



Google Cloud

Manage Data Pipelines
with Cloud Data Fusion
and Cloud Composer

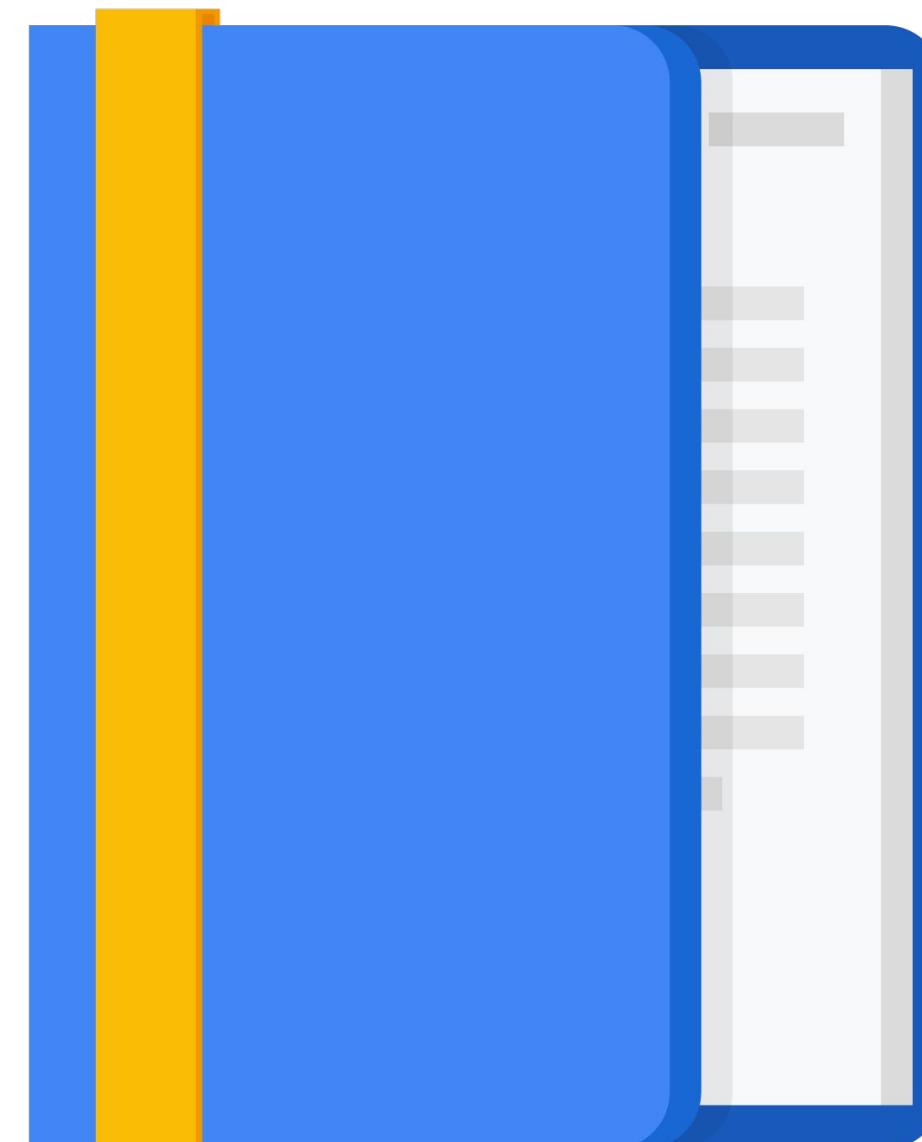
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

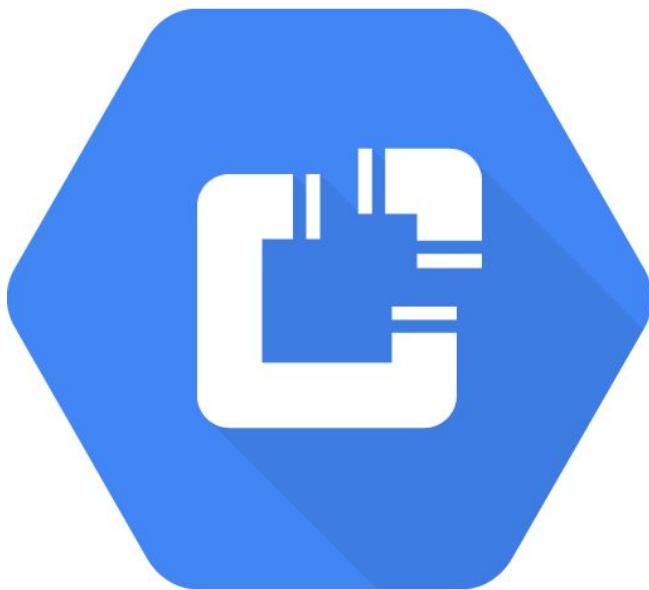
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

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



Cloud Data Fusion



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



Developer, Data Scientist and Business Analyst



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

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

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



Benefits

Integrate with any data

Increase productivity

Reduce complexity

Increase flexibility

The screenshot shows a data preview of a file named "sales_clean.txt" from a "File System". The interface has a blue header bar with the title "Data Fusion | Preparation" and a "DASHBOARD" tab. Below the header is a sidebar with a tree view showing "File System" and "sales_clean.txt". The main area is titled "sales_clean.txt" and contains a table with 11 rows of data. The columns are labeled: Double (price), String (city), String (zip), String (type), String (beds), Double (baths), Integer (size), Long (lot_size), and Integer (stories). The data represents real estate sales information.

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

Build data pipelines with a friendly UI



Rich graphical interface

100+ plugins - connectors, transforms & actions

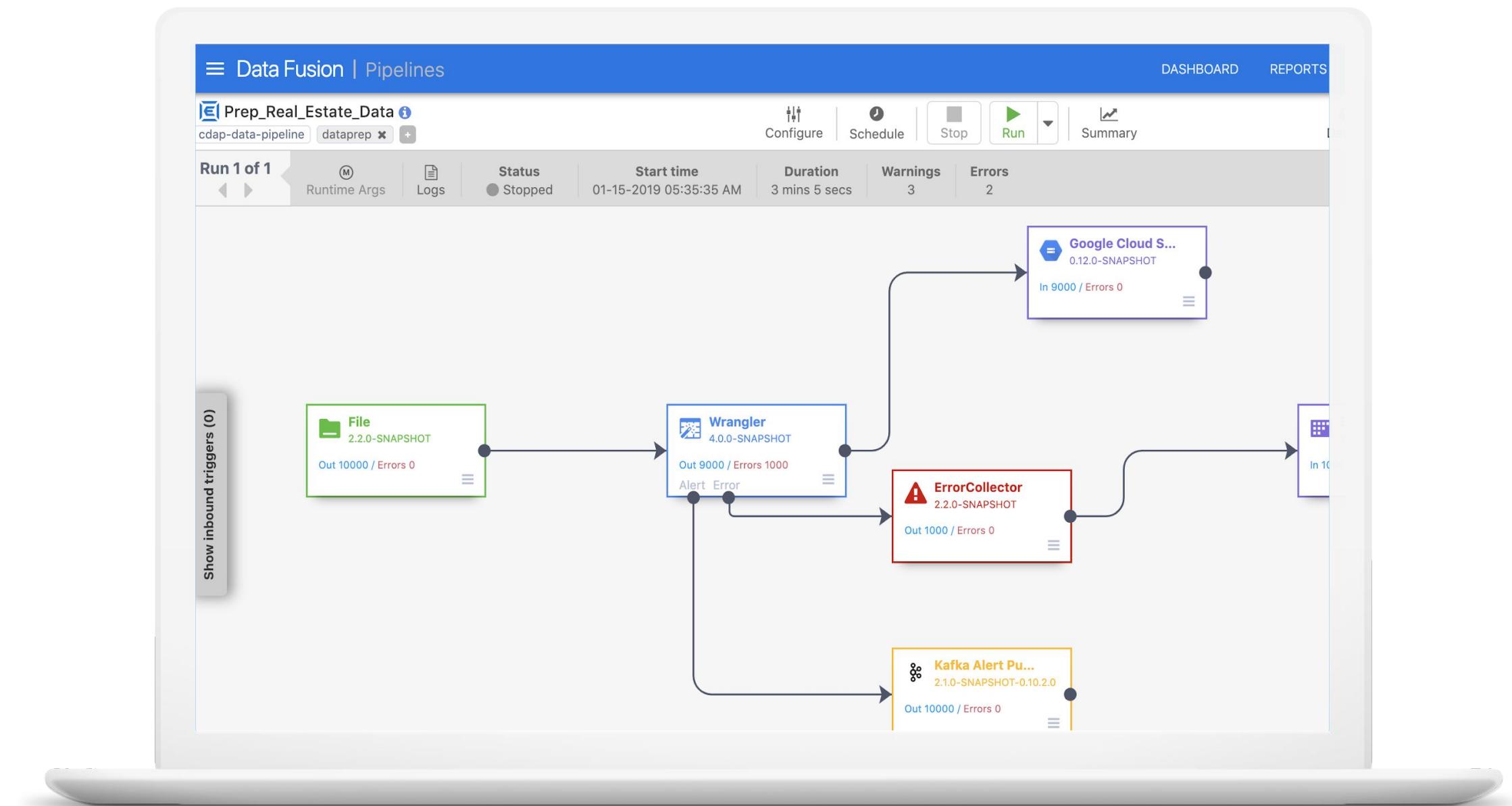
Code free visual transformations

1000+ transforms, data quality

Test and debug pipeline

Pre-built pipelines

Developer SDK



Integration Metadata



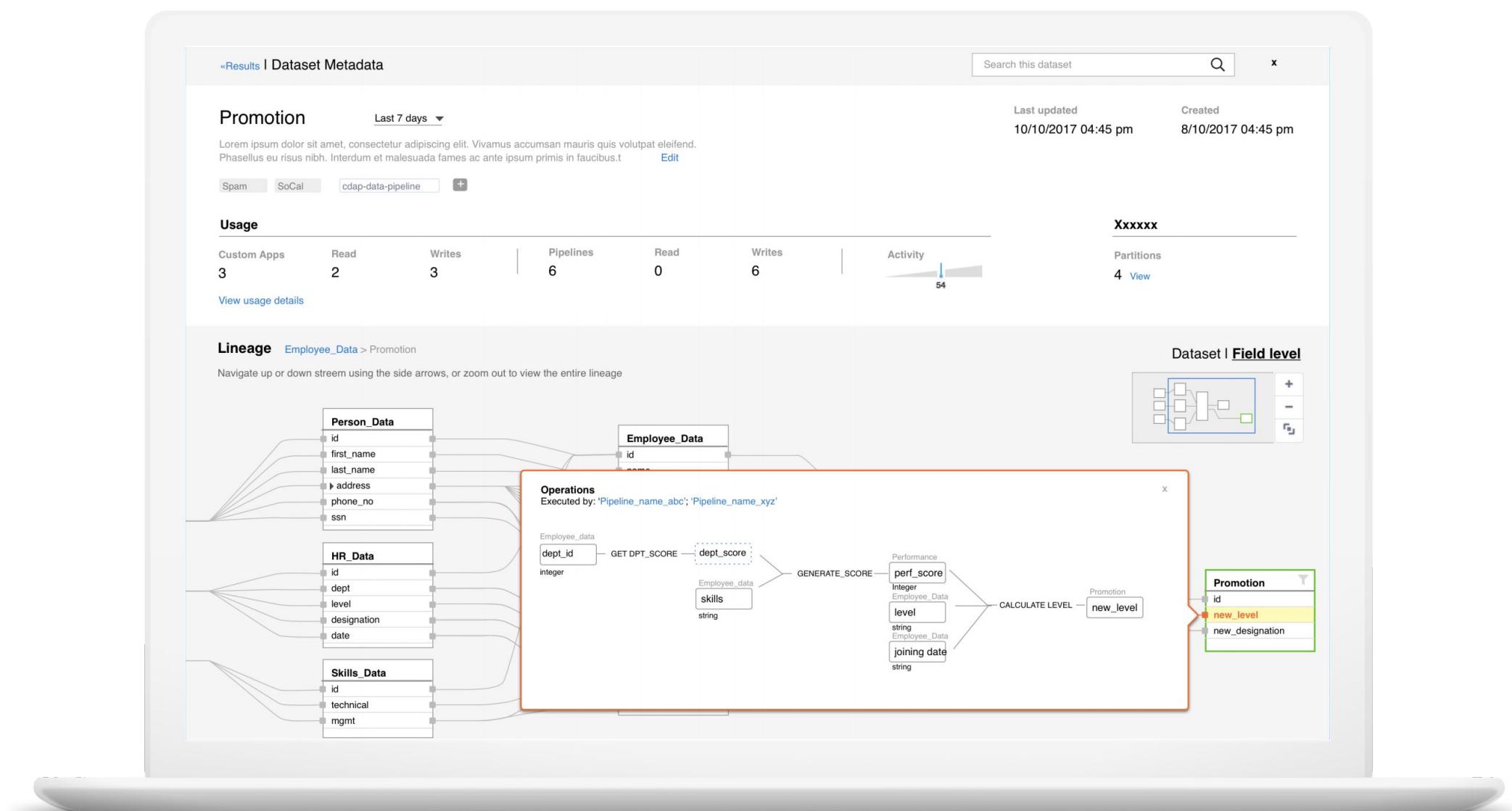
Tags and Properties support

- Pipeline
- Dataset
- Schema

Search integrated entities by

- Keyword
- Schema name and type

Dataset level and Field level Lineage



Extensible



Pipeline templatization

Conditional Pipeline Triggers

Plugin Management

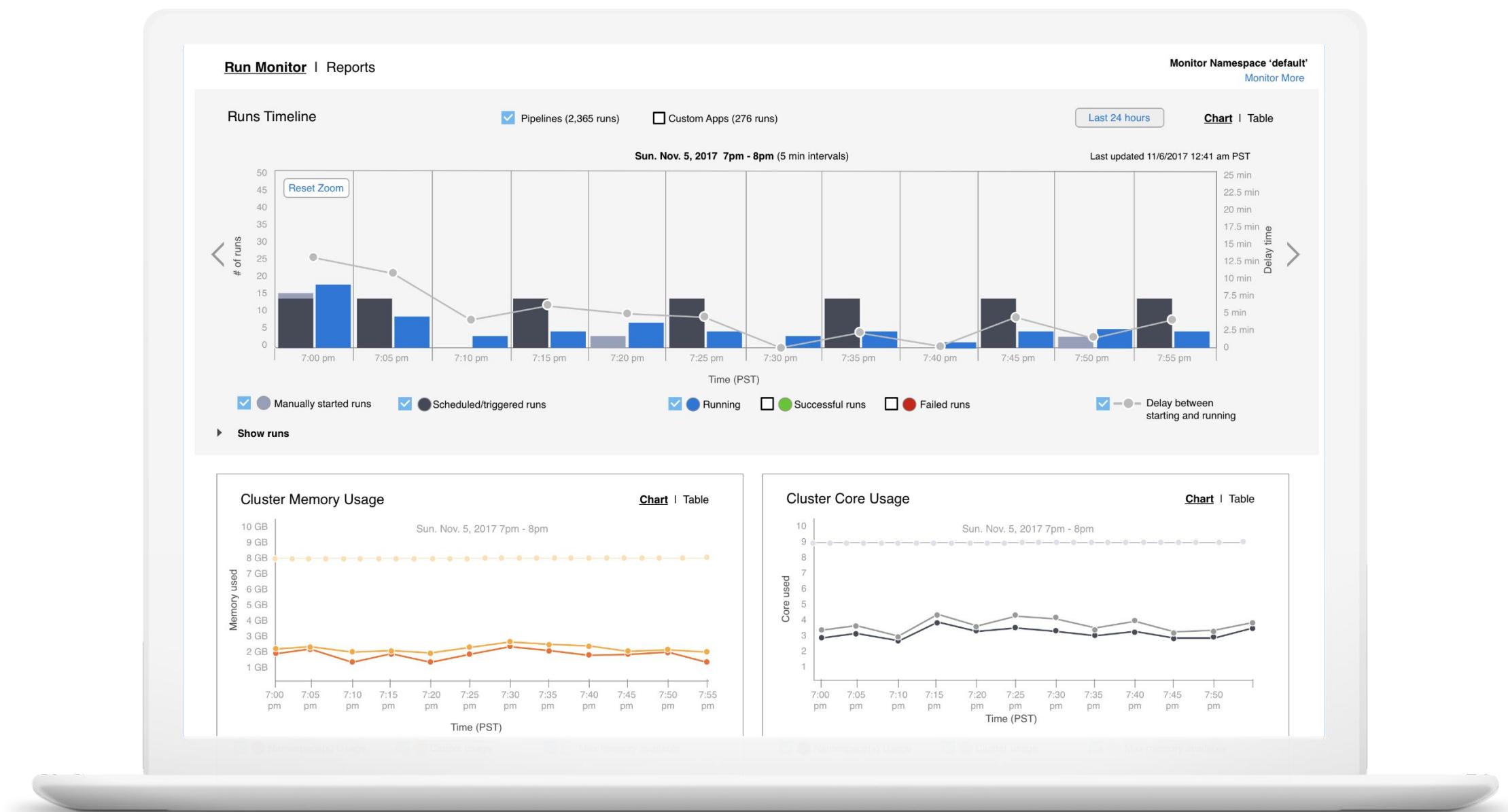
Plugin templatization

Plugin UI Widget

Custom Provisioners

Custom Compute Profiles

Hub Integration



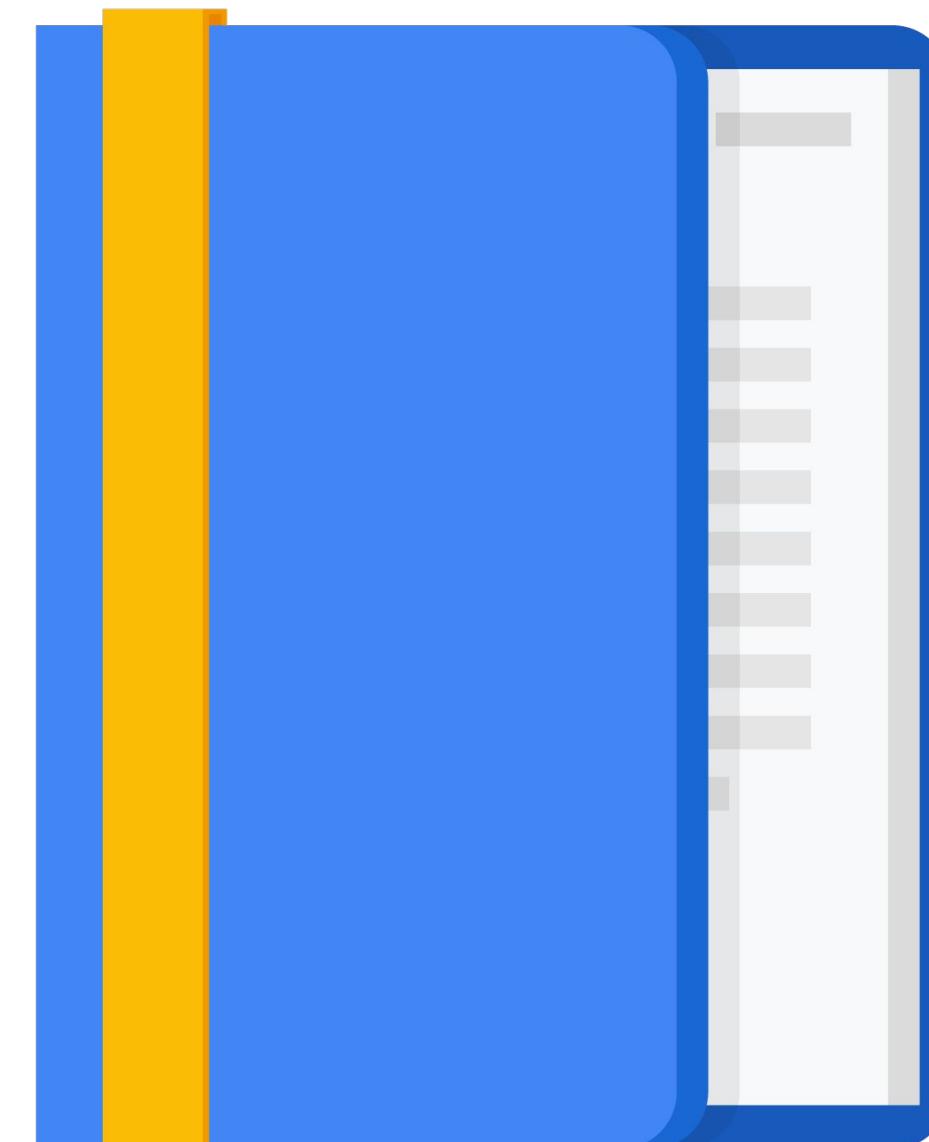
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

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

Orchestrating work between GCP services with Cloud Composer

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

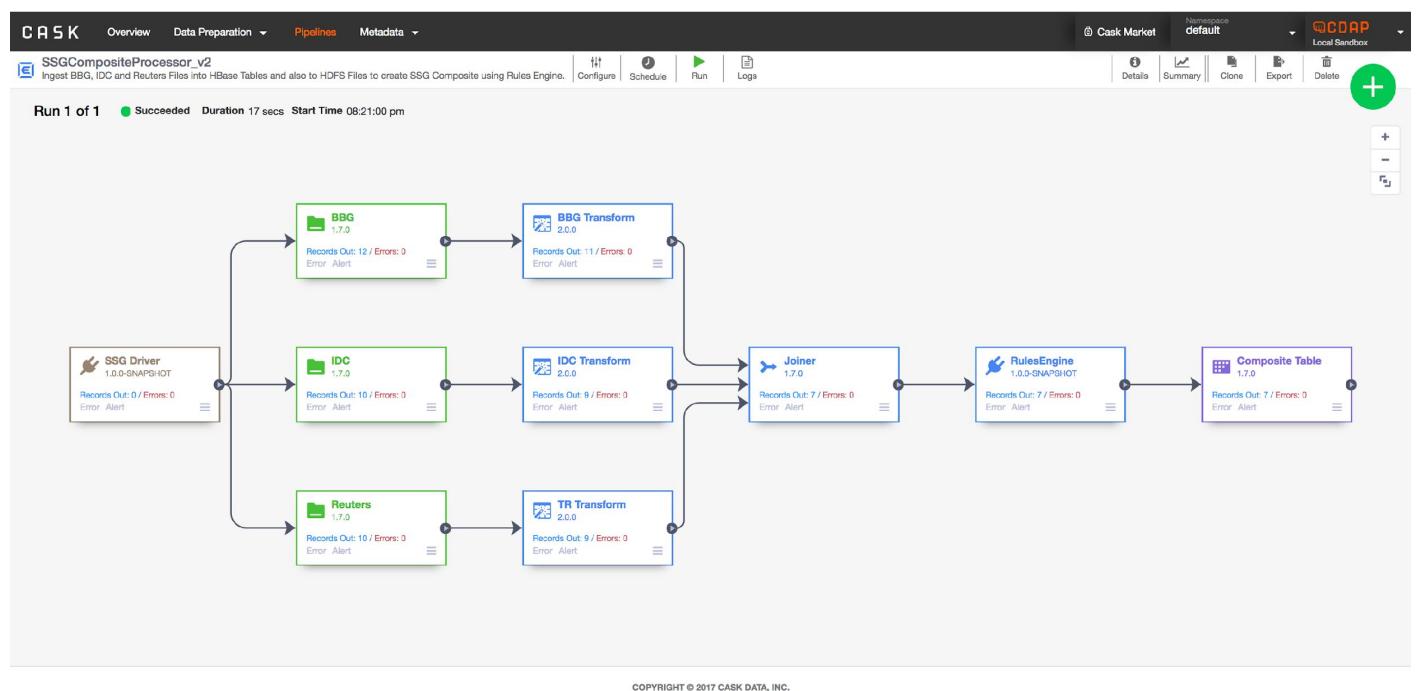




Components of Cloud Data Fusion

Wrangler — Framework

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



Data Pipeline — Framework

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



Components of Cloud Data Fusion

The screenshot shows the Cask Rules Engine interface. At the top, there are tabs for Overview, Data Preparation, Pipelines, and Metadata. Below that, there are sections for Rules Books (5) and Rules (23). A search bar and a 'Create a New Rule' button are present. On the left, a sidebar lists various rule names and their last updated times. The main area displays a 'Client1CompositeRulebook' with 9 rules, each with a description and a 'Remove' link.

