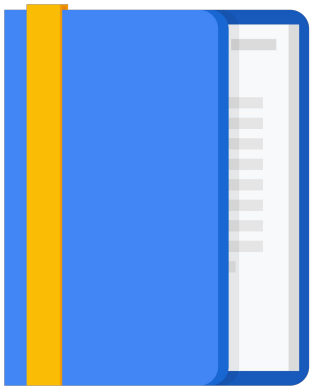
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

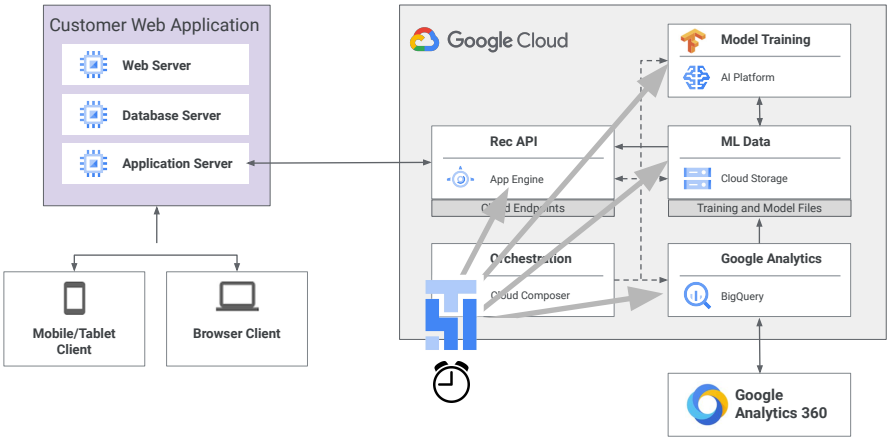
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



Google Cloud

The next big task for managing data pipelines is to orchestrate the work across multiple Google Cloud services. For example, if you had three Cloud Data Fusion Pipelines and two ML models that you want it to run in a certain order, you need an orchestration engine. In this module, we'll look at using Cloud Composer to help out with task like that.

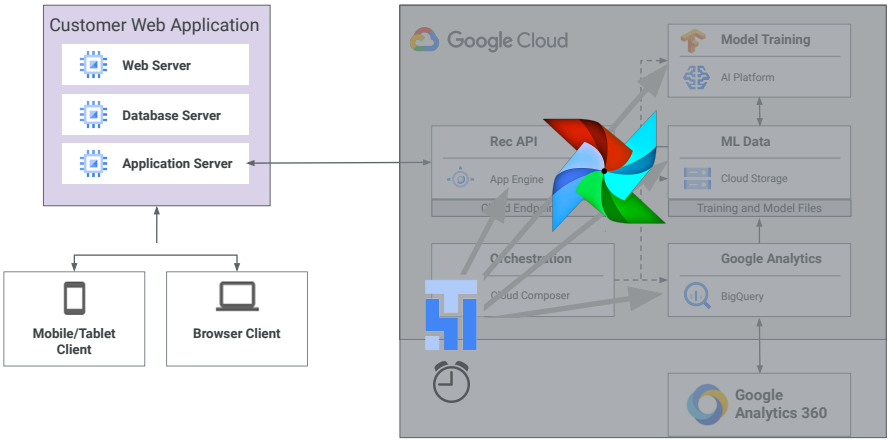
Cloud Composer orchestrates automatic workflows



Google Cloud

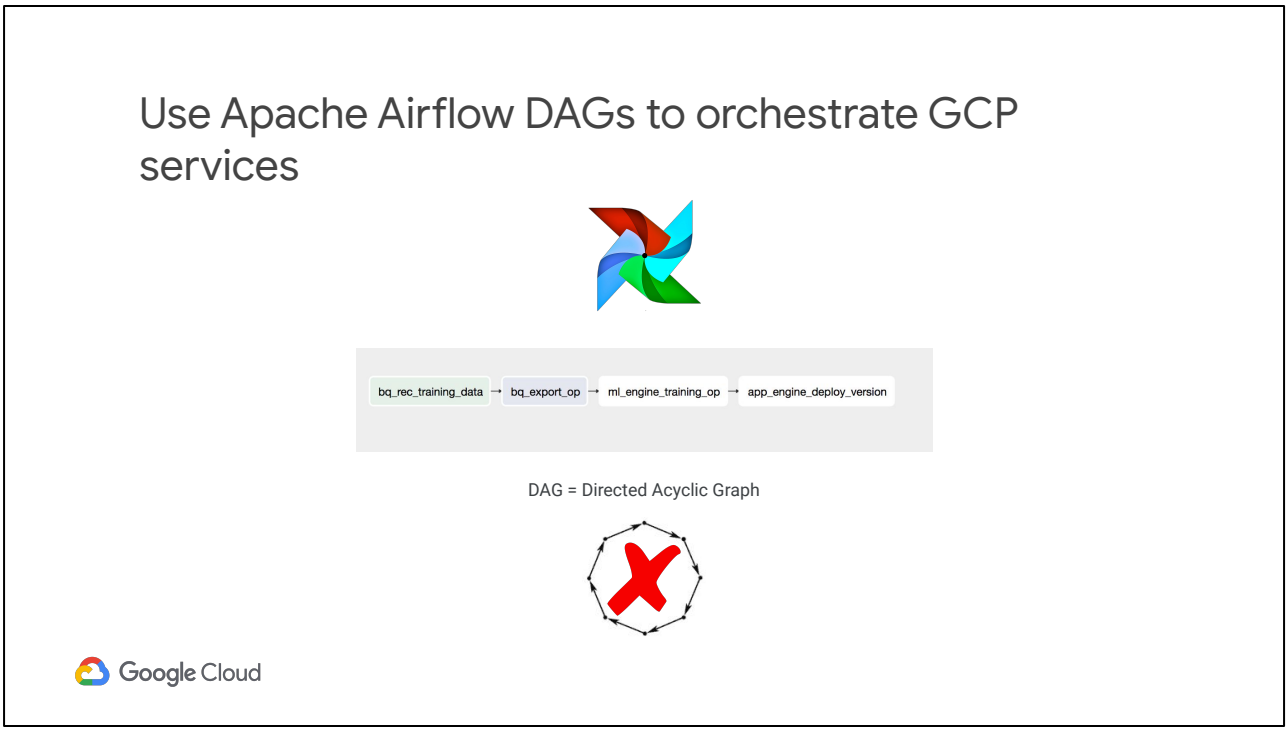
Cloud Composer will command the Google Cloud services that we need to run. But Cloud Composer is simply a serverless environment on which an open source workflow tool runs.

Cloud Composer is managed Apache Airflow



Google Cloud

That workflow tool is called Apache Airflow, which is an open-source orchestration engine.

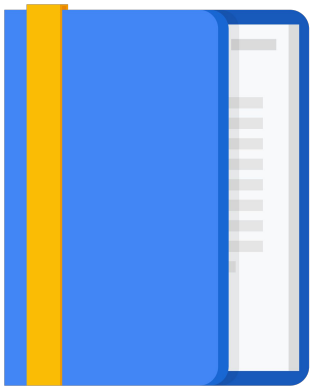


The heart of any workflow (workflow pipeline and DAG are interchangeable), is to DAG. As you saw with Cloud Data Fusion, you're also building DAGs with Apache Airflow as you see here. What's happening in this particular DAG are four tasks that update our training data, export it, we train our model, and we deploy it. You can tell your DAG to pretty much do anything you need it to do. Here it's sending tasks to BigQuery, Cloud Storage, and AI Platform, but yours could orchestrate among four completely different services.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

