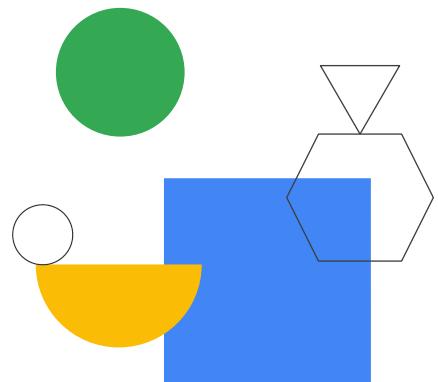


Manage Data Pipelines with Cloud Data Fusion and Cloud Composer



In this module, we will discuss how to manage data pipelines with Cloud Data Fusion and Cloud Composer.



Module agenda



01

Building Batch Data Pipelines Visually with Cloud Data Fusion

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

02

Orchestrating Work Between Google Cloud Services with Cloud Composer

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

Google Cloud

Specifically, we will look at how you can use Cloud Data Fusion to visually build data pipelines, and how you can use Cloud Composer to orchestrate work between Google Cloud services.

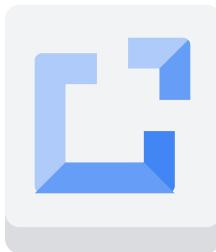


Building Batch Data Pipelines Visually with Cloud Data Fusion

Google Cloud

Let's start with an introduction to Cloud Data Fusion.

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.

Developer, data scientist, and business analyst



1
Need to cleanse, match, de-dupe, blend, transform, partition, transfer, standardize, automate, and monitor.

2
Use Cloud Data Fusion to visually build integration pipeline, test, debug and deploy.

3
Run it at scale on Google Cloud, operationalize (monitor, report) pipelines, inspect rich integration metadata.

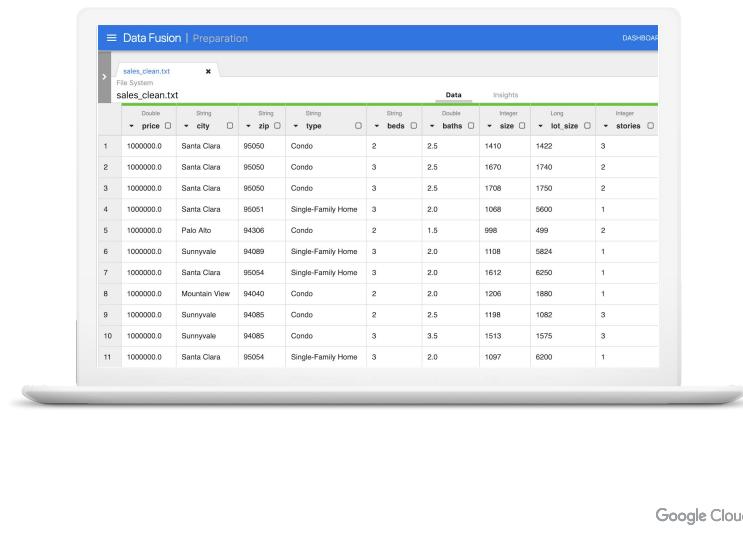
Google Cloud

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



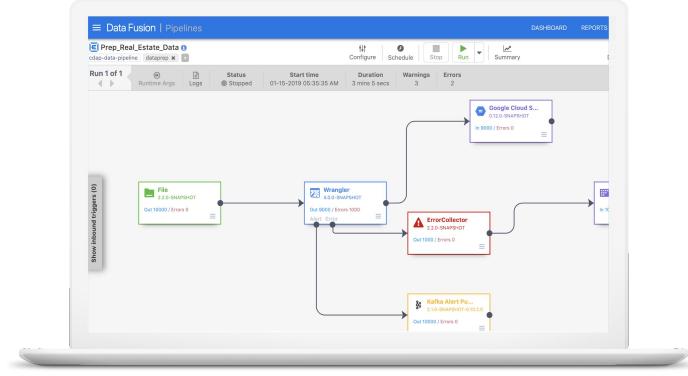
Google Cloud

Cloud Data Fusion offers a number of benefits:

- **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, Cloud 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 the Open source software CDAP.

Build data pipelines with a friendly UI

- Rich graphical interface
- [100+ plugins](#) - connectors, transforms, and 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

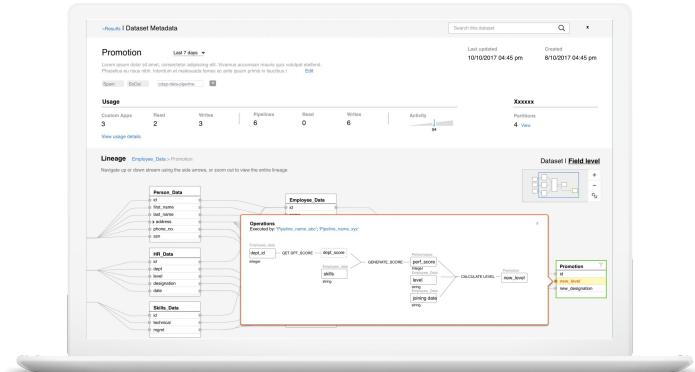
Tags and Properties support

- Pipeline
- Dataset
- Schema

Search integrated entities by

- Keyword
- Schema name and type

Dataset level and Field level lineage



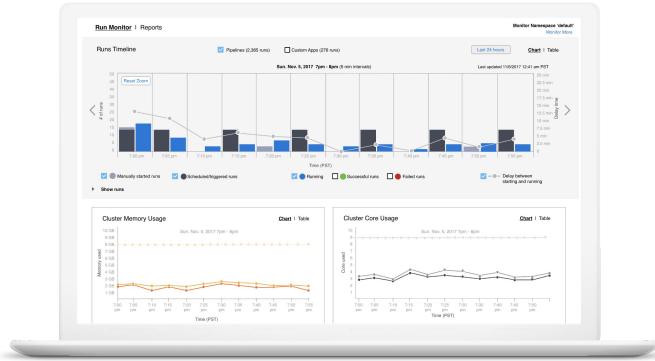
Google Cloud

As you will see in the next lesson, 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'll 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 template pipelines, create conditional triggers, and manage and template plugins. There is a UI widget plug-in as well as custom provisioners, custom compute profiles, and the ability to integrate to hubs.

Components of Cloud Data Fusion

The top screenshot shows the Wrangler interface with a data grid titled 'Starbucks'. The grid contains columns for EmployeeID, Name, Surname, Prefix, Title, Sex, Age, BirthDate, and HireDate. The bottom screenshot shows the Data Pipeline interface with a canvas displaying a complex data workflow involving multiple stages like 'Read from BigQuery', 'Map', 'Join', and 'Write to BigTable'.

01

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.

02

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.

Google Cloud

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

The screenshot shows the Cloud Data Fusion Rules Engine interface. It displays a list of rules under the heading 'Rules (0)'. Each rule entry includes a preview icon, the rule name, and a 'Details' link. The rules listed are:

- 1. **isMissingField**: If a field is missing, send it to an output.
- 2. **isNotPresent**: Create a new row if the field is not present.
- 3. **isPresent**: Insert a new row if the field is present.
- 4. **isPresentAndEmpty**: Insert a new row if the field is present and empty.
- 5. **isPresentOrEmpty**: Insert a new row if the field is present or empty.
- 6. **isPresentOrEmptyAndNull**: Insert a new row if the field is present or empty and null.
- 7. **isPresentOrEmptyOrNull**: Insert a new row if the field is present or empty or null.
- 8. **isPresentOrEmptyOrNullOrEmpty**: Insert a new row if the field is present or empty or null or empty.
- 9. **isPresentOrEmptyOrNullOrEmptyOrNull**: Insert a new row if the field is present or empty or null or empty or null.
- 10. **isPresentOrEmptyOrNullOrEmptyOrNullOrEmpty**: Insert a new row if the field is present or empty or null or empty or null or empty.
- 11. **isPresentOrEmptyOrNullOrEmptyOrNullOrEmptyOrEmpty**: Insert a new row if the field is present or empty or null or empty or null or empty or empty.

At the bottom of the interface, there is a note: 'Copyright © 2017 Google Inc.'

03

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.

04

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.

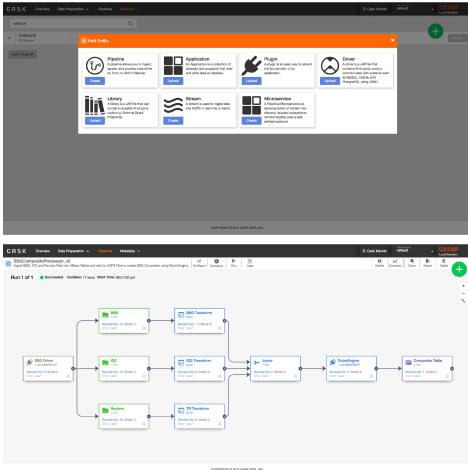
Google Cloud

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



05

Microservice — Framework

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

06

Event Condition Action (ECA) — Application

- Delivers a specialized solution 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.

Google Cloud

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.

Cloud Data Fusion - UI overview

- Control Center
- Pipelines
- Wrangler
- Metadata
- Hub
- Entities
- Administration

Type	Status	Last start time	Next run in	Total runs	Tags
Batch	Failed	10-18-2018 05:01:35 PM	58 sec	2	Wrangler, Real_Estate, cdap-data-pipeline
Batch	Running	10-18-2018 05:01:35 PM	1 hr	10	Wrangler, Real_Estate, cdap-data-pipeline
Batch	Succeeded	10-18-2018 05:01:35 PM	1 day	13	cdap, Real_Estate, SoCal, cdap-data-pipeline
Realtime	Deployed	cdap-data-pipeline
Realtime	Running	10-18-2018 05:01:35 PM	...	234	Wrangler, Real_Estate, news, Spam, Social, cdap-data-pipeline
Batch	Failed	10-18-2018 05:01:35 PM	...	35	cdap-data-pipeline
Batch	Succeeded	10-18-2018 05:01:35 PM	1 hr	5,678	cdap-data-pipeline
Realtime	Succeeded	10-18-2018 05:01:35 PM	1 month	345	cdap-data-pipeline
Batch	Succeeded	10-18-2018 05:01:35 PM	...	1	cdap-data-pipeline
Batch	Succeeded	10-18-2018 05:01:35 PM	...	24	cdap-data-pipeline

Google Cloud

Managing your pipelines is easiest when you have the right tools. We'll now take a high-level look at the Cloud Data Fusion UI as you saw in the component overview.