The screenshot shows the Cask Metadata Aggregator interface. At the top, there are tabs for rulebook, dataset, summary, lineage, audit log, preview, usage, and compliance. The lineage tab is selected. A search bar and a green '+' button are at the top right. The main area shows a lineage diagram where a 'rulebook' table is connected to a 'yare service' and a 'rules' table. The URL at the bottom is 'localhost:11011/metadata/ns/default/entity/datasets/rulebook/lineage?searchTerm=rulebook'.

Rules Engine — Tool*

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

Metadata Aggregator — Tool

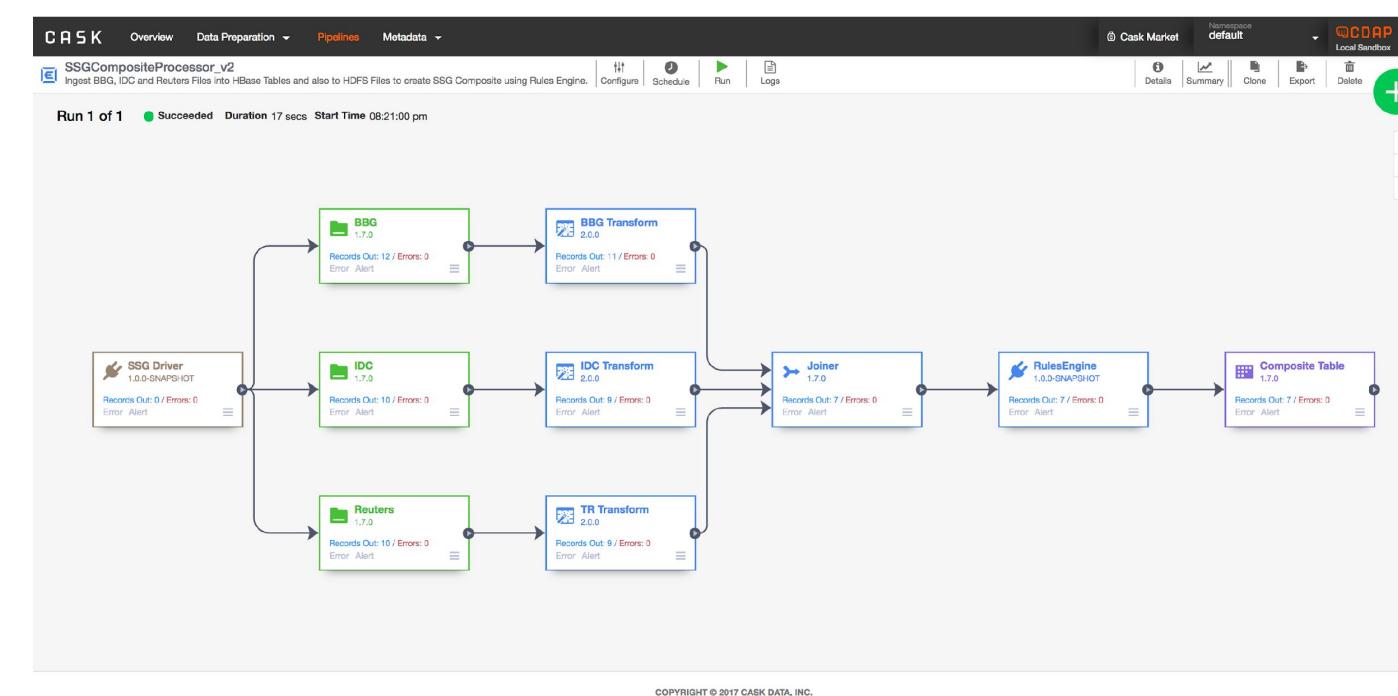
- Aggregate Business, Technical and Operational Metadata
- Track the flow of data (Lineage) for richer data needed for governance
- Create Data Dictionary and Metadata Repository
- Integrate with enterprise MDM solutions.
- Integrates with Data Pipeline, Rules Engine



Components of Cloud Data Fusion

Microservice — Framework*

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



Event Condition Action (ECA) — Application*

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

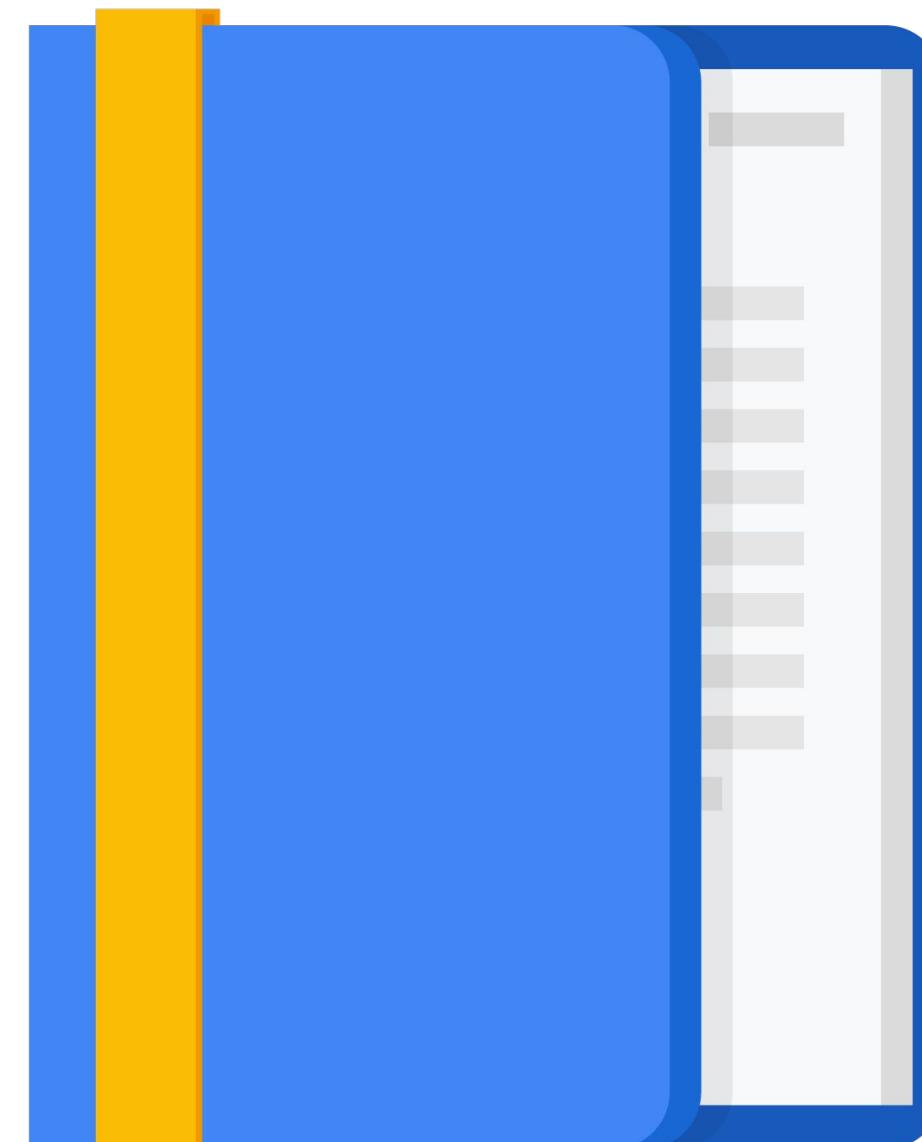
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

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

Orchestrating work between GCP services with Cloud Composer

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





Cloud Data Fusion - UI Overview

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

Cloud Data Fusion | Pipelines

DASHBOARD REPORTS HUB SYSTEM ADMIN

Namespace Admin

Type	Status	Last start time	Next run in	Total runs	Tags	Actions
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 new Spam SoCal 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	



Control Center

- Application
- Artifact
- Dataset

Cloud Data Fusion | Control Center

DASHBOARD REPORTS HUB SYSTEM ADMIN

Entities in namespace "default"

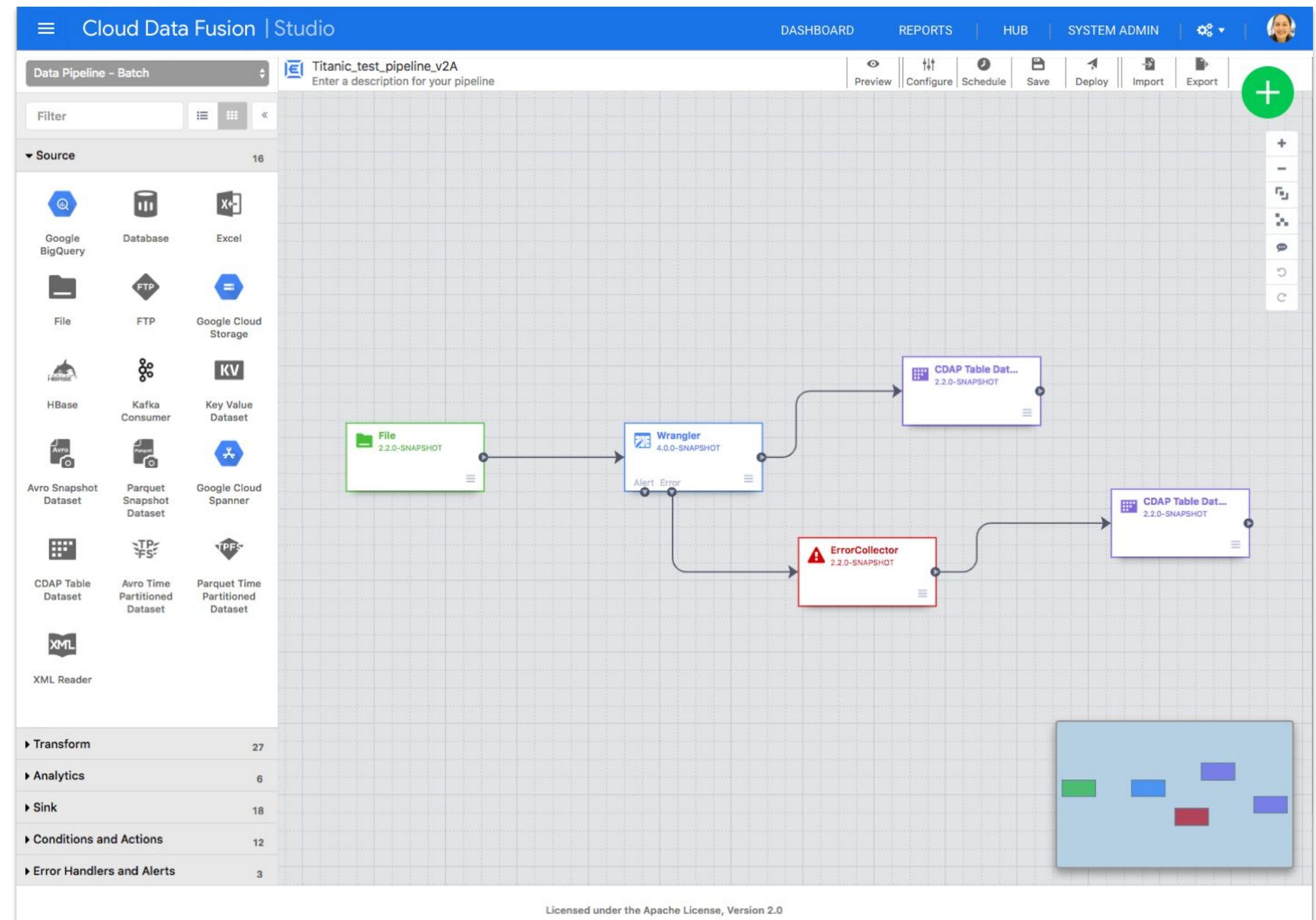
Displaying Applications, Datasets, sorted by Newest

Type	Name	Version	Programs	Running	Failed	Operations	Writes
Data Pipeline	join-w-customer	1.0.0-SNAPSHOT	2	0	0	10	3
Dataset	sales-with-zip		1	10	3	0	0
Dataset	customers.csv		1	0	0	0	0
Dataset	sales.csv		1	0	0	0	0
Data Pipeline	sales-ingest	1.0.0-SNAPSHOT	2	0	0	0	0
Dataset	Mask_data	1.0.0-SNAPSHOT	2	15	10	0	0
Dataset	customers-ingest	1.0.0-SNAPSHOT	2	0	0	12	9
Dataset	customers		2	0	0	0	0
Dataset	U.S._Chronic_Disease_Indicator...		1	0	0	0	0
Data Pipeline	Mask_data	1.0.0-SNAPSHOT	2	0	0	0	0
Dataset	experiment_model_meta		1	0	0	0	0
Dataset	experiment_model_components		1	0	0	0	0
Dataset	experiment_meta		1	1	0	0	0
Dataset	experiment_splits		1	0	0	0	0
Data Pipeline	Titanic_test_pipeline_v1_test_v1	1.0.0-SNAPSHOT	2	0	0	0	0
Dataset	titanic.csv		0	0	0	0	0
Data Pipeline	Titanic_test_pipeline_v1_test	1.0.0-SNAPSHOT	2	0	0	0	0
Dataset	Titanic_test		0	0	0	0	0
Dataset	connections		1	0	0	0	0
Dataset	dataprepfs		1	0	0	0	0
Dataset	workspace		1	2,499	239	0	0
Error_sink_titanic			0	0	0	0	0
Application	ModelManagementApp	1.0.0-SNAPSHOT	1	0	0	0	0
Application	dataprep	1.0.0-SNAPSHOT	1	1	0	0	0



Pipelines

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





Wrangler

- Connections
- Transforms
- Data Quality
- Insights
- Functions

Cloud Data Fusion | Wrangler

DASHBOARD HUB SYSTEM ADMIN Enterprise Edition

titanic.csv Google Cloud Storage titanic.csv

Data Insights

Create a Pipeline More

Columns (16) Transformation steps (20)

Search Column names ▾

	Integer	String	String	String	Integer	String	String	String	Double	String	String
	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	1	0	3	male	22	1	0	A/5 21171	7.25	none	S
2	2	1	1	female	38	1	0	PC 17599	71.2833	C85	C
3	3	1	3	female	26	0	0	STON/O2. 3101282	7.925	none	S
4	4	1	1	female	35	1	0	113803	53.1	C123	S
5	5	0	3	male	35	0	0	373450	8.05	none	S
6	7	0	1	male	54	0	0	17463	51.8625	E46	S
7	8	0	3	male	2	3	1	349909	21.075	none	S
8	9	1	3	female	27	0	2	347742	11.1333	none	S
9	10	1	2	female	14	1	0	237736	30.0708	none	C
10	11	1	3	female	4	1	1	PP 9549	16.7	G6	S
11	12	1	1	female	58	0	0	113783	26.55	C103	S
12	13	0	3	male	20	0	0	A/5. 2151	8.05	none	S
13	14	0	3	male	39	1	5	347082	31.275	none	S
14	15	0	3	female	14	0	0	350406	7.8542	none	S
15	16	1	2	female	55	0	0	248706	16.0	none	S
16	17	0	3	male	2	4	1	382652	29.125	none	Q

\$



Integration Metadata

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

Cloud Data Fusion | Metadata

DASHBOARD REPORTS HUB SYSTEM ADMIN

NAMESPACE: default

Control Center

Pipelines

List

Studio

Transform

Metadata

Sort by Oldest first 1-10 of 18 results

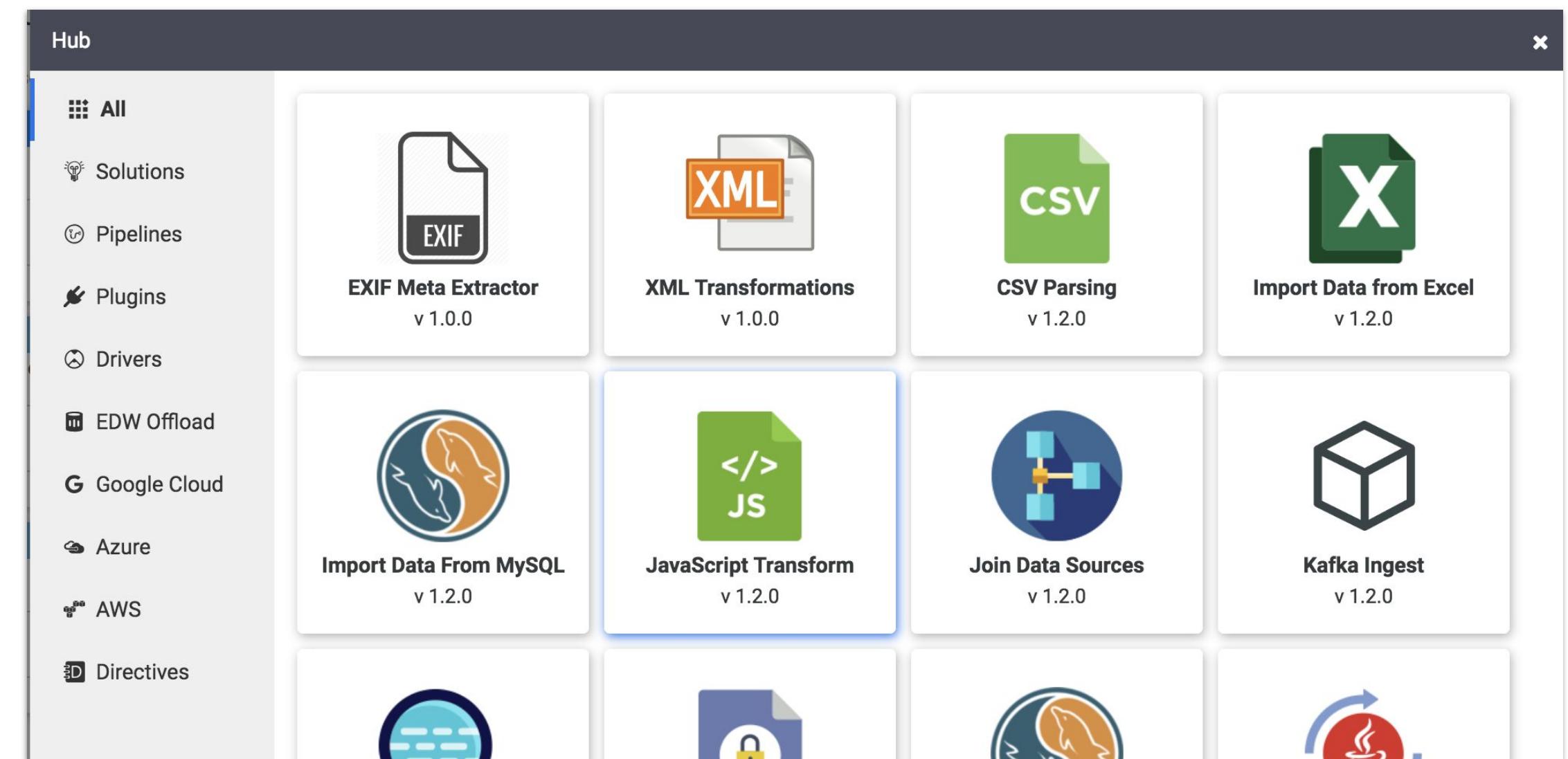
18	recipes	Dataset Created: Oct 12, 2018 Recipe store.
2	workspace	Dataset Created: Oct 12, 2018 Dataprep workspace dataset
18	dataprefs	Dataset Created: Oct 12, 2018 Store Dataprep Index files
1	connections	Dataset Created: Oct 12, 2018 DataPrep connections store.
	Error_sink_titanic	Dataset Created: Oct 17, 2018 No description provided for this Dataset.
	Titanic_test	Dataset Created: Oct 17, 2018 No description provided for this Dataset.
	titanic.csv	Dataset Created: Oct 17, 2018 No description provided for this Dataset.

