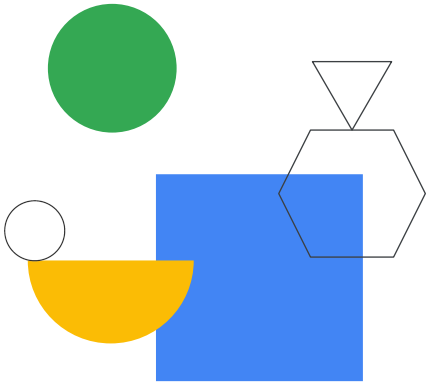
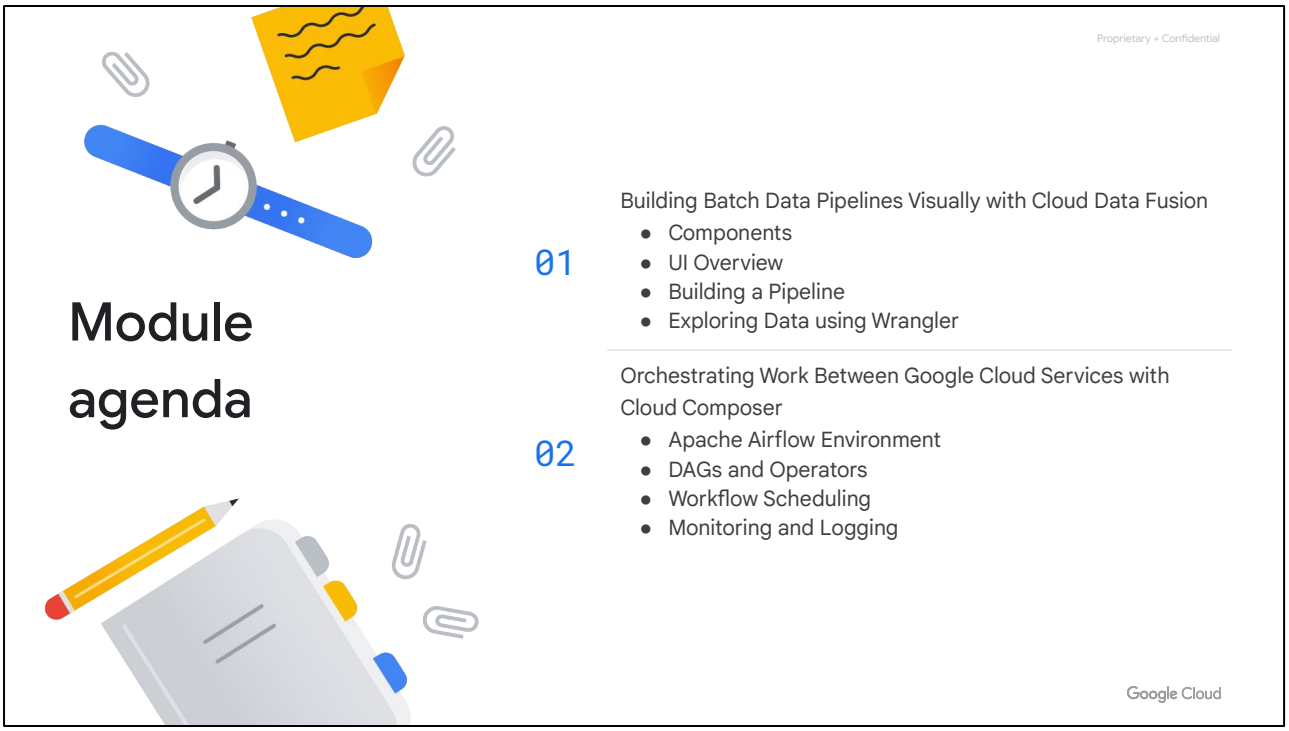
Google Cloud

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.



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.

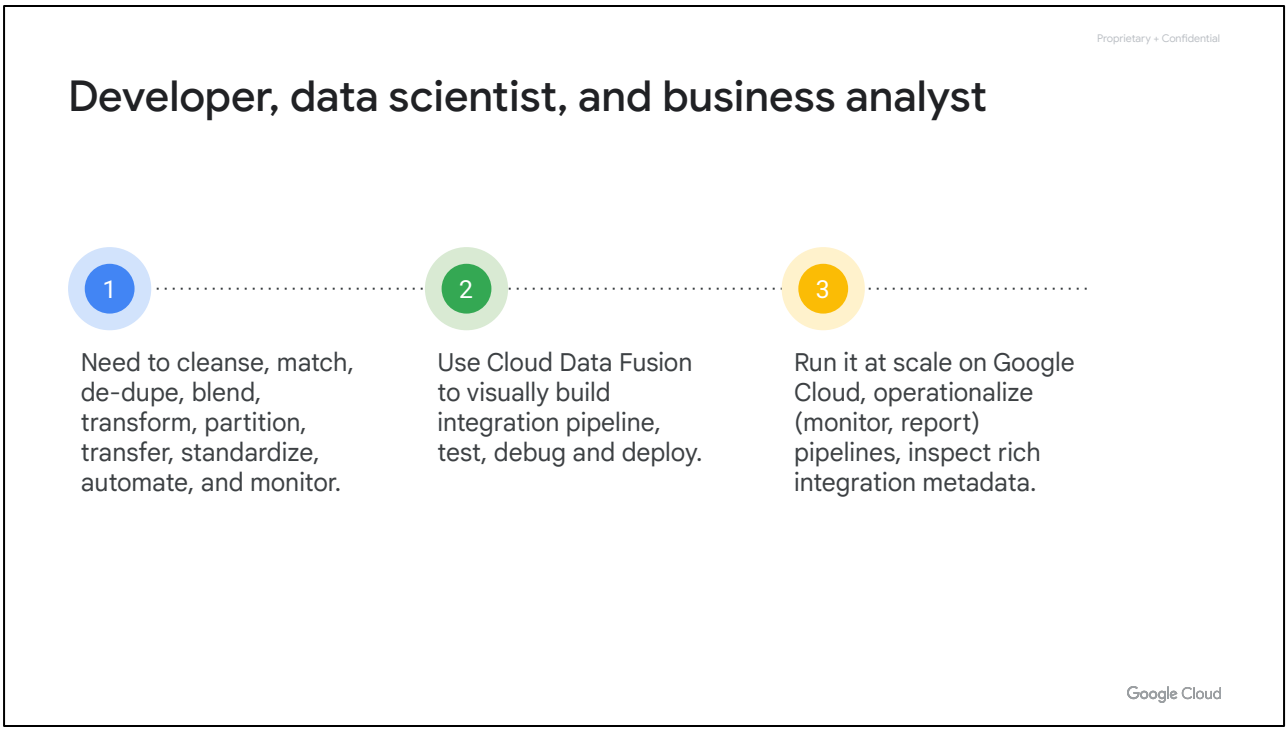
Building Batch Data Pipelines Visually with Cloud Data Fusion

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.

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.

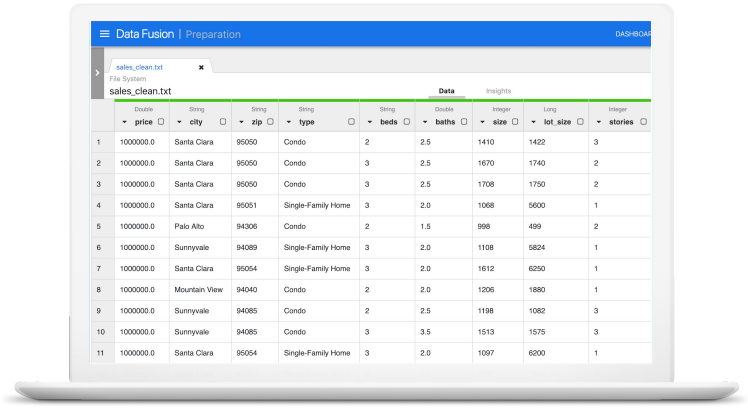


Cloud Data Fusion is used by developers, data scientists, and business analysts alike.

- 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

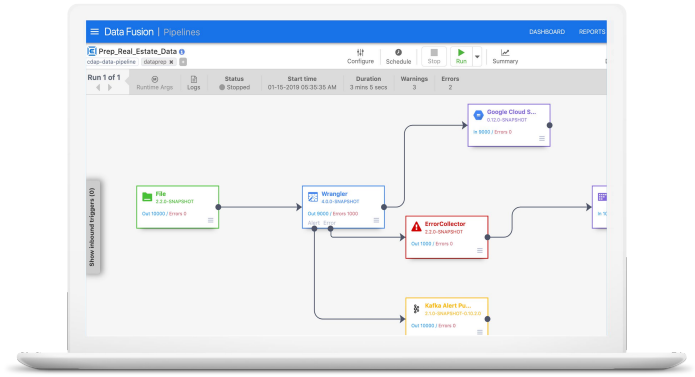


Cloud Data Fusion offers a number of benefits:

- **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



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

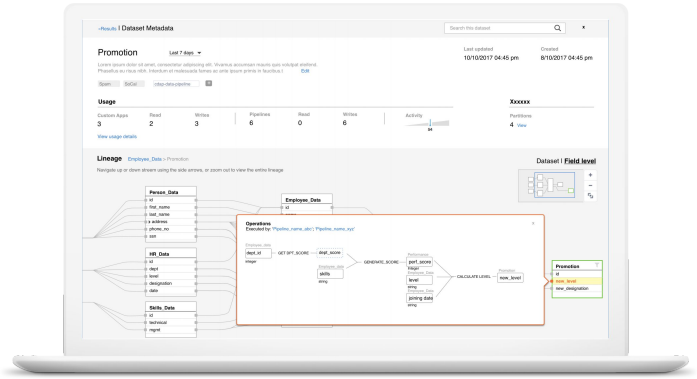
Tags and Properties support

- Pipeline
- Dataset
- Schema

Search integrated entities by

- Keyword
- Schema name and type

Dataset level and Field level lineage

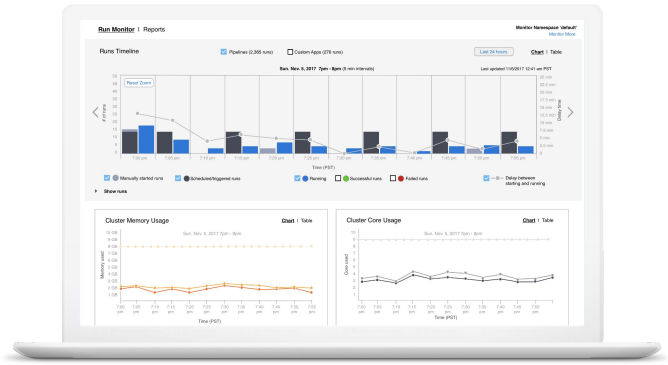


As you will see in the next lesson, you can tag pipelines to help organize them more efficiently for your team, and you can use the unified search functionality to quickly find field values or other keywords across your pipelines and schemas.