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

Control Center

- Application
- Artifact
- Dataset

The screenshot shows the Cloud Data Fusion Control Center interface. At the top, there's a search bar, filter dropdown, and sort dropdown. Below that, a header reads "Entities in namespace 'default'" with a count of "26 entities". There are two pages shown, with page 1 selected.

The main area displays a grid of 26 entities, each represented by a card:

- Data Pipeline:** join-w-customer (1.0.0)

	Programs	Running	Failed
2	0	0	0
- Dataset:** sales-with-zip

	Programs	Operations	Writes
1	10	3	0
- Dataset:** sales.csv

	Programs	Operations	Writes
1	0	0	0
- Data Pipeline:** sales-ingest (1.0.0)

	Programs	Running	Failed
2	0	0	0
- Dataset:** sales

	Programs	Operations	Writes
2	15	10	0
- Dataset:** customers-ingest

	Programs	Running	Failed
2	0	0	0
- Dataset:** customers

	Programs	Operations	Writes
1	12	9	0
- Dataset:** U.S._Chronic_Disease_Indicator...

	Programs	Operations	Writes
1	0	0	0
- Data Pipeline:** Mask_data (1.0.0)

	Programs	Running	Failed
2	0	0	0
- Dataset:** Mask_data

	Programs	Operations	Writes
1	42,119,097	42,119,097	0
- Dataset:** experiment_splits

	Programs	Operations	Writes
1	0	0	0
- Data Pipeline:** Titanic_test_pipeline_v1_test_v1 (1.0.0)

	Programs	Running	Failed
2	0	0	0
- Dataset:** experiment_model_meta

	Programs	Operations	Writes
1	0	0	0
- Dataset:** experiment_model_components

	Programs	Operations	Writes
1	0	0	0
- Dataset:** experiment_meta

	Programs	Operations	Writes
1	1	0	0
- Data Pipeline:** Titanic.csv (1.0.0)

	Programs	Operations	Writes
0	0	0	0
- Dataset:** Titanic.csv

	Programs	Operations	Writes
0	0	0	0
- Data Pipeline:** Titanic_test_pipeline_v1_test (1.0.0)

	Programs	Running	Failed
2	0	0	0
- Dataset:** Titanic_test

	Programs	Operations	Writes
0	0	0	0
- Dataset:** connections

	Programs	Operations	Writes
1	0	0	0
- Dataset:** dataprepfs

	Programs	Operations	Writes
1	0	0	0
- Dataset:** workspace

	Programs	Operations	Writes
1	2,499	239	0
- Dataset:** Error_sink_titanic

	Programs	Operations	Writes
0	0	0	0

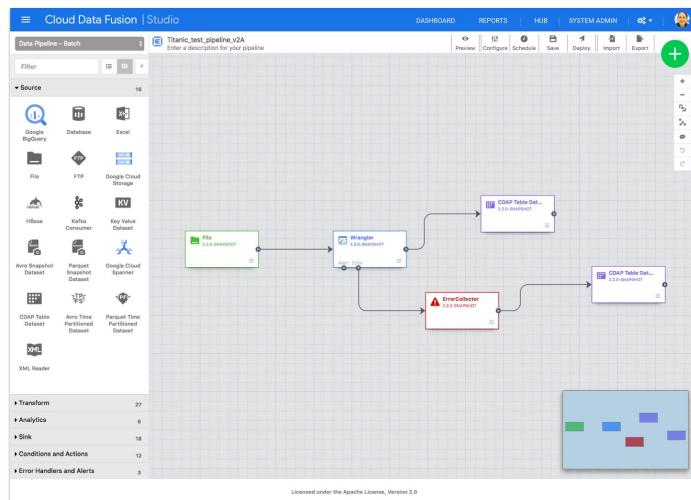
Google Cloud

Under Control Center, there's 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

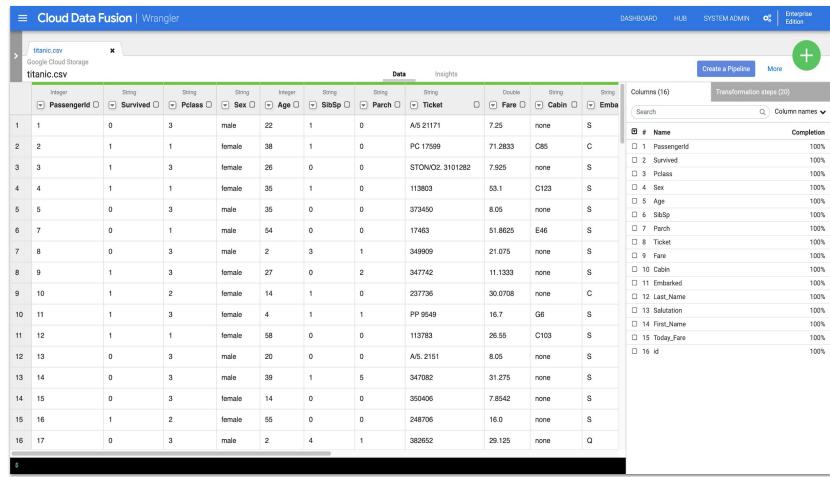


Google Cloud

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



The screenshot shows the Cloud Data Fusion Wrangler interface. At the top, there's a navigation bar with 'DASHBOARD', 'HUB', 'SYSTEM ADMIN', and 'Enterprise Edition'. Below the navigation is a search bar and a 'Create a Pipeline' button. A sidebar on the right lists 'Transformation steps (20)' with completion percentages for each step. The main area displays a table of data from 'titanic.csv' in Google Cloud Storage. The table has 16 columns: PassengerId, Survived, Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked, and Id. The data consists of 16 rows, each representing a passenger. The last row is highlighted in yellow.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Id
1	1	0	3	male	22	1	0	0	A/5 21171	7.25	none	S	
2	2	1	1	female	38	1	0	0	PC 17599	71.233	C85	C	
3	3	1	3	female	26	0	0	0	STON/O2. 3101282	7.925	none	S	
4	4	1	1	female	35	1	0	0	13803	53.1	C123	S	
5	5	0	3	male	35	0	0	0	373450	8.05	none	S	
6	7	0	1	male	54	0	0	0	17463	51.8625	E46	S	
7	8	0	3	male	2	3	1	1	349909	21.075	none	S	
8	9	1	3	female	27	0	2	0	347742	11.1333	none	S	
9	10	1	2	female	14	1	0	0	237736	30.0708	none	C	
10	11	1	3	female	4	1	1	1	PP 9649	16.7	G6	S	
11	12	1	1	female	58	0	0	0	113783	26.55	C103	S	
12	13	0	3	male	20	0	0	0	A/5. 2151	8.05	none	S	
13	14	0	3	male	39	1	5	5	347082	31.275	none	S	
14	15	0	3	female	14	0	0	0	350406	7.8542	none	S	
15	16	1	2	female	55	0	0	0	248706	16.0	none	S	
16	17	0	3	male	2	4	1	1	382652	29.125	none	Q	

Google Cloud

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

Integration metadata

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

The screenshot shows the Cloud Data Fusion Metadata interface. On the left, there's a sidebar with a 'Control Center' section containing 'Pipelines', 'List', 'Studio', 'Transform', and 'Metadata'. Below that is a 'Namespace Admin' section. The main area is titled 'Cloud Data Fusion | Metadata' and shows a list of datasets under the 'default' namespace. The list includes:

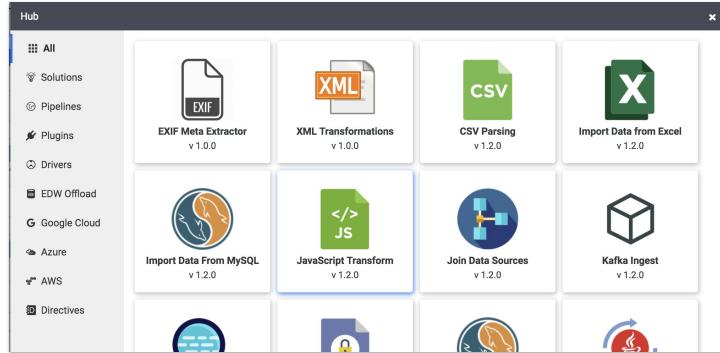
- recipes**: 18 datasets, created on Oct 12, 2018. Description: Recipe store.
- workspace**: 4 datasets, created on Oct 12, 2018. Description: Dataprep workspace dataset.
- dataprepfs**: 1 dataset, created on Oct 12, 2018. Description: Store Dataprep Index files.
- connections**: 1 dataset, created on Oct 12, 2018. Description: Dataprep connections store.
- Error_sink_titanic**: 1 dataset, created on Oct 17, 2018. Description: No description provided for this Dataset.
- Titanic_test**: 1 dataset, created on Oct 17, 2018. Description: No description provided for this Dataset.
- titanic.csv**: 1 dataset, created on Oct 19, 2018. Description: No description provided for this Dataset.

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

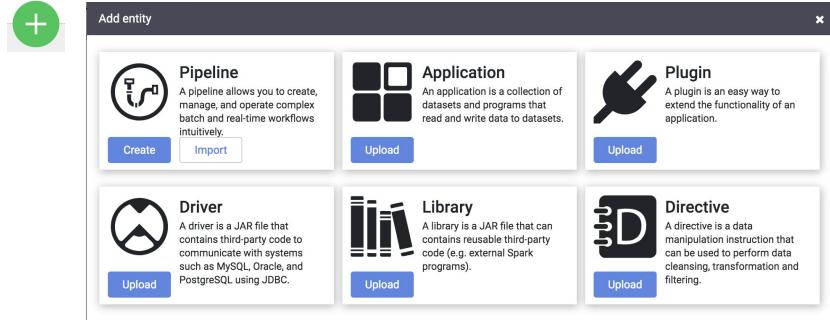


Google Cloud

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

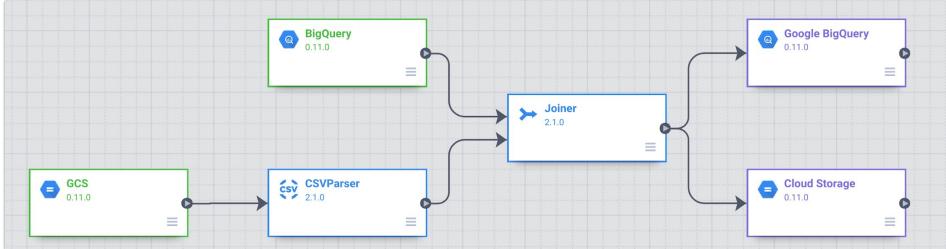
- Configuration

- 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.