Namespace Admin



Hub

- Plugins
- Use cases
- Pre-built pipelines





Entities



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

Add entity

 Pipeline A pipeline allows you to create, manage, and operate complex batch and real-time workflows intuitively. Create Import	 Application An application is a collection of datasets and programs that read and write data to datasets. Upload	 Plugin A plugin is an easy way to extend the functionality of an application. Upload
 Driver A driver is a JAR file that contains third-party code to communicate with systems such as MySQL, Oracle, and PostgreSQL using JDBC. Upload	 Library A library is a JAR file that can contain reusable third-party code (e.g. external Spark programs). Upload	 Directive A directive is a data manipulation instruction that can be used to perform data cleansing, transformation and filtering. Upload



Administration

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

Cloud Data Fusion | Administration

Management | Configuration

Uptime 3 hours 5 mins 32 secs Version - 6.0.0-SNAPSHOT

Services

Status	Name ▾	Provisioned	Requested	Action
Green	App Fabric	1	1	View Logs
Green	Dataset Executor	1	1	View Logs
Green	Log Saver	1	1	View Logs
Green	Messaging Service	1	1	View Logs
Green	Metadata Service	1	1	View Logs
Green	Metrics	1	1	View Logs
Green	Metrics Processor	1	1	View Logs
Green	Transaction	1	1	View Logs

System metrics

Entities	Last hour load
Datasets	0
Programs	0
Namespaces	1
Artifacts	36
Applications	0
ClientErrors	2
ServerError	0
ErrorLogs	2
WarnLogs	35
TotalRequests	192,059
Successful	192,057

Transactions

NumCommittingChangeSets	0
NumInProgressTransactions	0
NumInvalidTransactions	0

Cloud Data Fusion | Administration

Management | Configuration

Reload System Artifacts Make HTTP Calls

▶ Namespaces (1) Create, view, and manage namespaces

▼ System Compute Profiles (1) Manage compute profiles available to launch programs in all namespaces

Create New Profile Import

Default	Profile name ▾	Provisioner	Scope	Last 24 hrs runs	Last 24 hrs node hours	Total node hours	Schedules	Triggers	Status
★	Dataproc	Google Cloud Dataproc	SYSTEM	--	--	--	0	0	Enabled

Pipeline usage

▶ System Preferences (1) Manage system preferences available as runtime arguments to programs across all namespaces

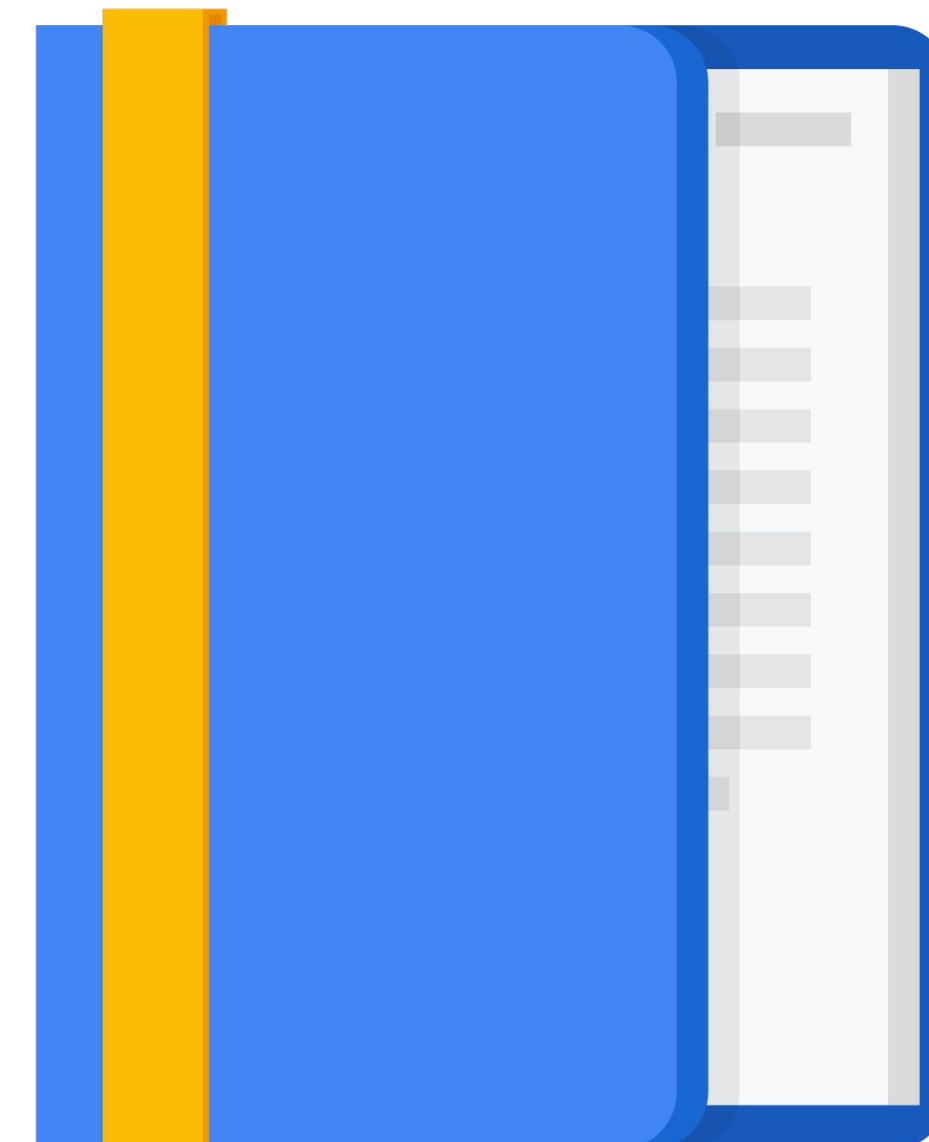
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

- Components
- UI Overview
- [Building a Pipeline](#)
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

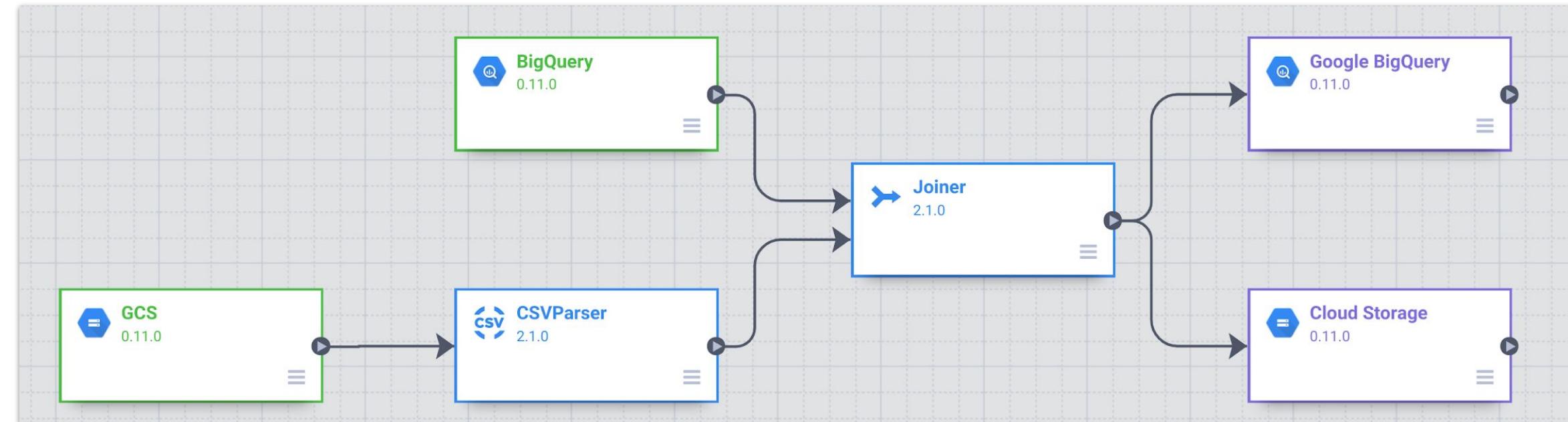
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



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

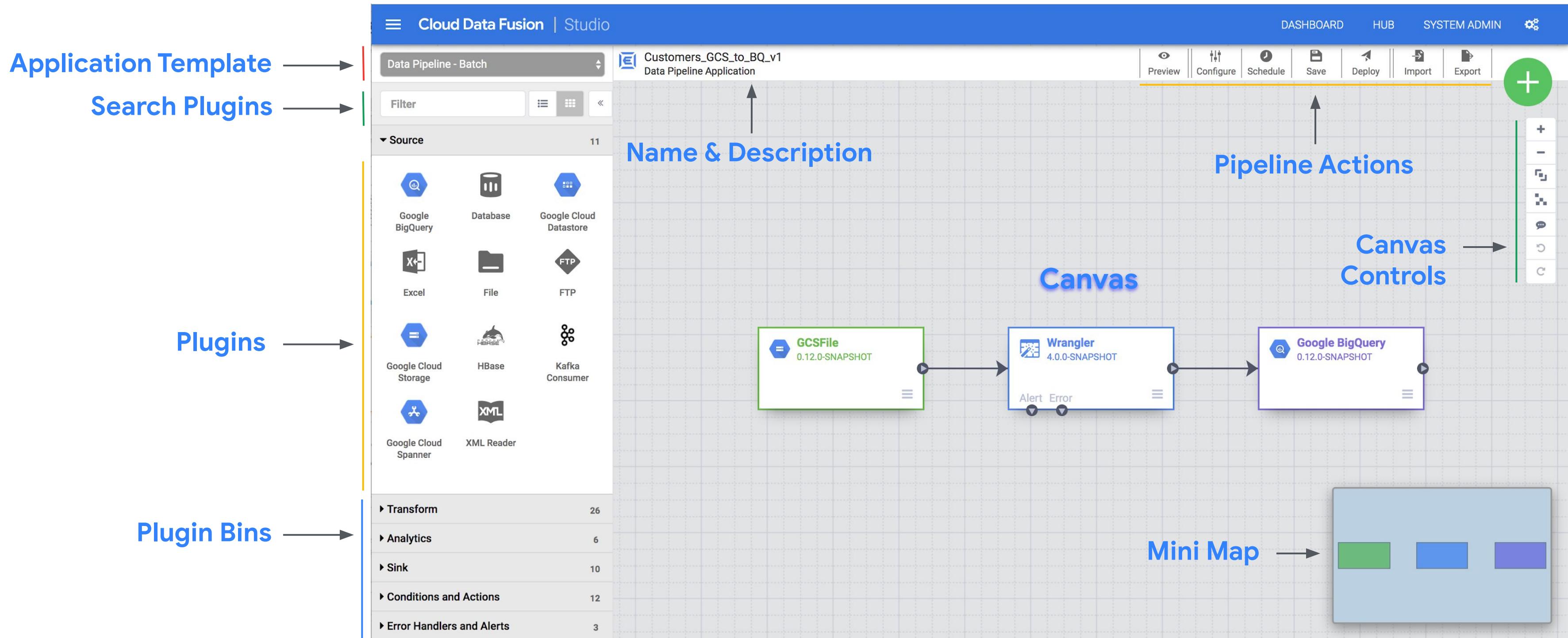




Data Pipeline

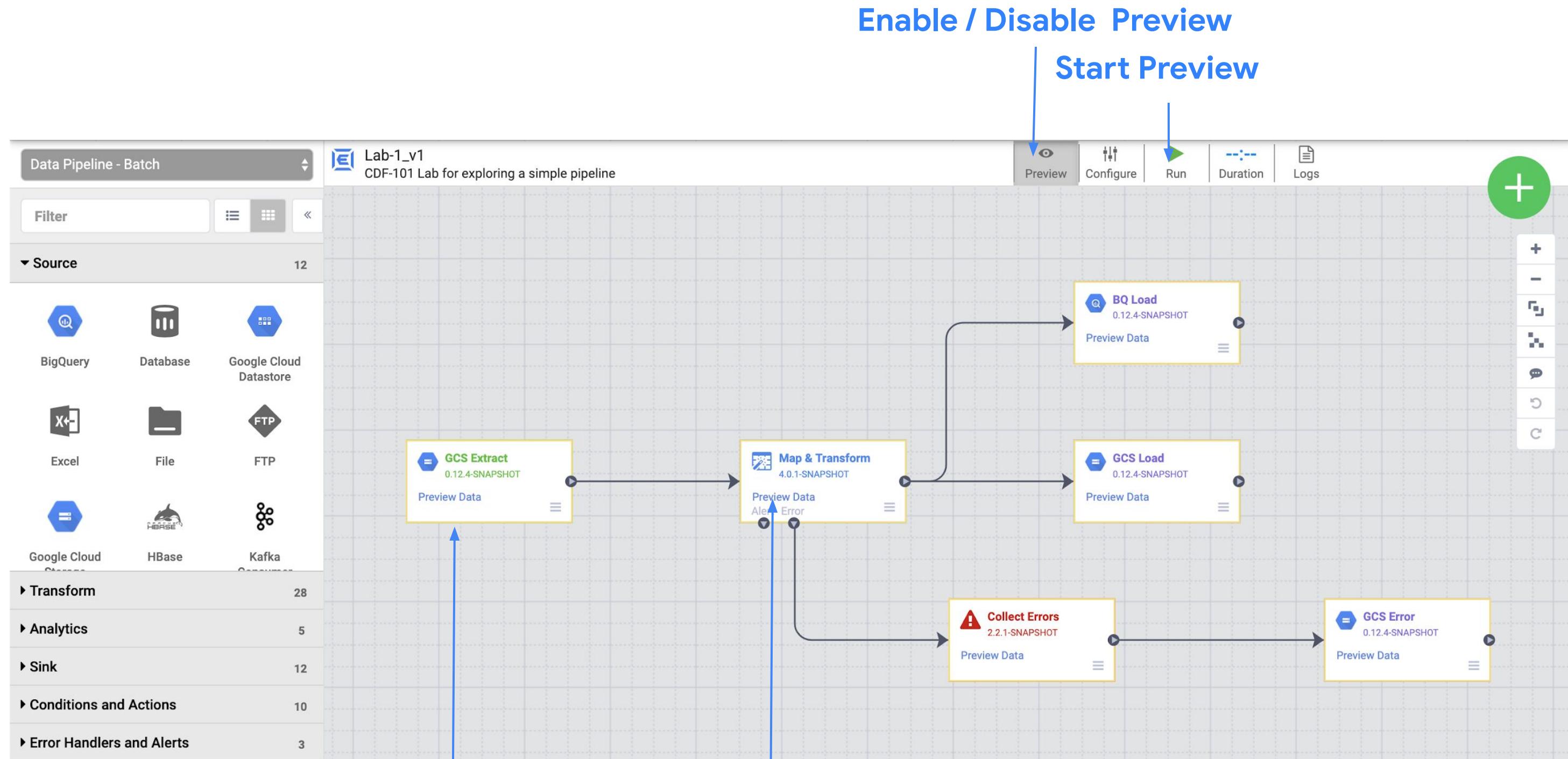
- Allows non-linear pipelines.
- Can fork from a node, where output from a node can be sent to two or more stages.
- Two or more forked nodes can merge at a transform or a sink node.

Studio is the UI where you create new pipelines





Use Preview Mode to see how the pipeline will run

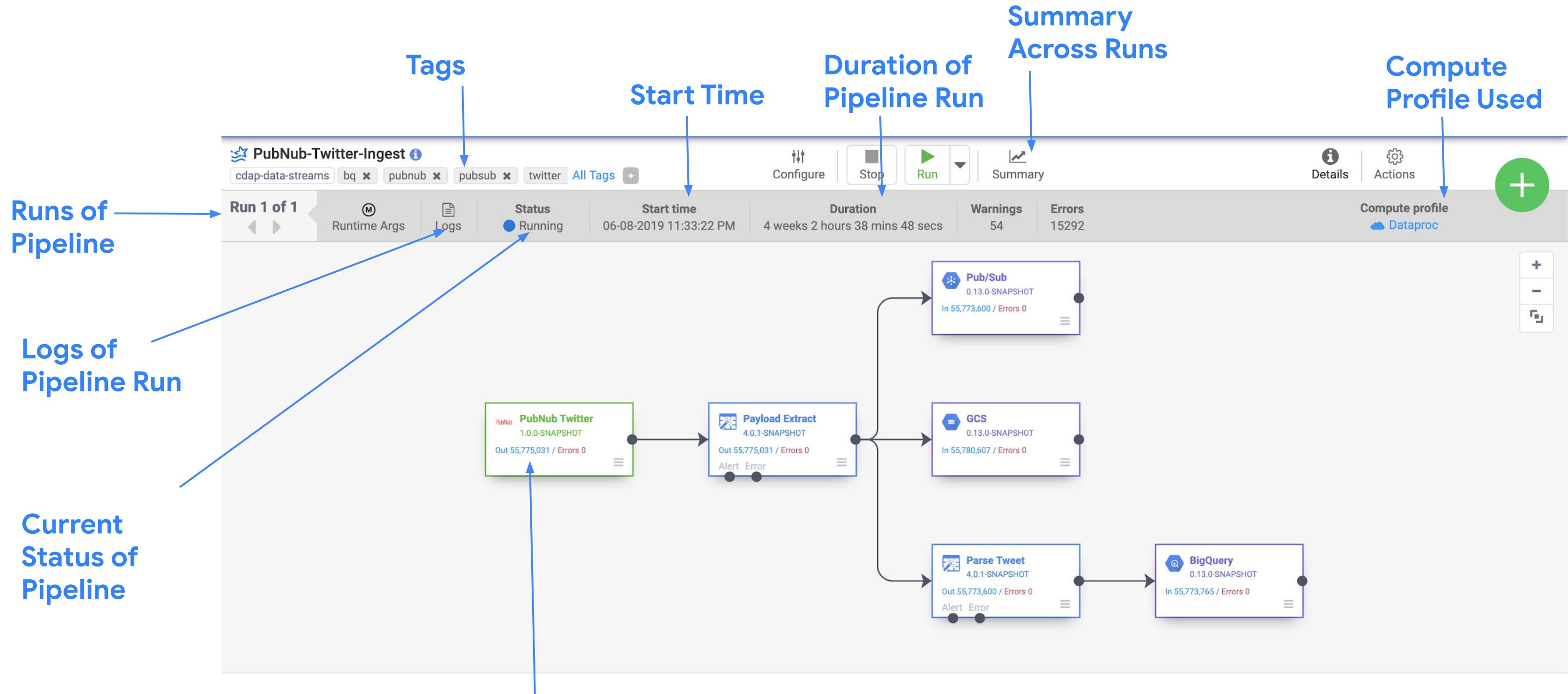


Nodes
highlighted -
Yellow glow

View Sample
Data



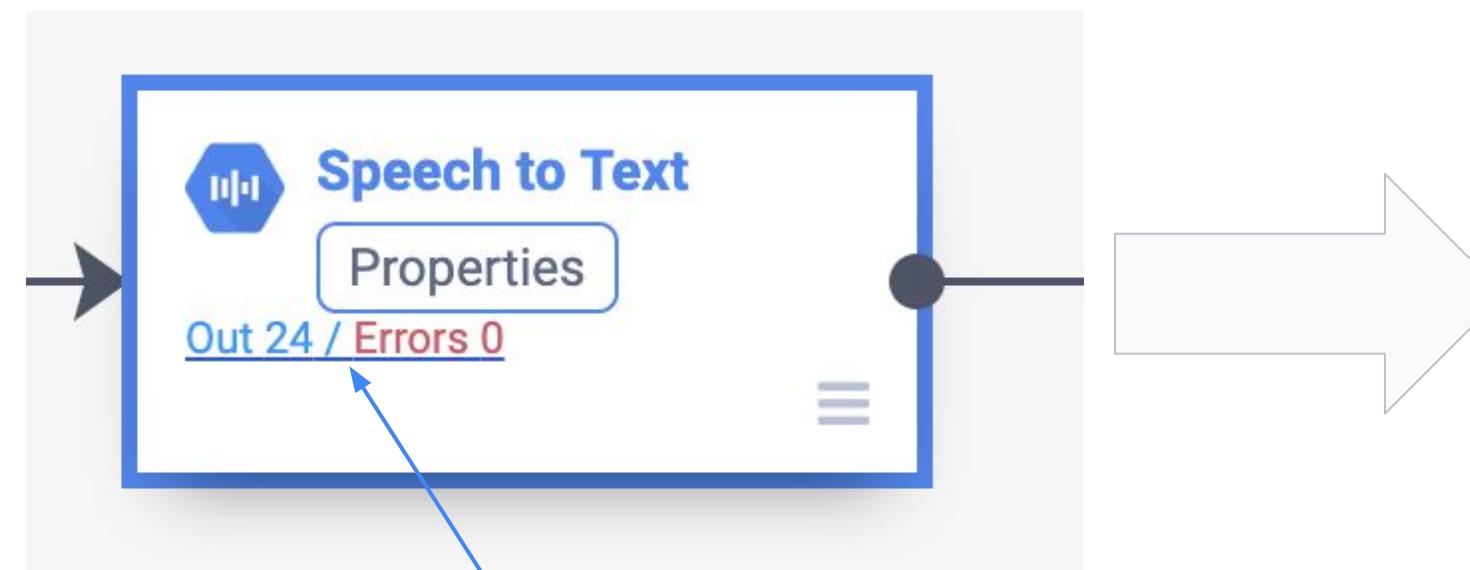
Monitor the health of the entire pipeline