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.

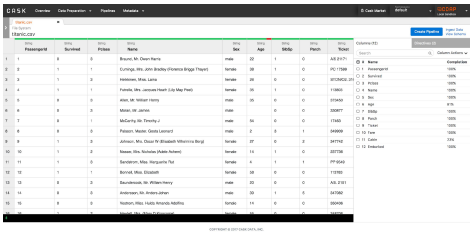
Extensible

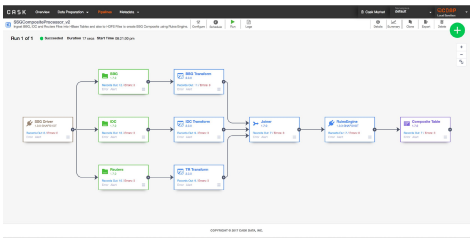
- Pipeline templatization
- Conditional pipeline triggers
- Plugin management
- Plugin templatization
- Plugin UI widget
- Custom provisioners
- Custom compute profiles
- Hub integration



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 hubs.

Components of Cloud Data Fusion





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

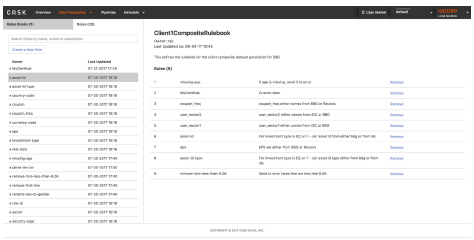
Data Pipeline — Framework

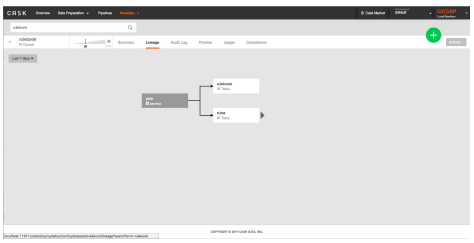
- User interface for building complex data workflows.
- Join, Lookup, Aggregate, Filtering data in-flight.
- Building complex workflows with 100s of connectors.
- Extend Data Pipeline using simple APIs.
- **Integrates with Dataprep, Rule Engine and Metadata Aggregator.**

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





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

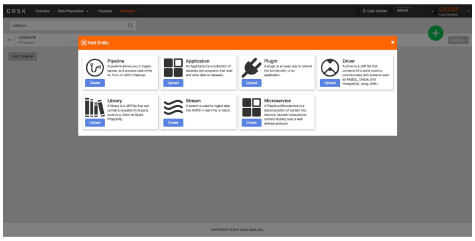
- Aggregate Business, Technical and Operational Metadata.
- Track the flow of data (Lineage) for richer data needed for governance.
- Create Data Dictionary and Metadata Repository.
- Integrate with enterprise MDM solutions.
- Integrates with Data Pipeline, Rules Engine.

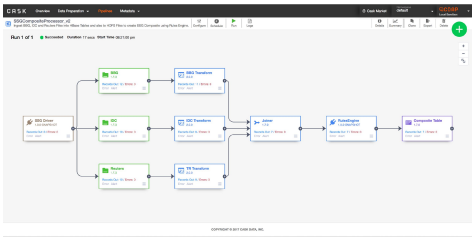
There are other features of Cloud Data Fusion that you should be aware of too.

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

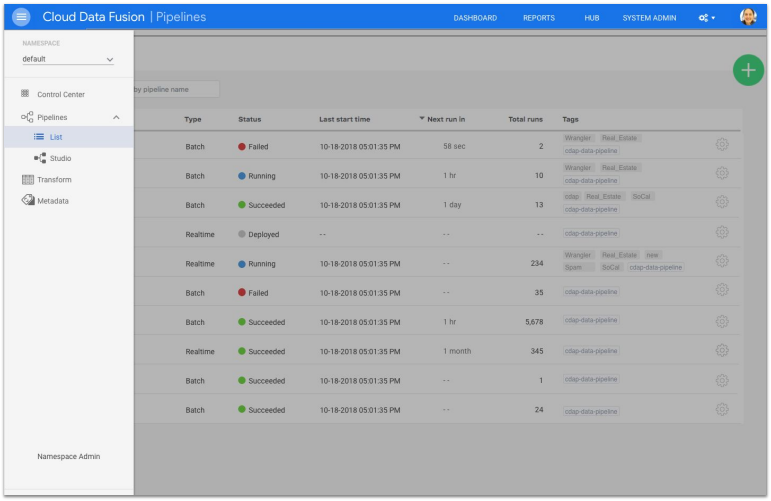
- Delivers a specialized solutions for IoT event processing.
- Parses any events, triggers conditions and executes Action.
- Real-time notification system, with easy-to-use user interface for configuring event parsing, condition and actions.

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.

Cloud Data Fusion - UI overview

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

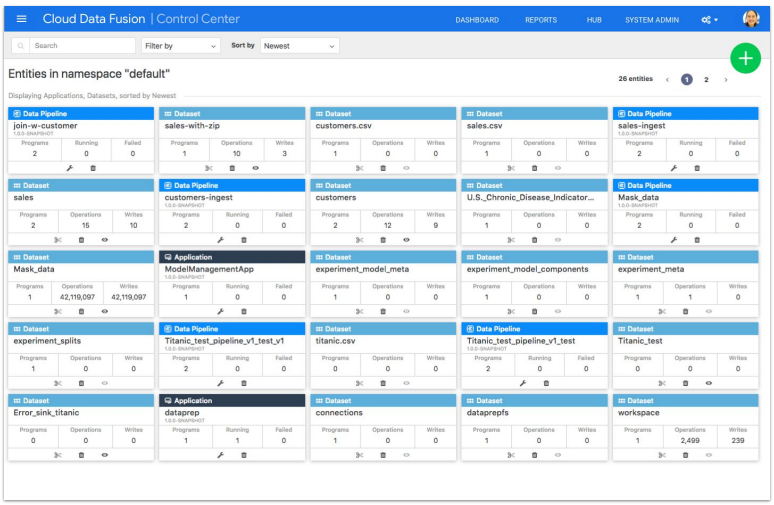


Managing your pipelines is easiest when you have the right tools. We'll now take a high-level look at the Cloud Data Fusion UI as you saw in the component overview.

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

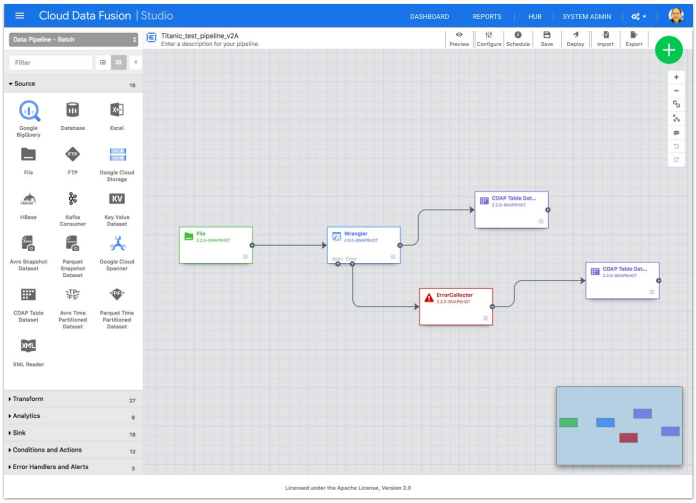


Under Control Center, there's a section for applications, artifacts and a dataset. Here you could have multiple pipelines associated with a particular application.

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

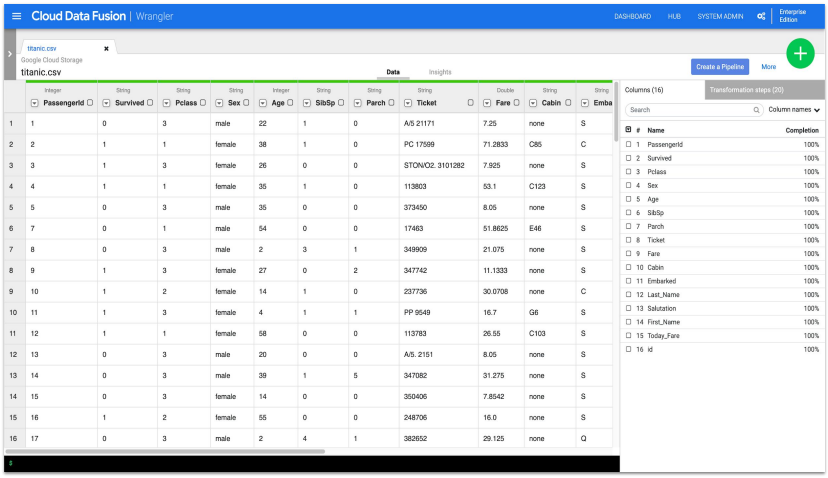
- Developer Studio
- Preview
- Export
- Schedule
- Connector and function palette
- Navigation



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

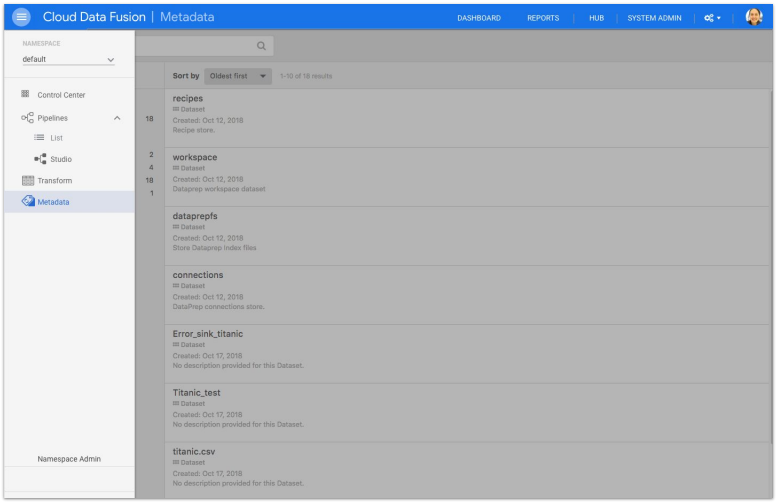
- Connections
- Transforms
- Data Quality
- Insights
- Functions