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

Now that we've looked at the components in the UI, we'll discuss the process of building a data pipeline.

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, nodes can be of a different type. You may start with a node that pulls data from 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.

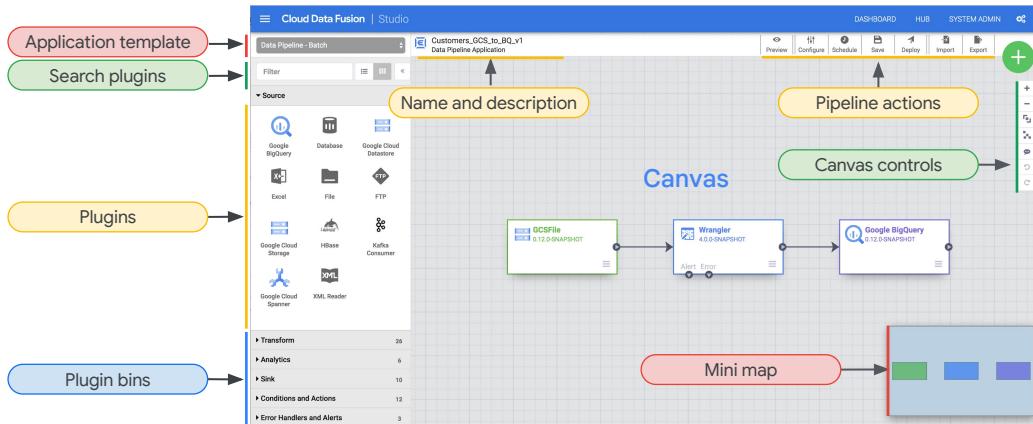
Data pipeline

- 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.

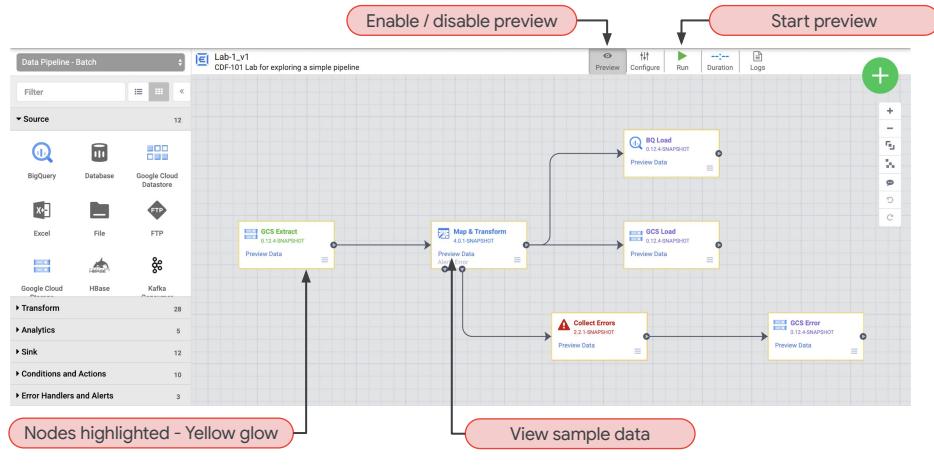
You can interact with the canvas and add objects by using the **Canvas Control Panel**.

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 pre-existing **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.

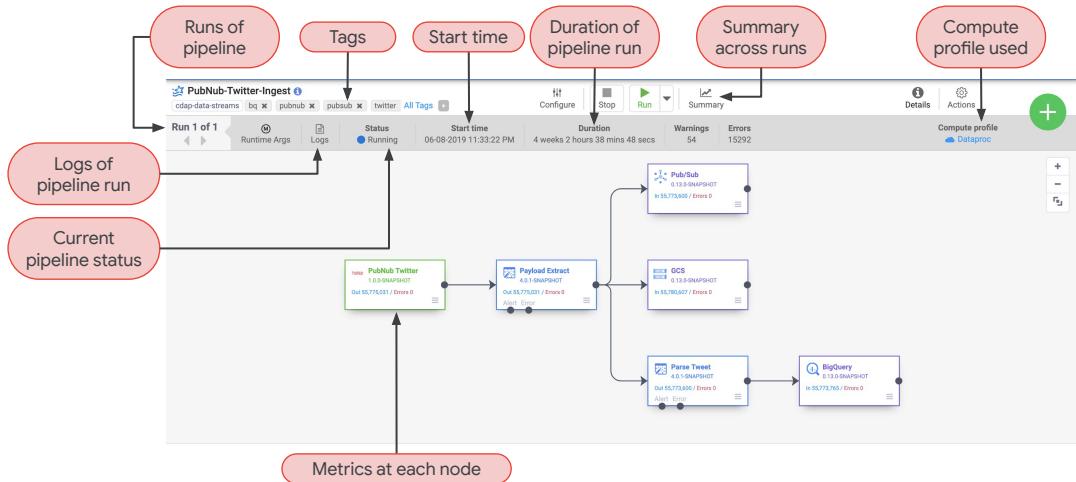
Use Preview Mode to see how the pipeline will run



Google Cloud

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.

Monitor the health of the entire pipeline



Google Cloud

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 datasinks.

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.

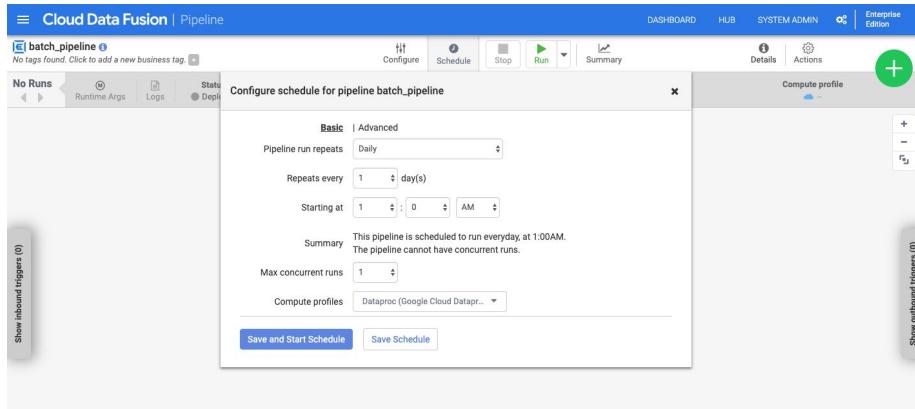
Monitor the health of a single node



Google Cloud

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

The screenshot shows the Cloud Data Fusion Field Level Lineage interface. At the top, there's a navigation bar with 'DASHBOARD', 'HUB', 'SYSTEM ADMIN', and 'Enterprise Edition'. Below the navigation is a breadcrumb trail: 'Cloud Data Fusion | Field Level Lineage > doubleclick > Dataset'. The main area is titled 'Field level lineage' with the sub-section 'Cause for: doubleclick: campaign'. It shows a table with one dataset row containing three fields: 'campaign', 'offset', and 'body'. To the right, under 'Impact for: doubleclick: campaign', it shows a table with one dataset row containing two fields: 'hits' and 'campaign'. A central section is labeled 'Dataset: doubleclick' and lists '8 Fields'. Below this are two lists of operations: 'Incoming operations' (with 'campaign' highlighted) and 'Outgoing operations' (listing 'referrer_url', 'campaign_id', 'landing_page_url', 'advertiser', and 'landing_page_url_id'). A search bar 'Search by field name' is also present.

Google Cloud

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

The screenshot shows the 'Field level lineage' section of the Cloud Data Fusion interface. It displays a table of operations between the 'campaign' field and the 'doubleclick' dataset. The table has columns for Input, Input fields, Operation, Description, Output fields, and Output.

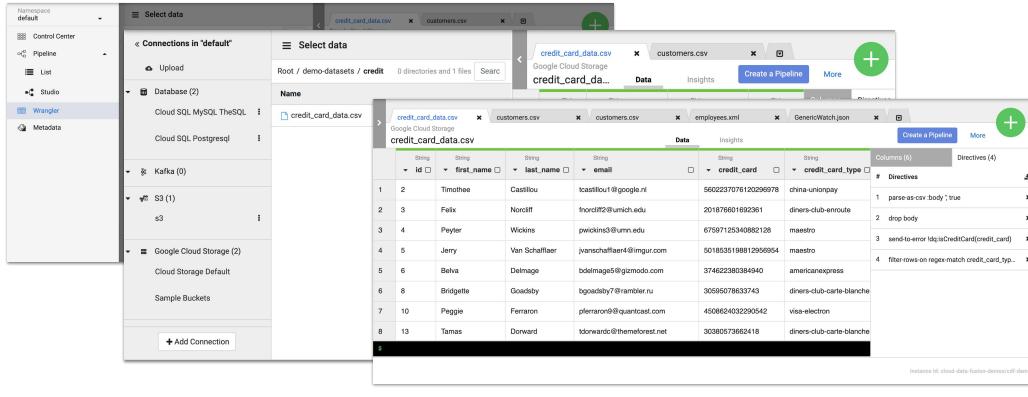
Input	Input fields	Operation	Description	Output fields	Output
1 campaign	-	campaign.Re...	Read from Google Cloud Storage.	offset, body	-
2 -	[offset], [body]	parse	campaigns.P... Data	advertiser_id, campaign_id, campaign	-
3 -	[campaign]	Joiner3.Iden... parse	parse-as-csv 'body', true; drop 'body'; campaigns.c...	campaign	-
			Unchanged as part of a join		

Google Cloud

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.

Data analysts can explore datasets in the Wrangler

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



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

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.