Metrics at
each node



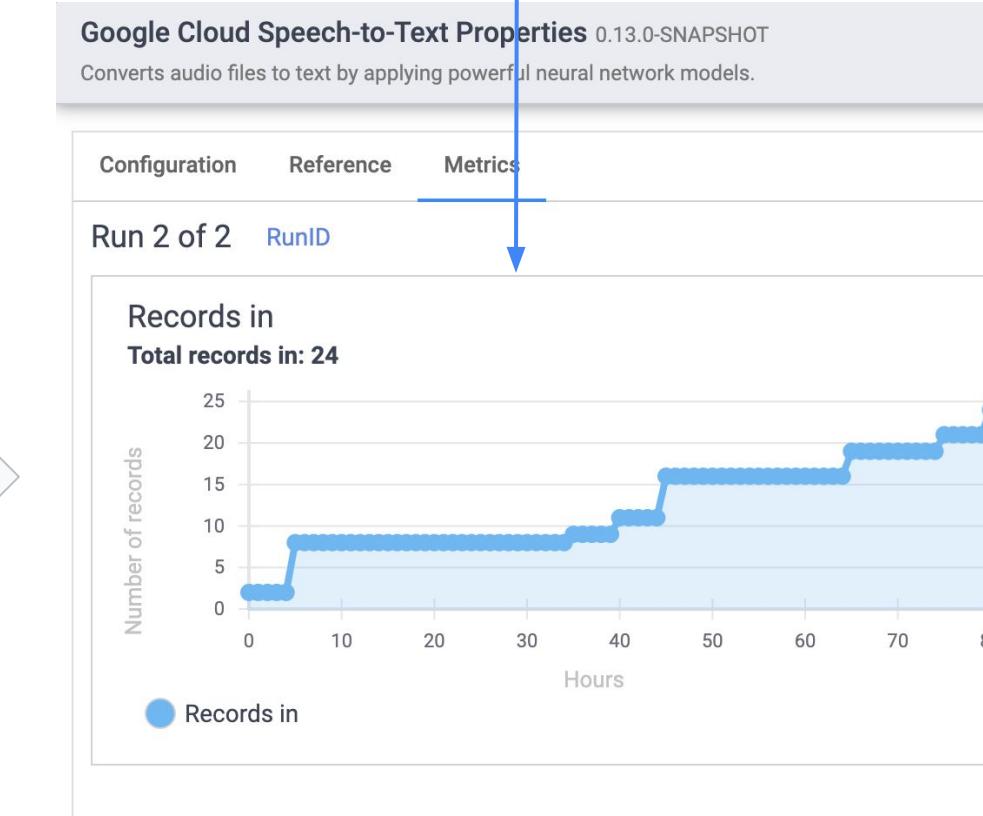
Monitor the health of a single node



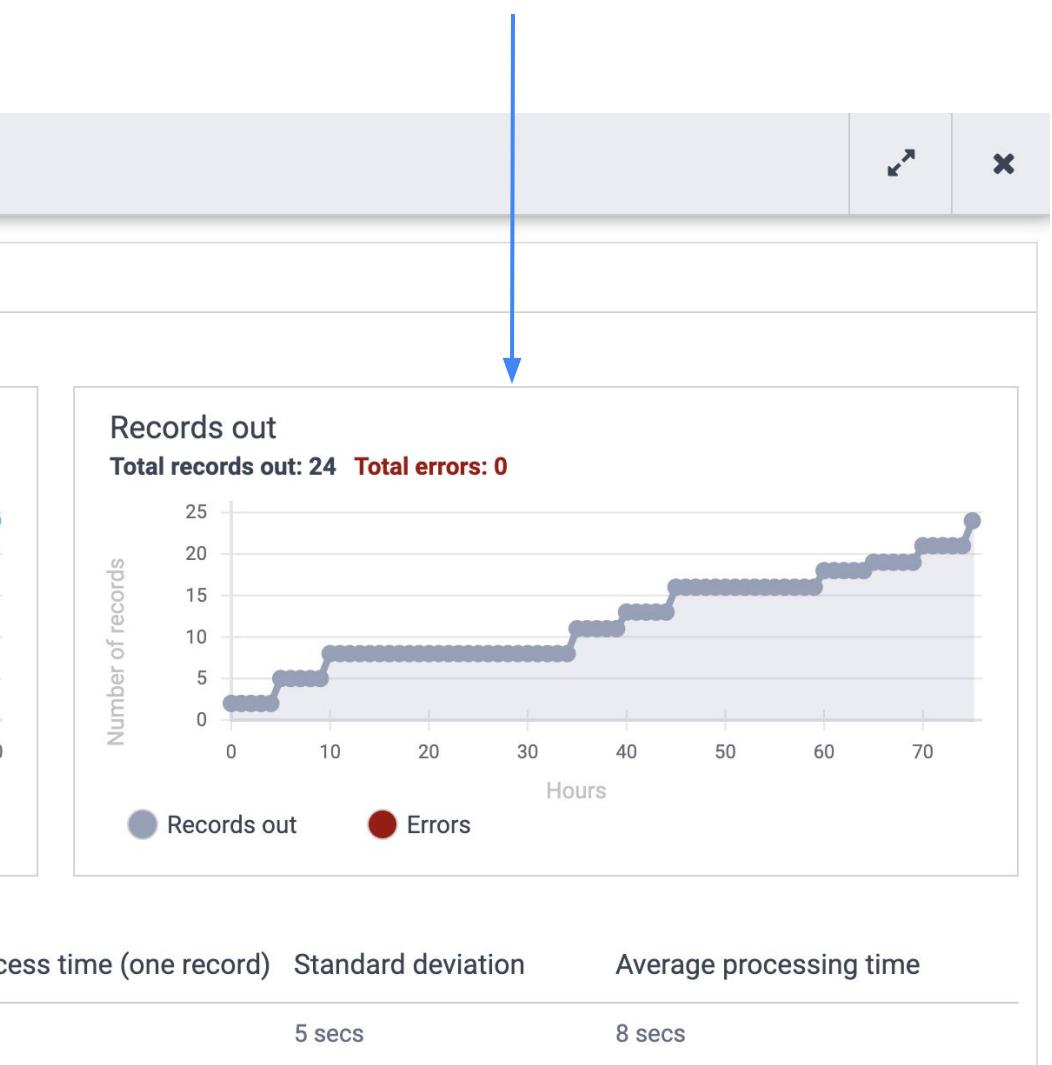
Click to see each node detail



Incoming records



Outgoing records



Processing Statistics at each node

You can schedule batch pipelines



Cloud Data Fusion | Pipeline

batch_pipeline 1
No tags found. Click to add a new business tag. +

Configure Schedule Stop Run Summary Details Actions

No Runs Runtime Args Logs Status Deploy

Configure schedule for pipeline batch_pipeline

Basic | Advanced

Pipeline run repeats: Daily

Repeats every: 1 day(s)

Starting at: 1 : 0 AM

Summary: This pipeline is scheduled to run everyday, at 1:00AM.
The pipeline cannot have concurrent runs.

Max concurrent runs: 1

Compute profiles: Dataproc (Google Cloud Dataproc)

Show inbound triggers (0) Show outbound triggers (0)

Save and Start Schedule Save Schedule

A screenshot of the Cloud Data Fusion interface showing the configuration of a batch pipeline named "batch_pipeline". The pipeline has no runs. The "Schedule" tab is active, displaying a modal window titled "Configure schedule for pipeline batch_pipeline". In the "Basic" tab, the pipeline is set to run daily at 1:00 AM. The summary notes that the pipeline cannot have concurrent runs. There is one concurrent run profile assigned. Buttons at the bottom allow saving and starting the schedule.

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



Relationship between fields of datasets: Provenance, Impact

☰ Cloud Data Fusion | Field Level Lineage DASHBOARD HUB SYSTEM ADMIN ⚙️ Enterprise Edition

« Back | doubleclick
Dataset

Field level lineage

Explore root cause and impact for each of the fields of the dataset

View field level info in the Last 7 days

Cause for: doubleclick: campaign

1 Dataset

Dataset name	Field name
1 campaign	offset
	body

Dataset: doubleclick

8 Fields

Search by field name

Field name
advertiser_id
timestamp
campaign
referrer_url
campaign_id
landing_page_url
advertiser
landing_page_url_id

Impact for: doubleclick: campaign

1 Dataset

Dataset name	Field name
1 hits	campaign

The diagram illustrates the lineage flow. It shows three main components: 'Cause for: doubleclick: campaign', 'Dataset: doubleclick', and 'Impact for: doubleclick: campaign'. In the 'Dataset: doubleclick' section, the 'campaign' field is highlighted in yellow and labeled 'Incoming operations'. In the 'Impact for: doubleclick: campaign' section, the 'campaign' field is also highlighted in yellow and labeled 'Outgoing operations'. Arrows point from the 'campaign' field in the dataset to the 'campaign' field in the impact section, indicating the flow of lineage from the source dataset field to the target dataset field.



See every operation that is made on a field

Operations applied to a field

☰ Cloud Data Fusion | Field Level Lineage DASHBOARD HUB SYSTEM ADMIN ⚙️ Enterprise Edition

« Back | doubleclick Dataset

Field level lineage

Cause operations for field 'campaign' ×

Operations between 'campaign' and 'doubleclick'

◀ 1 of 1 ▶

Last executed by '00-BigQuery_SQL_DoubleClick_v1' on 03-27-2019 09:48:28 AM

	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	parse-as-csv :body ', true; drop :body;	advertiser_id, campaign_id, campaign	--
3	-	[campaign]	Joiner3.Iden... parse campaigns.c...	Unchanged as part of a join	campaign	--

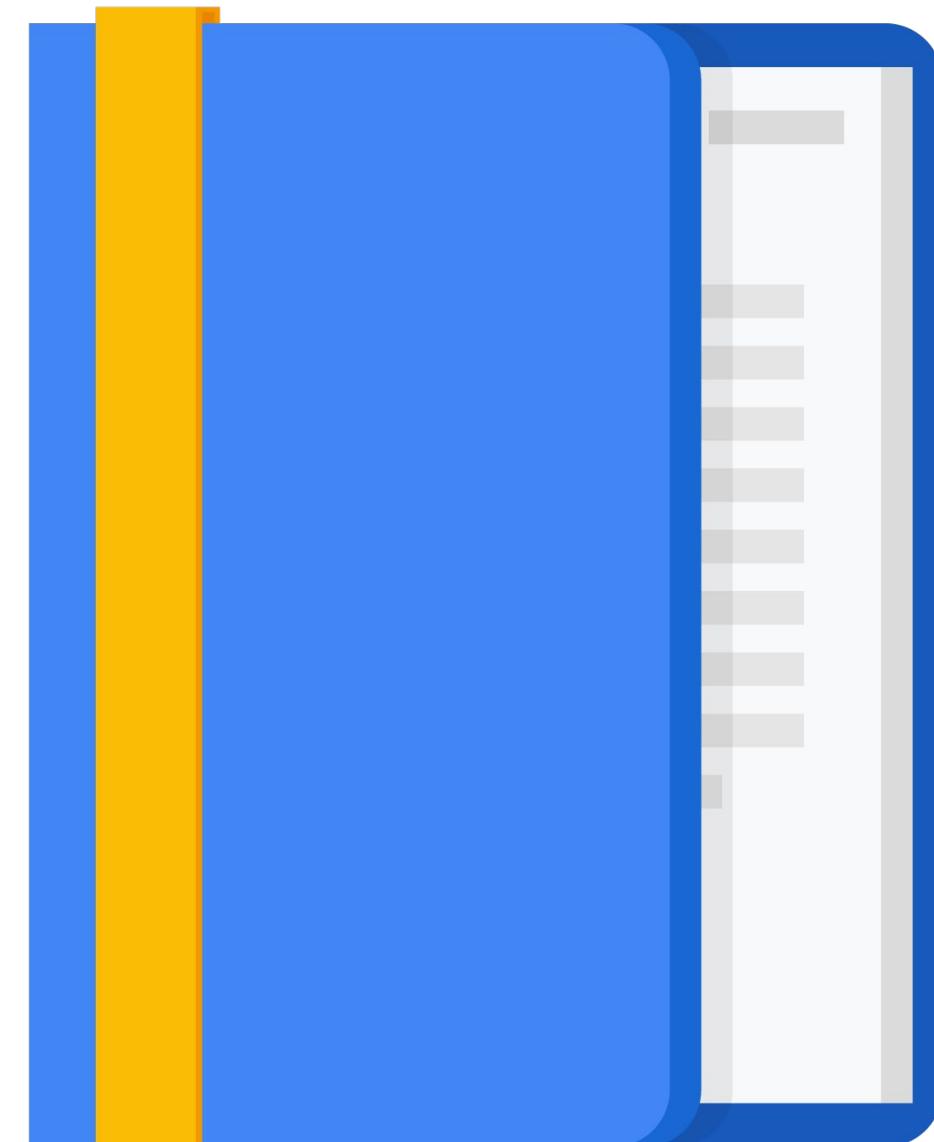
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

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

Orchestrating work between GCP services with Cloud Composer

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





Data analysts can explore datasets in the Wrangler

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

The screenshot shows the Google Cloud Wrangler interface. On the left, the sidebar includes options for Namespace (default), Control Center, Pipeline, List, Studio, Wrangler (which is selected), and Metadata. The main area displays a data pipeline titled "credit_card_data.csv" being processed from "Google Cloud Storage". The pipeline consists of three stages: "credit_card_data.csv" (Input), "customers.csv" (Transformation), and "customers.csv" (Output). The transformation stage shows a preview of the data with columns: id (String), first_name (String), last_name (String), email (String), credit_card (String), and credit_card_type (String). The data preview shows 8 rows of sample data. To the right of the preview is a "Directives" section containing the following JSON:

```
# Directives
1 parse-as-csv :body "true"
2 drop body
3 send-to-error !dq:isCreditCard(credit_card)
4 filter-rows-on regex-match credit_card_type "visa-electron"
```

At the bottom right, there is a note: "Instance Id: cloud-data-fusion-demos/cdf-demo".



Wrangler UI Overview for exploring datasets

The screenshot illustrates the Wrangler UI interface with several key components and associated annotations:

- Connection**: A sidebar on the left containing a list of available connections and a section for adding new ones.
- Add New Connection Type**: An annotation pointing to the "Add Connection" button in the sidebar.
- Search**: An annotation pointing to the search bar in the "Select data" panel.
- Live Browsing of Connection**: An annotation pointing to the main data preview area.
- Workspace**: An annotation pointing to the workspace tab.
- Power Mode**: An annotation pointing to the bottom-most row of the data preview.
- Sample Insights**: An annotation pointing to the insights tab.
- Turn all transformation into Pipeline**: An annotation pointing to the "Create a Pipeline" button.
- Add Directive**: An annotation pointing to the green "+" button for adding directives.
- Recipe**: An annotation pointing to the "Directives" tab.
- Data Quality**: An annotation pointing to the quality report table.
- Schema**: An annotation pointing to the schema definition table.

Connection Sidebar:

- « Connections in "default"
- Upload
- Database (2):
 - Cloud SQL MySQL TheSQL
 - Cloud SQL Postgresql
- Kafka (0):
 - Select a source to connect
 - Database
 - Kafka
 - S3
 - Google Cloud Storage
 - Google BigQuery
 - Google Cloud Spanner

Main Workspace:

- Select data
- Root / demo-datasets / csv 1 directories and 35 fi... Search
- Name
- credit_card_data.csv x customers.csv x
- Google Cloud Storage Data Insights
- customers.csv
- credit_card_data.csv
- customers.csv
- String String String String
- body_1 body_2 body_3 body_4
- 1 id first_name last_name email
- 2 1 Joyce Taylor jtaylor0@bbc.co.uk
- 3 2 Katherine Wallace kwallace1@aristoteles.com
- 4 3 Sandra McDonald smcdonald2@texas.gov
- 5 4 Henry Crawford hcrawford3@upwork.com
- 6 5 Lawrence Lane llane4@mapquest.com
- 7 6 Jean Garza jgarza5@usatoday.com
- 8 7 Gerald Austin gaustin6@techcrunch.com
- 9 8 Lawrence Adams ladams7@scribd.com

Bottom Panel:

- Columns Directives
- Column names
- # Name Completion
- 1 body_1 100%
- 2 body_2 100%
- 3 body_3 100%
- 4 body_4 100%
- 5 Gender 100%
- 6 body_6 100%
- 7 body_7 100%
- 8 body_8 100%
- 9 body_9 100%

Google Cloud Logo:



Building and executing a pipeline graph in Cloud Data Fusion

Objectives

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

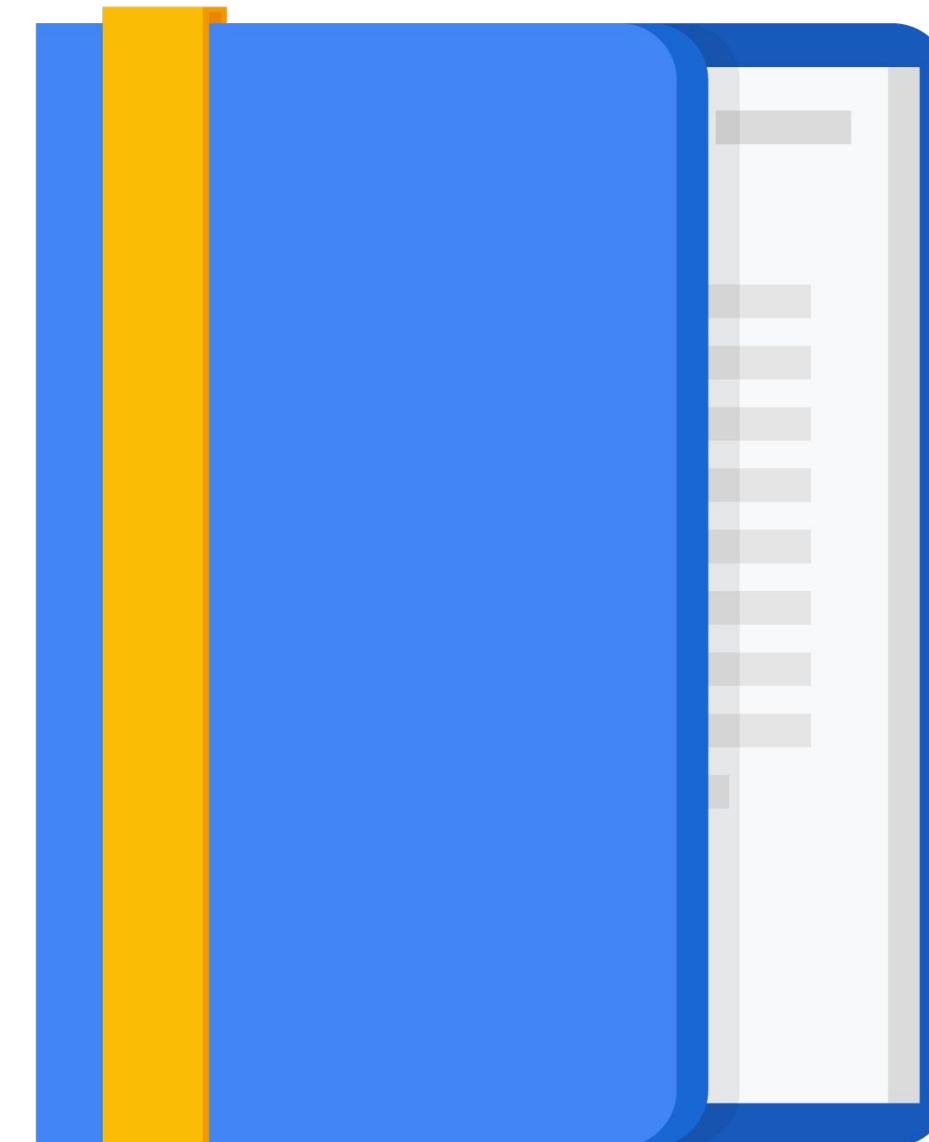
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

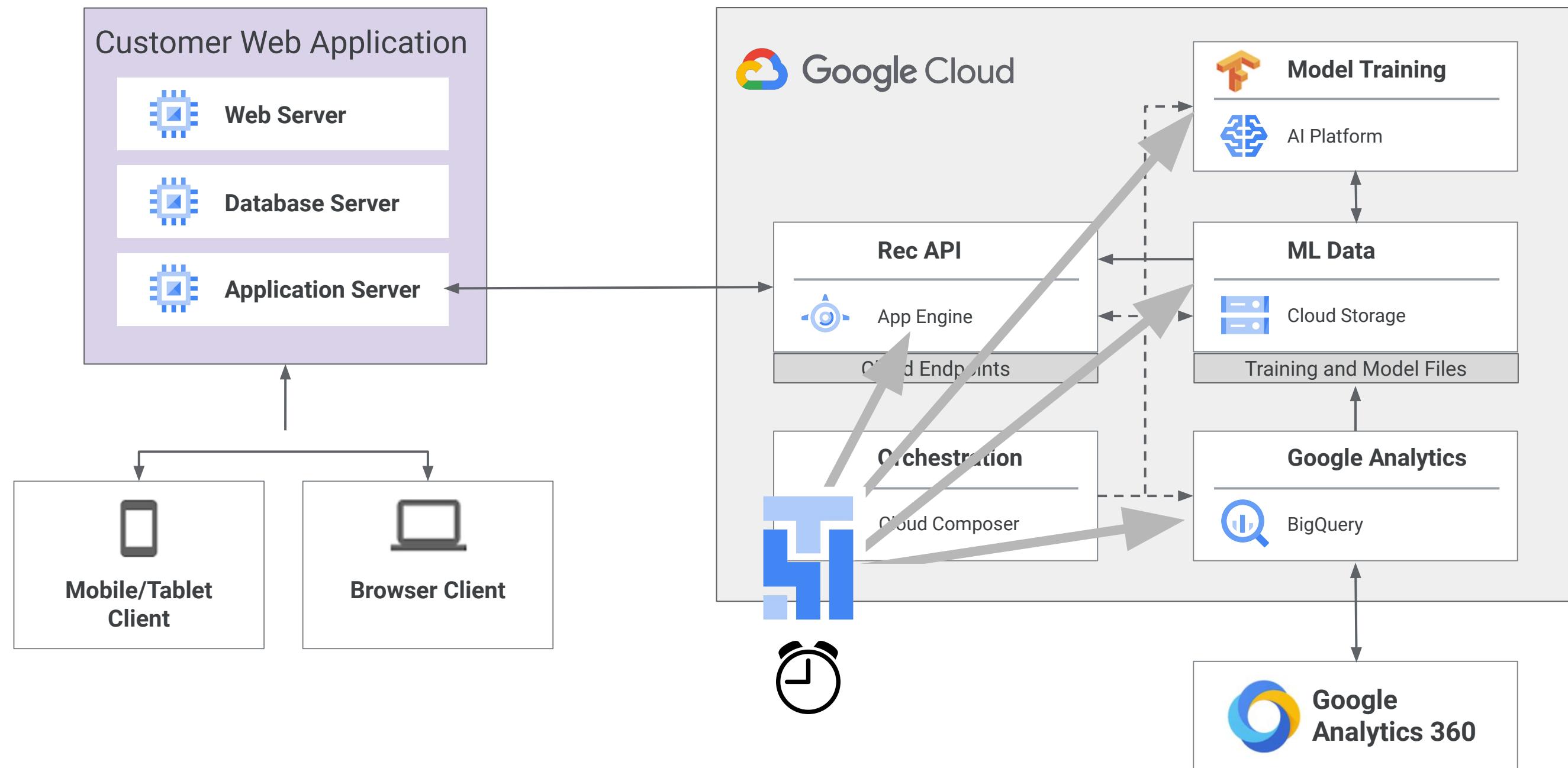
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

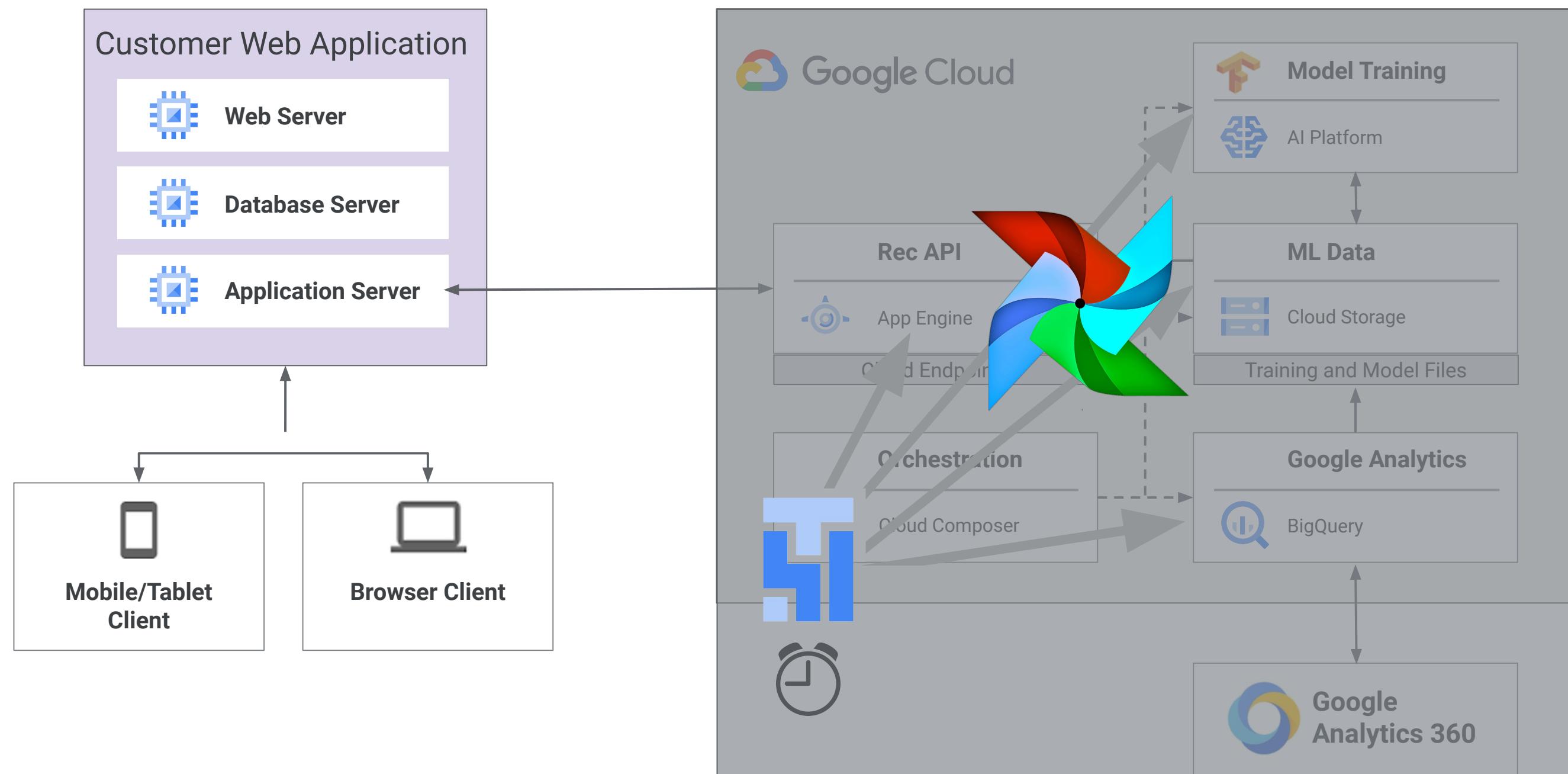
- Apache Airflow Environment
- DAGs and Operators
- Workflow Scheduling
- Monitoring and Logging