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

Integration metadata

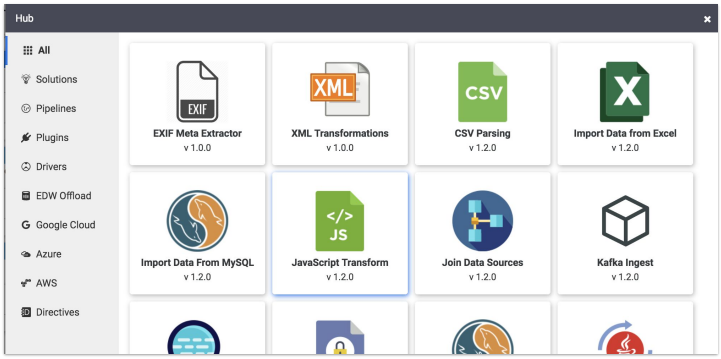
- Search
- Tags and Properties
- Lineage - Field and Data



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

Hub

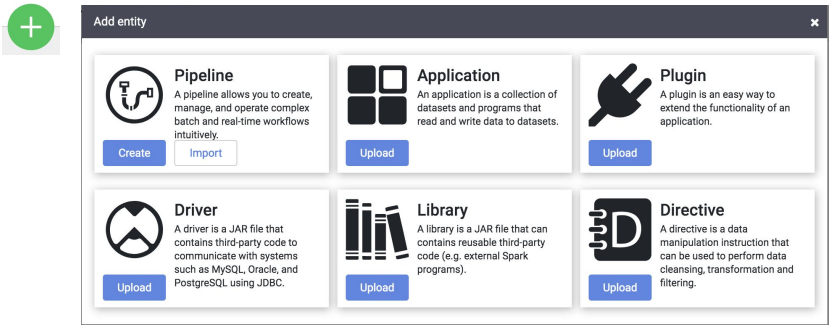
- Plugins
- Use cases
- Pre-built pipelines



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

Entities

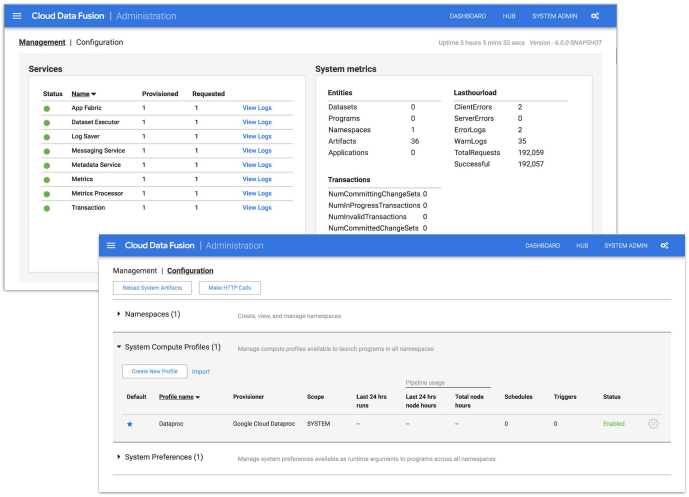
- Pipeline
- Application
- Plugin
- Driver
- Library
- Directive



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

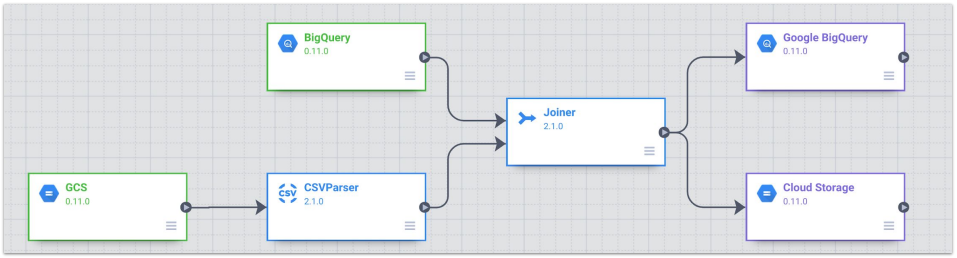
Administration

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



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.

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.

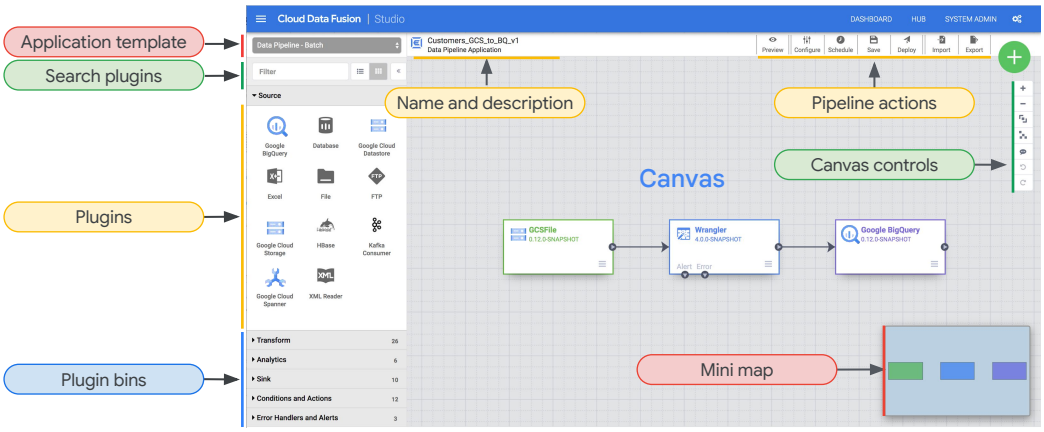
Each stage is a node, and as you can see here, nodes can be of a different type. You may start with a node that pulls data from Cloud Storage, then passes it on to a node that parses a CSV. The next node takes multiple nodes, has an input, and joins them together before passing the join data to two separate 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.

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



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.

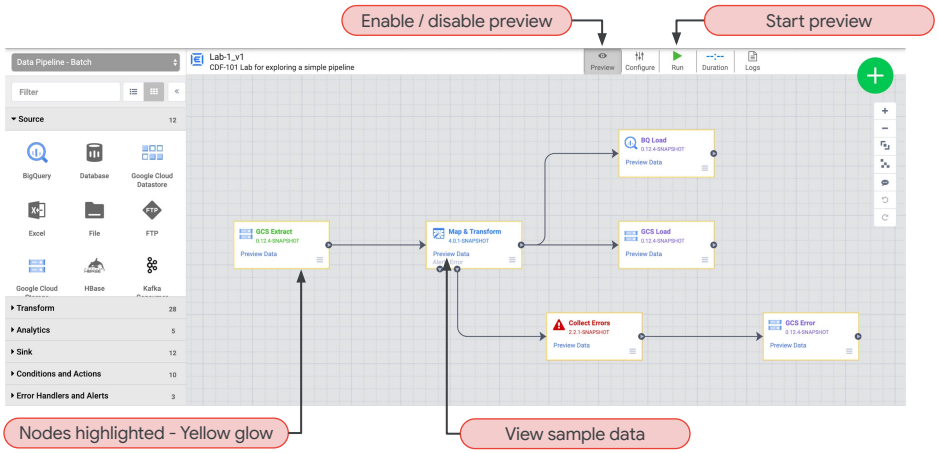
You can interact with the canvas and add objects by using the **Canvas Control Panel**.

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.

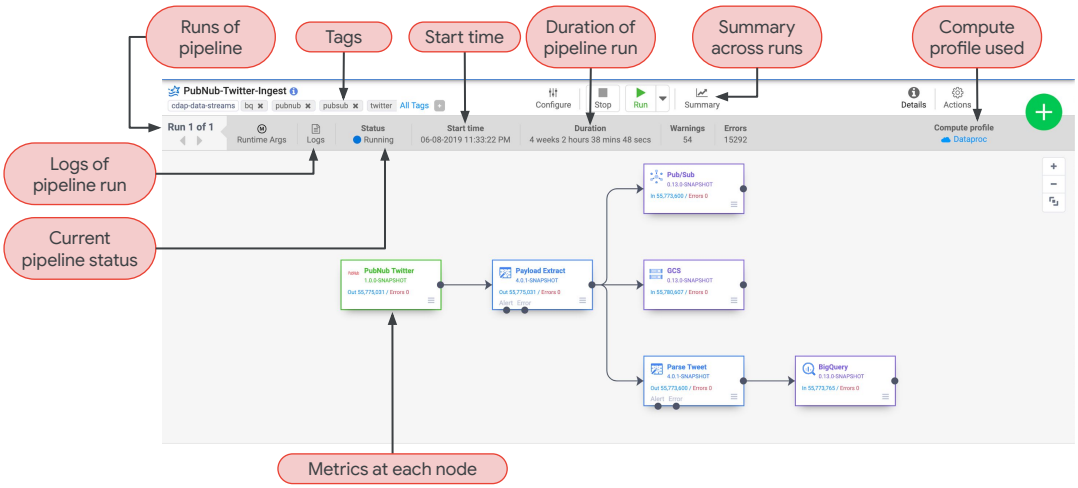
Here, we've used a template or data pipeline batch which gives us the three nodes you see here to move data from a Cloud Storage file, process it in a wrangler, and output it to BigQuery.