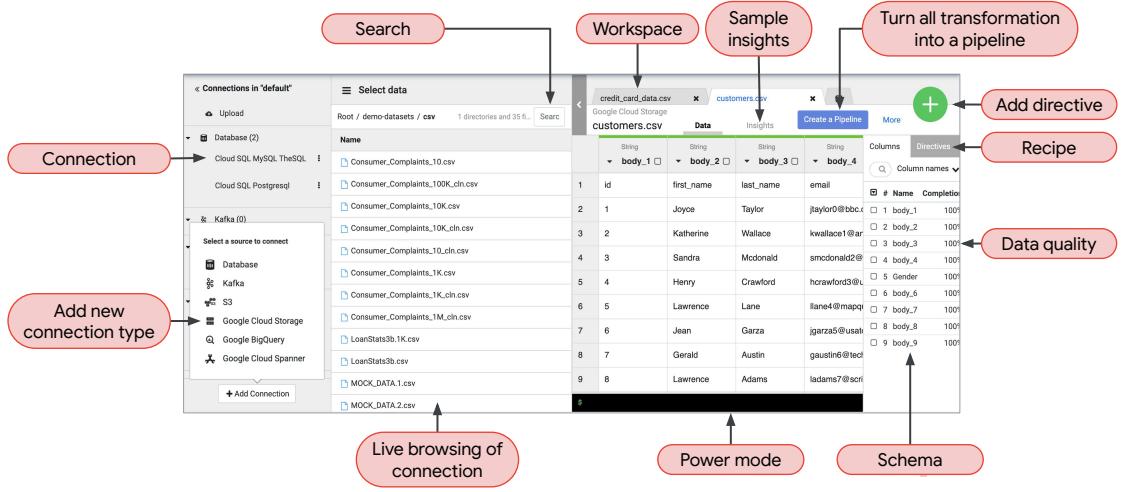
Wrangler UI overview for exploring datasets

The screenshot shows the Google Cloud Wrangler UI interface. On the left, there's a sidebar titled "Connections in 'default'" with sections for "Upload", "Database (2)", "Kafka (0)", "Select a source to connect" (with options for Database, Kafka, S3, Google Cloud Storage, Google BigQuery, and Google Cloud Spanner), and a "+ Add Connection" button. The main area is titled "Select data" and shows a file tree under "Root / demo-datasets / csv". The "customers.csv" file is selected, and its contents are displayed in a table view. The table has columns: id, first_name, last_name, email, body_1, body_2, body_3, body_4, gender, body_5, body_6, body_7, body_8, body_9, and completion. The table contains 9 rows of data. At the top right of the main area, there are tabs for "Data" (which is selected) and "Insights", along with buttons for "Create a Pipeline" and "More". Below the table, there are buttons for "Columns" and "Directives", and a "Column names" dropdown.

Google Cloud

Here's what the Wrangler UI looks like.

Wrangler UI overview for exploring datasets



Starting from the left, you have your **connections** to existing datasets.

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.

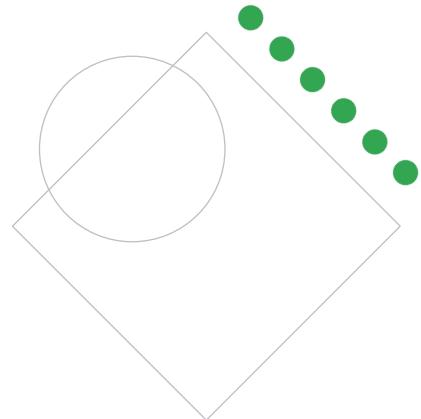
Here, you see a Cloud Storage bucket of demo datasets and all the CSV files of customer complaints. Once you've found an example dataset like `customers.csv` here, you can **explore** the rows and columns visually and view sample insights.

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.

Lab Intro

Building and Executing a Pipeline
Graph in Cloud Data Fusion



Google Cloud

Now it's time for you to practice building and executing a pipeline graph in Cloud Data Fusion.

Lab objectives

- 01 Connect Cloud Data Fusion to a couple of data sources
- 02 Apply basic transformations
- 03 Join two data sources
- 04 Write data to a sink



Google Cloud

In this lab, you will connect Cloud Data Fusion to a couple of data sources, apply basic transformations, join two data sources, and write data to a sink.

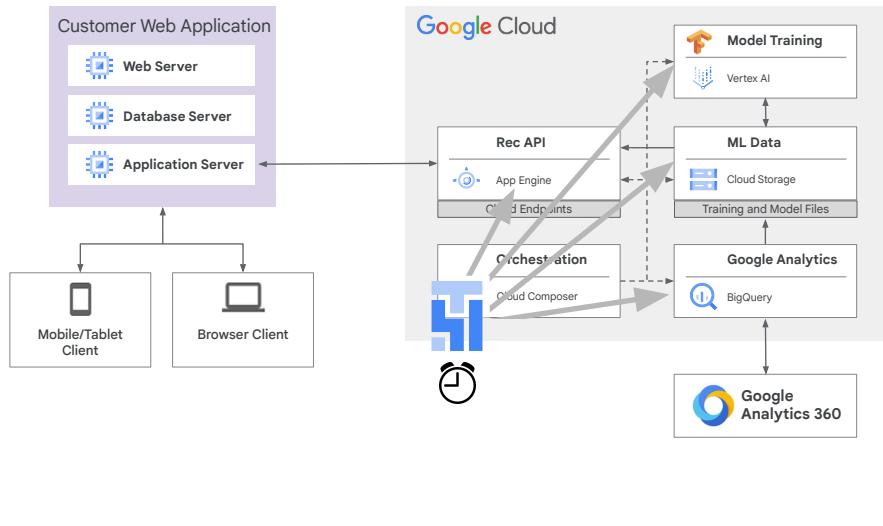


Orchestrating Work Between Google Cloud Services with Cloud Composer

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 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 tasks like that.

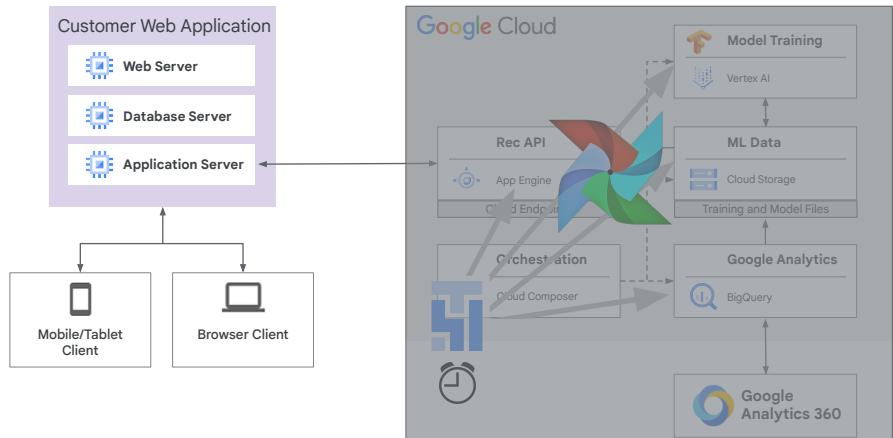
Cloud Composer orchestrates automatic workflows



Google Cloud

Cloud Composer will control 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.

Use Apache Airflow DAGs to orchestrate Google Cloud services



```
bq_rec_training_data → bq_export_op → ml_engine_training_op → app_engine_deploy_version
```

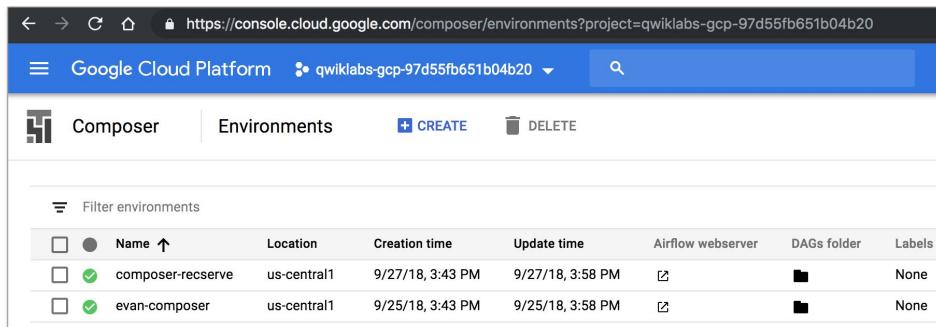
DAG = Directed Acyclic Graph



Google Cloud

The heart of any workflow is a 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 Vertex AI, but yours could orchestrate among four completely different services.

Cloud Composer creates managed Apache Airflow environments



The screenshot shows the Google Cloud Platform interface for managing Composer environments. The URL is https://console.cloud.google.com/composer/environments?project=qwiklabs-gcp-97d55fb651b04b20. The page title is "Google Cloud Platform" and the project is "qwiklabs-gcp-97d55fb651b04b20". The main section is titled "Environments" and includes "CREATE" and "DELETE" buttons. A "Filter environments" dropdown is open, showing two entries:

Name	Location	Creation time	Update time	Airflow webserver	DAGs folder	Labels
composer-reserve	us-central1	9/27/18, 3:43 PM	9/27/18, 3:58 PM	[edit]	[edit]	None
evan-composer	us-central1	9/25/18, 3:43 PM	9/25/18, 3:58 PM	[edit]	[edit]	None

Google Cloud

Let's preview the actual Cloud Composer environment.

Once you use the command line or Google Cloud 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 with 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 Google Cloud project account. Normally, you will not do that at the Cloud Composer level but on the actual Apache Airflow instance level. Again, generally you're only on the Cloud Composer page here to create new environments before you launch directly into the Airflow web server.

Each Airflow environment has a separate web server and folder in Cloud Storage for pipeline DAGs

	Location	Creation time	Update time	Airflow webserver	DAGs folder	Labels
<input type="checkbox"/>	us-central1	9/27/18, 3:43 PM	9/27/18, 3:58 PM			None
<input checked="" type="checkbox"/>	us-central1	9/25/18, 3:43 PM	9/25/18, 3:58 PM			None

Google Cloud

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 Cloud Storage bucket where you will load your pipeline code

Buckets / us-central1-evan-composer-0e85530c-bucket / dags						
Name	Size	Type	Storage class	Last modified	Public access	Encryption
dataflow/	—	Folder	—	—	Per object	—
simple_load_dag.py	6.79 KB	text/x-python-script	Multi-Regional	10/1/18, 1:11 PM	Not public	Google-managed key
simple.py	2.51 KB	text/x-python-script	Multi-Regional	10/1/18, 1:10 PM	Not public	Google-managed key

Google Cloud

The DAGs folder for each Airflow instance is simply a Cloud Storage bucket that is automatically created for you when you create your Cloud Composer instance. Here is where you upload your DAG files, written in Python, and bring your first workflow to life in Airflow.