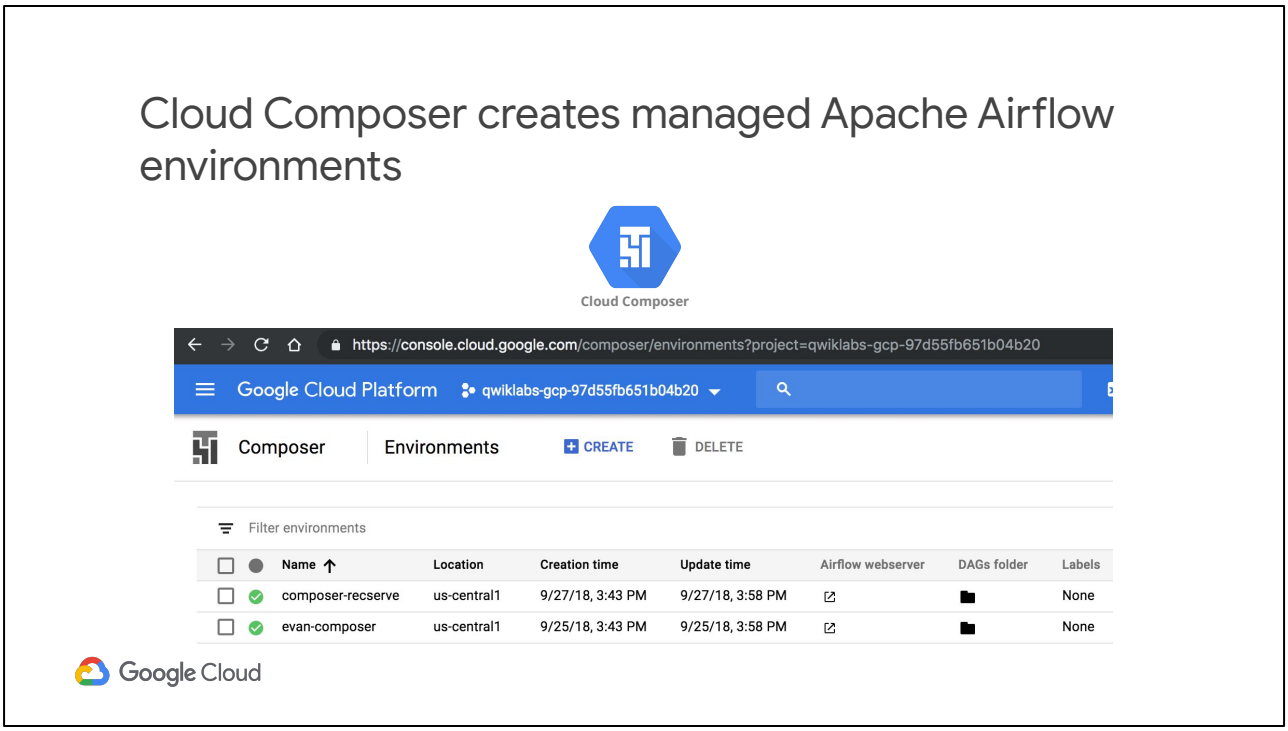
Orchestrating work between GCP services with Cloud Composer
- **Apache Airflow Environment**
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



Google Cloud

Building any workflow in Cloud Composer consists of these four steps and building a ML workflow for our recommendation engine is no different.

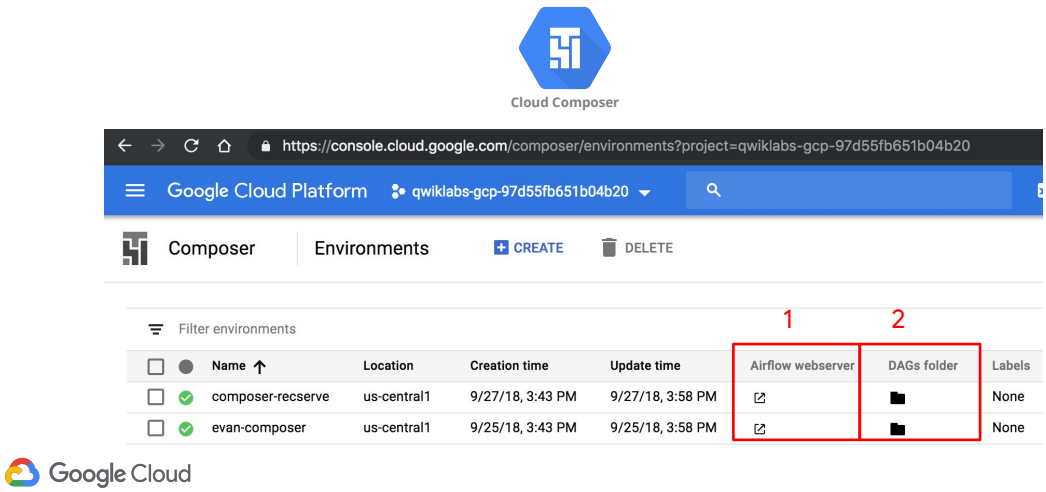
We'll start by previewing the actual Cloud Composer environment.



Once you use the command line or GCP web UI to launch a Cloud Composer instance, you'll be met with a screen like this. Keep in mind that you can have multiple Cloud Composer environments and which each environment you can have a separate Apache Airflow instance which could have zero to many DAGs.

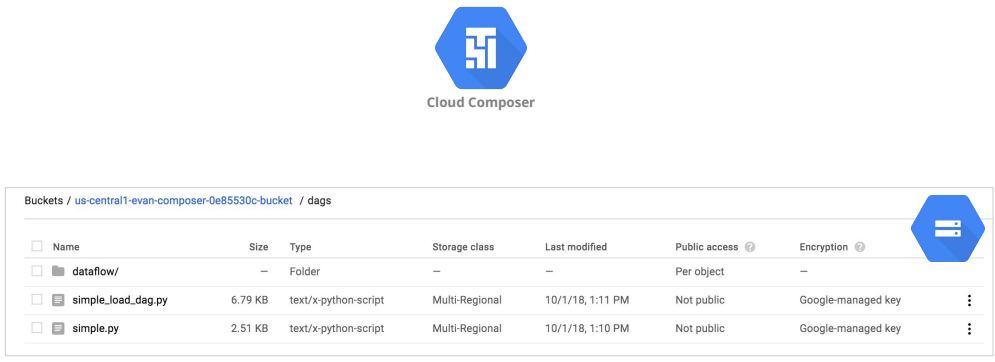
An important note here is that sometimes you'll be required to edit environment variables for your workflows (like specifying your specific GCP project account). Normally you will not do that at the Cloud Composer level but on the actual Apache Airflow instance level which we will show you in the next demo. Again, generally I'm only on the Cloud Composer page here to create new environments before I launch directly into the Airflow webserver.

Each Airflow environment has a separate webserver and folder in GCS for pipeline DAGs



To access the Airflow admin UI where you can monitor and interact with your workflows you'll click on the link underneath Airflow webserver. The second box you see is the DAGs folder which is where the code of your actual workflows will be stored.

The DAGs folder is simply a GCS bucket where you will load your pipeline code



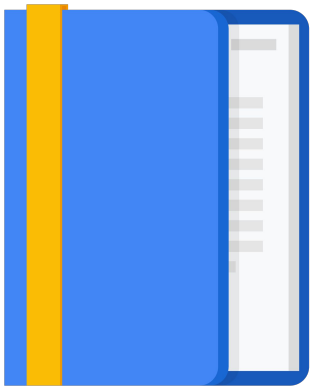
Google Cloud

The DAGs folder for each Airflow instance is simply a GCS bucket that is automatically created for you when you launch your Cloud Composer instance. Here is where you upload your DAG files, written in python, and bring your first workflow to life in Airflow. Let's take a look at a quick demo of what the simplest DAG looks like in the UI.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

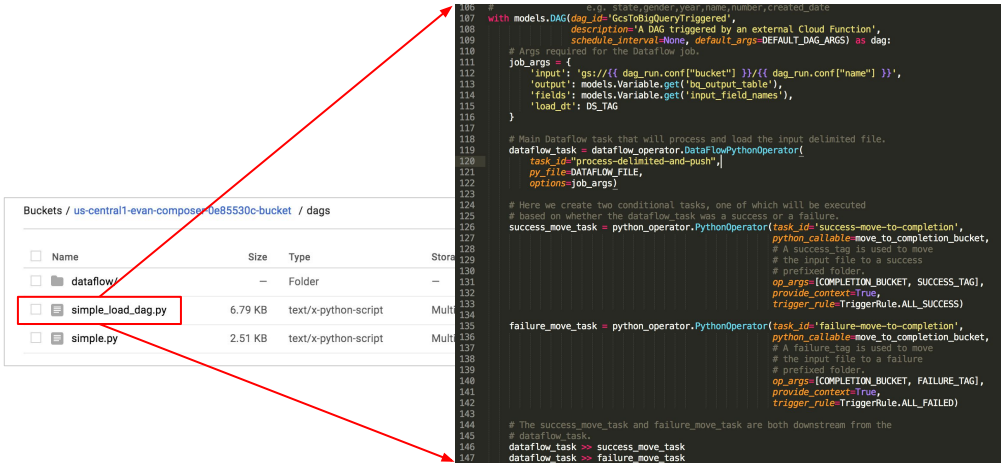
Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- **DAGs and Operators**
- Workflow Scheduling
- Monitoring and Logging



Google Cloud

Now that you're familiar with the basic environment setup, it's time to discuss your primary artifact which is your DAG and the operators you are using to call whichever services you want to send tasks to.