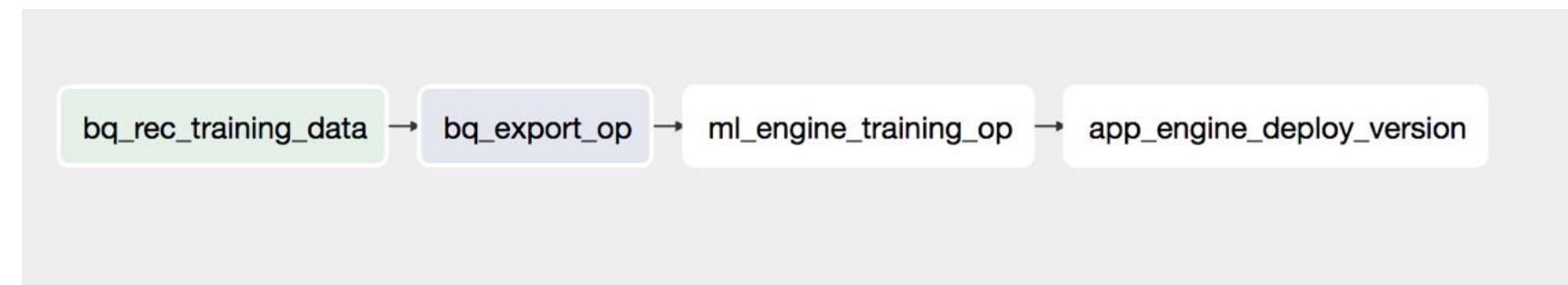
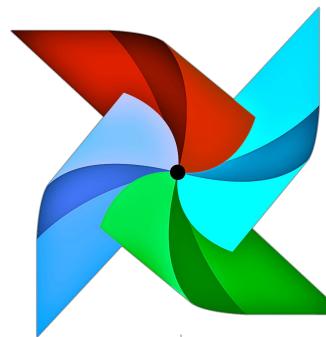
Cloud Composer orchestrates automatic workflows



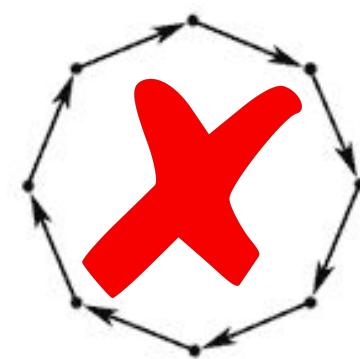
Cloud Composer is managed Apache Airflow



Use Apache Airflow DAGs to orchestrate GCP services



DAG = Directed Acyclic Graph



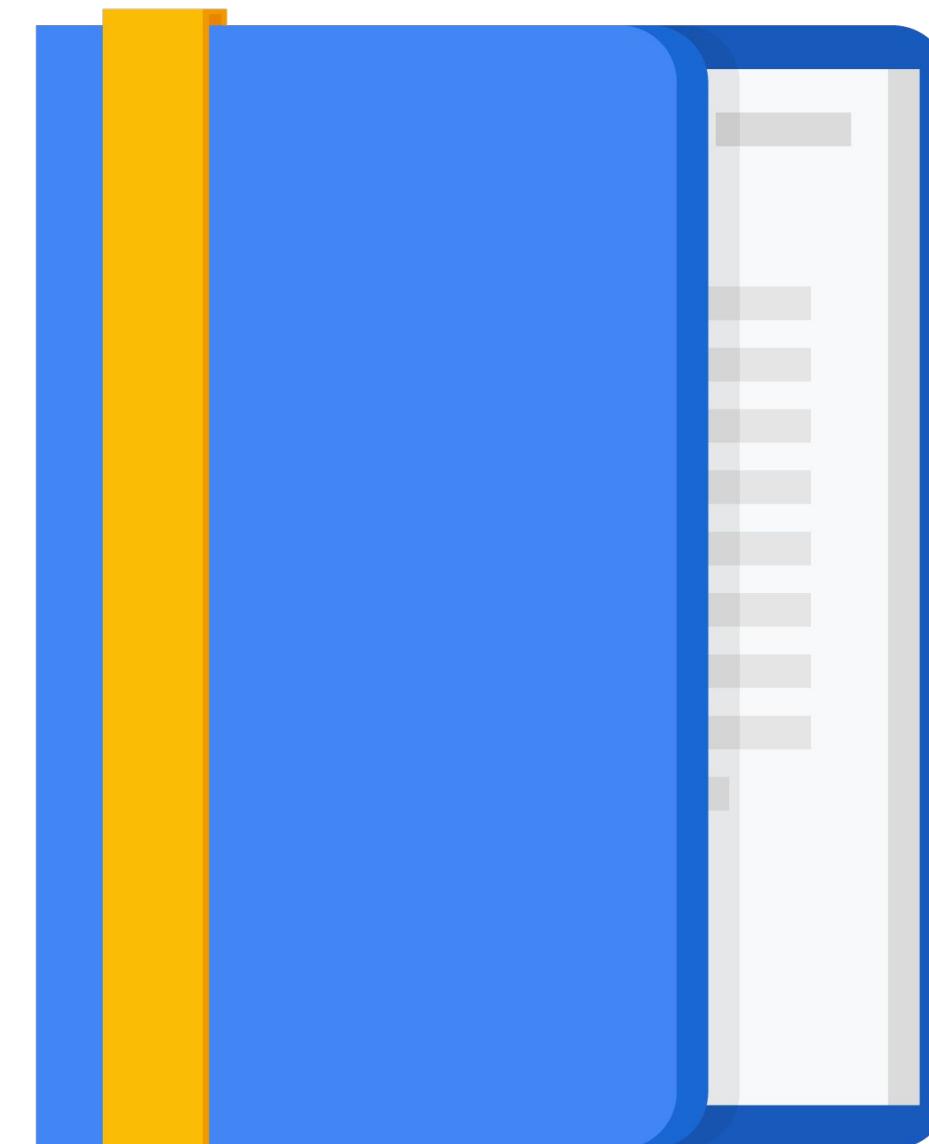
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

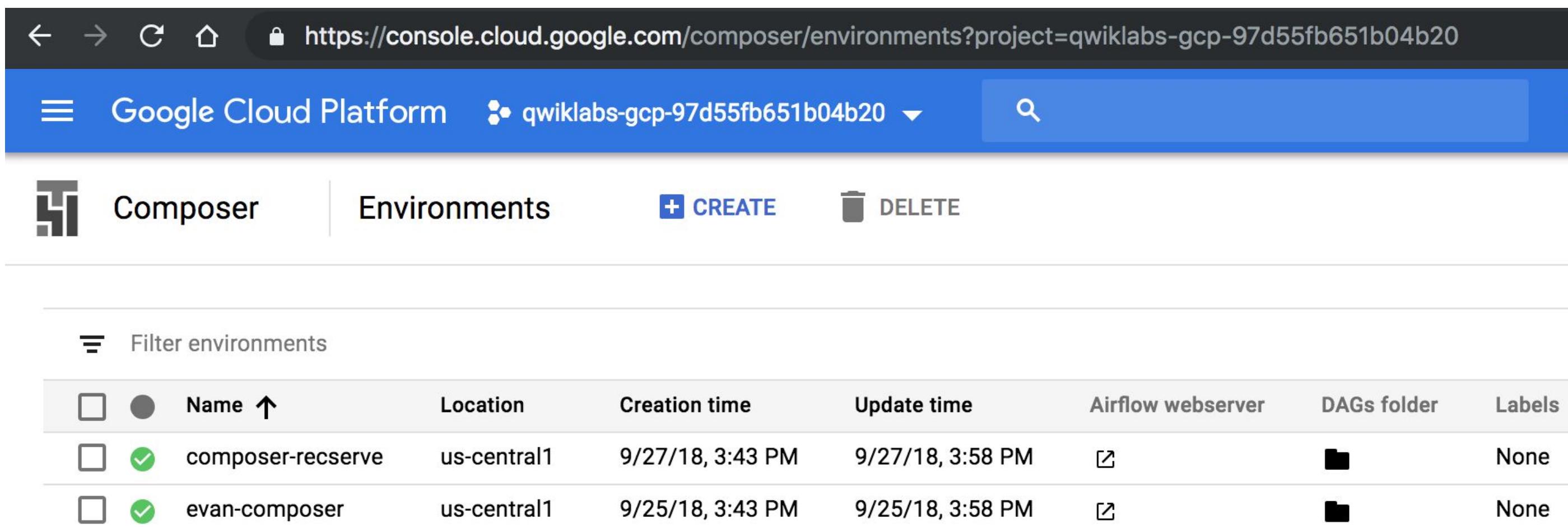
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

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



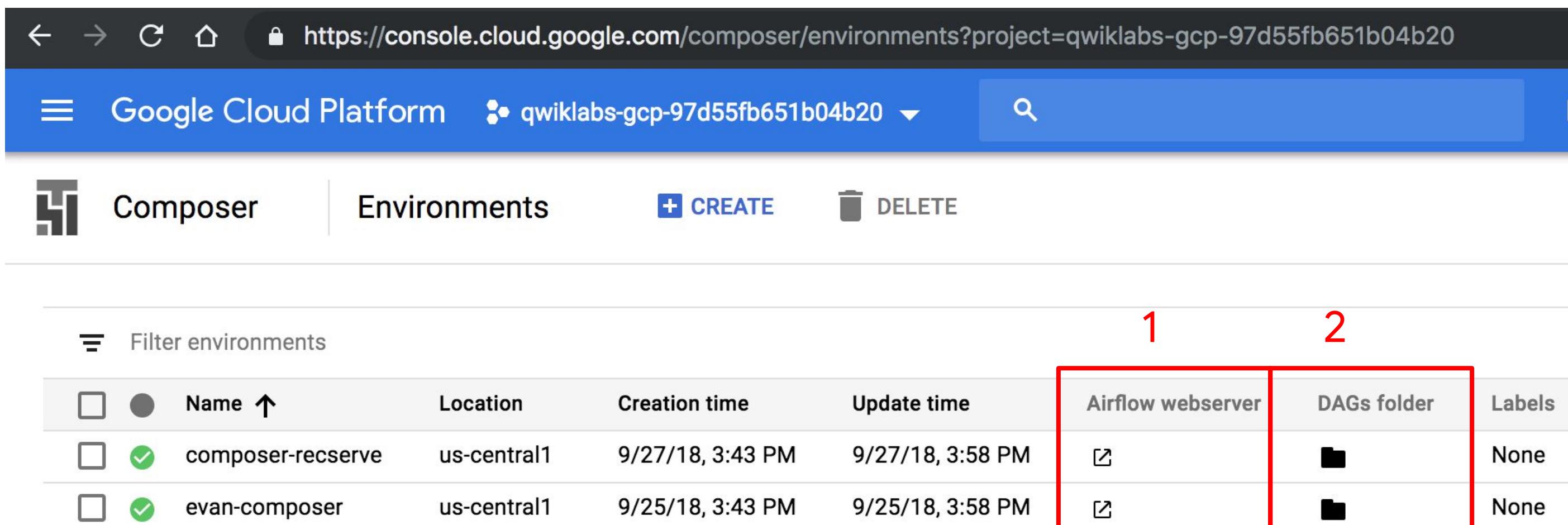
Cloud Composer creates managed Apache Airflow environments



The screenshot shows the Google Cloud Platform interface for Cloud Composer environments. At the top, there's a logo for Cloud Composer (a blue hexagon with a white 'H') and the text 'Cloud Composer'. Below the logo is a browser-style header with navigation icons, a URL bar showing <https://console.cloud.google.com/composer/environments?project=qwiklabs-gcp-97d55fb651b04b20>, and a search bar. The main navigation bar includes links for 'Google Cloud Platform', 'qwiklabs-gcp-97d55fb651b04b20', and a search icon. The main content area has tabs for 'Composer' (selected) and 'Environments'. It features a 'CREATE' button and a 'DELETE' button. A 'Filter environments' section allows filtering by name, location, creation time, update time, Airflow webserver, DAGs folder, and labels. Two environment entries are listed: 'composer-recserve' and 'evan-composer'. Both entries show their creation and update times, Airflow webserver status (indicated by a link icon), DAGs folder (indicated by a folder icon), and label status (None). The 'composer-recserve' entry is checked.

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

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



The screenshot shows the Google Cloud Platform Composer environments page. At the top, there's a logo for Cloud Composer and a navigation bar with links for 'Composer', 'Environments', 'CREATE', and 'DELETE'. Below the navigation bar is a search bar and a 'Filter environments' dropdown. The main table lists two environments:

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

Red numbers 1 and 2 are overlaid on the 'Airflow webserver' and 'DAGs folder' columns respectively, indicating they represent separate components for each environment.

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



Cloud Composer

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

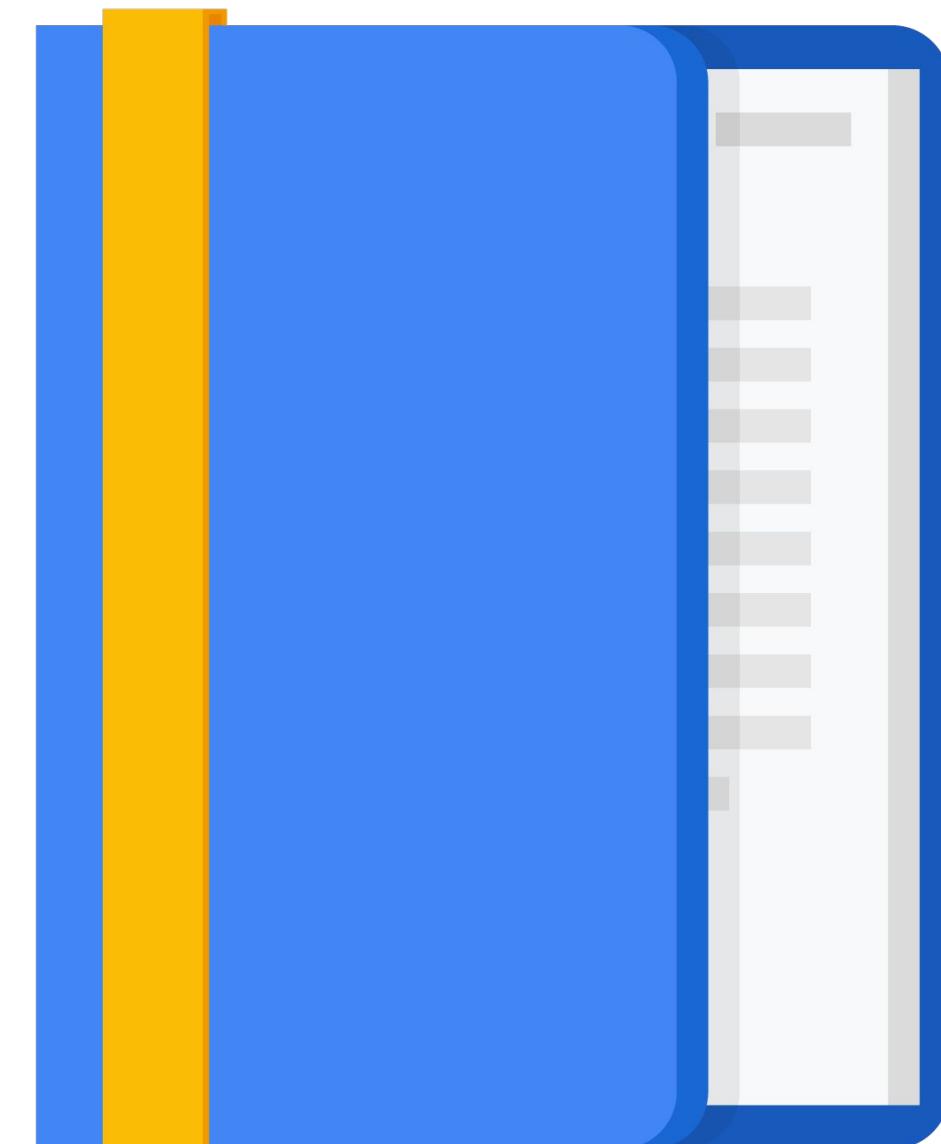
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

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

Orchestrating work between GCP services with Cloud Composer

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



Airflow workflows are written in Python

The screenshot shows a comparison between a Google Cloud Storage bucket listing and a Python script. A red arrow points from the 'simple_load_dag.py' file in the storage listing to the corresponding Python code on the right. Another red arrow points from the 'simple.py' file in the storage listing to the Python code.

Buckets / us-central1-evan-composer-0e85530c-bucket / dags

Name	Size	Type	Storage Class
dataflow/	—	Folder	—
simple_load_dag.py	6.79 KB	text/x-python-script	Multi
simple.py	2.51 KB	text/x-python-script	Multi

```
106 # e.g. state,gender,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('bq_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 = dataflow_operator.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
```

Airflow workflows are written in Python

The screenshot shows a comparison between a Google Cloud Storage bucket listing and a Python script.

Google Cloud Storage Bucket Listing:

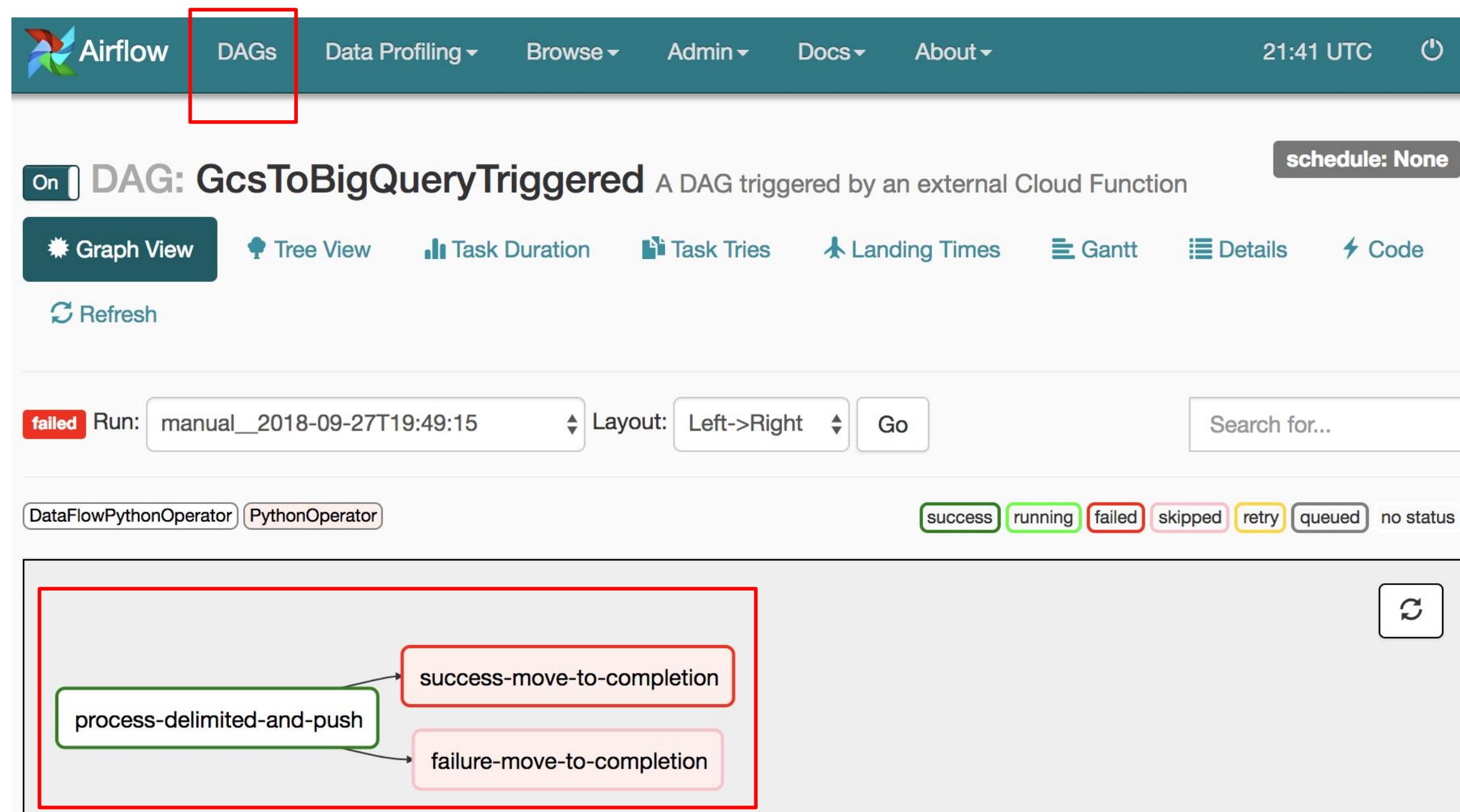
- Buckets / us-central1-evan-composer-0e85530c-bucket / dags
- dataflow/ (Folder)
- simple_load_dag.py (text/x-python-script, 6.79 KB)
- simple.py (text/x-python-script, 2.51 KB)

Python Script Content:

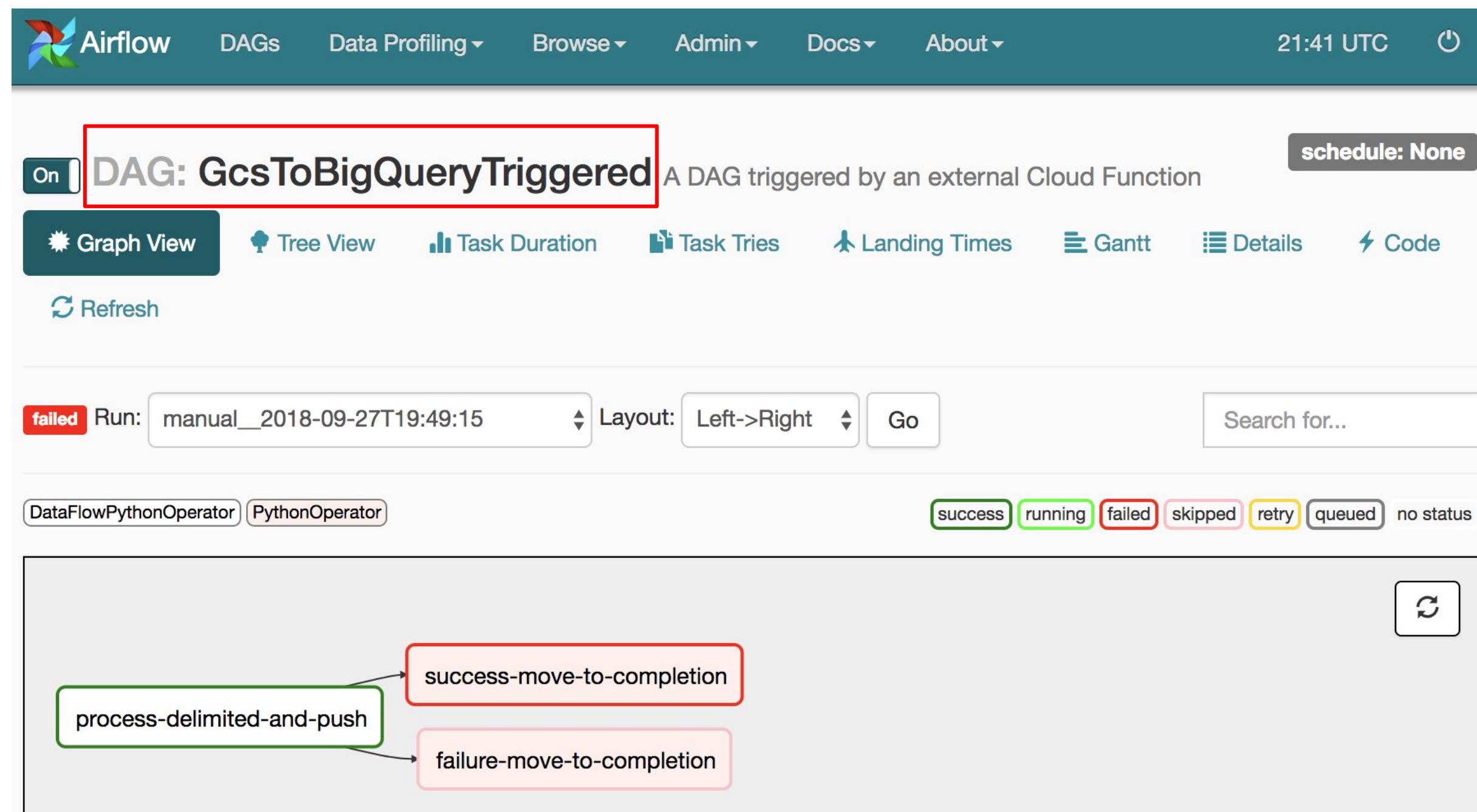
```
106 # e.g. state,gender,year,name,number,created_date
107 with models.DAG(dag_id='GcsToBigQueryTriggered',
108     description='A DAG triggered by an external Cloud Function',
109     schedule_interval=None, default_args=DEFAULT_DAG_ARGS) as dag:
110     # Args required for the Dataflow job.
111     job_args = {
112         'input': 'gs://{{ dag_run.conf["bucket"] }}/{{ dag_run.conf["name"] }}',
113         'output': models.Variable.get('bq_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 = dataflow_operator.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
```

Red arrows point from the "simple_load_dag.py" file in the storage listing to the corresponding code block in the script, and from the "simple.py" file to the "simple.py" code block in the script.

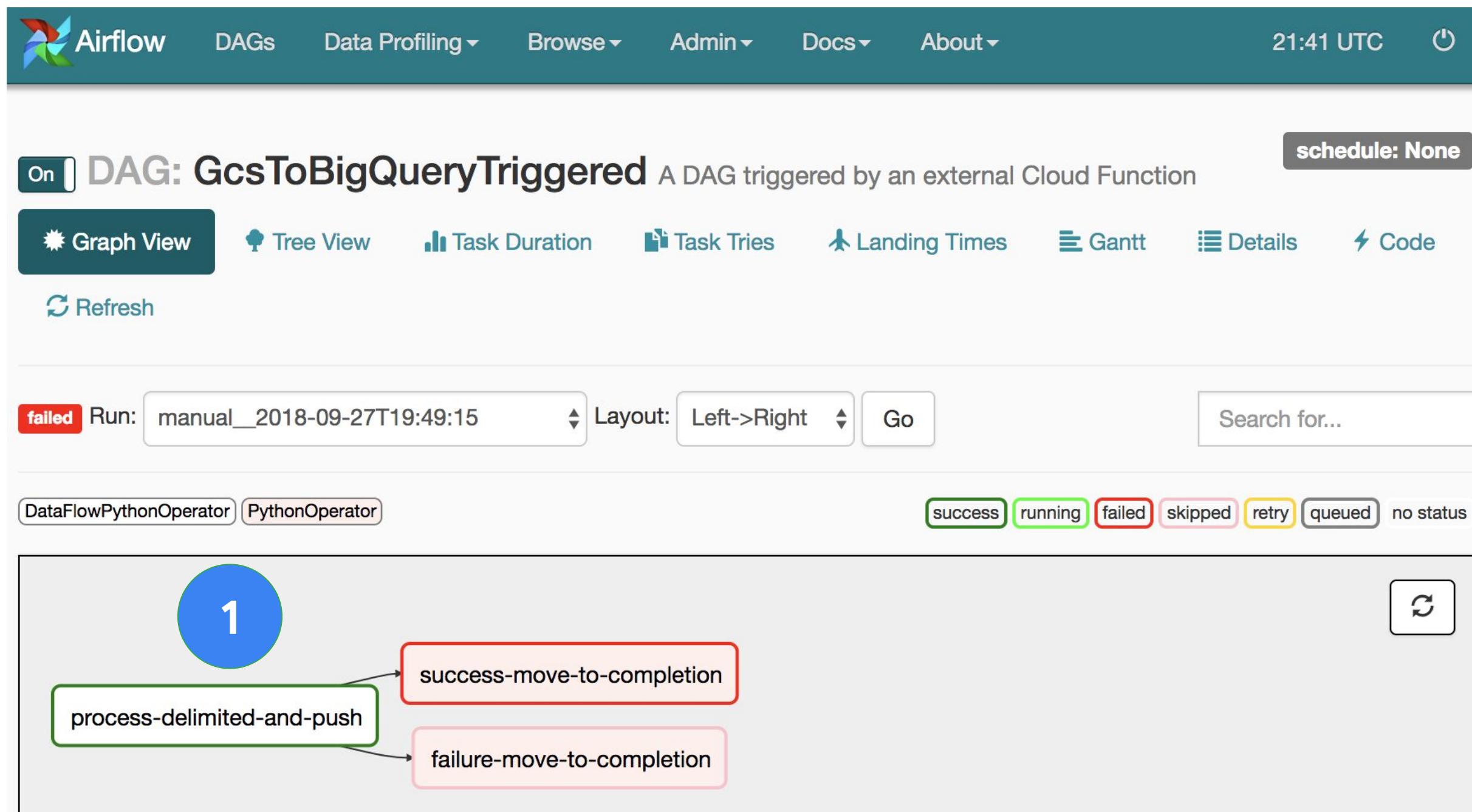
The python file creates a DAG



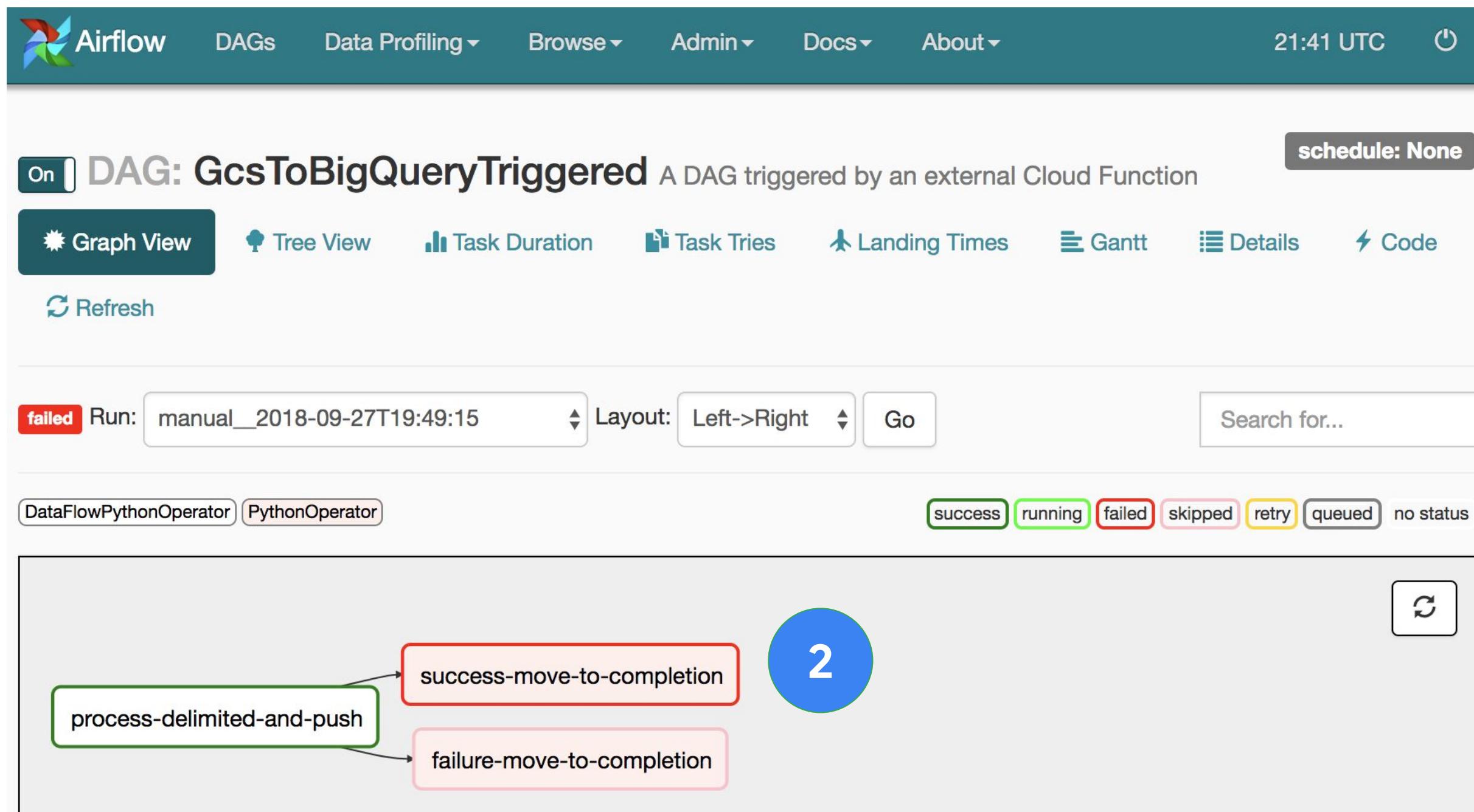
Airflow webserver UI overview



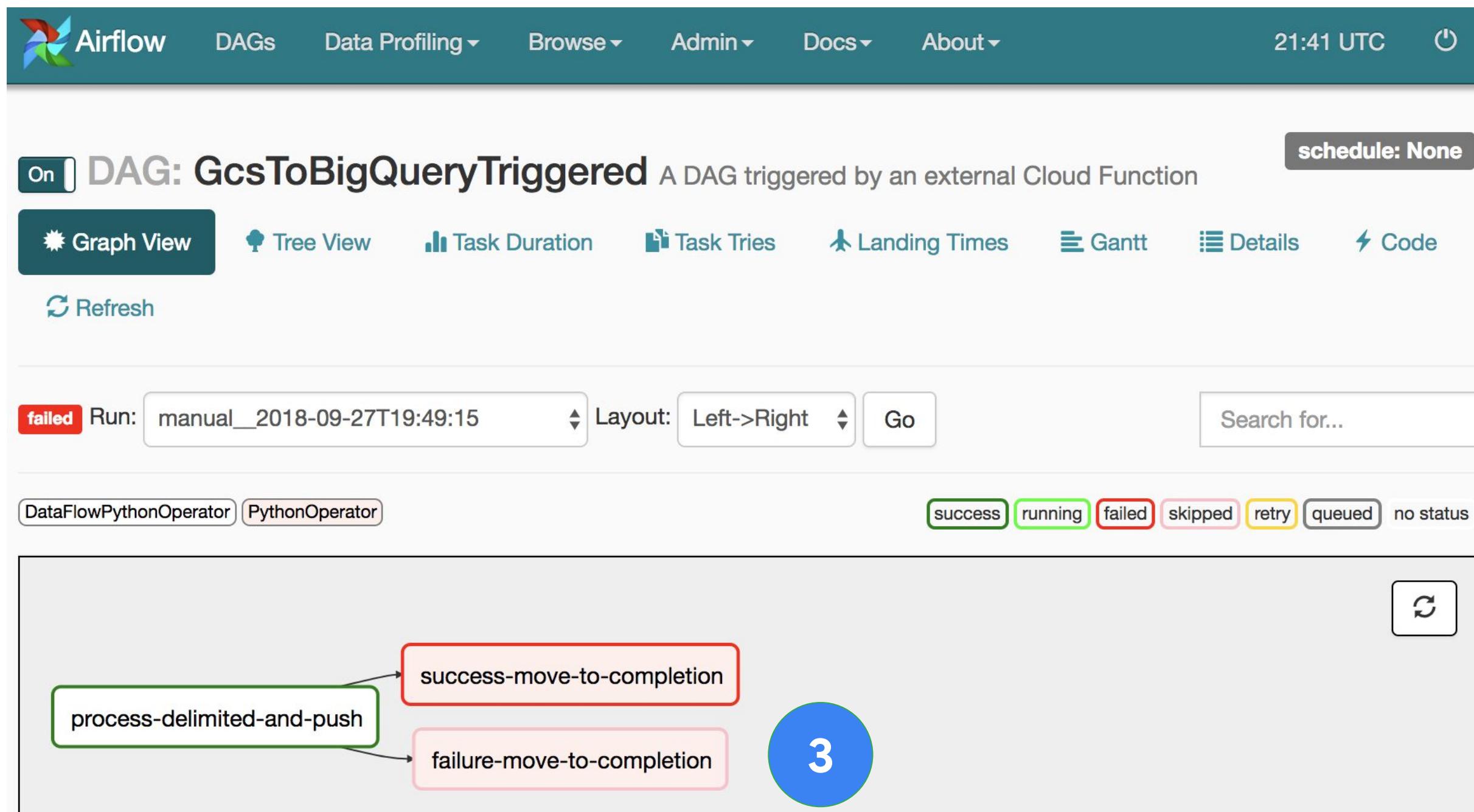
The python file creates a DAG (Directed Acyclic Graph)



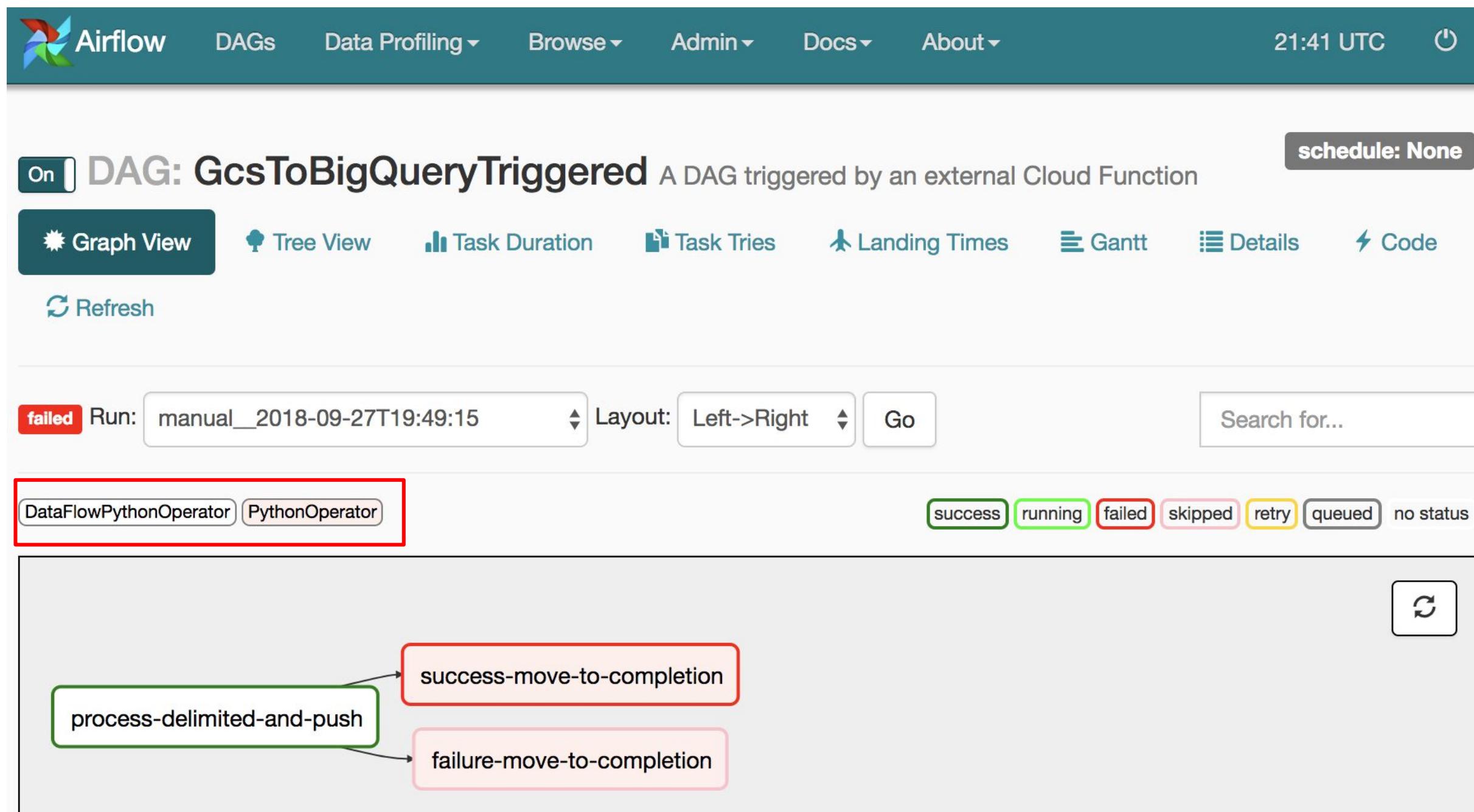
The python file creates a DAG (Directed Acyclic Graph)



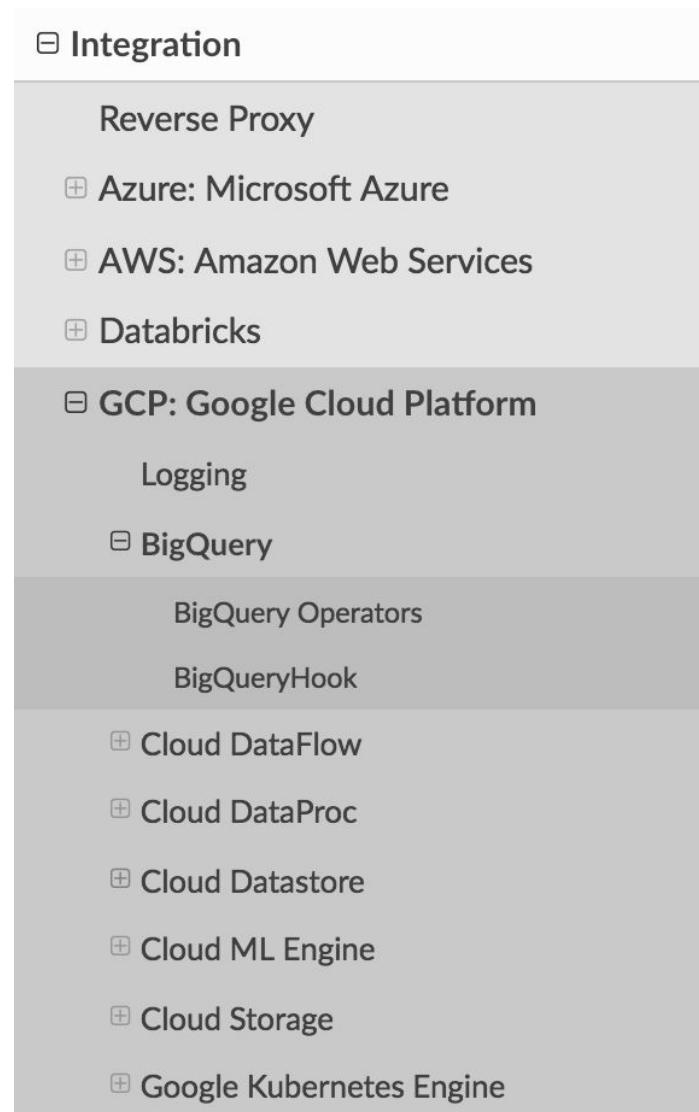
The python file creates a DAG (Directed Acyclic Graph)



The python file creates a DAG (Directed Acyclic Graph)