Airflow workflows are written in Python

The screenshot shows a Google Cloud Storage interface with a list of files in a folder named 'dags'. The files listed are 'simple_load_dag.py' (6.79 KB) and 'simple.py' (2.51 KB). A green box highlights 'simple_load_dag.py'. An arrow points from this file to a block of Python code.

```

106 # e.g. state,person_year,name,number,created_date
107 with models.DAG(dag_id='gcsToBigQueryTriggered',
108     description='A DAG triggered by an external Cloud Function',
109     schedule_interval=None, default_args=DEFAULT_DAG_ARGS) as dag:
110     # Args required for the Dataflow job.
111     job_args = {
112         'input': 'gs://{{ dag_run.conf["bucket"] }}/{{ dag_run.conf["name"] }}',
113         'output': models.Variable.get('bc_output_table'),
114         'fields': models.Variable.get('input_field_names'),
115         'load_dt': DS_TAG
116     }
117
118     # Main Dataflow task that will process and load the input delimited file.
119     dataflow_task = DataFlowPythonOperator(
120         task_id='process-delimited-and-push',
121         py_file=DATAFLOW_FILE,
122         options=job_args)
123
124     # Here we create two conditional tasks, one of which will be executed
125     # based on whether the dataflow_task was a success or a failure.
126     success_move_task = python_operator.PythonOperator(task_id='success-move-to-completion',
127             python_callable=move_to_completion_bucket,
128             # A success_tag is used to move
129             # the input file to a success
130             # prefixed folder.
131             op_args=[COMPLETION_BUCKET, SUCCESS_TAG],
132             provide_context=True,
133             trigger_rule=TriggerRule.ALL_SUCCESS)
134
135     failure_move_task = python_operator.PythonOperator(task_id='failure-move-to-completion',
136             python_callable=move_to_completion_bucket,
137             # A failure_tag is used to move
138             # the input file to a failure
139             # prefixed folder.
140             op_args=[COMPLETION_BUCKET, FAILURE_TAG],
141             provide_context=True,
142             trigger_rule=TriggerRule.ALL_FAILED)
143
144     # The success_move_task and failure_move_task are both downstream from the
145     # dataflow_task.
146     dataflow_task >> success_move_task
147     dataflow_task >> failure_move_task

```

Google Cloud

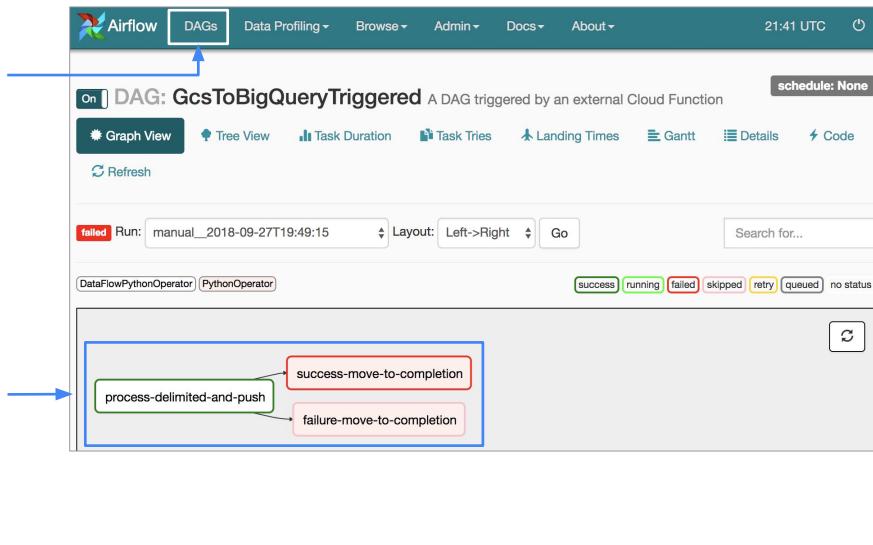
Now that you're familiar with the basic environment setup, it's time to discuss your primary artifact, which is your DAG, and the operators you are using to call whichever services you want to send tasks to.

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 Cloud Storage 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.

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 its components a little later.

The Python file creates a DAG



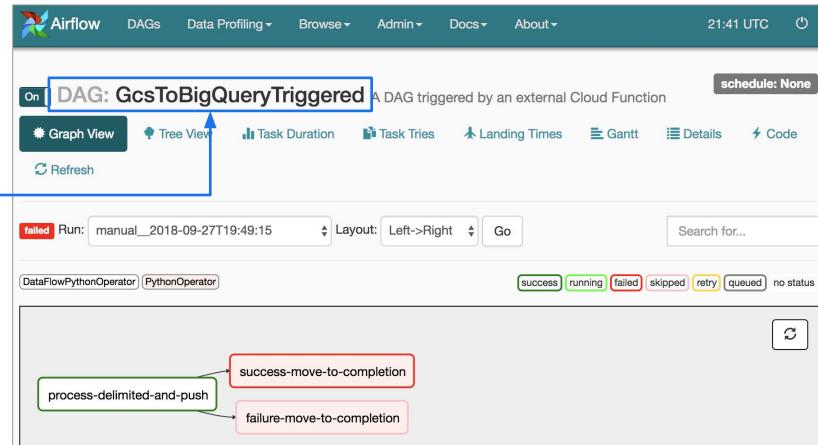
Google Cloud

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, with 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.

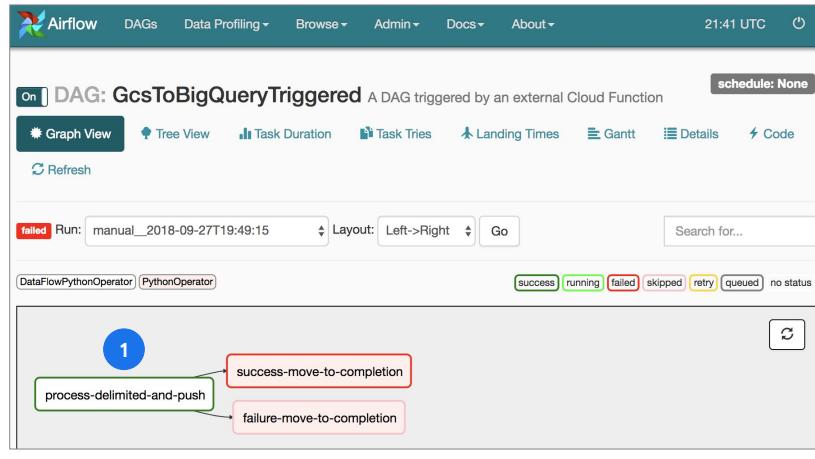
Let's explore a bit more of the Airflow web UI.

Airflow web server UI overview



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

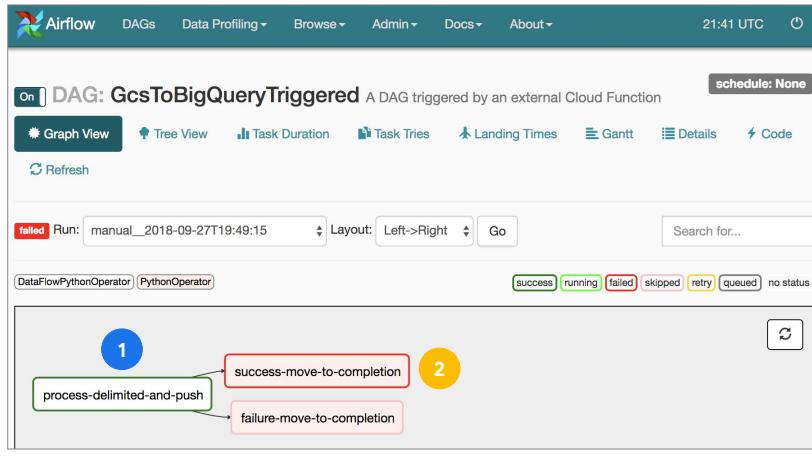
The Python file creates a DAG



Google Cloud

One, process-delimited-and-push. 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 Cloud Storage bucket, processes it, and writes the output to BigQuery.

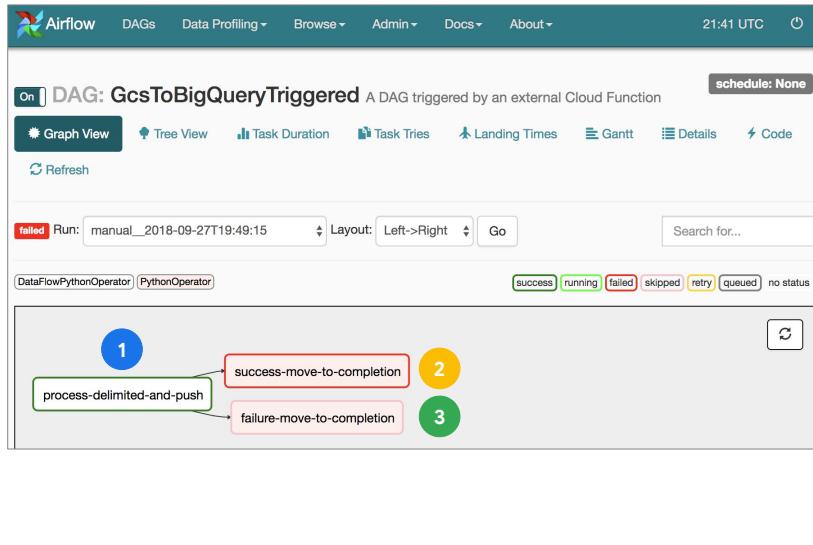
The Python file creates a DAG



Google Cloud

Two, success-move-to-completion, which moves the CSV file from an input Cloud Storage bucket to a processed or completed Cloud Storage bucket for archiving, or

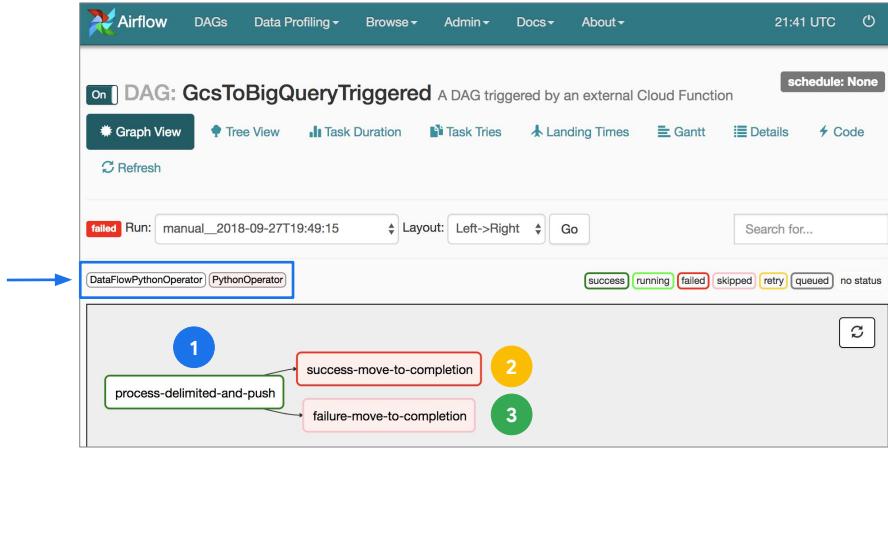
The Python file creates a DAG



Google Cloud

Three, 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 is made to run one node or a different one based on the outcome of the parent node.