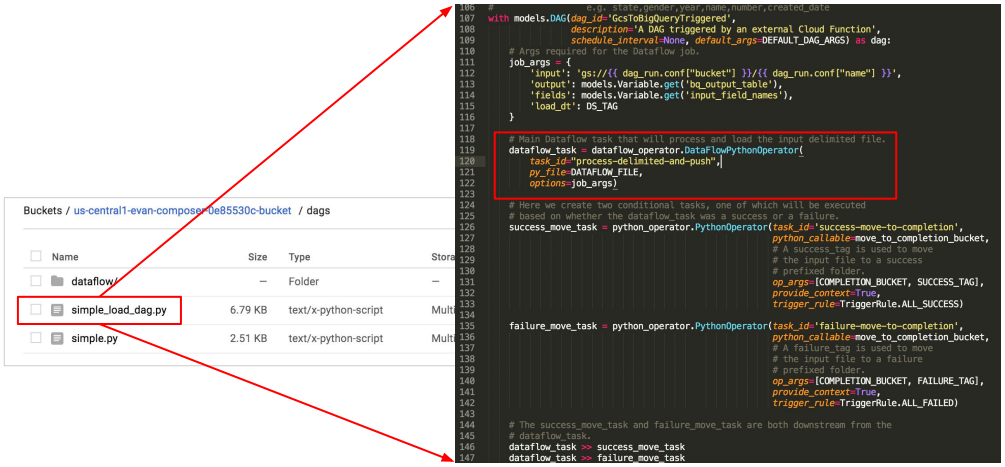
Airflow workflows are written in Python



Google Cloud

First, Airflow workflows are written in python. You'll have one python file for each DAG. For example, here we have simple_load_dag.py in our DAG Folder GCS bucket and you can see a preview of what the DAG file looks like. Don't worry about reading the code, we'll go into that later. It's sufficient enough for now to just know that there are a series of user-created tasks in each DAG file that invoke predefined operators.

Airflow workflows are written in Python

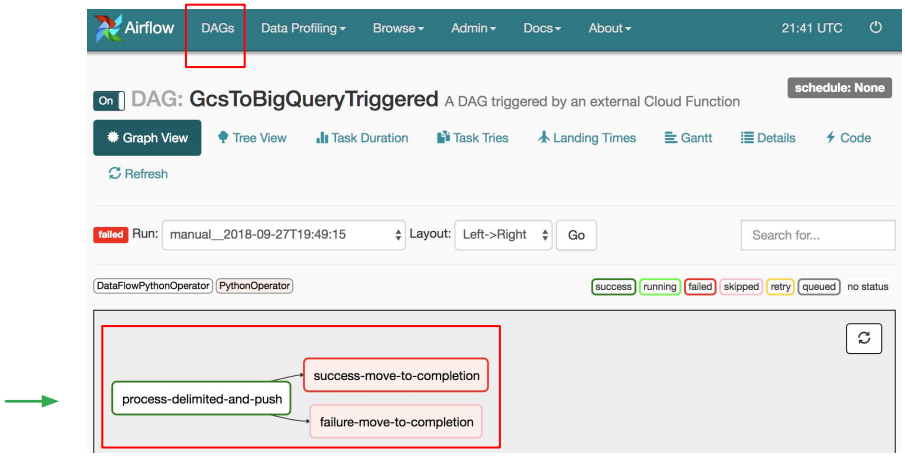


Google Cloud

Like this task which uses the DataFlowPythonOperator and is given the task_id of “process-delimited-and-push”.

We’ll go over creating a DAG file and it’s components a little later

The python file creates a DAG



Google Cloud

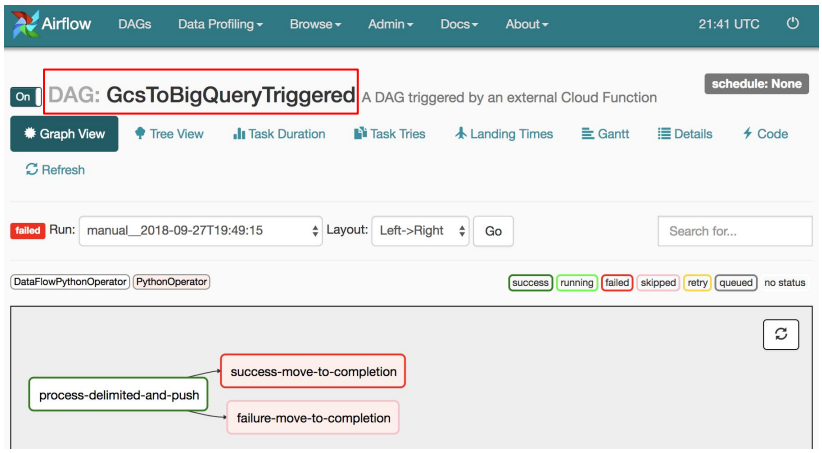
Once you've uploaded the python file to the DAGs folder, you can navigate back to the Airflow webserver and under DAGs you'll see the DAG you created with code represented visually as a directed graph (nodes and edges).

You'll remember that the python code that defined a task we called "process-delimited-and-push" is now a node in our graph here.

Let's explore a bit more of the Airflow web UI.

// Great DAG tutorial: https://airflow.apache.org/tutorial.html

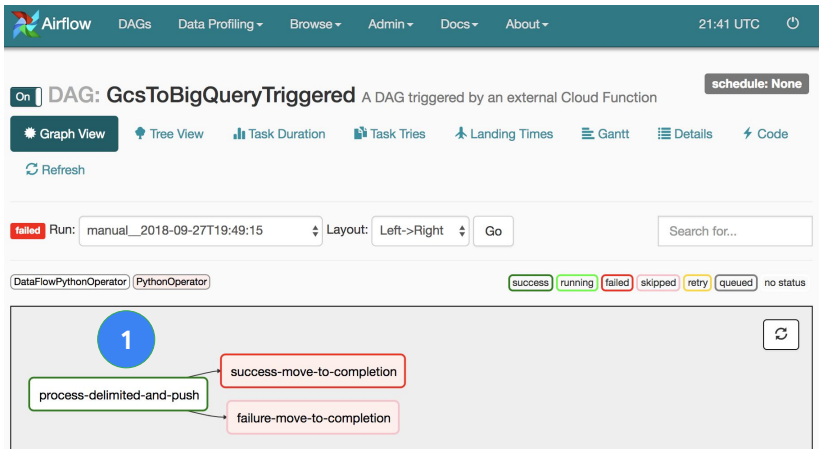
Airflow webserver UI overview



Google Cloud

You can see that this particular workflow is called GcsToBigQueryTriggered and it has three TASKS when it runs.

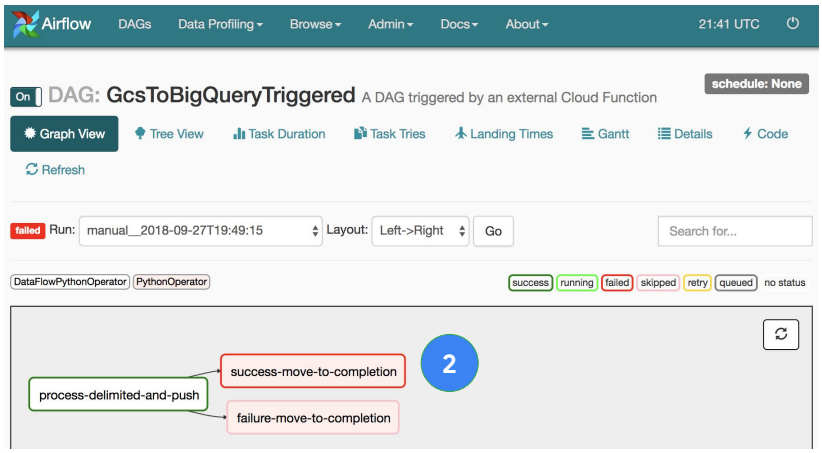
The python file creates a DAG (Directed Acyclic Graph)



Google Cloud

1) process-delimited-and-push which I just happen to know from that python file you saw earlier that this task invokes a Dataflow job to read in a new CSV file from a GCS bucket, processes it, and writes the output to BigQuery

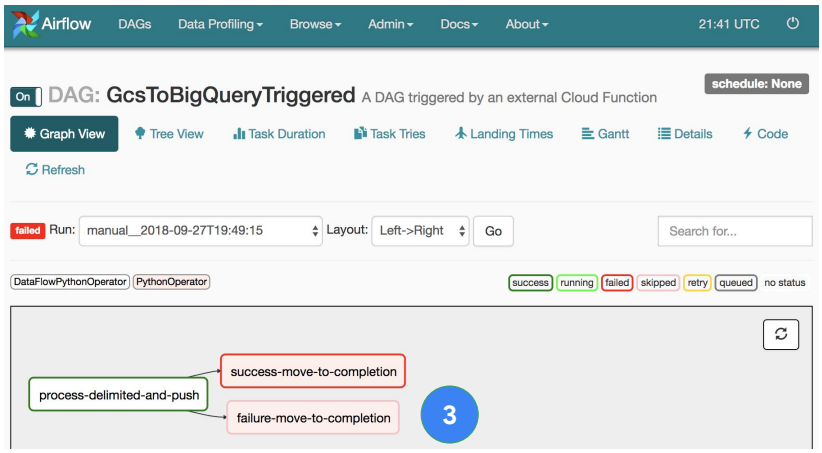
The python file creates a DAG (Directed Acyclic Graph)



Google Cloud

2) success-move-to-completion which moves the CSV file from an input GCS bucket to a processed or completed GCS bucket for archiving

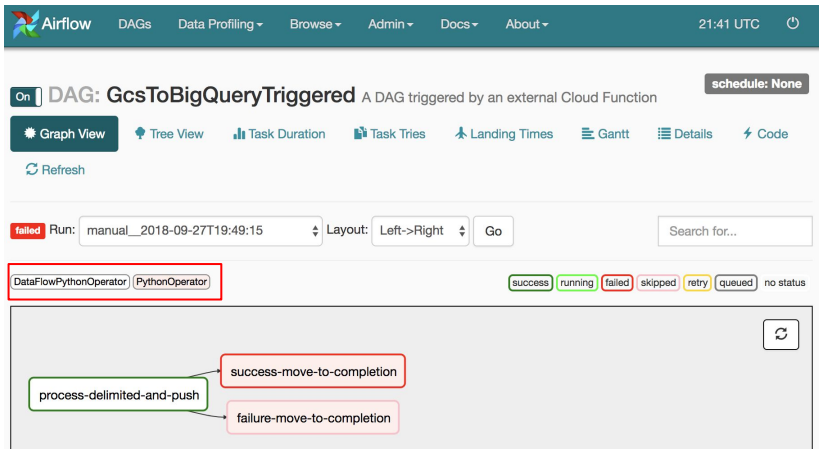
The python file creates a DAG (Directed Acyclic Graph)



Google Cloud

or 3) if the pipeline fails part way the file is moved to the completion bucket but tagged as failure. This is an example of a DAG which isn't strictly sequential. There a decision made to run one node or a different one based on the outcome of the parent node.