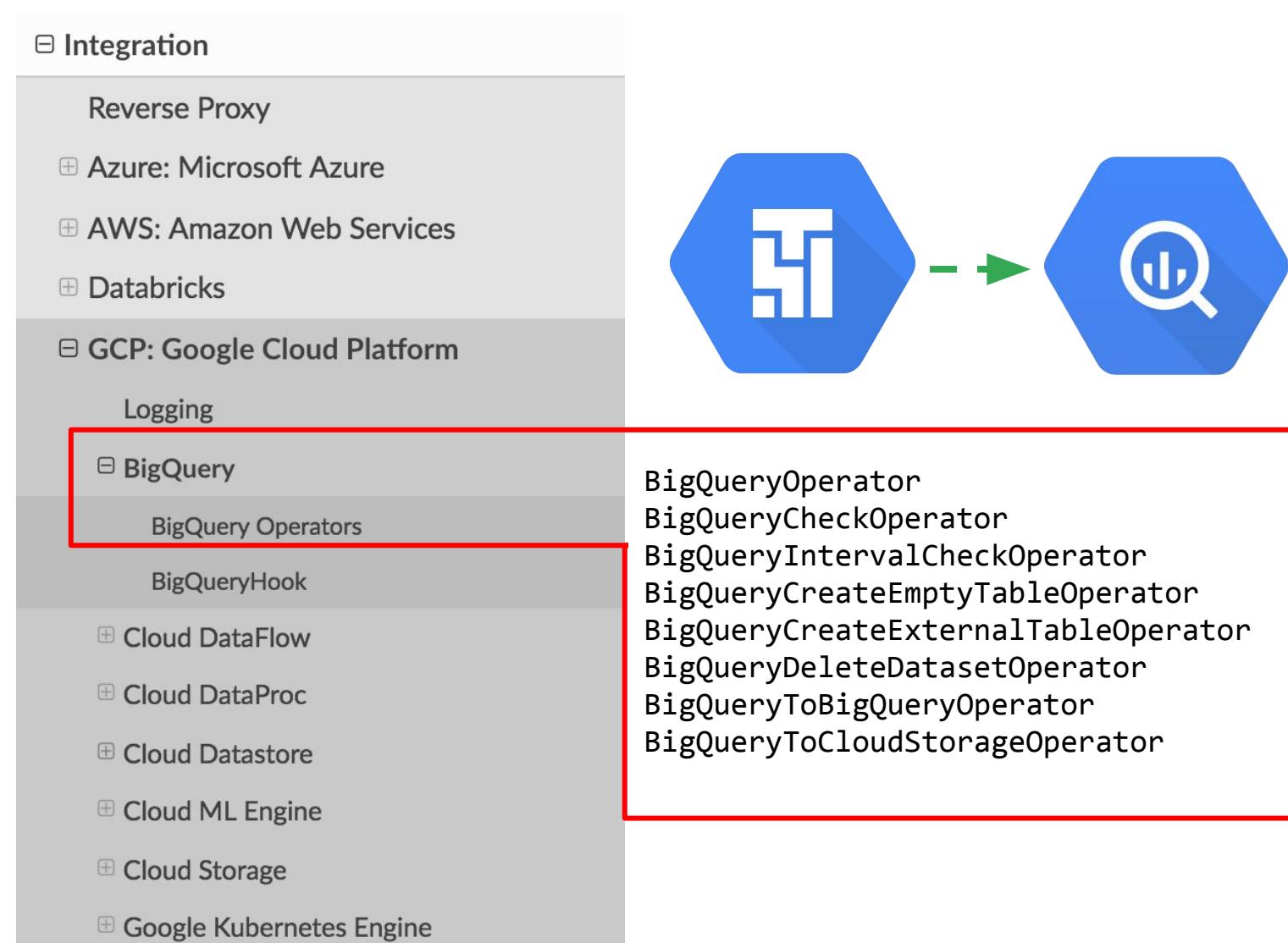
Airflow uses operators in your DAG to orchestrate other GCP services



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

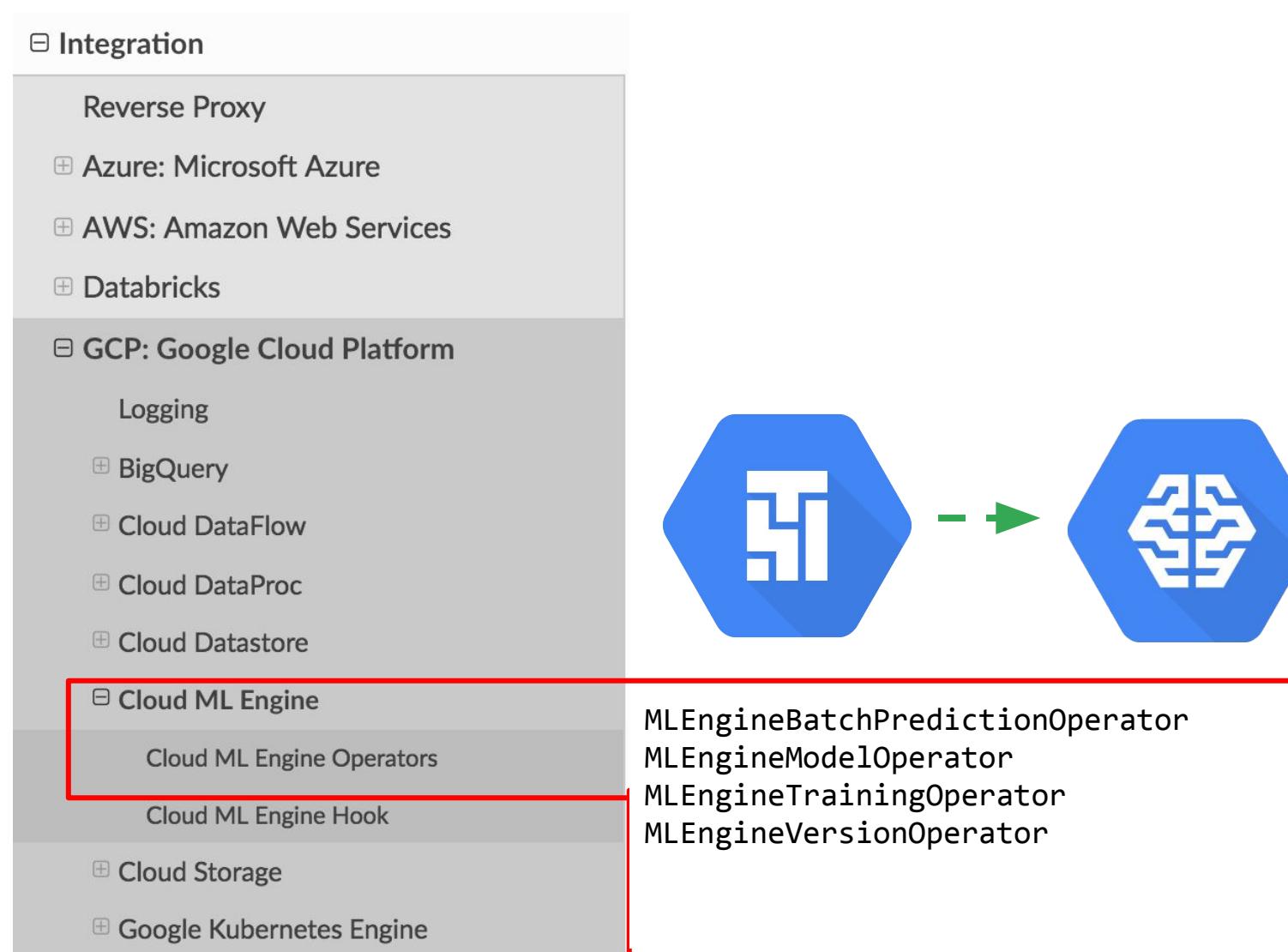
+ sample code

BigQuery Operators are popular for updating your ML training dataset



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

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



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

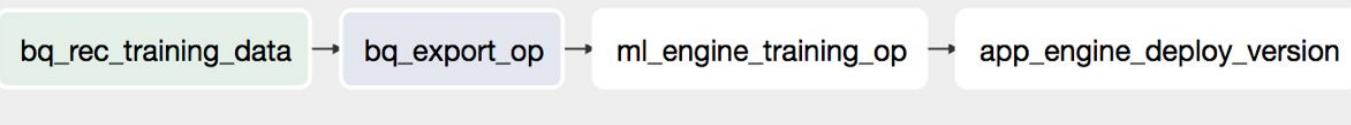
Apache Airflow is open source and cross-platform for hybrid pipelines

The screenshot shows a portion of the Apache Airflow documentation under the 'Integration' heading. The 'GCP: Google Cloud Platform' section is highlighted with a red box. The visible items in the 'Integration' list are:

- Reverse Proxy
 - ⊕ Azure: Microsoft Azure
 - ⊕ AWS: Amazon Web Services
 - ⊕ Databricks
 - ⊖ GCP: Google Cloud Platform
- Logging
 - ⊕ BigQuery
 - ⊕ Cloud DataFlow
 - ⊕ Cloud DataProc
 - ⊕ Cloud Datastore
 - ⊖ Cloud ML Engine
- Cloud ML Engine Operators
- Cloud ML Engine Hook
- Cloud Storage
 - ⊕ Google Kubernetes Engine

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

Example: Common Machine Learning workflow DAG Operators



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

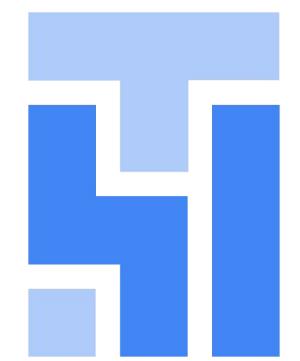
1

2

3

4

BigQuery and Cloud Storage operators get us fresh training data



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

1

2

3

4

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)
```

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)
```

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',
    bqql="""
        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)
```

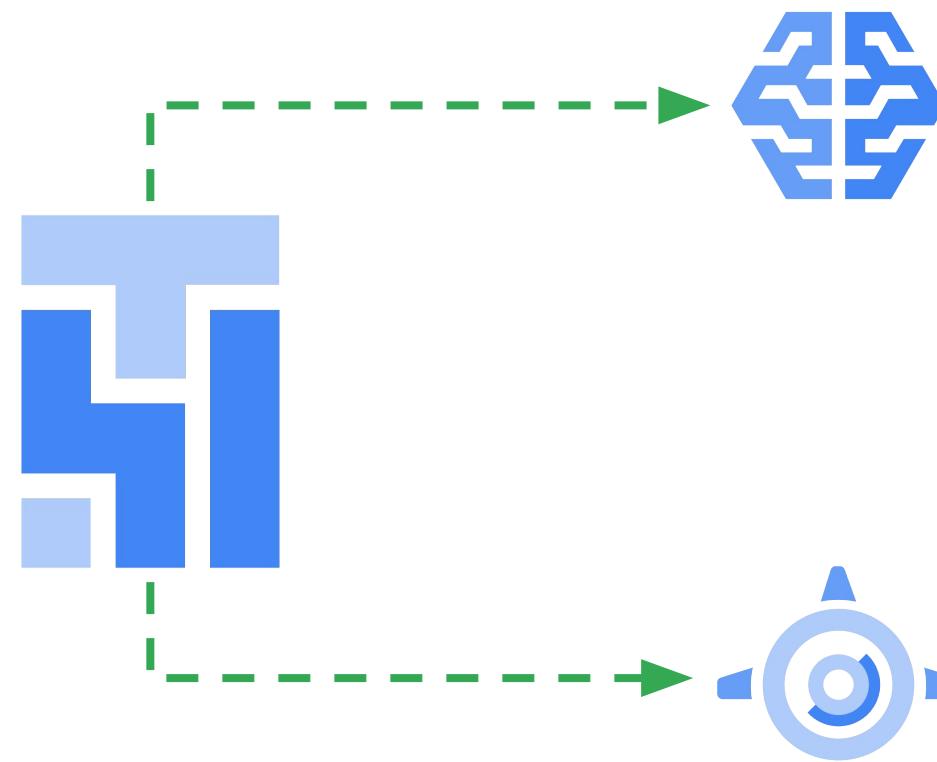
You could even set a rolling window with macros

```
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

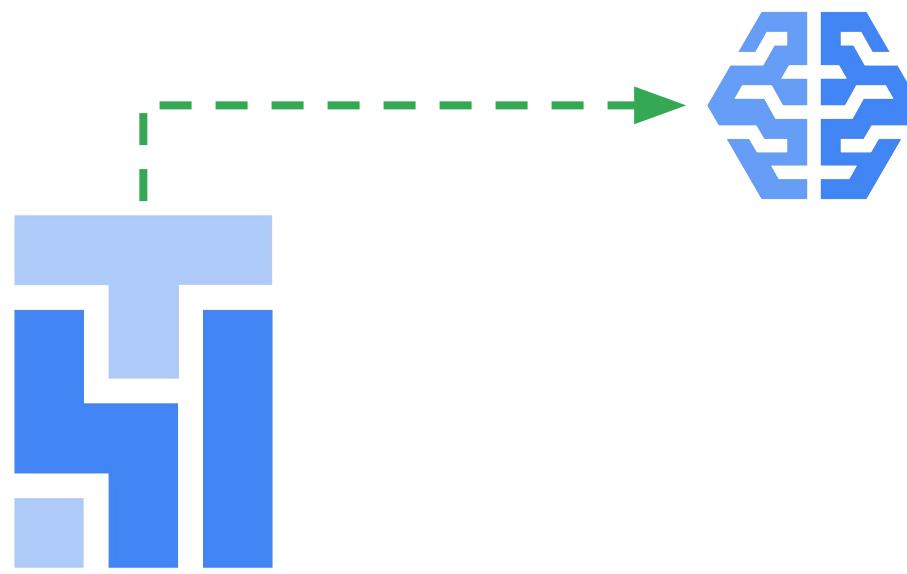
# Query recent StackOverflow questions.
bq_recent_questions_query = bigquery_operator.BigQueryOperator(
    task_id='bq_recent_questions_query',
    bqql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < 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)
```

AI Platform and App Engine operators retrain and redeploy our model



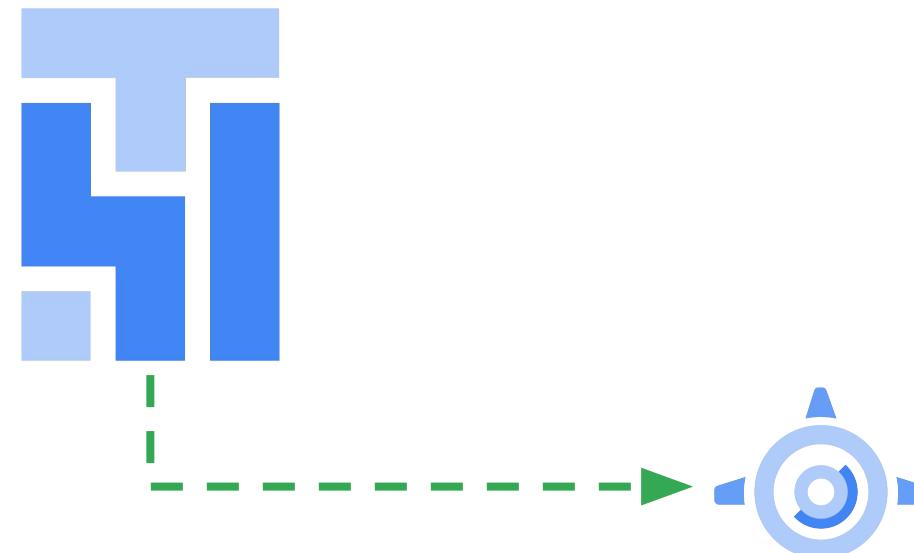
```
# update training data  
t1 = BigQueryOperator(  
)  
  