Use Preview Mode to see how the pipeline will run



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



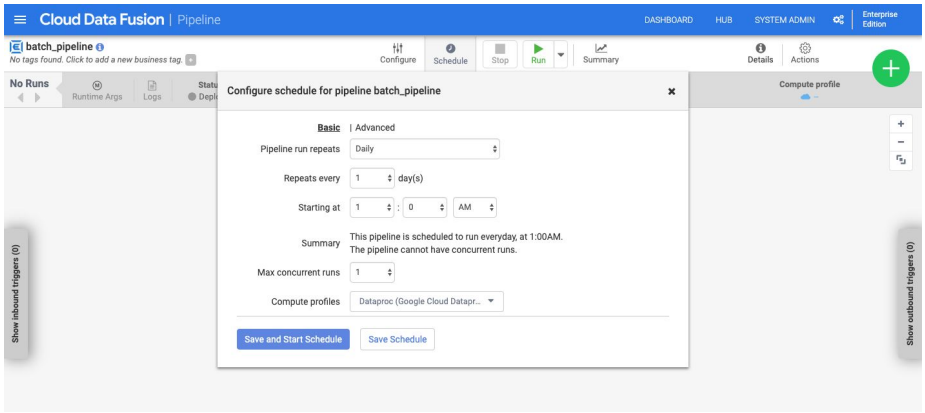
After deployment, you can monitor the health of your pipeline and collect key summary stats of each execution. Here, we are ingesting data from Twitter and Google Cloud, and parsing each tweet before loading them into a variety of datasinks.

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

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.

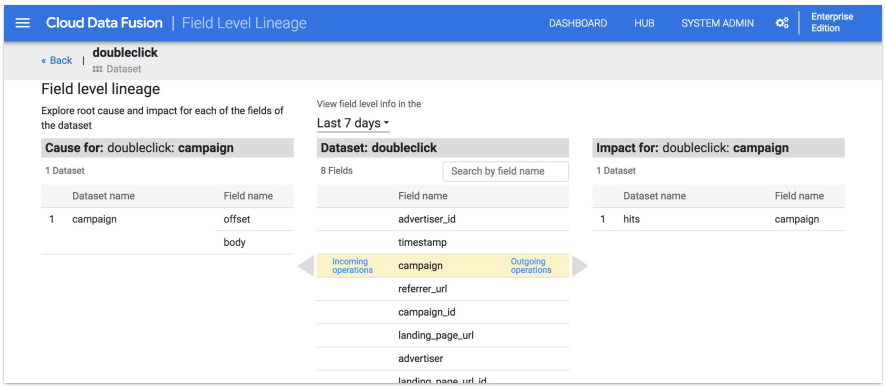
You can schedule batch pipelines



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

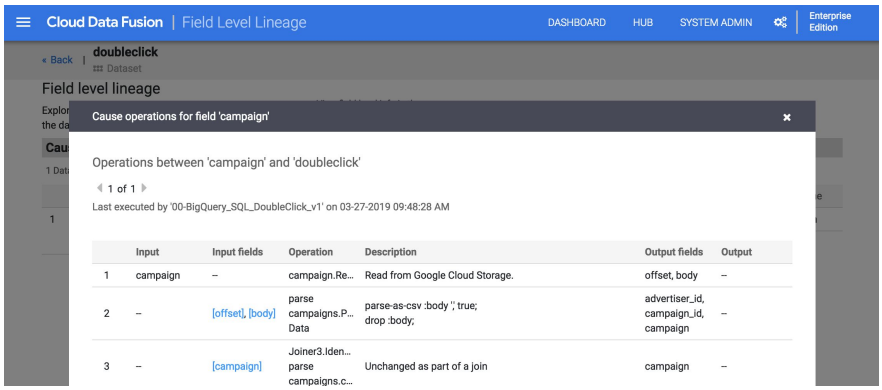
Relationship between fields of datasets: Provenance, Impact



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

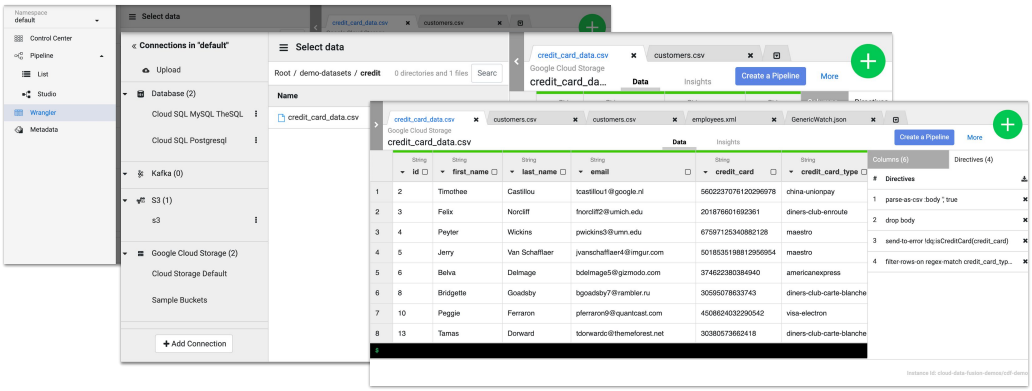
Operations applied to a field



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.

Data analysts can explore datasets in the Wrangler

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



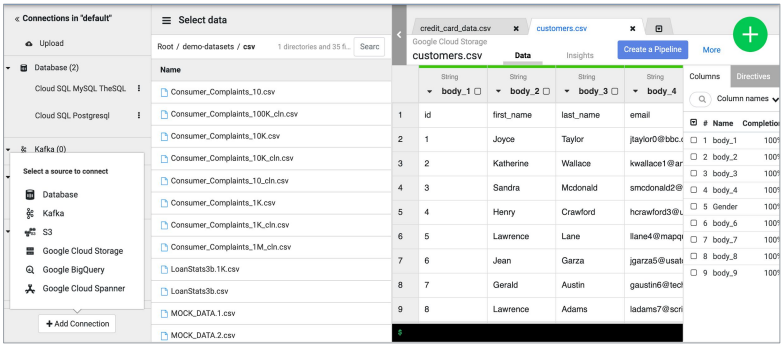
Google Cloud

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

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.

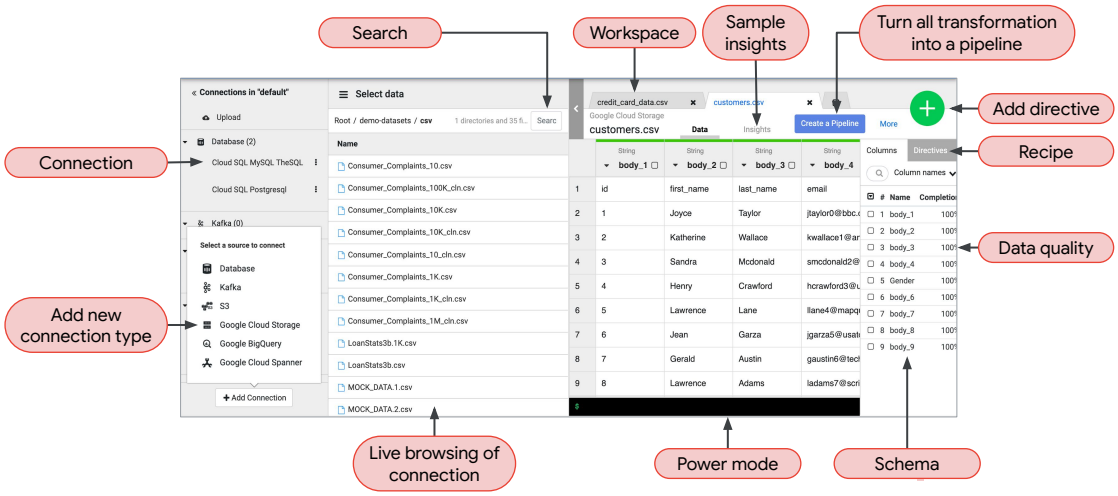
The Wrangler UI is the Cloud Data Fusion environment for exploring new datasets visually for insights. Here, you can inspect the dataset and build a series of transformation steps called directives to stitch together a pipeline.

Wrangler UI overview for exploring datasets



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.

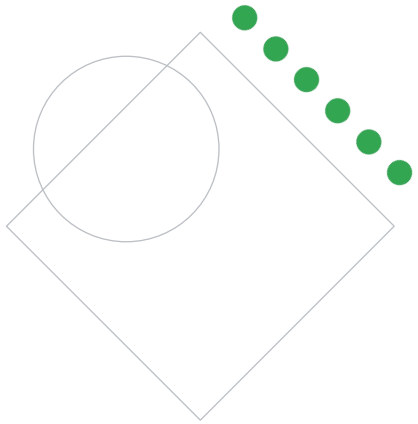
Here, you see a Cloud Storage bucket of demo datasets and all the CSV files of customer complaints. Once you've found an example dataset like customers.csv here, you can **explore** the rows and columns visually and view sample insights.

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



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

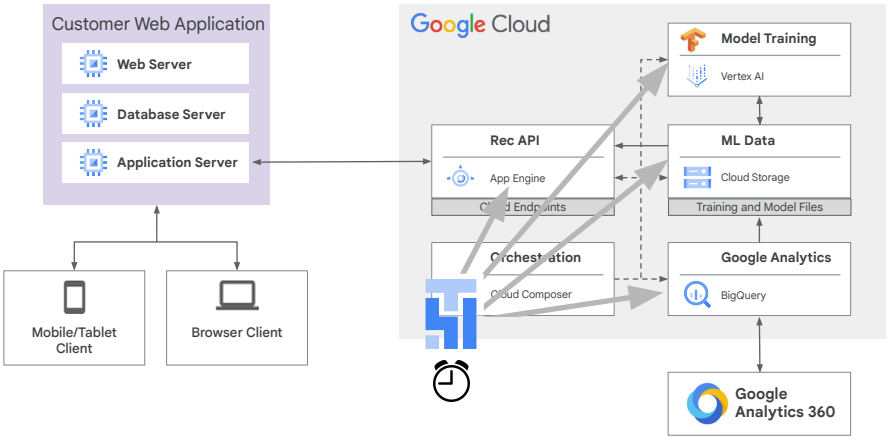
04 Write data to a sink

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

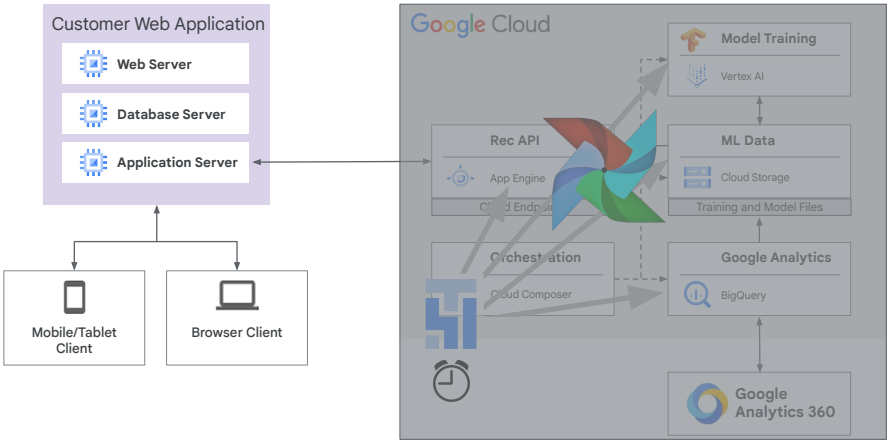
The next big task for managing data pipelines is to orchestrate the work across multiple Google Cloud services. For example, if you had three Cloud Data Fusion Pipelines and two ML models that you want to run in a certain order, you need an orchestration engine. In this module, we'll look at using Cloud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

Cloud Composer orchestrates automatic workflows



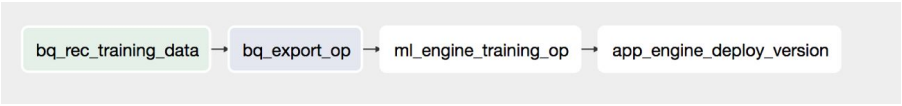
Cloud Composer will control the Google Cloud services that we need to run. But Cloud Composer is simply a serverless environment on which an open source workflow tool runs.