The python file creates a DAG (Directed Acyclic Graph)

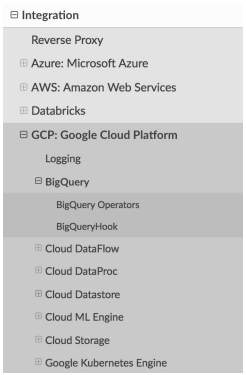


Google Cloud

But regardless of the size and shape of your workflow DAG, one common thread for all workflows is the common Operators used. If the DAG itself is HOW to run the workflow (first do step 1 then either move to step 2 or 3) the operators specify WHAT actually gets done as part of the task.

In this simple example we're calling on the DataFlowPythonOperator and the general PythonOperator. Those are by no means the only operators so let's pause here and look at all the operators at our disposal to achieve our goal of automatic retraining and deployment of our ML model.

Airflow uses operators in your DAG to orchestrate other GCP services



View all GCP services that Airflow can orchestrate:
http://airflow.apache.org/integration.html#gcp

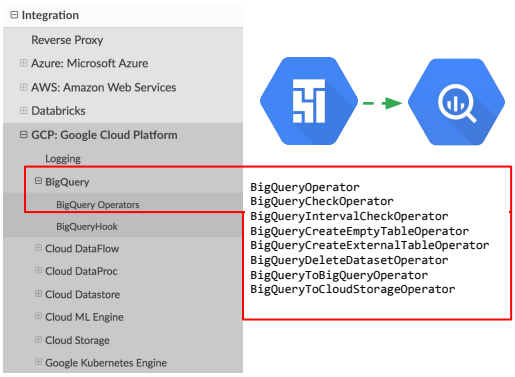
+ sample code

Google Cloud

Airflow has many operators which you can invoke the TASKS you want to complete. Operators are usually atomic in a task (which means generally you only see one operator per task). I've taken this list directly from the Apache Airflow docs of all services that Airflow can orchestrate to. Let's take a look at the ones that are most relevant to us as ML Engineers.

List of all operators:
https://cloud.google.com/composer/docs/how-to/using/writing-dags#gcp_name_operators
https://airflow.apache.org/integration.html#gcp
https://airflow.apache.org/concepts.html#operators

BigQuery Operators are popular for updating your ML training dataset



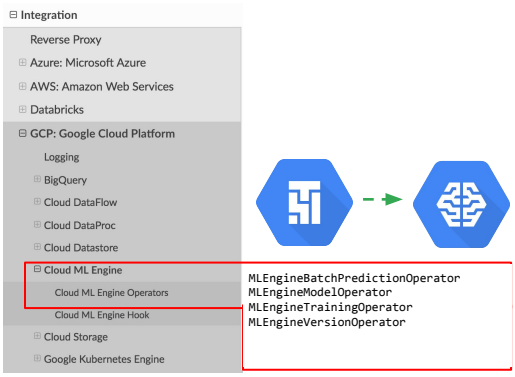
https://airflow.apache.org/integration.html#gcp

Google Cloud

As you might have guessed, we'll certainly be making use of the BigQuery operators since our workflows live and die by the data that is fed into them through GCS and BigQuery. Here's a list of the specific operators that we can invoke in a task to call on the BigQuery service for querying and other data-related tasks. You'll be mainly working with the first three in this course but I encourage you to skim the resource link on all the operators so you can get a feel for what is possible.

List of all operators:
https://cloud.google.com/composer/docs/how-to/using/writing-dags#gcp_name_operators
https://airflow.apache.org/integration.html#gcp

Cloud AI Platform (formerly Cloud ML Engine) operators can launch training and deployment jobs



https://airflow.apache.org/integration.html#cloud-ml-engine-operators

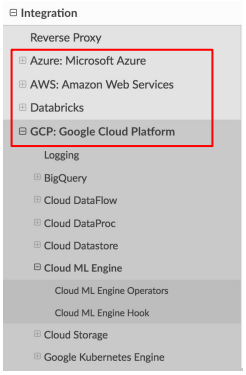
Google Cloud

Once we have our training data in a good place, the next logical step in our workflow is to retrain and redeploy our model. In the same DAG file after the BigQuery operators complete we can make a service call through a Cloud ML Engine operator to kick off a new training job and manage our model like incrementing the version.

https://airflow.apache.org/integration.html#cloud-ml-engine-operators

List of all operators:
https://cloud.google.com/composer/docs/how-to/using/writing-dags#gcp_name_operators
https://airflow.apache.org/integration.html#gcp

Apache Airflow is open source and cross-platform for hybrid pipelines



https://airflow.apache.org/integration.html#cloud-ml-engine-operators

Google Cloud

You might have noticed that your Airflow DAG can have operators that send tasks out to other cloud providers. This is great for hybrid workflows where you have components across multiple cloud platforms or even on-premise. Apache Airflow is open source and continually adds more operators to other services so be sure to check out the list in the documentation periodically if you're waiting for a new service to be added.

https://airflow.apache.org/integration.html#cloud-ml-engine-operators

Example: Common Machine Learning workflow DAG Operators


```
# update training data
t1 = BigQueryOperator(
)

# BigQuery training data export to GCS
t2 = BigQueryToCloudStorageOperator(
)

# AI Platform training job
t3 = MLEngineTrainingOperator(
)

# App Engine deploy new version
t4 = AppEngineVersionOperator(
)

# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

1 2 3 4

Google Cloud

Here you see four tasks t1, t2, t3, and t4 and four operators corresponding to four Google Cloud Platform services. The names should look familiar and you can probably start to guess what this pipeline does at a high level just by reading them in order.

BigQuery and Cloud Storage operators get us fresh training data

```
# update training data
t1 = BigQueryOperator(
)

# BigQuery training data export to GCS
t2 = BigQueryToCloudStorageOperator(
)

# AI Platform training job
t3 = MLEngineTrainingOperator(
)

# App Engine deploy new version
t4 = AppEngineVersionOperator(
)

# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

1 2 3 4

Google Cloud

The first two are concerned with *getting fresh model data* from a BigQuery dataset and into Cloud Storage for consumption by our ML model later in the pipeline.

In the lab you're going to work on later the dataset will be the one you're already familiar with, the Google Analytics news articles sample dataset.

Let's see the parameters the BigQueryOperator takes.

Use the BigQueryOperator to run SQL

```
from airflow.contrib.operators import bigquery_operator

# constants or can be dynamic based on Airflow macros
max_query_date = '2018-02-01'
min_query_date = '2018-01-01'

    # Query recent StackOverflow questions.
    bq_recent_questions_query = bigquery_operator.BigQueryOperator(

        task_id='bq_recent_questions_query',

        bql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < CAST('{max_date}' AS TIMESTAMP)
            AND creation_date >= CAST('{min_date}' AS TIMESTAMP)
        ORDER BY view_count DESC
        LIMIT 100
        """.format(max_date=max_query_date, min_date=min_query_date),

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

Google Cloud

The BigQueryOperator allows you to specify a SQL query to run against a BigQuery dataset.

BigQuery Operator
https://cloud.google.com/composer/docs/how-to/using/writing-dags#gcp_name_operators
https://airflow.apache.org/integration.html#bigqueryoperator

StackOverflow email notifier:
https://github.com/GoogleCloudPlatform/python-docs-samples/blob/master/composer/workflows/bq_notify.py

SQL commands can hold parameters (from Python)

```
from airflow.contrib.operators import bigquery_operator

# constants or can be dynamic based on Airflow macros
max_query_date = '2018-02-01'
min_query_date = '2018-01-01'

    # Query recent StackOverflow questions.
    bq_recent_questions_query = bigquery_operator.BigQueryOperator(

        task_id='bq_recent_questions_query',

        bql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < CAST('{max_date}' AS TIMESTAMP)
            AND creation_date >= CAST('{min_date}' AS TIMESTAMP)
        ORDER BY view_count DESC
        LIMIT 100
        """.format(max_date=max_query_date, min_date=min_query_date),

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

Google Cloud

In this quick example we're passing in a query which returns the top 100 most popular Stackoverflow posts from the BigQuery public dataset for a specified date range you see there in the WHERE clause. Notice anything different about the filters in the WHERE clause?