The Python file creates a DAG

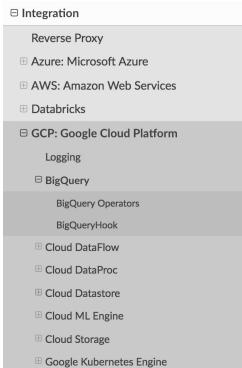


Google Cloud

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 Google Cloud services



View all Google Cloud services that Airflow can orchestrate:

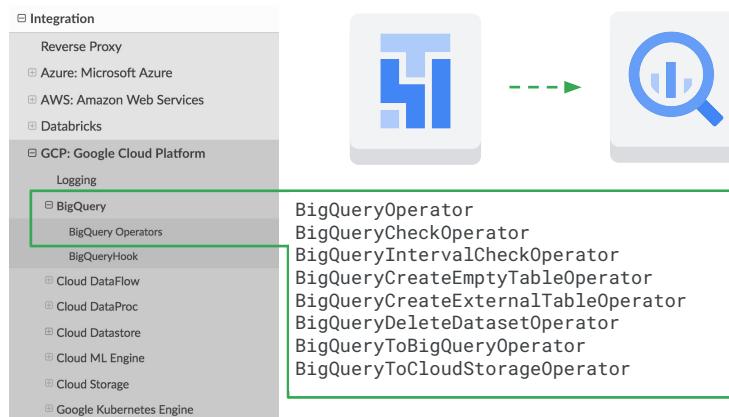
<http://airflow.apache.org/integration.html#gcp>

+ sample code

Google Cloud

Airflow has many operators which you can use to invoke the TASKS you want to complete. Operators are usually atomic in a task, which means generally you only see one operator per task. This list of all services that Airflow can orchestrate to is taken directly from the Apache Airflow documentation. Let's take a look at the ones that are likely most relevant to us as data engineers.

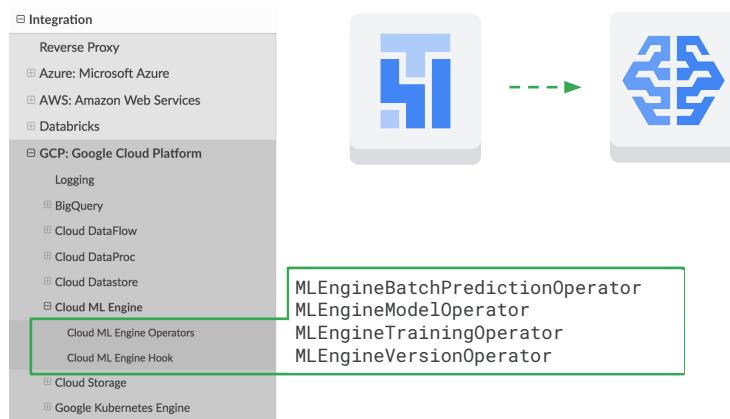
BigQuery Operators are popular for updating your ML training dataset



Google Cloud

As you might have guessed, we'll certainly be making use of the BigQuery operators since our workflows depend on the data that is fed into them through Cloud Storage 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.

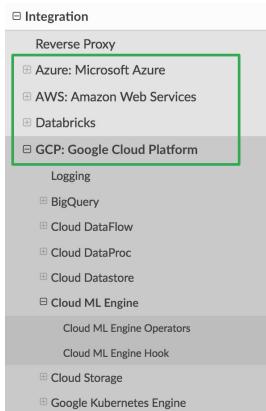
Vertex AI (formerly Cloud ML Engine) operators can launch training and deployment jobs



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 Vertex AI operator to kick off a new training job and manage our model like incrementing the version.

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



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

Example: Common Machine Learning workflow DAG Operators

```
bq_res_training_data → bq_export_op → ml_engine_training_op → app_engine_deploy_version
```

```
# update training data 1
t1 = BigQueryOperator(
    )

# BigQuery training data export to GCS 2
t2 = BigQueryToCloudStorageOperator(
    )

# AI Platform training job 3
t3 = MLEngineTrainingOperator(
    )

# App Engine deploy new version 4
t4 = AppEngineVersionOperator(
    )

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

Google Cloud

Here you see four tasks t1, t2, t3, and t4, and four operators corresponding to four Google Cloud 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 1
t1 = BigQueryOperator(
    )

# BigQuery training data export to GCS 2
t2 = BigQueryToCloudStorageOperator(
    )

# AI Platform training job 3
t3 = MLEngineTrainingOperator(
    )

# App Engine deploy new version 4
t4 = AppEngineVersionOperator(
    )

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

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.

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',
    bqj="""
    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 parsing 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',
    bqj="""
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

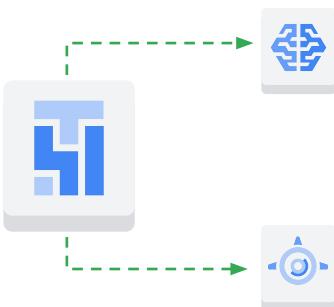
```
from airflow.contrib.operators import bigquery_operator
# constants or can be dynamic based on Airflow macros
max_query_date = {{ macros.ds_add(ds, -1) }} # one day prior
min_query_date = {{ macros.ds_add(ds, -7) }} # seven days prior

# 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 is really powerful is that 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 underscore add ds dash 7, which is a week before the DAG scheduled run date.

Vertex AI and App Engine operators retrain and redeploy our model



```
# update training data 1  
t1 = BigQueryOperator()  
  
# BigQuery training data export to GCS 2  
t2 = BigQueryToCloudStorageOperator()  
  
# AI Platform training job 3  
t3 = MLEngineTrainingOperator()  
  
# App Engine deploy new version 4  
t4 = AppEngineVersionOperator()  
  
# DAG dependencies  
t2.set_upstream(t1)  
t3.set_upstream(t2)  
t4.set_upstream(t3)
```

Google Cloud

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

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)
```

Google Cloud

At the end of almost all DAG files you'll find 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.

Two scheduling options for Cloud Composer workflows



Periodic



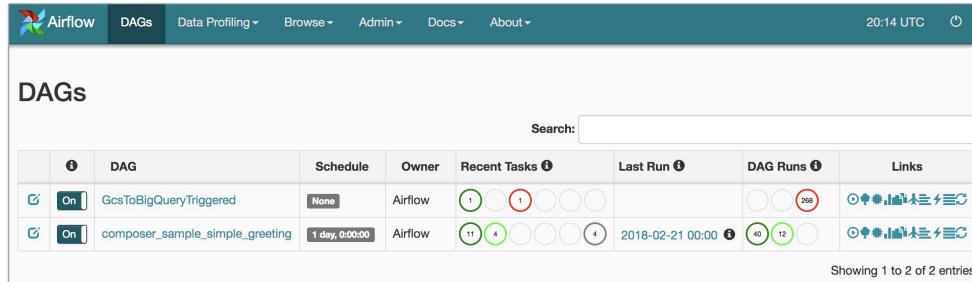
Event-driven

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 Google Cloud services in your DAG, it is time to discuss a really important topic: workflow scheduling.

As we hinted at earlier, there are two different ways your workflow can be run 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 Cloud Storage bucket or if new data came in from a Pub/Sub topic you've subscribed to.

Airflow scheduling basics



The screenshot shows the Airflow web interface with the 'DAGs' tab selected. At the top, there are navigation links: Airflow, DAGs, Data Profiling, Browse, Admin, Docs, About, and a power icon. The time is displayed as 20:14 UTC. Below the header is a search bar labeled 'Search:'.

The main content area is titled 'DAGs'. It displays a table with the following data:

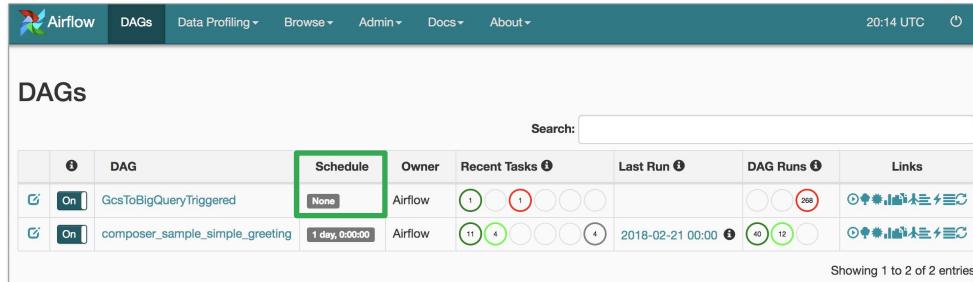
	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	GcsToBigQueryTriggered	None	Airflow	1 2 3 4 5 6 7 8 9 10		200	
	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 5 6 7 8 9 10	2018-02-21 00:00	40 12	

At the bottom of the table, it says 'Showing 1 to 2 of 2 entries'.

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?



DAGs

Search:

	DAG	Schedule	Owner	Recent Tasks ⓘ	Last Run ⓘ	DAG Runs ⓘ	Links
<input checked="" type="checkbox"/>	GcsToBigQueryTriggered	None	Airflow	1 1 1 1 1 1		1 1 200	0 * 1 1 1 1
<input checked="" type="checkbox"/>	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 1 1 1 4	2018-02-21 00:00 ⓘ	40 12 1	0 * 1 1 1 1

Showing 1 to 2 of 2 entries

Google Cloud

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

Option 1: Event-driven scheduling with Cloud Functions

The screenshot shows the Airflow web interface with the 'DAGs' tab selected. A green arrow points from the 'DAGs' section to a 'Cloud Functions' icon. The 'Cloud Functions' icon is a blue server-like icon with three dots above it. Below the icon, there is a search bar labeled 'Search:'.

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	GcsToBigQueryTriggered	None	Airflow	1 0 1 0 0		0 0 200	
	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 0 0 4	2018-02-21 00:00	1 40 12 0	

Showing 1 to 2 of 2 entries

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 Cloud Storage bucket for new CSV files.

Option 2: Specify pipeline schedule_interval in your DAG

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	GcsToBigQueryTriggered	None	Airflow	1 2 3 4 5 6 7 8 9 10		1 2 3 4 5 6 7 8 9 10 200	
	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 12 13 14 15 16 17 18 19 20	2018-02-21 00:00:00	40 12 13 14 15 16 17 18 19 20	

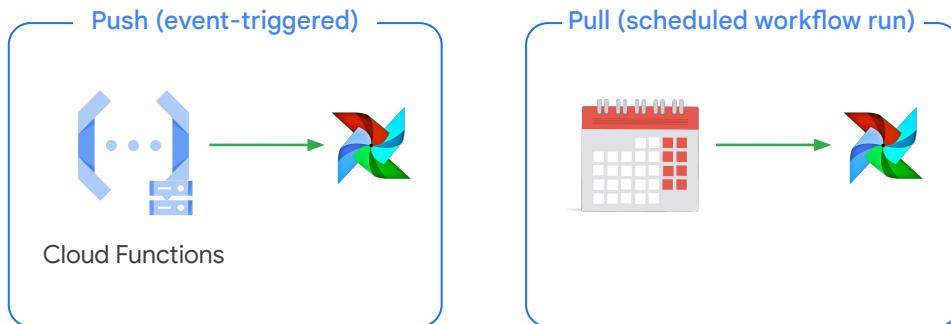
Showing 1 to 2 of 2 entries