Cloud Composer is managed Apache Airflow



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

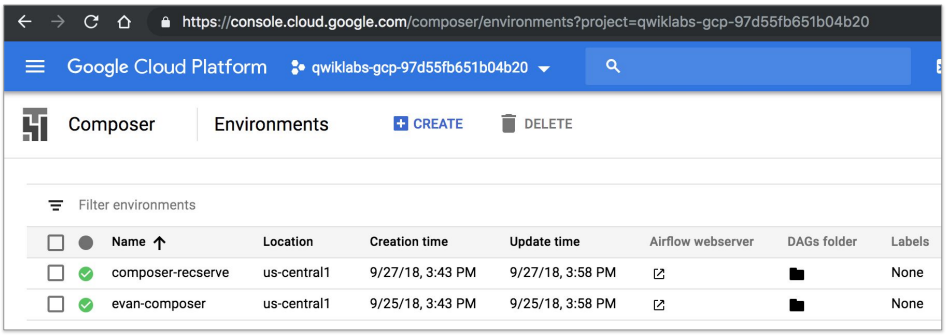
Use Apache Airflow DAGs to orchestrate Google Cloud services



DAG = Directed Acyclic Graph

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

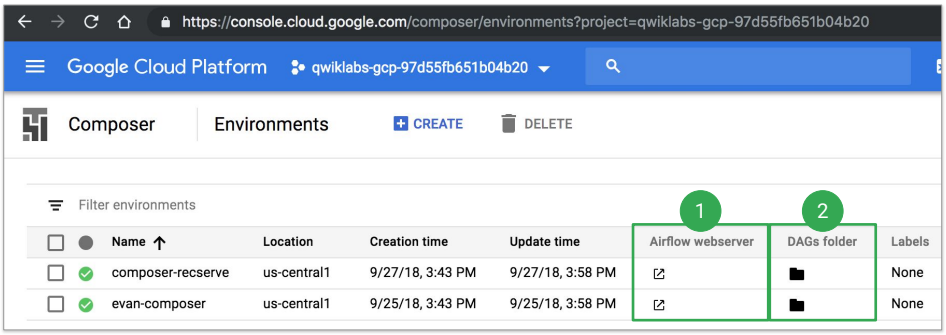


Let's preview the actual Cloud Composer environment.

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.

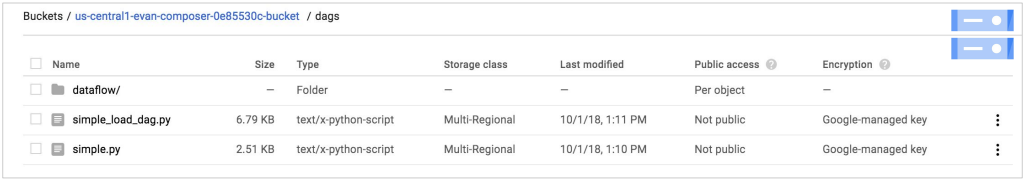
Each Airflow environment has a separate web server and folder in Cloud Storage for pipeline DAGs



To access the Airflow admin UI where you can monitor and interact with your workflows you'll click on the link underneath Airflow webserver.

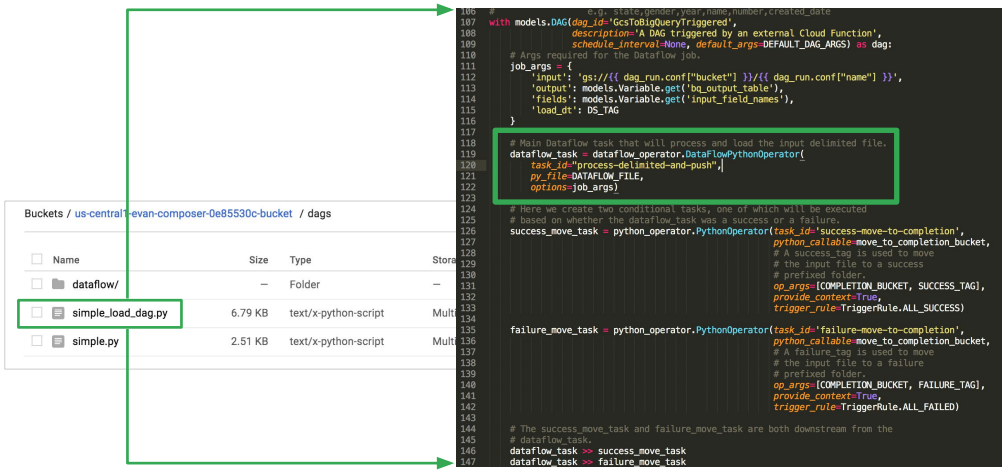
The second box you see is the DAGs folder which is where the code of your actual workflows will be stored.

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



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



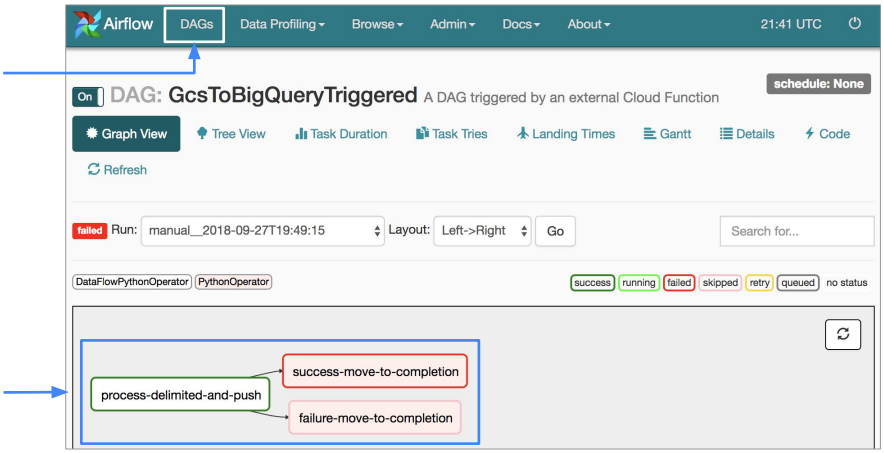
Now that you're familiar with the basic environment setup, it's time to discuss your primary artifact, which is your DAG, and the operators you are using to call whichever services you want to send tasks to.

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 it's components a little later.

The Python file creates a DAG

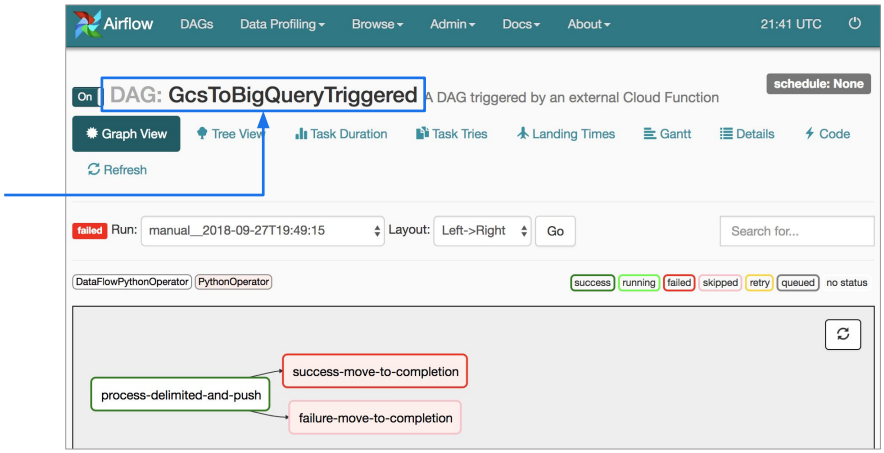


Once you've uploaded the Python file to the DAGs folder, you can navigate back to the Airflow webserver and under DAGs you'll see the DAG you created with code represented visually as a directed graph, with nodes and edges.

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.

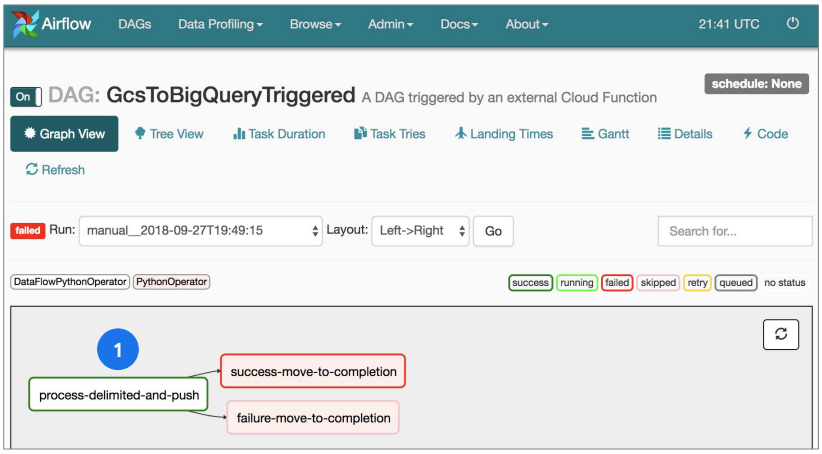
Let's explore a bit more of the Airflow web UI.