Yes! they are parameters in this case for max_date and min_date

Note the Python constants for a date range here

```
from airflow.contrib.operators import bigquery_operator

# constants or can be dynamic based on Airflow macros
max_query_date = '2018-02-01'
min_query_date = '2018-01-01'

    # Query recent StackOverflow questions.
    bq_recent_questions_query = bigquery_operator.BigQueryOperator(

        task_id='bq_recent_questions_query',

        bql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < CAST('{max_date}' AS TIMESTAMP)
            AND creation_date >= CAST('{min_date}' AS TIMESTAMP)
        ORDER BY view_count DESC
        LIMIT 100
        """.format(max_date=max_query_date, min_date=min_query_date),

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

Google Cloud

You can parameterize pieces of the SQL statement like what we did here to only return posts for January 2018 with min_query_date and max_query_date.

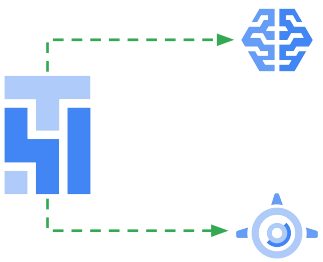
You could even set a rolling window with macros

```
from airflow.contrib.operators import bigquery_operator

# constants or can be dynamic based on Airflow macros
max_query_date = {{ macros.ds_add(ds, -1) }}  # one day prior
min_query_date = {{ macros.ds_add(ds, -7) }}  # seven days prior

    # Query recent StackOverflow questions.
    bq_recent_questions_query = bigquery_operator.BigQueryOperator(

        task_id='bq_recent_questions_query',

        bql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < CAST('{max_date}' AS TIMESTAMP)
            AND creation_date >= CAST('{min_date}' AS TIMESTAMP)
        ORDER BY view_count DESC
        LIMIT 100
        """.format(max_date=max_query_date, min_date=min_query_date),

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

Google Cloud

What's really neat is you can even make the parameters dynamic (instead of the static ones shown here) and have them be based on the DAG schedule date like {{ macros.ds_add(ds, -7) }} which is a week before the DAG scheduled run date.

AI Platform and App Engine operators retrain and redeploy our model



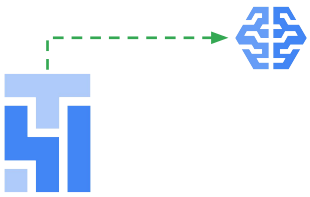
```
# update training data
t1 = BigQueryOperator(
)

# BigQuery training data export to GCS
t2 = BigQueryToCloudStorageOperator(
)

# AI Platform training job
t3 = MLEngineTrainingOperator(
)

# App Engine deploy new version
t4 = AppEngineVersionOperator(
)

# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

1 2 3 4

Google Cloud

The next two operators handle re-training the model by submitting a job to Cloud Machine Learning Engine and then deploying the updated model to App Engine.

Use Cloud ML Engine operators to periodically submit new training jobs



```
t3 = MLEngineTrainingOperator(
    task_id='ml_engine_training_op',
    project_id=PROJECT_ID,
    job_id=job_id,
    package_uris=[PACKAGE_URI],
    training_python_module='trainer.task',
    training_args=training_args,
    region=REGION,
    scale_tier='CUSTOM',
    master_type='complex_model_m_gpu',
    dag=dag
)
```

Google Cloud

The MLEngineTrainingOperator is how Cloud Composer interacts with Cloud Machine Learning Engine and thus gives you the ability to periodically schedule new jobs to be sent to CMLE as part of your automated workflow. Take a look through the parameters you provide to the operator.

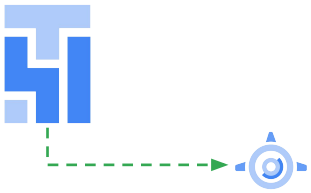
First we specify an ID for the task that we're running and then the project we want to run it on. For our lab, this will be the same project that is also managing the Airflow Instance and everything else. Then you have the option of creating your own job id, which usually involves a concatenated timestamp or similar unique identifier. After that is path to the actual model code which we'll have in a zipped file which is created by a shell script after training is run in our lab.

Then the actual Python module name within your code to run within the MLEngine training job after installing the package. In this example we've called it trainer.task

Next is a series of arguments for training which we store as an array called "training_args" which includes the location of the job directory, where the training files are, where the output directory is, and what data we're using. The next three: region, scale_tier, and master_type are GCP infrastructure parameters for where you want the job to be ran and with what speciality hardware like GPUs or TPUs if any.

https://airflow.apache.org/_modules/airflow/contrib/operators/mlengine_operator.html

Use App Engine operators to periodically deploy and redeploy models



```
t4 = AppEngineVersionOperator(
    task_id='app_engine_deploy_version',
    project_id=PROJECT_ID,
    service_id='default',
    region=REGION,
    service_spec=None,
    dag=dag
)
```

Google Cloud

Lastly, once your model is retrained and ready to be delivered as an API endpoint for serving, we need to redeploy our app engine project with the latest model. Here the parameters are simple since we're just redeploying our existing app engine instance associated with our project. And that's it! Well almost. That's the last TASK but at the end of most all DAG files you'll find

Manage pipelines and dependencies as code

```
# update training data
t1 = BigQueryOperator(
)

# BigQuery training data export to GCS
t2 = BigQueryToCloudStorageOperator(
)

# AI Platform training job
t3 = MLEngineTrainingOperator(
)

# App Engine deploy new version
t4 = AppEngineVersionOperator(
)

# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

The actual order in which we want these operators to run. This is what the D in DAG is for: Directed. For our example, t2 or task 2 won't run until t1 has completed. This is what gives our graph the dependencies of a workflow. I can probably guess what you're thinking at this point, you could build some cool branching of multiple child nodes per upstream parent and that is totally possible. Just don't forget to comment your code so you know where one branch begins, where it's going, and what tasks are involved. As a tip, after you load your DAG file into the DAG folder you can see the visualization of your DAG in the airflow UI as a directed graph, a gantt chart, or a list if you wanted. Reviewing the visual representation of the ordered tasks will help you confirm your tasks are ordered properly.

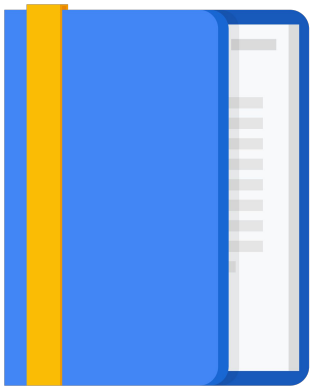
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

- Apache Airflow Environment
- DAGs and Operators
- **Workflow Scheduling**
- Monitoring and Logging



Google Cloud

Now that you're familiar with the Cloud Composer and Apache Airflow environments AND the basics of building a list of tasks for GCP services in your DAG, it is time to discuss a really important topic: workflow scheduling.