```
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 what 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

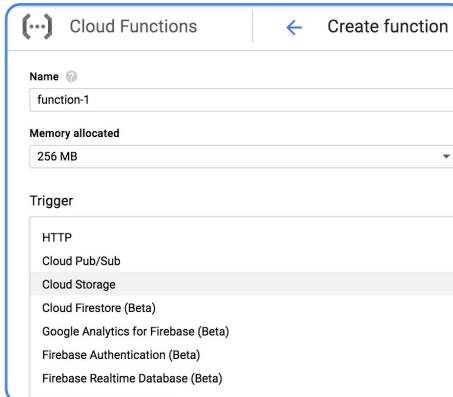


Google Cloud

As you saw earlier in this course, there are two general patterns for ETL workflows:

- **Event triggered or Push** - as in you push a new file to Cloud Storage and your workflow kicks off.
- Or **Pull**, which is where Airflow at a set time could look in your 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



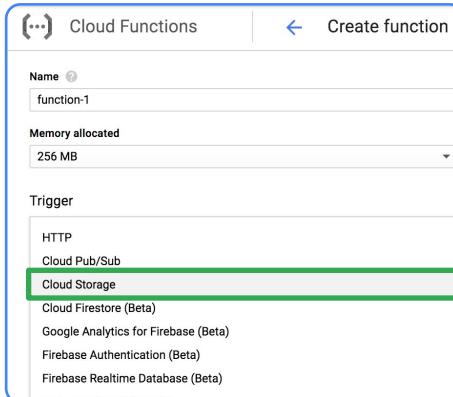
Google Cloud

We can use Cloud Functions to create our event-driven or push architecture workflow. I mentioned triggering on events within a Cloud Storage 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 doesn't 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.

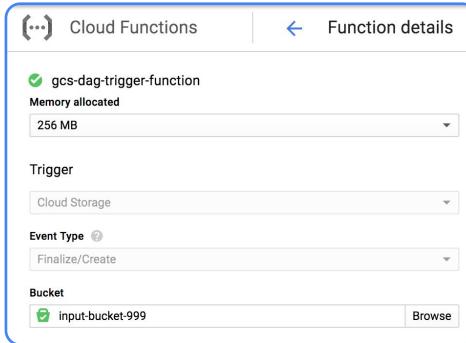
Example: Trigger an event when new data is loaded to Cloud Storage



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.

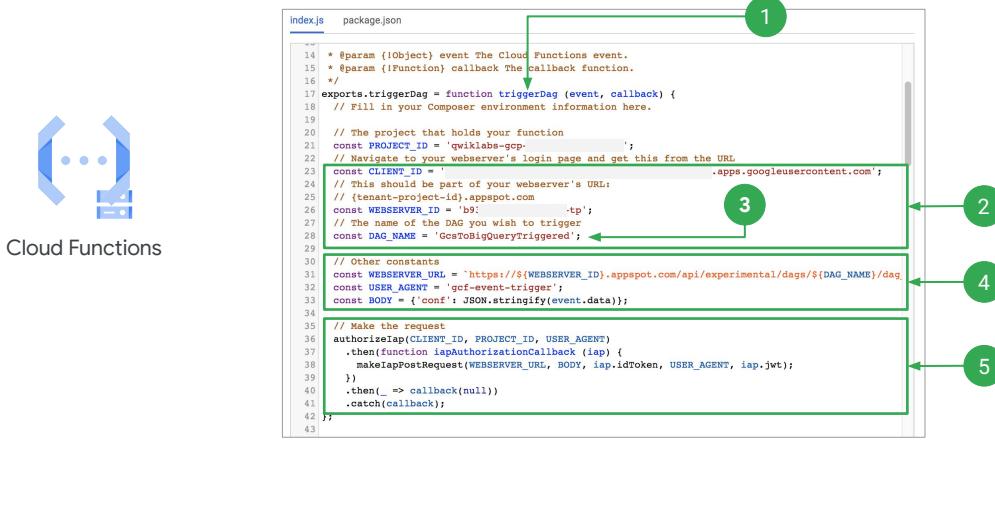
Creating a Cloud Function to trigger an Airflow DAG



Google Cloud

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

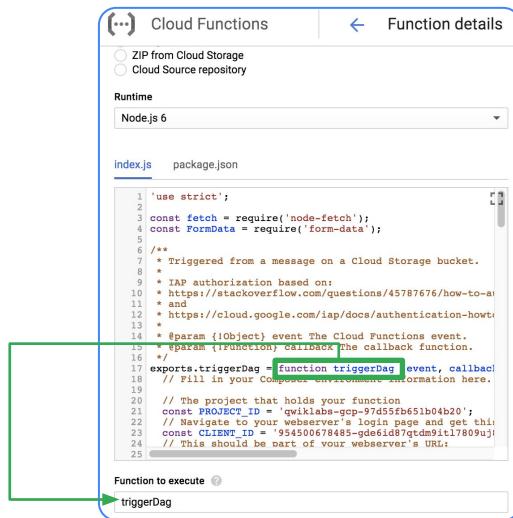
Creating a Cloud Function Cloud Storage trigger an Airflow DAG



As part of the Cloud Function, we need to create an actual function in 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.

1. Here we specify a name for our function called triggerDag.
2. 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.
3. 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!
4. 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.
5. 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



The screenshot shows the 'Function details' page for a Cloud Function. The 'Runtime' is set to Node.js 6. The code editor contains an index.js file with the following content:

```

1 'use strict';
2 const fetch = require('node-fetch');
3 const FormData = require('form-data');
4
5 /**
6 * Triggered from a message on a Cloud Storage bucket.
7 * IAM authorization based on:
8 * https://stackoverflow.com/questions/45787676/how-to-authenticate-a-cloud-storage-trigger-with-iam
9 * https://cloud.google.com/iap/docs/authentication-howto
10 * #param {Object} event The Cloud Functions event.
11 * #param {function} callback The callback function.
12 * exports.triggerDag = function triggerDag(event, callback) {
13 *   // Fill in your Cloud environment information here.
14 *   // The project that holds your function
15 *   const PROJECT_ID = 'qwiklabs-gcp-97d5fb651b04b20';
16 *   // Navigate to your webserver's login page and get this:
17 *   const CLIENT_ID = '954500678485-gde6id87qtdm9it17809uj';
18 *   // This should be part of your webserver's URL:
19 *
20 }
21
22
23
24
25

```

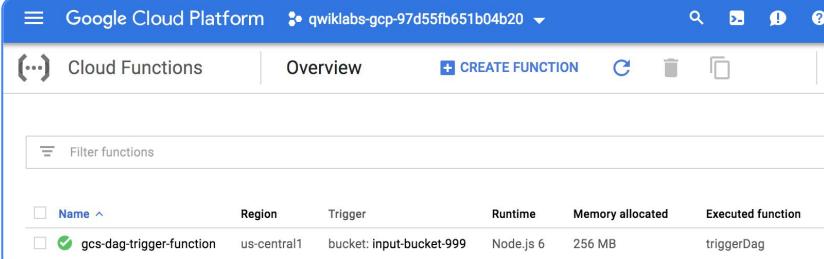
The 'Function to execute' dropdown at the bottom is set to 'triggerDag'. A green box highlights this dropdown.

Google Cloud

Once you have 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 one 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 D-A-G 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



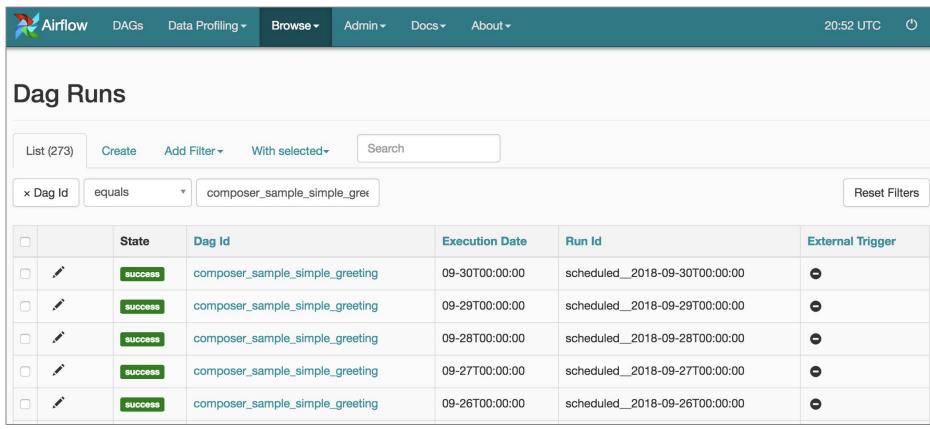
The screenshot shows the Google Cloud Platform Cloud Functions Overview page. At the top, there is a navigation bar with the Google Cloud logo, the project name 'qwiklabs-gcp-97d55fb651b04b20', and various icons for search, refresh, and help. Below the navigation bar, there is a header with a three-dot menu icon, the text 'Cloud Functions', and a 'Overview' tab. To the right of the overview tab are buttons for '+ CREATE FUNCTION', a refresh icon, a trash can icon, and a copy icon. Underneath the header, there is a search bar labeled 'Filter functions'. A table follows, with columns: Name, Region, Trigger, Runtime, Memory allocated, and Executed function. The table contains one row for the function 'gcs-dag-trigger-function', which is highlighted with a green checkmark in the 'Name' column. The details for this function are: Region 'us-central1', Trigger 'bucket: input-bucket-999', Runtime 'Node.js 6', and Memory allocated '256 MB'. The Executed function column shows 'triggerDag'.