Airflow web server UI overview



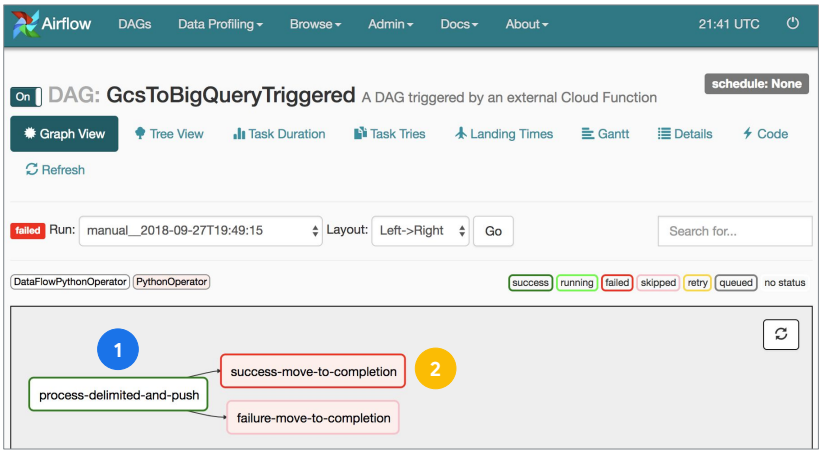
You can see that this particular workflow is called GcsToBigQueryTriggered and it has three TASKS when it runs.

The Python file creates a DAG



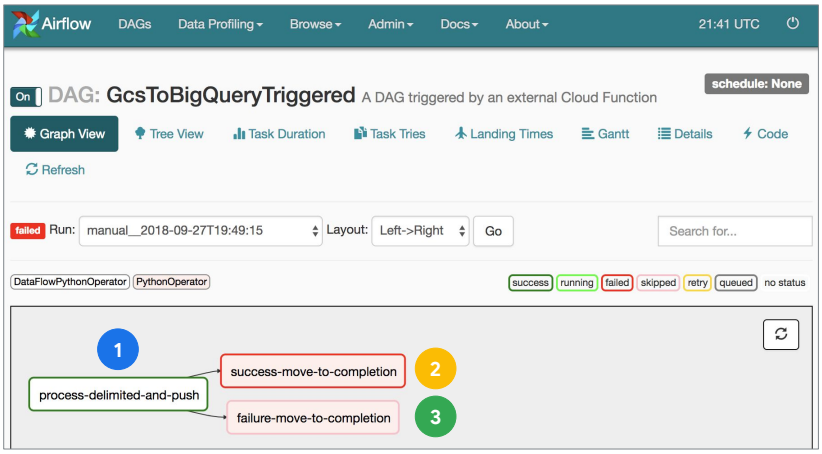
One, process-delimited-and-push. I just happen to know from that Python file you saw earlier that this task invokes a Dataflow job to read in a new CSV file from a Cloud Storage bucket, processes it, and writes the output to BigQuery.

The Python file creates a DAG



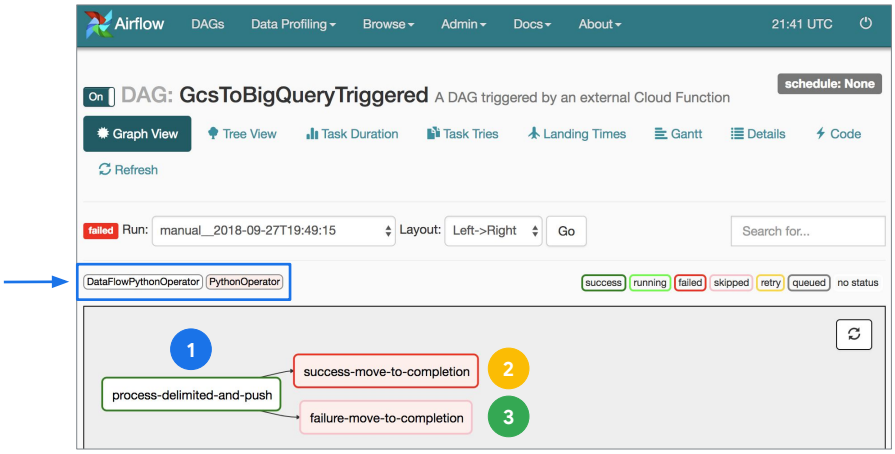
Two, success-move-to-completion, which moves the CSV file from an input Cloud Storage bucket to a processed or completed Cloud Storage bucket for archiving, or

The Python file creates a DAG



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



But regardless of the size and shape of your workflow DAG, one common thread for all workflows is the common Operators used. Another 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

Google Cloud Operators

- Google Cloud AutoML Operators
- Google Cloud BigQuery Operators
- Google Cloud BigQuery Data Transfer Service Operators
- Google Cloud Bigtable Operators
- Google Cloud Build Operators
- Google Cloud Composer Operators
- Google Cloud Memorystore Operators
- Google Cloud Memorystore Memcached Operators
- Google Cloud SQL Operators
- Google Cloud Transfer Service Operators
- Google Compute Engine Operators
- Google Compute Engine SSH Operators
- Google Cloud Data Loss Prevention Operator
- Google Cloud Data Catalog Operators
- Google Cloud Dataflow Operators
- Google Dataform Operators
- Google DataFusion Operators
- Google Dataplex Operators
- Google Dataprep Operators
- Google Cloud Dataproc Operators
- Google Cloud Dataproc Metastore Operators
- Google Cloud Datastore Operators

View all Google Cloud services that Airflow can orchestrate:

https://airflow.apache.org/docs/apache-airflow-providers-google/stable/operators/cloud/index.html

\+ sample code

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.

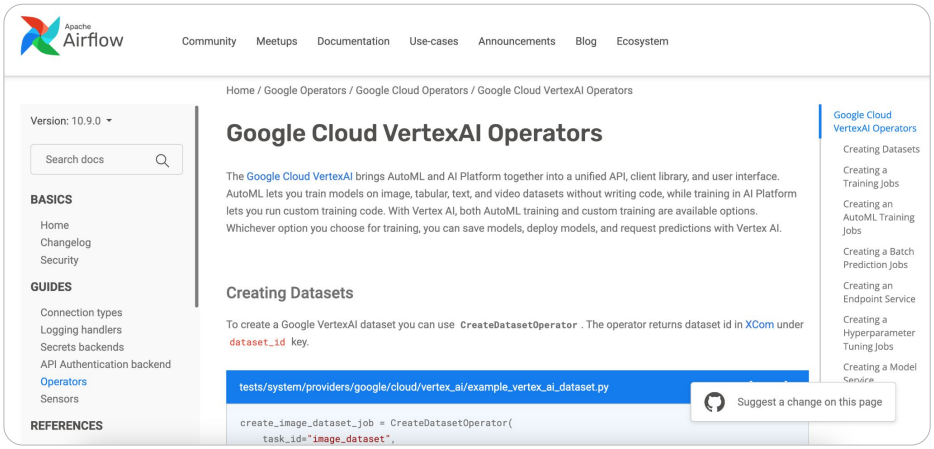
BigQuery Operators are popular for updating your ML training dataset

Task	Operator
Create dataset	BigQueryCreateEmptyDatasetOperator
Get dataset details	BigQueryGetDatasetOperator
List tables in dataset	BigQueryGetDatasetTablesOperator
Update table	BigQueryUpdateTableOperator
Update dataset	BigQueryUpdateDatasetOperator
Delete dataset	BigQueryDeleteDatasetOperator
Create native table	BigQueryCreateEmptyTableOperator
Create external table	BigQueryCreateExternalTableOperator
Fetch data from table	**BigQueryGetDataOperator**
Upsert table (either update table or create new one)	BigQueryUpsertTableOperator

Task	Operator
Update table schema	BigQueryUpdateTableSchemaOperator
Delete table	BigQueryDeleteTableOperator
Execute BigQuery jobs	BigQueryInsertJobOperator
Check if query result has data	**BigQueryCheckOperator**
Compare query result to pass value	BigQueryValueCheckOperator
Compare metrics over time	**BigQueryIntervalCheckOperator or BigQueryIntervalCheckAsyncOperator**
Check columns with predefined tests	BigQueryColumnCheckOperator
Check table level data quality	BigQueryTableCheckOperator
Check that a Table exists	BigQueryTableExistenceSensor
Check that a Table Partition exists	BigQueryTablePartitionExistenceSensor

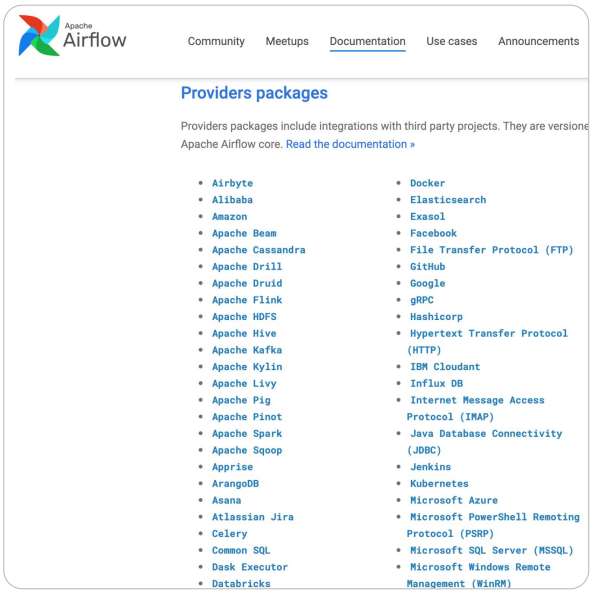
As you might have guessed, we'll certainly be making use of the BigQuery operators since our workflows depend on the data that is fed into them through Cloud Storage and BigQuery. Here's a list of the specific operators that we can invoke in a task to call on the BigQuery service for querying and other data-related tasks. You'll be mainly working with the BigQueryGetDataOperator, BigQueryCheckOperator, and BigQueryIntervalCheckOperator 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 operators can launch training and deployment jobs



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



You might have noticed that your Airflow DAG can have operators that send tasks out to other cloud and software 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.