Two scheduling options for Cloud Composer workflows

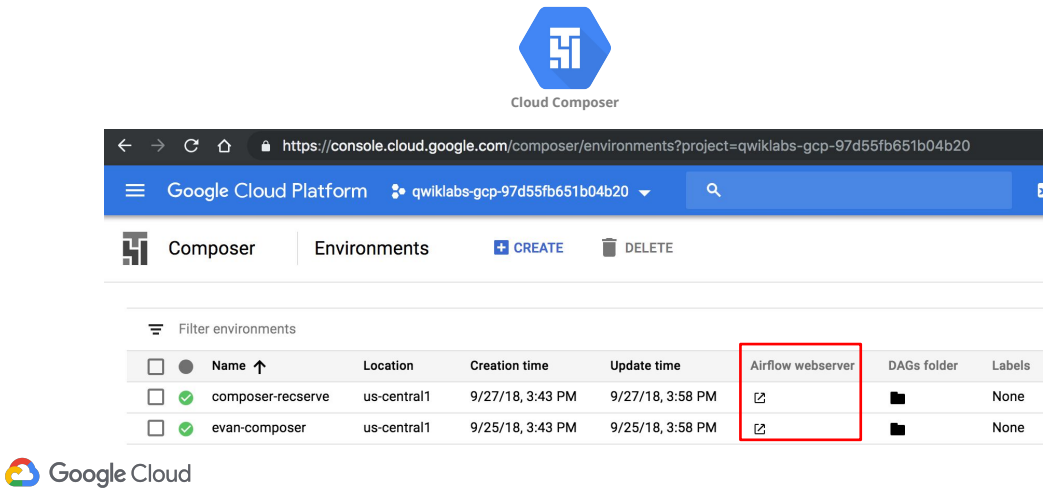
Periodic

Event-driven

Google Cloud

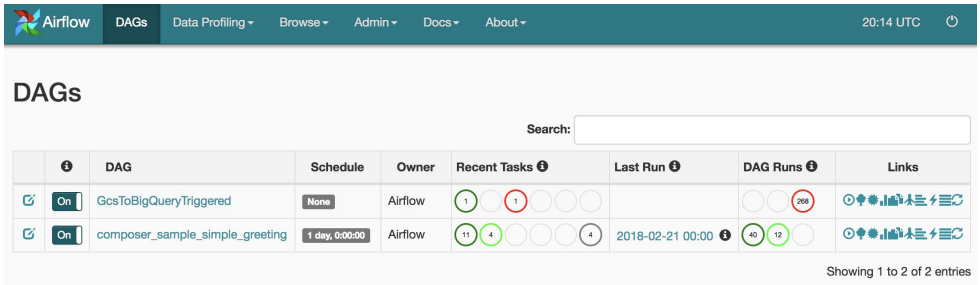
As we hinted at earlier, there are two different ways your workflow can be ran without you sitting there manually clicking “run DAG”. The first and most common is a set schedule or periodic run of a workflow like once a day at 6am or weekly on Saturdays. The second way is trigger-based like if you wanted to run your workflow whenever new CSV data files were loaded into a GCS bucket or if new data came in from a PUB/SUB topic you’ve subscribed to.

Launch the Airflow webserver to interact with your DAGs



To view the schedules for your DAGs you first launch the Airflow webserver from within Cloud Composer. This is a great link to bookmark.

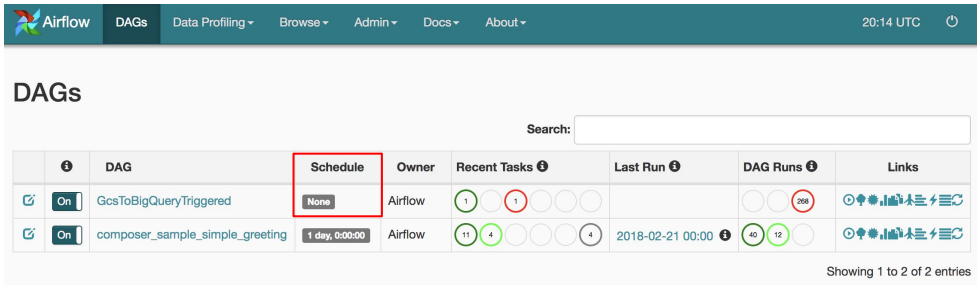
Airflow scheduling basics



Google Cloud

Then navigate to the DAGs tab to view the existing workflows that you have python DAG files for. Here we have two DAGs. The bottom one, composer_sample_simple_greeting has a daily schedule but..

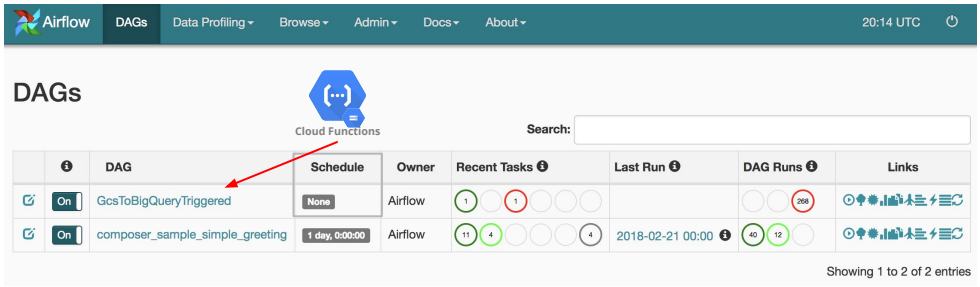
Why is this DAG missing a schedule?



Google Cloud

..why is this top DAG missing a schedule? How would it ever get run?

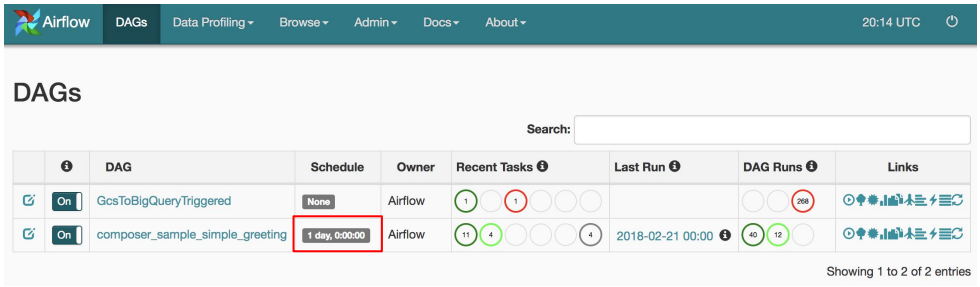
Option 1: Event-driven scheduling with Cloud Functions



Google Cloud

The answer is the fact that it's not on a set schedule at all. It's event driven. The driver of when this workflow runs is a Cloud Function that we create. In the next lesson, we'll actually create our own Cloud Function that watches a GCS bucket for new CSV files.

Option 2: Specify pipeline schedule_interval in your DAG

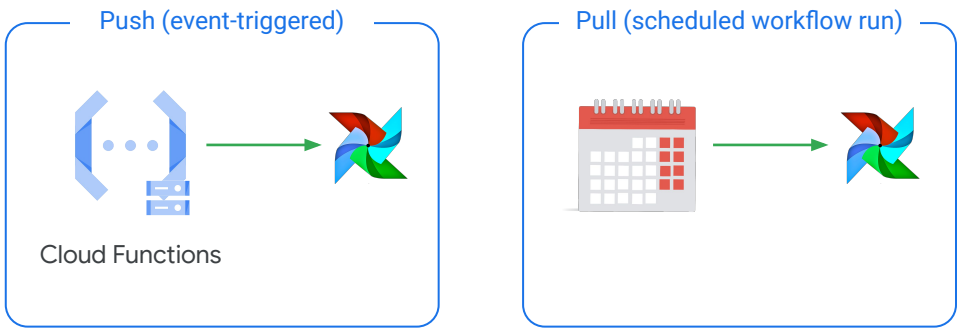



```
with models.DAG(
        'composer_sample_simple_greeting',
        schedule_interval=datetime.timedelta(days=1),
        default_args=default_dag_args) as dag:
```

Google Cloud

If you wanted to go the regular schedule route you can specify the schedule_interval in your DAG code like waht you see here. By the way, clicking on the schedule of 1 day here in the UI won't allow you to edit it there but instead will take you to the history of all the runs for that workflow.

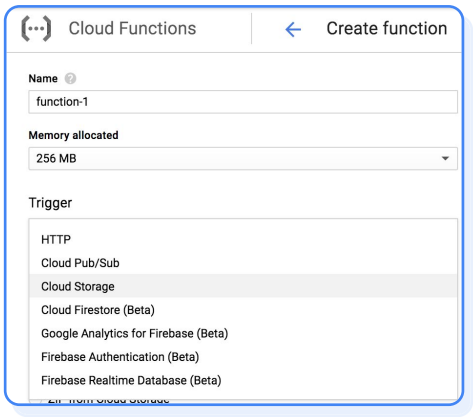
Two types of workflow ETL patterns



Google Cloud

As you saw earlier in this course, there are two general patterns for ETL workflows. Event triggered or push (as-in I push a new file to Cloud Storage and my workflow kicks off) or PULL which is where Airflow at a set time could look in my Cloud Storage folder and take all the contents that are found for it's workflow run.

Use Cloud Functions to create an event-based push architecture



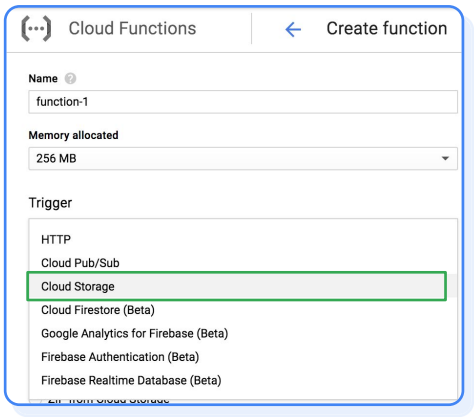
Google Cloud

We can use Cloud Functions to create our event-driven or push architecture workflow. I mentioned triggering on events within a GCS bucket but you can also trigger based on HTTP requests, Pub/sub, Firestore, Firebase and more as you see here.

Generally, push technology is GREAT when wanting to distribute transactions as-they-happen. Stock tickers, and other types of financial institution transactions are very important when it comes to push technology. How about disasters and notification? Again, important.

For ML workflows where your upstream data doesnt arrive at a regular pace (like get all the transactions at the end of each day) consider experimenting with a push architecture. Your final lab, since it's based on regular Google Analytics news article data will be a pull architecture but I've added in an optional lab for you to get practice with Cloud Functions and event-driven workflows for those interested so let's talk through it more now.