Name	Region	Trigger	Runtime	Memory allocated	Executed function
<input checked="" type="checkbox"/> gcs-dag-trigger-function	us-central1	bucket: input-bucket-999	Node.js 6	256 MB	triggerDag

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.

Monitor current and historical workflow progress



The screenshot shows the Airflow interface with the 'Dag Runs' page selected. The top navigation bar includes links for Airflow, DAGs, Data Profiling, Browse, Admin, Docs, and About, along with the time '20:52 UTC' and a power icon. Below the navigation is a search bar with filters. A specific filter for 'Dag Id' is set to 'equals' and 'composer_sample_simple_greeting'. The main table displays five successful runs for this DAG, each with a checkmark icon, a pencil icon, and a green 'SUCCESS' status box. The columns are labeled: State, Dag Id, Execution Date, Run Id, and External Trigger.

	State	Dag Id	Execution Date	Run Id	External Trigger
<input type="checkbox"/>		SUCCESS composer_sample_simple_greeting	09-30T00:00:00	scheduled_2018-09-30T00:00:00	
<input type="checkbox"/>		SUCCESS composer_sample_simple_greeting	09-29T00:00:00	scheduled_2018-09-29T00:00:00	
<input type="checkbox"/>		SUCCESS composer_sample_simple_greeting	09-28T00:00:00	scheduled_2018-09-28T00:00:00	
<input type="checkbox"/>		SUCCESS composer_sample_simple_greeting	09-27T00:00:00	scheduled_2018-09-27T00:00:00	
<input type="checkbox"/>		SUCCESS composer_sample_simple_greeting	09-26T00:00:00	scheduled_2018-09-26T00:00:00	

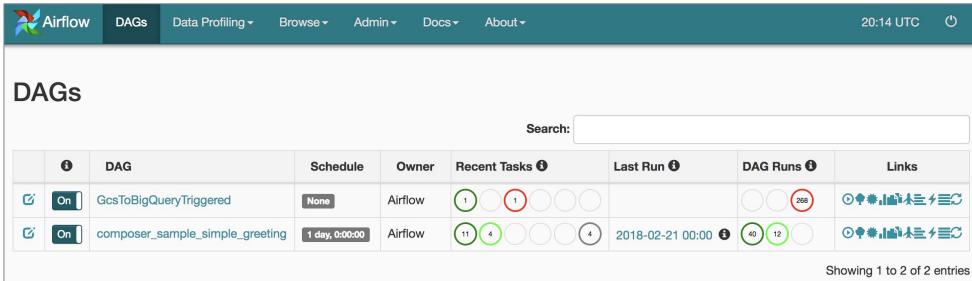
Google Cloud

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.

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 can't get your workflow to run at all in the beginning.

In the "Dag Runs" you can monitor when your pipelines run 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



	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	On GcsToBigQueryTriggered	None	Airflow	1 1 1 1 1		1 1 268	DAG failed
<input checked="" type="checkbox"/>	On composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 1 1 1 4	2018-02-21 00:00:00	40 12 1	DAG succeeded

Showing 1 to 2 of 2 entries

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

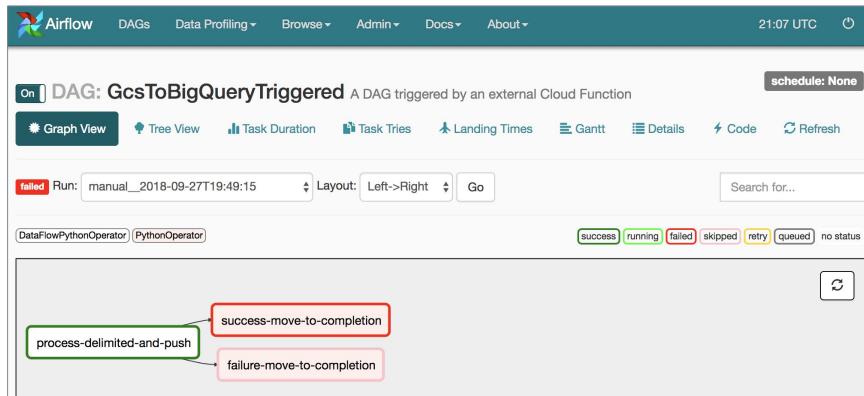
	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
<input checked="" type="checkbox"/>	On GcsToBigQueryTriggered	None	Airflow	1 1 268	2018-02-21 00:00:00	268	268
<input checked="" type="checkbox"/>	On composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 4		40 12	40 12

Showing 1 to 2 of 2 entries

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

The screenshot shows the Airflow web interface with the following details:

- DAG:** GcsToBigQueryTriggered
- Schedule:** None
- Task Instance:** success-move-to-completion (2018-09-27 19:49:15)
- Log tab selected:** Log, Rendered Template, XCom
- Log by attempts:** Attempt 1
- Log Output:**

```
*** Reading remote log from gs://us-central1-evan-composer-0e85530c-bucket/logs/GcsToBigQueryTriggered/success-move-to-completion/2018-09-27 20:03:38,633 [clL.py:374] INFO - Running on host airflow-worker-7bcfc477b-hdgv7 [2018-09-27 20:03:38,698] [models.py:1196] INFO - Dependencies all met for <TaskInstance: GcsToBigQueryTriggered.success-move-to-completion> [2018-09-27 20:03:38,729] [models.py:1196] INFO - Dependencies all met for <TaskInstance: GcsToBigQueryTriggered.success-move-to-completion> [2018-09-27 20:03:38,729] [models.py:1486] INFO - Starting attempt 1 of
_____
[2018-09-27 20:03:39,751] [models.py:1427] INFO - Executing <Task(PythonOperator): success-move-to-completion> on 2018-09-27 : [2018-09-27 20:03:39,751] [base_task_runner.py:115] INFO - Running: ['!hash', '-c', 'uairflow run GcsToBigQueryTriggered success-move-to-completion --subtask=Subtask: (2018-09-27 20:03:40,439) __init__.py:45] INFO - Using ex]
```

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

The screenshot shows the Google Cloud Platform Cloud Logging interface. The left sidebar has 'Logs' selected. The main area has a dropdown menu set to 'Dataflow Step'. A list of log entries is displayed, each with a timestamp, log level (PDT), error message, and a detailed description of the error. The logs show multiple instances of errors where Dataflow pods are syncing and failing to download packages, specifically mentioning 'apache_beam-2.5.0-0cp1-cp27mu-manylinux_x86_64.wheel'.

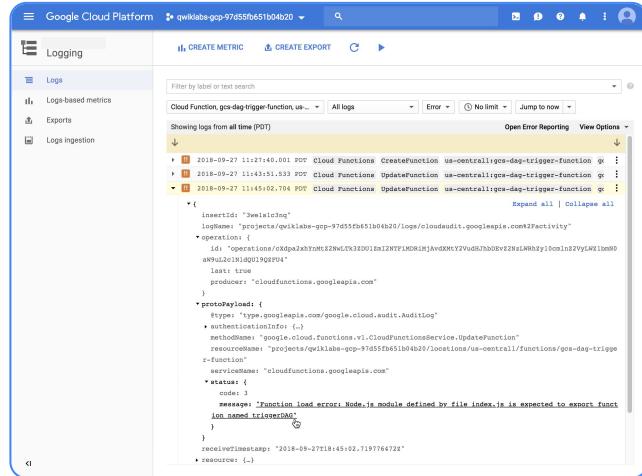
Timestamp	Error Message	Detailed Description
2018-09-27 12:22:23.144 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:22:24.170 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:22:25.187 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:23:13.190 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:23:25.320 PDT	Error: download of	01apache_beam-2.5.0-0cp1-cp27mu-manylinux_x86_64.whe...
2018-09-27 12:23:25.521 PDT	Error: download of	01dataflow_python_sdk.tgz failed (#0): object.gzip(/...)
2018-09-27 12:23:25.521 PDT	Failed to download packages:	download incomplete: 2/2 files failed
2018-09-27 12:23:26.096 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:23:42.187 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:23:54.187 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:24:06.188 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:24:20.187 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:24:26.186 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:24:51.322 PDT	Error: download of	01dataflow_python_sdk.tgz failed (#0): object.gzip(/...)
2018-09-27 12:24:51.419 PDT	Error: download of	01apache_beam-2.5.0-0cp1-cp27mu-manylinux_x86_64.whe...
2018-09-27 12:24:51.619 PDT	Failed to download packages:	download incomplete: 2/2 files failed
2018-09-27 12:24:52.038 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:25:05.191 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")
2018-09-27 12:25:18.187 PDT	Error syncing pod	076cbab870a706d93de43711149130af3 ("dataflow-processes-...")

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.

Monitor Cloud Function health in Cloud Logging



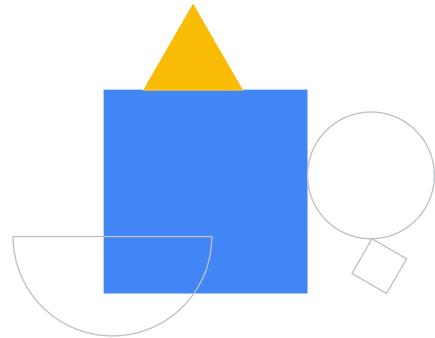
Google Cloud

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 upload a CSV file to my Cloud Storage bucket hoping to trigger a Cloud Function and then my DAG, I get an error message that includes: '*expected to export function named 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 Functions for the first time.

Lab Intro

An Introduction to
Cloud Composer



Google Cloud

Let's now take some time to practice what you learned in this module by creating an environment in Cloud Composer, running a DAG and Airflow, and analyzing the results.

Lab objectives

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



Google Cloud

The objectives of this lab are to:

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