Two scheduling options for Cloud Composer workflows

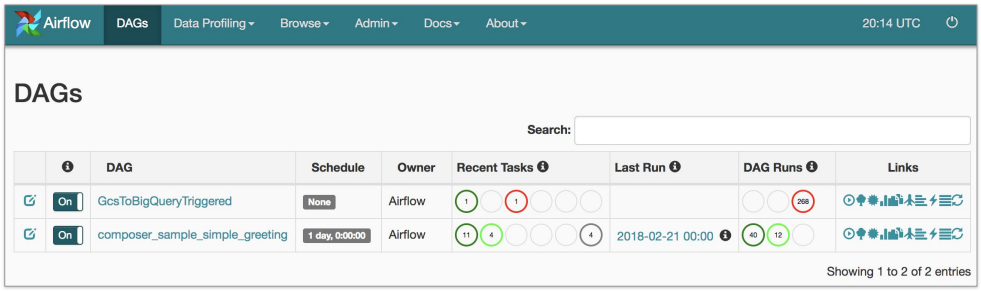
Periodic

Event-driven

Now that you're familiar with the Cloud Composer and Apache Airflow environments, it's time to discuss a really important topic: workflow scheduling.

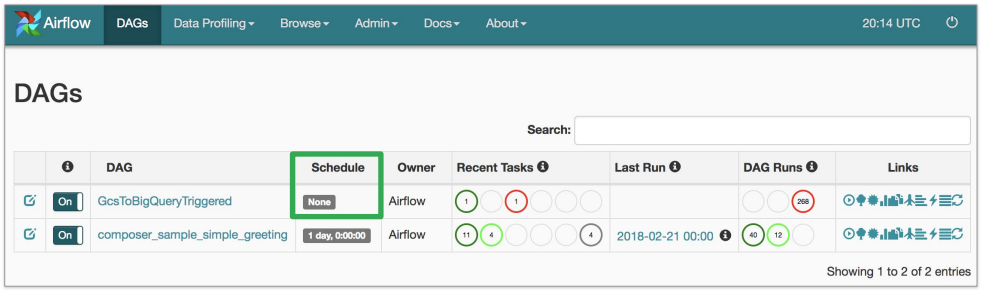
As mentioned earlier, there are two different ways your workflow can be run without you sitting there manually clicking "run DAG". The first and most common is a set schedule or periodic run of a workflow like once a day at 6am, or weekly on Saturdays. The second way is trigger-based, like if you wanted 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



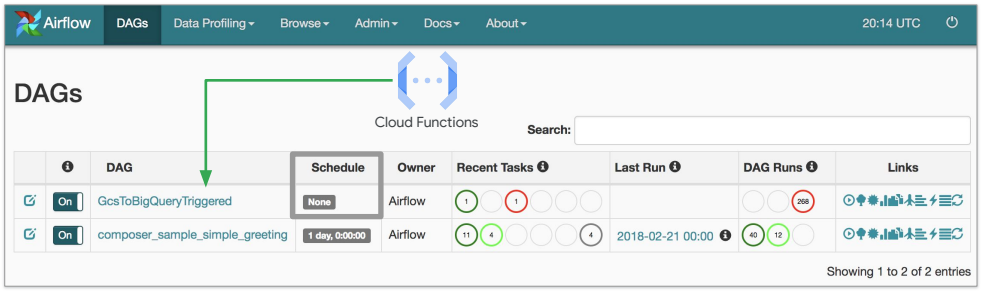
Then navigate to the DAGs tab to view the existing workflows that you have python DAG files for. Here we have two DAGs. The bottom one, composer_sample_simple_greeting has a daily schedule but...

Why is this DAG missing a schedule?



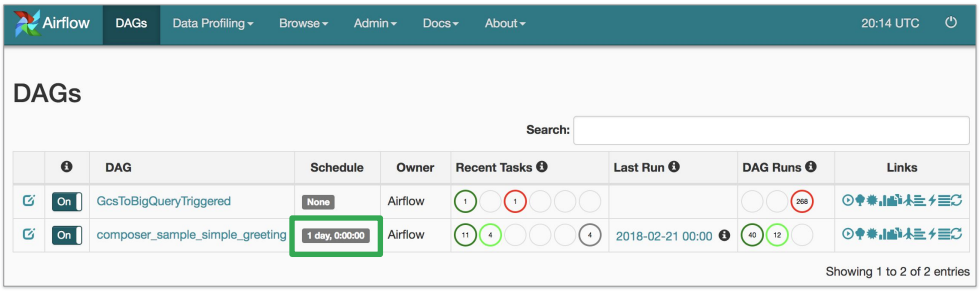
...why is this top DAG missing a schedule? How would it ever get run?

Option 1: Event-driven scheduling with Cloud Functions



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.

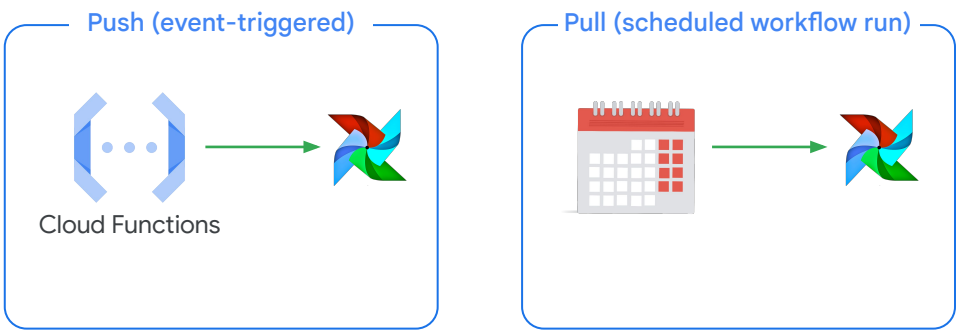
Option 2: Specify pipeline schedule_interval in your DAG




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


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.

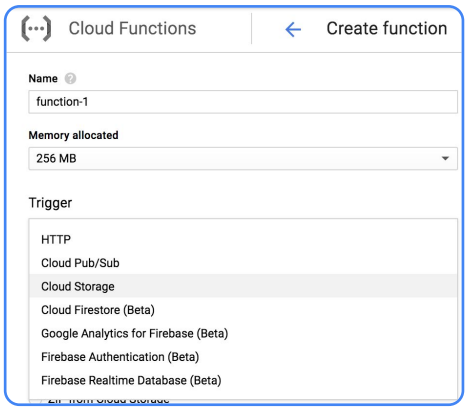
Two types of workflow ETL patterns



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

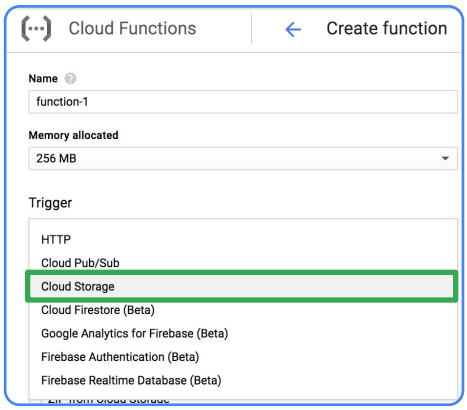


We can use Cloud Functions to create our event-driven or push architecture workflow. I mentioned triggering on events within a Cloud Storage bucket but you can also trigger based on HTTP requests, Pub/sub, Firestore, Firebase and more as you see here.

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.

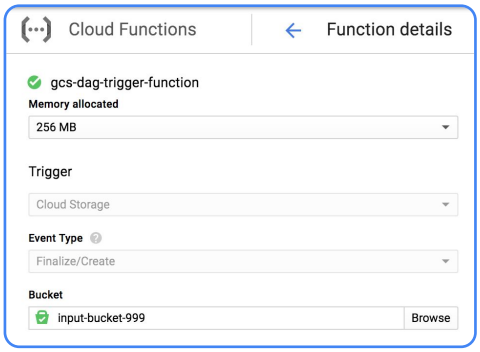
For ML workflows where your upstream data doesn't arrive at a regular pace (like get all the transactions at the end of each day) consider experimenting with a push architecture. Your final lab, since it's based on regular Google Analytics news article data, will be a pull architecture but I've added 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



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.

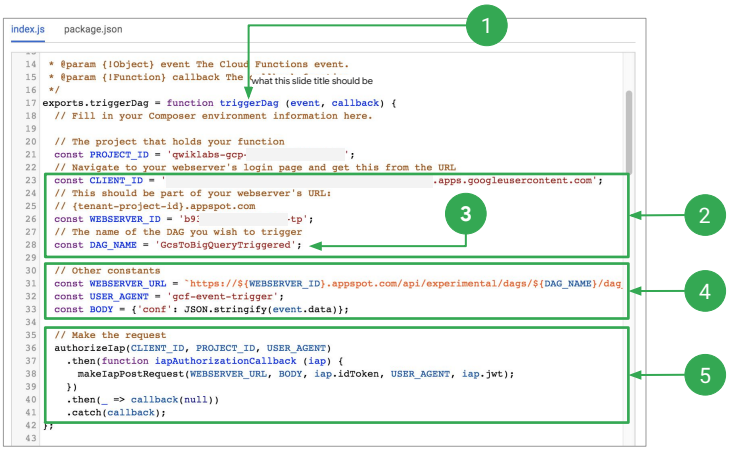
Creating a Cloud Function to trigger an Airflow DAG



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

Creating a Cloud Storage Cloud Function to trigger an Airflow DAG

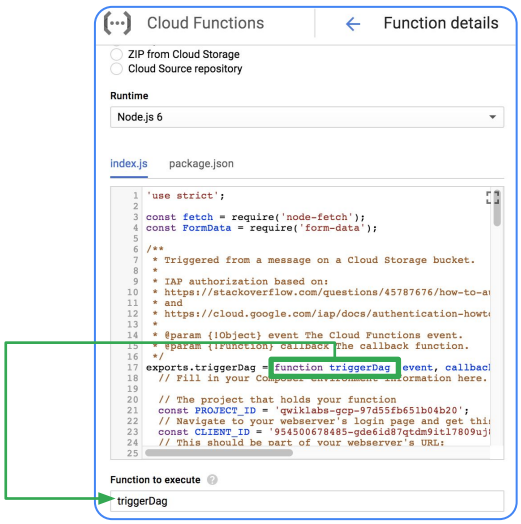
Cloud Functions



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

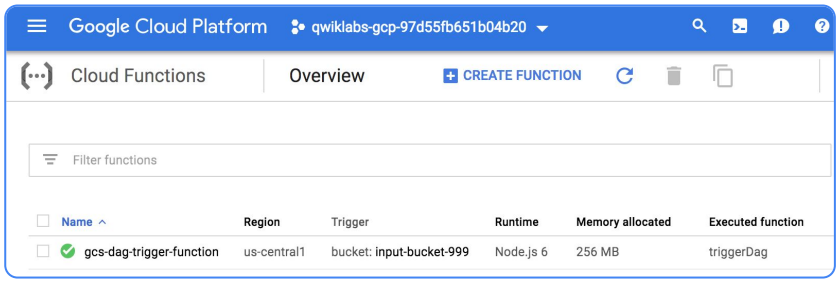
Be sure to specify the function you want Cloud Functions to call in your script



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.