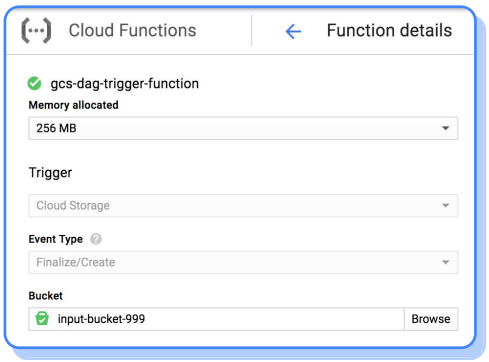
Example: Trigger an event when new data is loaded to Cloud Storage



Google Cloud

For our example let's assume we have a CSV file or set of files loaded to Cloud Storage so we'll choose a Cloud Storage trigger for our function.

Creating a Cloud Function to trigger an Airflow DAG



Google Cloud

Then we specify an Event Type (Finalize/Create new files) and a bucket to watch.

https://cloud.google.com/composer/docs/how-to/using/triggering-with-gcf

Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
/**
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} callback The callback function.
 */
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};

```

Google Cloud

As part of the Cloud Function, we need to well… create an actual function is javascript that we want called. The good news is most of this code for triggering Airflow DAGs in a function is all boilerplate for you to copy from as a starting point.

Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} callback The callback function.
 */
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};
```

Google Cloud

Here we specify a name for our function called triggerDag

Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
* @param {!Object} event The Cloud Functions event.
* @param {!Function} callback The callback function.
*/
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};
```

Google Cloud

Then we tell it where your Airflow environment is to be triggered and which DAG in that Airflow environment. In this case it's looking for one called GcsToBigQueryTriggered

Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
/**
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} callback The callback function.
 */
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};

```

Google Cloud

Keep in mind you can have multiple workflows or DAGs in a single Airflow environment so be sure you specify the correct DAG_NAME to trigger!

Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
* @param {!Object} event The Cloud Functions event.
* @param {!Function} callback The callback function.
*/
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};
```

Google Cloud

Then we have a few constants that are provided which construct the Airflow URL that we're going to trigger a POST request to as well as who's making the request and what is the body of the request is.

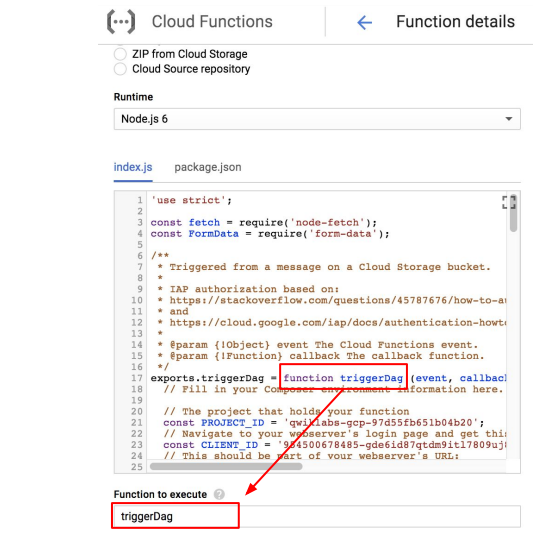
Creating a Cloud Function GCS trigger an Airflow DAG

index.js package.json

```
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} callback The callback function.
 */
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};
```

Google Cloud

Lastly, the triggerDag function makes the actual request against the Airflow server to kick-off a workflow DAG.



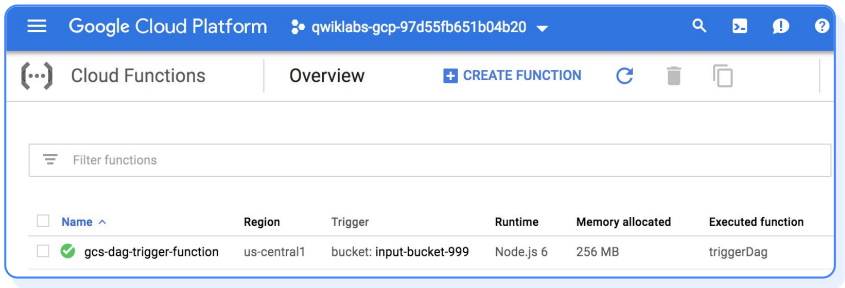
Be sure to specify the function you want Cloud Functions to call in your script

Google Cloud

Once you've got the Cloud Function code ready in your index.js file and the metadata about the function in package.json (which contains code dependency and versioning information) you still need to specify which function you actually want executed. In this case we created on called triggerDag so we just copy that down.

I'll also save you about 20 minutes of frustration and tell you that the 'function to execute' box is case sensitive so all capital letters DAG is different than capital D lowercase a and g.

Cloud Function is triggered whenever a new file is loaded to a specific Cloud Storage bucket



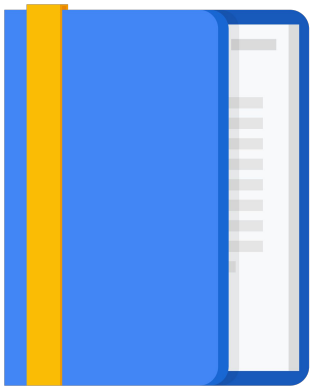
Google Cloud

And there you have it! Your Cloud Function has been created and is actively watching your Cloud Storage bucket for file uploads. But how can you be sure everything is working as intended? For that, check out the next topic on monitoring and logging.

Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

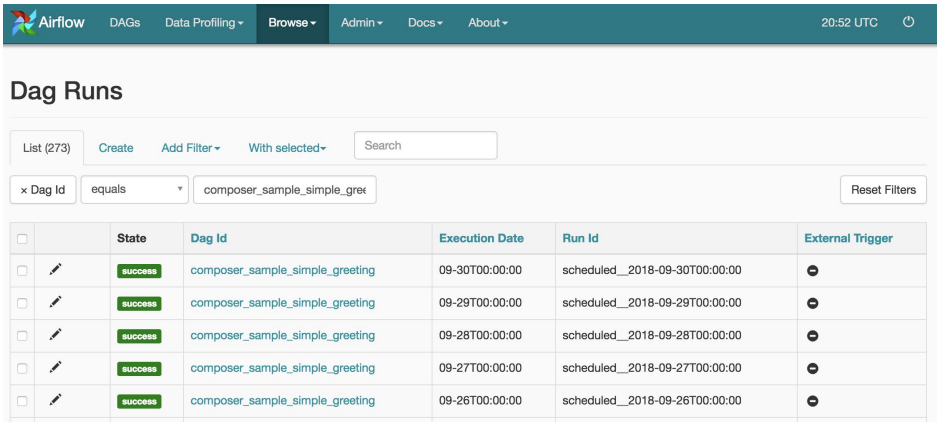
Orchestrating work between GCP services with Cloud Composer
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- **Monitoring and Logging**



Google Cloud

By this point, we've got our environment setup with our DAGs running at predefined schedule or with triggered events. The last topic we'll cover before you practice what you've learned in your labs is how to monitor and troubleshoot your cloud functions and Airflow workflows.

Monitor current and historical workflow progress

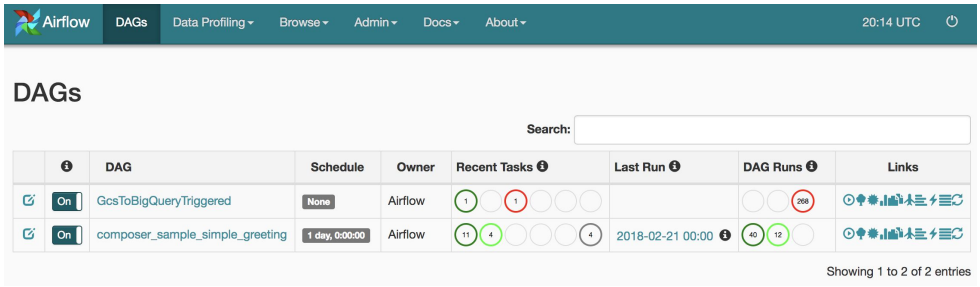


Google Cloud

One of the most common reasons you'll want to investigate the historical runs of your DAGs is in the event that your workflow simply stops working. Note that you can have it auto-retry for a set number of attempts in case it's a transient bug but sometimes you just cant get your workflow to run at all in the beginning.

In the Dag Runs you can monitor when your pipelines ran and in what State like success, running, or failure. The quickest way to get to this page is clicking on the schedule for any of your DAGs from the main DAGs page. Here we have 5 successful runs over 5 days for this DAG so this one seems to be running just fine.

Quickly gauge pipeline health of DAG Runs

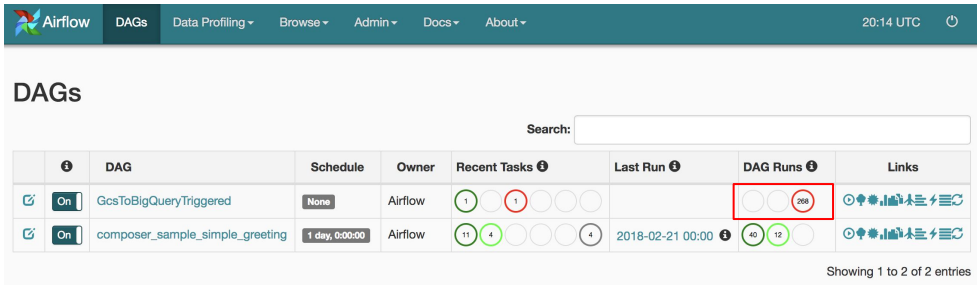


Which pipeline is healthier?

Google Cloud

Back on the main page for DAGs, we see some red which indicates trouble with some of our recent DAG runs.

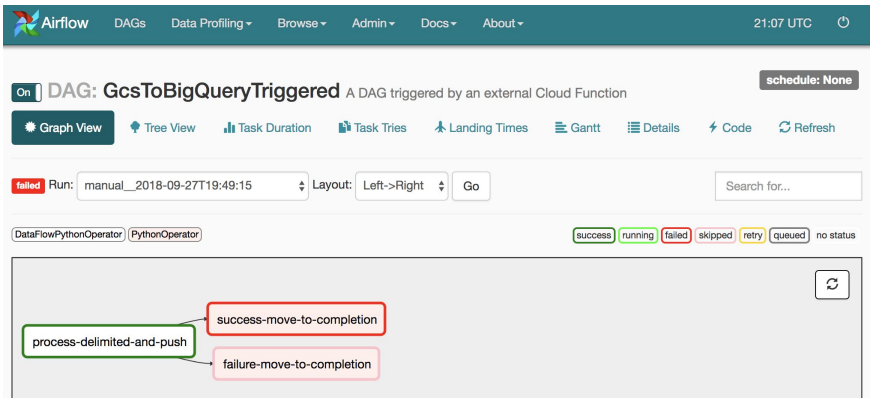
Quickly gauge pipeline health of DAG Runs



Google Cloud

Speaking of DAG Runs, you'll note the three circles below which indicates how many runs passed, are currently active, or have failed. It certainly doesn't look good for 268 runs failed and 0 passed for this first DAG. Let's see what happened. We click on the name of the DAG to get to the visual representation.

Quickly gauge pipeline health of DAG Runs

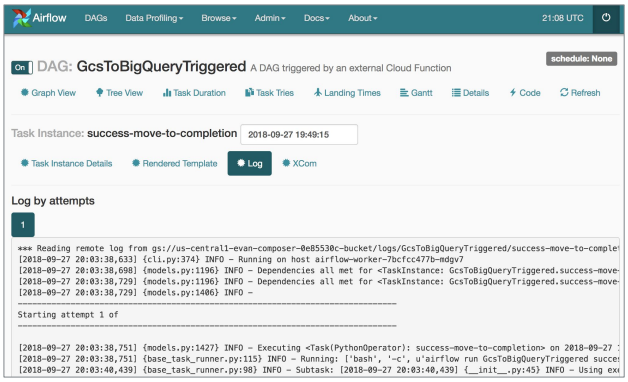


Google Cloud

It looks like the first task is succeeding, judging by the green border, but the next task ‘success-move-to-completion’ is failing. Note that the lighter pink color for the ‘failure-move-to-completion’ node means that node was skipped. So reading into this a bit, the CSV file was correctly processed by Dataflow in the first task but there is some issue moving the CSV file to a different Cloud Storage bucket as part of task two.

To troubleshoot, click on the node of a particular task and then click LOGS.

Monitor and troubleshoot Airflow step errors in logs

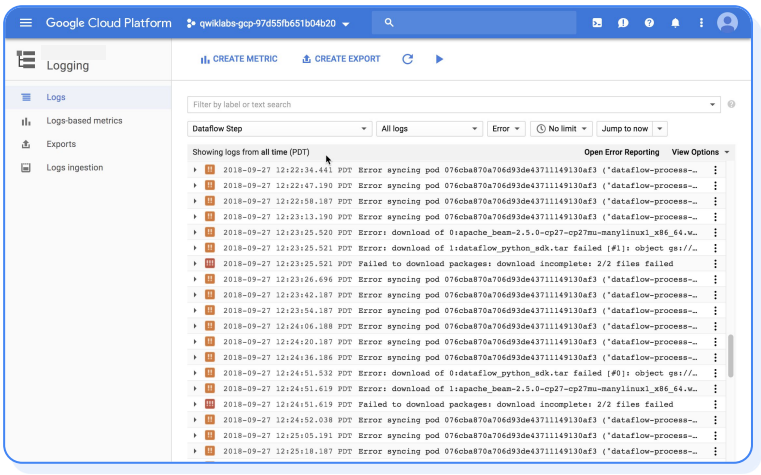



```
errors.HttpError: <HttpError 400 when requesting
https://www.googleapis.com/storage/v1/b/input-bucket-999/o/usa_names.csv/copyTo/b/gs%3A%2F%2Foutput-bucket-999
%2F/o/success%2F2018-09-27%2Fusa_names.csv?alt=json returned "Invalid bucket name: 'gs://output-bucket-999/'">
```

Google Cloud

Here you will find the logs for that specific Airflow run. I search for the word “error” and then start my diagnosis there. Here this was a pretty simple error where it was trying to copy a file from an input bucket to an output bucket and the output bucket didn't exist or was named poorly.

Monitor Dataflow health in Cloud Logging



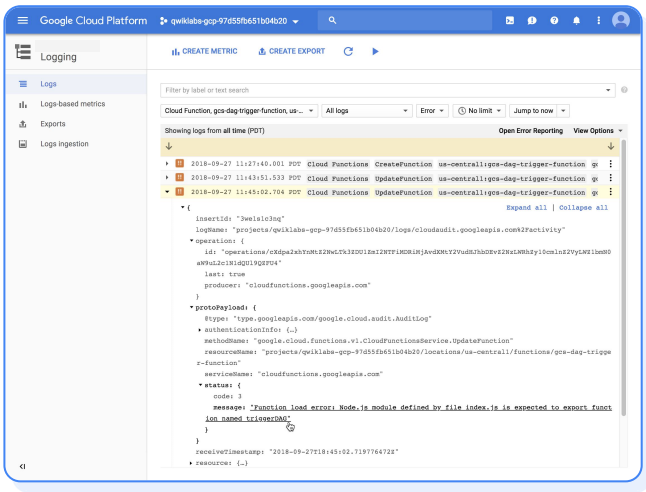
Google Cloud

Another tool in your toolkit for diagnosing Airflow failures is the general Google Cloud logs. Since Airflow launches other Google Cloud services through tasks, you can see and filter for errors for those services in Cloud Logging as you would debugging any other normal application. Here I've filtered for Dataflow step errors to troubleshoot why my workflow is failing.

It turns out that I had not changed the name of the output bucket for the CSV file so after the file was processed by Dataflow as part of step 1, it dumped the completed file back into the input bucket which triggered another Dataflow job for processing and so on.

https://console.cloud.google.com/logs/viewer?project=qwiklabs-gcp-97d55fb651b04b20&minLogLevel=400&expandAll=false&timestamp=2018-10-01T21:03:39.539000000Z&customFacets=&limitCustomFacetWidth=true&interval=NO_LIMIT&resource=cloud_function%2Ffunction_name%2Fgcs-dag-trigger-function%2Fregion%2Fus-central1&scrollTimestamp=2018-09-27T18:45:02.704000000Z

Monitor Cloud Function health in Cloud Logging



Google Cloud

You might be wondering, if there's an error with my Cloud Function my Airflow instance would never have been triggered or issued any logs at all (since it was unaware we were trying to trigger it) and you're exactly right. If you're using Cloud Functions, be sure to check the normal Google Cloud logs for errors and warnings in addition to your Airflow logs.

In this example, each time I uploaded a CSV file to my Cloud Storage bucket hoping to trigger a Cloud Function and then my DAG I got an error expected to find function triggerDAG. Remember way back when I said Cloud Functions were case-sensitive? Looking for a function with capital DAG doesn't exist if it's capital D lowercase a and g so be sure to be mindful when setting up your Cloud Function for the first time.

https://console.cloud.google.com/logs/viewer?project=qwiklabs-gcp-97d55fb651b04b20&minLogLevel=400&expandAll=false&timestamp=2018-10-01T21:03:39.539000000Z&customFacets=&limitCustomFacetWidth=true&interval=NO_LIMIT&resource=cloud_function%2Ffunction_name%2Fgcs-dag-trigger-function%2Fregion%2Fus-central1&scrollTimestamp=2018-09-27T18:45:02.704000000Z

An Introduction to Cloud Composer

Objectives

- Use GCP Console to create a Cloud Composer environment
- View and run a DAG in the Airflow web interface
- View the results of the wordcount job in storage

Module Summary

Cloud Data Fusion for building data pipelines.

Cloud Composer, Cloud Functions and Cloud Scheduler are the glue for your data pipelines.