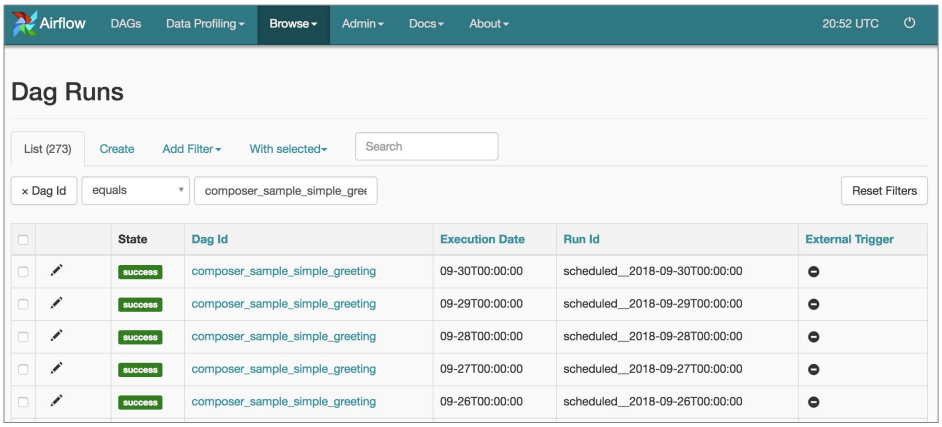
I'll also save you about 20 minutes of frustration and tell you that the 'function to execute' box is case sensitive so all capital letters D-A-G is different than capital D lowercase a and g.

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



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

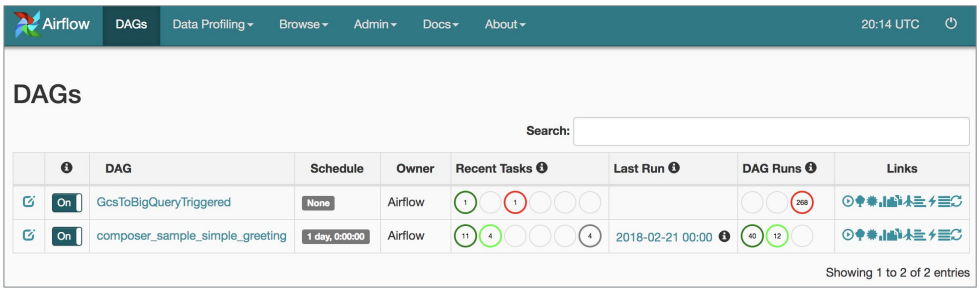


By this point, we've got our environment setup with our DAGs running at predefined schedule or with triggered events. The last topic we'll cover before you practice what you've learned in your labs is how to monitor and troubleshoot your Cloud Functions and Airflow workflows.

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.

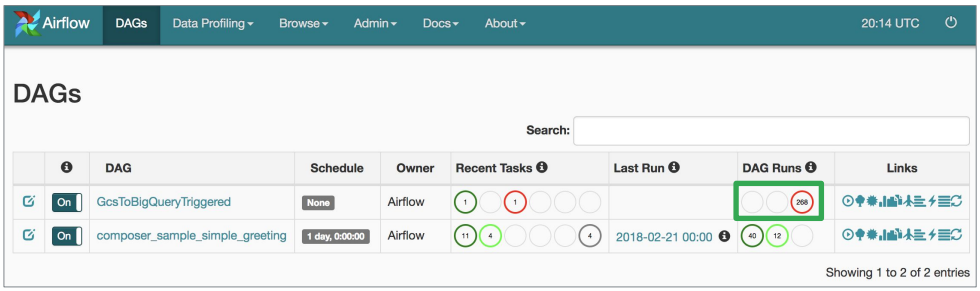
Quickly gauge pipeline health of DAG runs



Which pipeline is healthier?

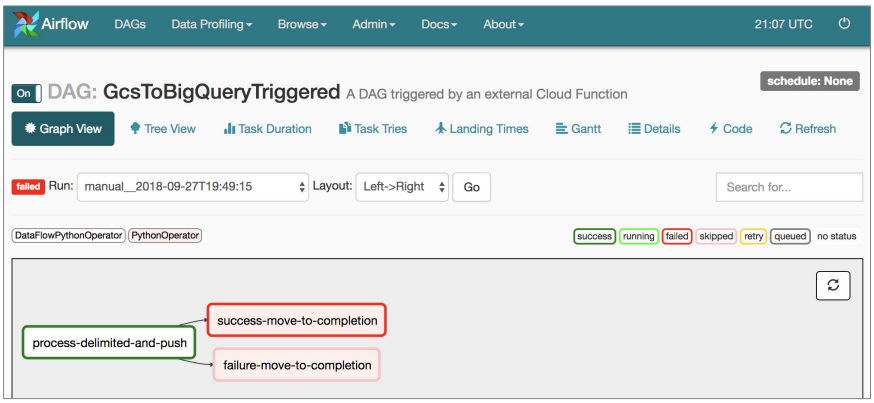
Back on the main page for DAGs, we see some red which indicates trouble with some of our recent DAG runs.

Quickly gauge pipeline health of DAG runs



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.

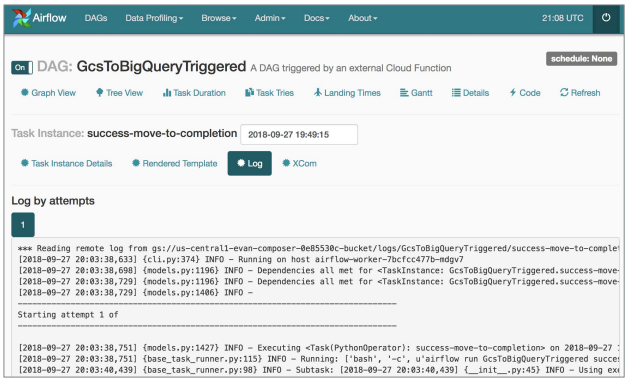
Quickly gauge pipeline health of DAG runs



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.

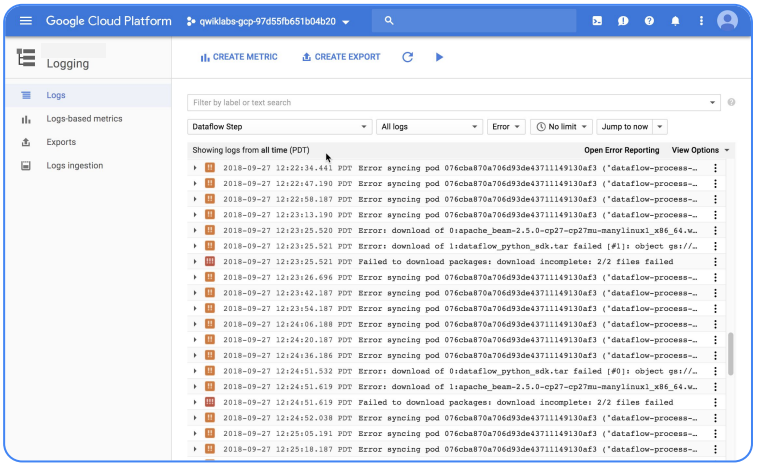
Monitor and troubleshoot Airflow step errors in logs



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/'">

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.

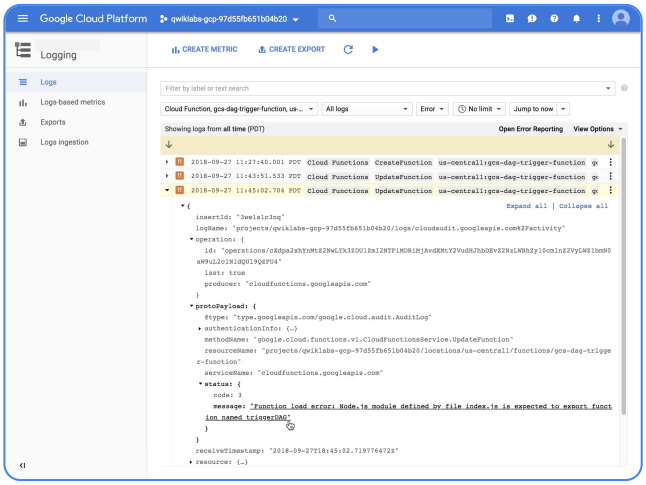
Monitor Dataflow health in Cloud Logging



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

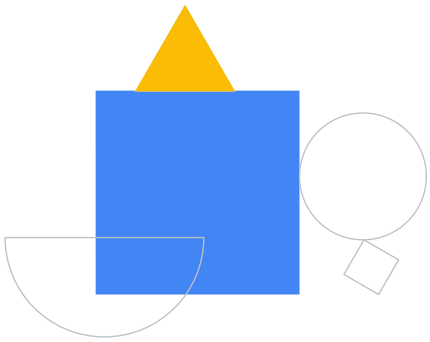


You might be wondering, if there's an error with my Cloud Function my Airflow instance would never have been triggered or issued any logs at all (since it was unaware we were trying to trigger it) and you're exactly right. If you're using Cloud Functions, be sure to check the normal Google Cloud logs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



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

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.