# BigQuery training data export to GCS  
t2 = BigQueryToCloudStorageOperator(  
)  
  
# AI Platform training job  
t3 = MLEngineTrainingOperator(  
)  
  
# App Engine deploy new version  
t4 = AppEngineVersionOperator(  
)  
  
# DAG dependencies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

Use Cloud ML Engine operators to periodically submit new training jobs



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

Use App Engine operators to periodically deploy and redeploy models



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

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)
```

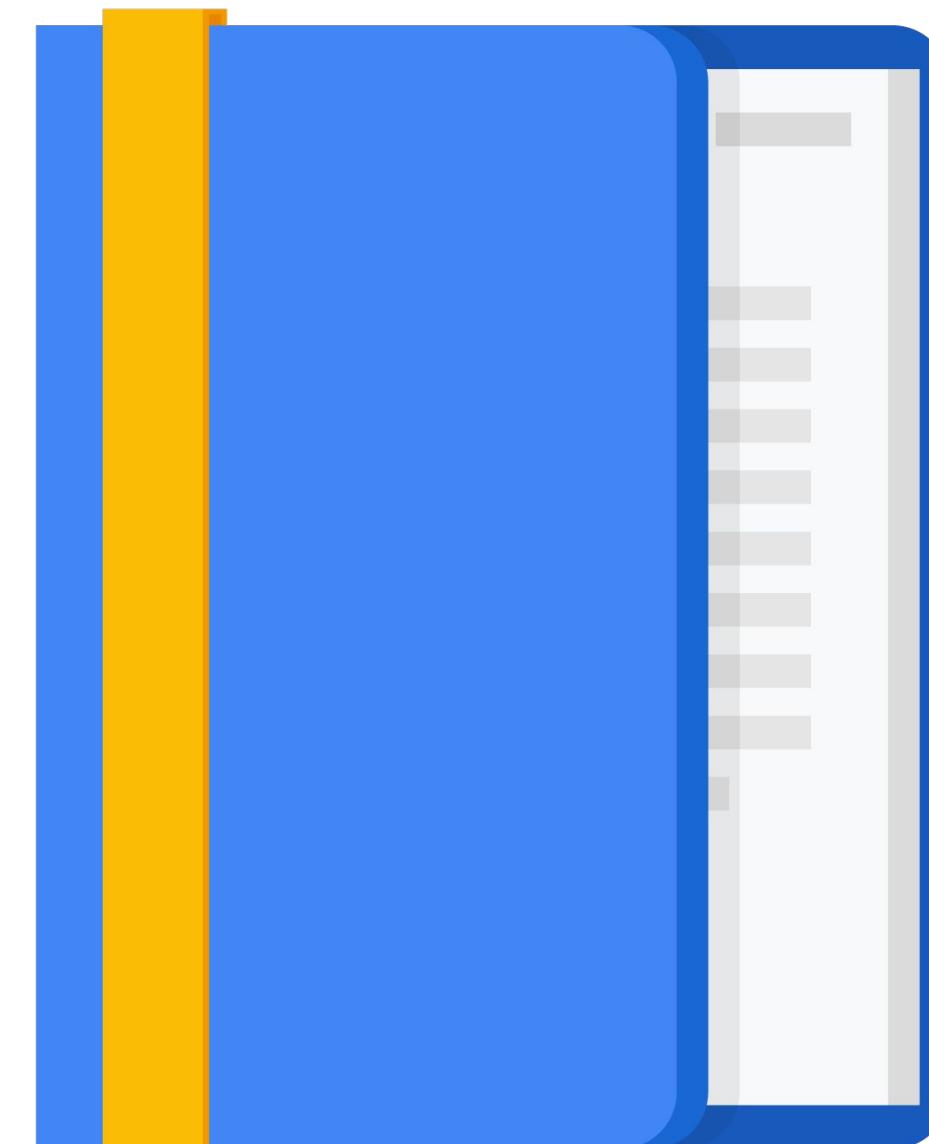
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

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

Orchestrating work between GCP services with Cloud Composer

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



Two scheduling options for Cloud Composer workflows

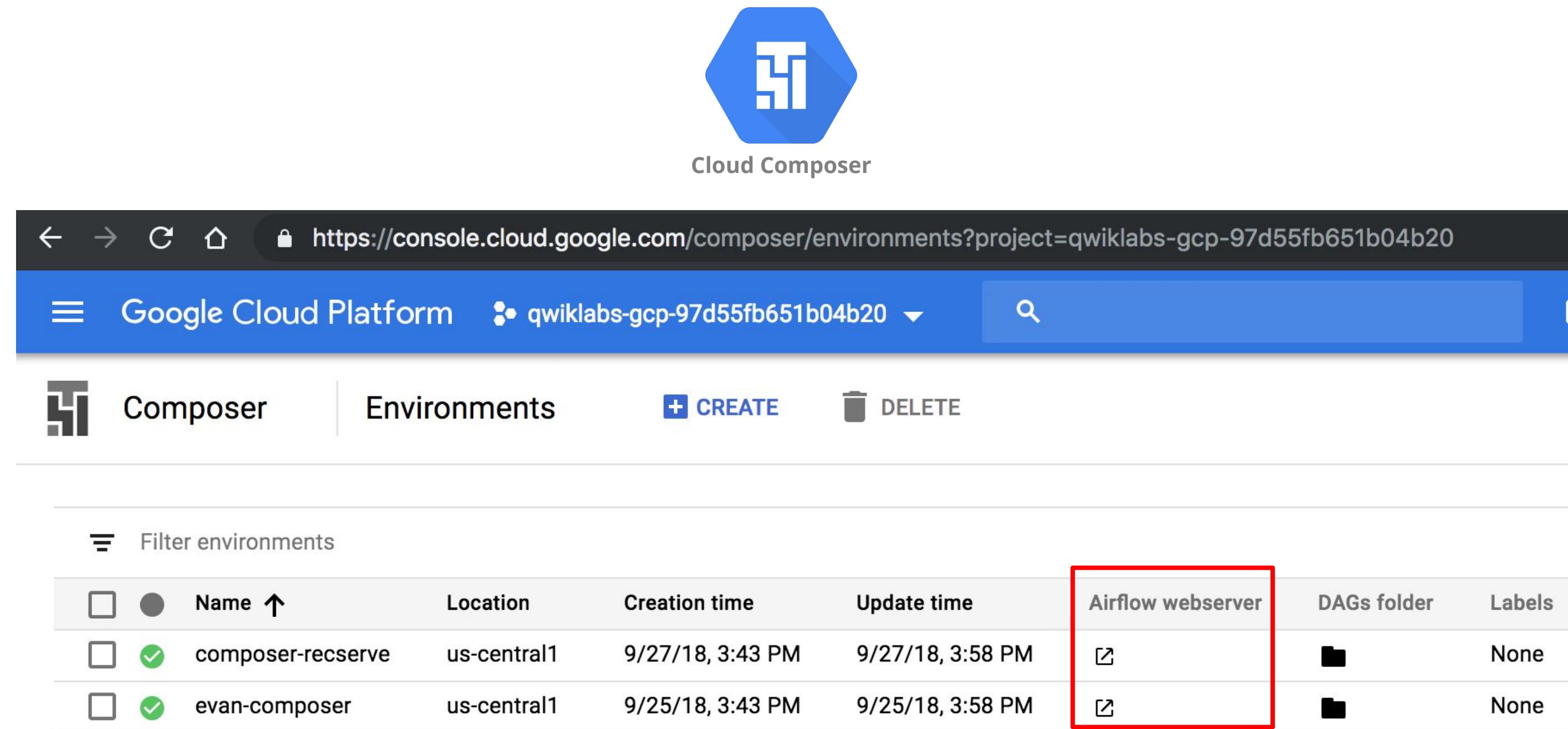


Periodic



Event-driven

Launch the Airflow webserver to interact with your DAGs



The screenshot shows the Google Cloud Platform interface for Cloud Composer environments. At the top, there's a logo for Cloud Composer and a navigation bar with links for 'Composer', 'Environments', 'CREATE', and 'DELETE'. Below this is a search bar labeled 'Filter environments' with a dropdown menu showing 'Name ↑'. A table lists two environments:

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

The 'Airflow webserver' column for both rows is highlighted with a red box.

Airflow scheduling basics

Screenshot of the Airflow web interface showing the DAGs page.

The top navigation bar includes:

- Airflow logo
- DAGs
- Data Profiling ▾
- Browse ▾
- Admin ▾
- Docs ▾
- About ▾
- 20:14 UTC
- Power icon

The main content area is titled "DAGs". It features a search bar labeled "Search:" and a table listing two DAGs:

		DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
		GcsToBigQueryTriggered		Airflow				
		composer_sample_simple_greeting		Airflow		2018-02-21 00:00		

Text at the bottom right: "Showing 1 to 2 of 2 entries"

Why is this DAG missing a schedule?

Screenshot of the Airflow web interface showing the DAGs page.

The top navigation bar includes links for Airflow, DAGs, Data Profiling, Browse, Admin, Docs, About, the current time (20:14 UTC), and a power icon.

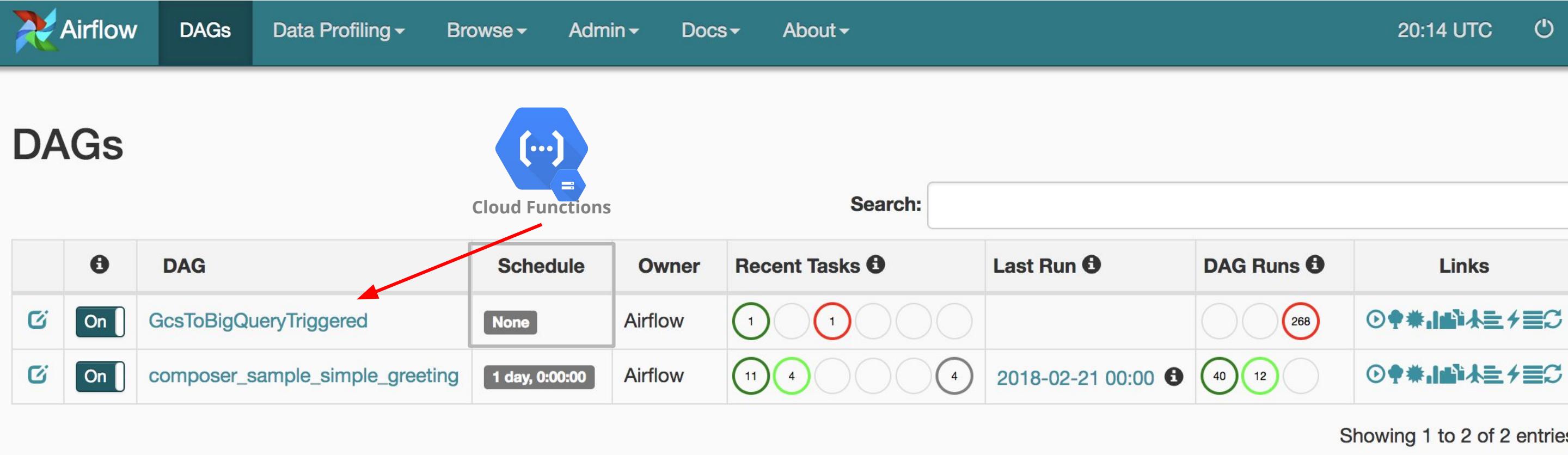
DAGs

Search:

	i	DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
		GcsToBigQueryTriggered	None	Airflow	1 1 1 1 1 1		268	
		composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 1 1 1 4	2018-02-21 00:00 i	40 12 1	

Showing 1 to 2 of 2 entries

Option 1: Event-driven scheduling with Cloud Functions



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

- Header:** Airflow, DAGs, Data Profiling ▾, Browse ▾, Admin ▾, Docs ▾, About ▾, 20:14 UTC, Power button.
- Section Title:** DAGs
- Cloud Functions Icon:** A blue hexagon icon with three dots and an equals sign, located above the table.
- Search Bar:** Search: [empty]
- DAG Table:**

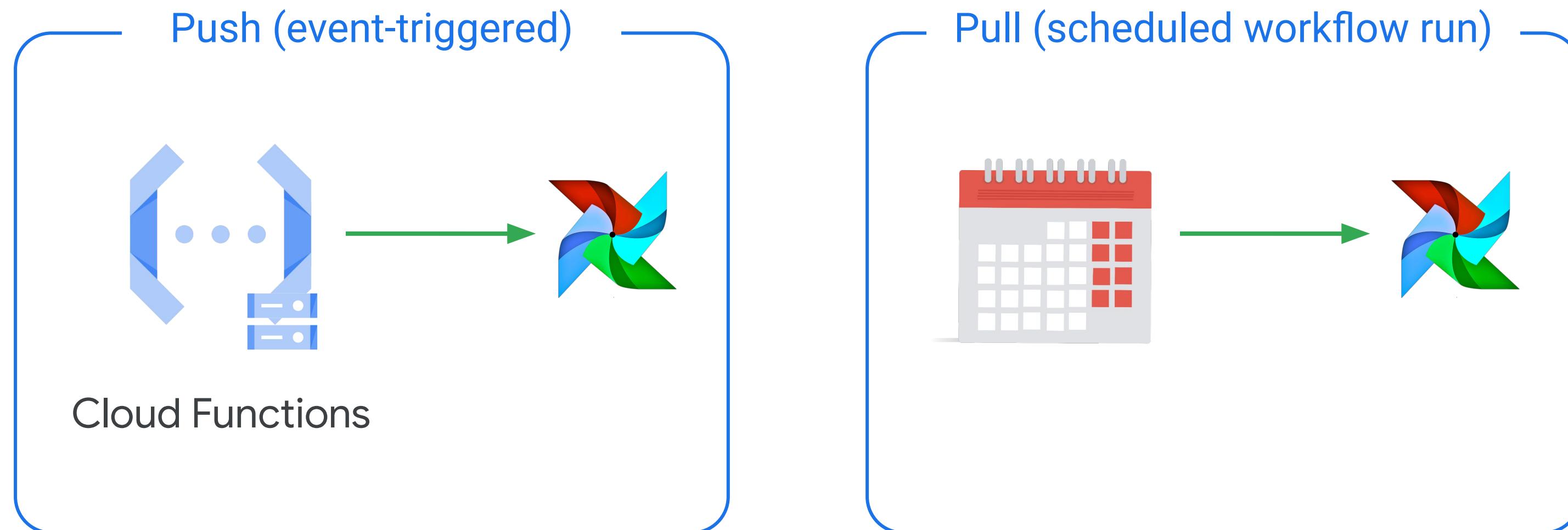
	i	DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
		GcsToBigQueryTriggered	None	Airflow				
		composer_sample_simple_greeting	1 day, 0:00:00	Airflow		2018-02-21 00:00 i		
- Text at bottom:** Showing 1 to 2 of 2 entries

Option 2: Specify pipeline schedule_interval in your DAG

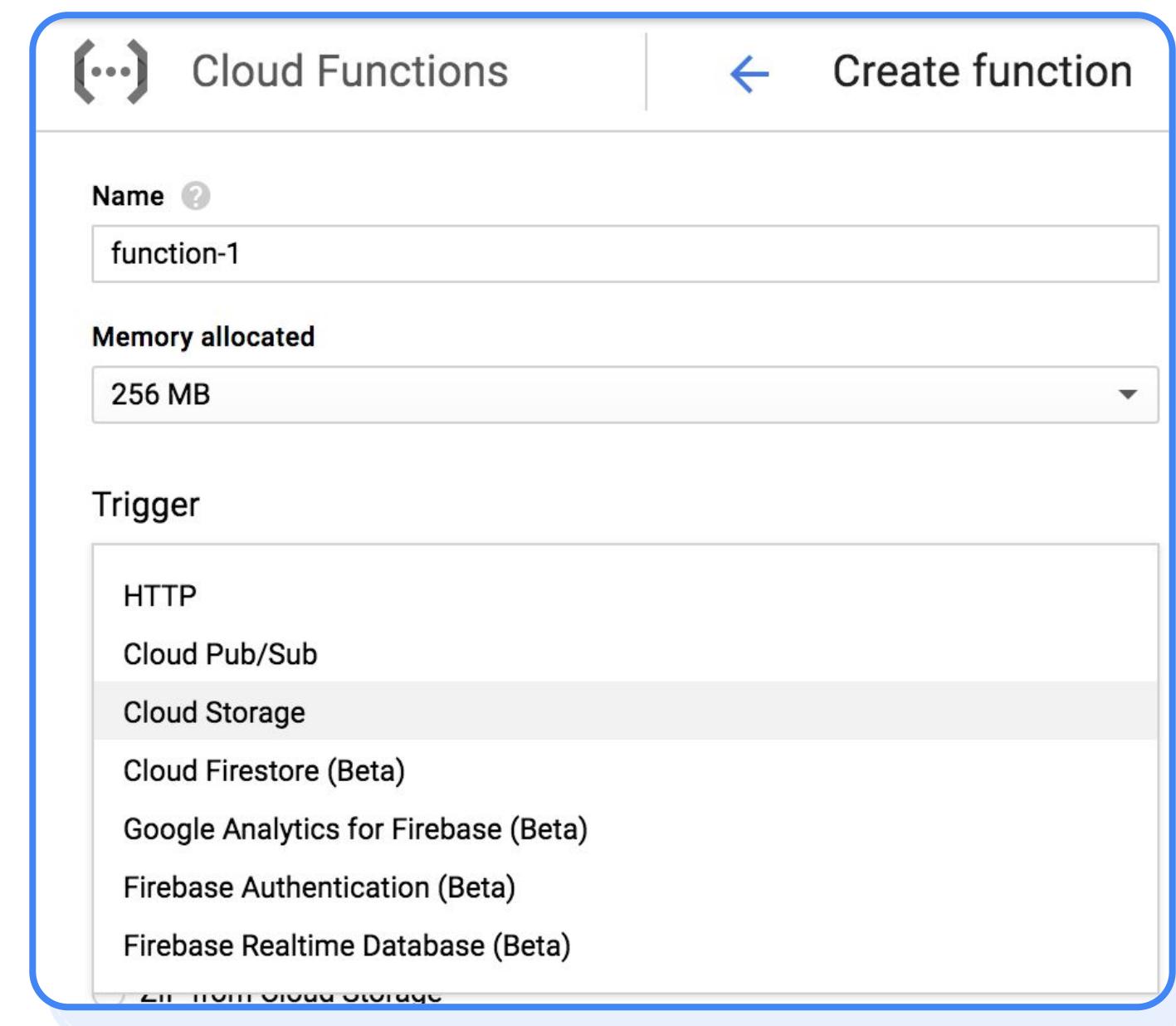
	i	DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
	On	GcsToBigQueryTriggered	None	Airflow	1 1 1 1 1 1		268	View Logs Metrics Tasks DAG API
	On	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 4 4 4 4	2018-02-21 00:00 i	40 12 1	View Logs Metrics Tasks DAG API

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

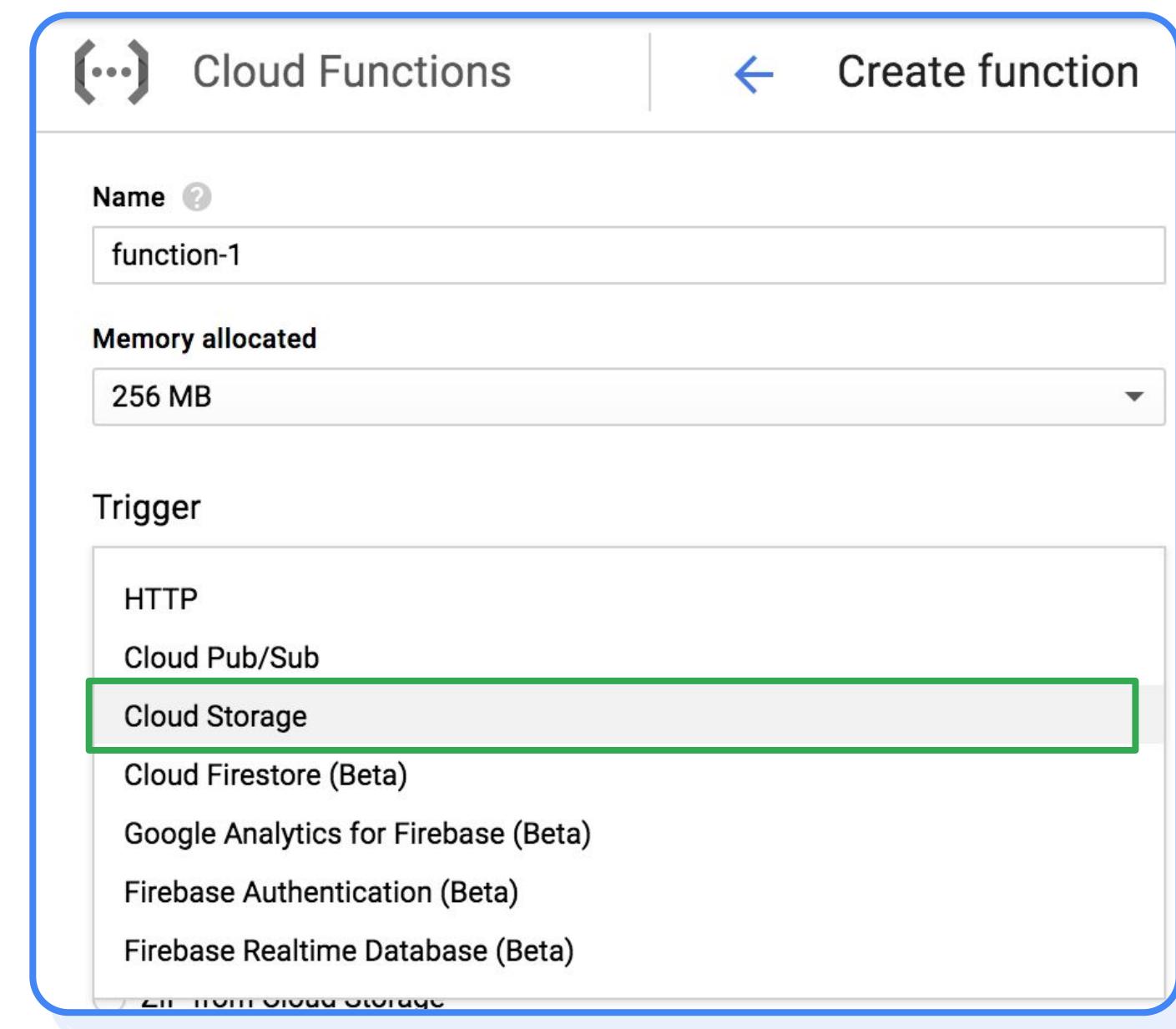
Two types of workflow ETL patterns



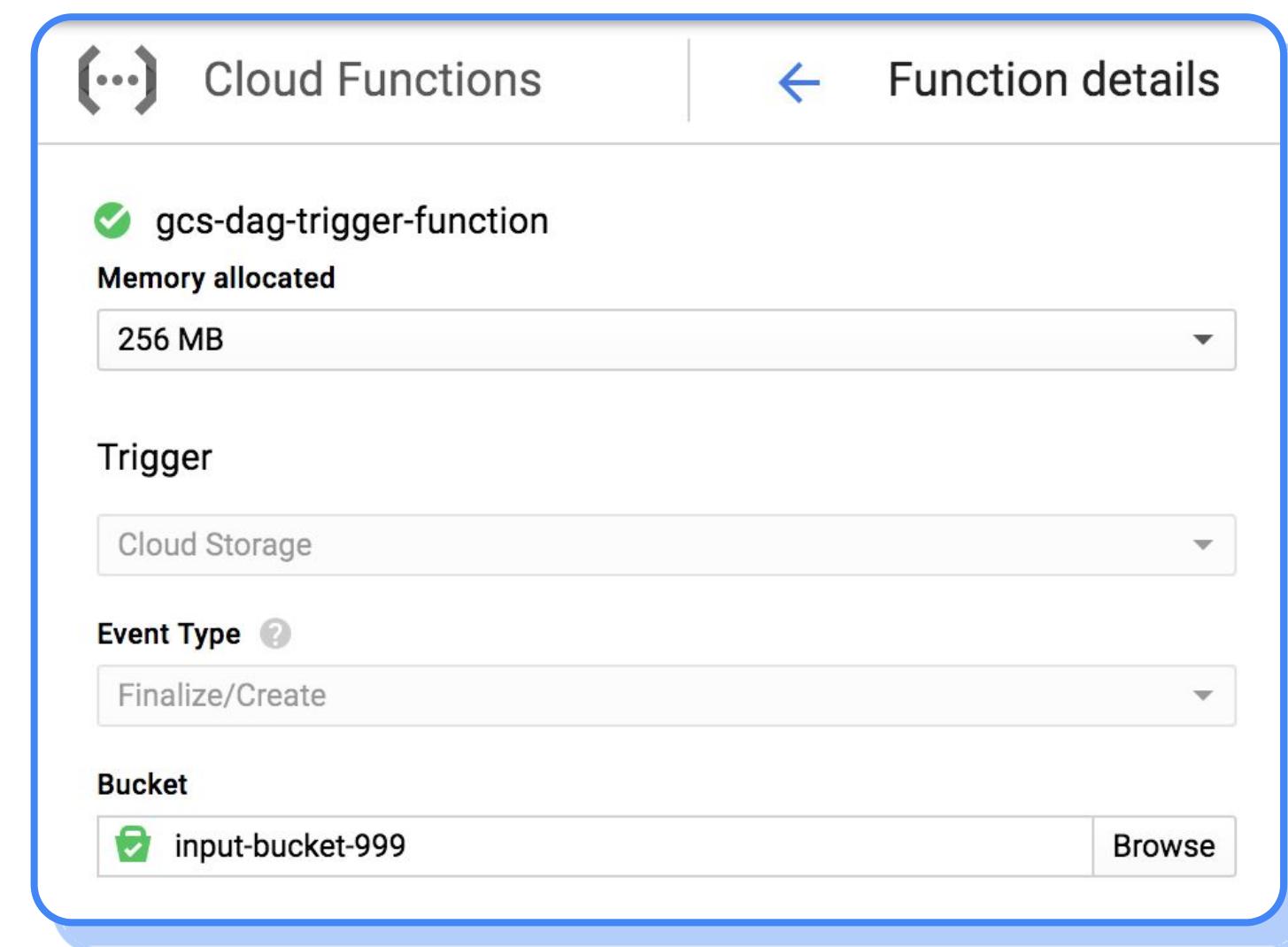
Use Cloud Functions to create an event-based push architecture



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



Creating a Cloud Function to trigger an Airflow DAG



Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

```
index.js    package.json

14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag (event, callback) {
18     // Fill in your Composer environment information here.
19
20     // The project that holds your function
21     const PROJECT_ID = 'qwiklabs-gcp';
22     // Navigate to your webserver's login page and get this from the URL
23     const CLIENT_ID = '';
24     // This should be part of your webserver's URL:
25     // {tenant-project-id}.appspot.com
26     const WEBSERVER_ID = 'b9';
27     // The name of the DAG you wish to trigger
28     const DAG_NAME = 'GcsToBigQueryTriggered';
29
30     // Other constants
31     const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32     const USER_AGENT = 'gcf-event-trigger';
33     const BODY = {'conf': JSON.stringify(event.data)};
34
35     // Make the request
36     authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37         .then(function iapAuthorizationCallback (iap) {
38             makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39         })
40         .then(_ => callback(null))
41         .catch(callback);
42     };
43 }
```

Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

```
index.js    package.json

14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag(event, callback) {
18     // Fill in your Composer environment information here.
19
20     // The project that holds your function
21     const PROJECT_ID = 'qwiklabs-gcp';
22     // Navigate to your webserver's login page and get this from the URL
23     const CLIENT_ID = '';
24     // This should be part of your webserver's URL:
25     // {tenant-project-id}.appspot.com
26     const WEBSERVER_ID = 'b9';
27     // The name of the DAG you wish to trigger
28     const DAG_NAME = 'GcsToBigQueryTriggered';
29
30     // Other constants
31     const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32     const USER_AGENT = 'gcf-event-trigger';
33     const BODY = {'conf': JSON.stringify(event.data)};
34
35     // Make the request
36     authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37         .then(function iapAuthorizationCallback (iap) {
38             makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39         })
40         .then(_ => callback(null))
41         .catch(callback);
42     };
43 }
```

Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

index.js package.json

```
14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag (event, callback) {
18   // Fill in your Composer environment information here.
19
20   // The project that holds your function
21   const PROJECT_ID = 'qwiklabs-gcp';
22   // Navigate to your webserver's login page and get this from the URL
23   const CLIENT_ID = 'REDACTED.apps.googleusercontent.com';
24   // This should be part of your webserver's URL:
25   // {tenant-project-id}.appspot.com
26   const WEBSERVER_ID = 'b9: -tp';
27   // The name of the DAG you wish to trigger
28   const DAG_NAME = 'GcsToBigQueryTriggered';
29
30   // Other constants
31   const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32   const USER_AGENT = 'gcf-event-trigger';
33   const BODY = {'conf': JSON.stringify(event.data)};
34
35   // Make the request
36   authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37     .then(function iapAuthorizationCallback (iap) {
38       makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39     })
40     .then(_ => callback(null))
41     .catch(callback);
42 };
43
```

Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

index.js package.json

```
14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag (event, callback) {
18   // Fill in your Composer environment information here.
19
20   // The project that holds your function
21   const PROJECT_ID = 'qwiklabs-gcp';
22   // Navigate to your webserver's login page and get this from the URL
23   const CLIENT_ID = 'REDACTED.apps.googleusercontent.com';
24   // This should be part of your webserver's URL:
25   // {tenant-project-id}.appspot.com
26   const WEBSERVER_ID = 'b9: -tp';
27   // The name of the DAG you wish to trigger
28   const DAG_NAME = 'GcsToBigQueryTriggered';
29
30   // Other constants
31   const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32   const USER_AGENT = 'gcf-event-trigger';
33   const BODY = {'conf': JSON.stringify(event.data)};
34
35   // Make the request
36   authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37     .then(function iapAuthorizationCallback (iap) {
38       makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39     })
40     .then(_ => callback(null))
41     .catch(callback);
42 };
43
```

Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

index.js package.json

```
14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag (event, callback) {
18     // Fill in your Composer environment information here.
19
20     // The project that holds your function
21     const PROJECT_ID = 'qwiklabs-gcp';
22     // Navigate to your webserver's login page and get this from the URL
23     const CLIENT_ID = '';
24     // This should be part of your webserver's URL:
25     // {tenant-project-id}.appspot.com
26     const WEBSERVER_ID = 'b9';
27     // The name of the DAG you wish to trigger
28     const DAG_NAME = 'GcsToBigQueryTriggered';
29
30     // Other constants
31     const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32     const USER_AGENT = 'gcf-event-trigger';
33     const BODY = {'conf': JSON.stringify(event.data)};
34
35     // Make the request
36     authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37         .then(function iapAuthorizationCallback (iap) {
38             makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39         })
40         .then(_ => callback(null))
41         .catch(callback);
42     };
43 }
```

Creating a Cloud Function GCS trigger an Airflow DAG



Cloud Functions

index.js package.json

```
14 * @param {!Object} event The Cloud Functions event.
15 * @param {!Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag (event, callback) {
18   // Fill in your Composer environment information here.
19
20   // The project that holds your function
21   const PROJECT_ID = 'qwiklabs-gcp';
22   // Navigate to your webserver's login page and get this from the URL
23   const CLIENT_ID = '';
24   // This should be part of your webserver's URL:
25   // {tenant-project-id}.appspot.com
26   const WEBSERVER_ID = 'b9';
27   // The name of the DAG you wish to trigger
28   const DAG_NAME = 'GcsToBigQueryTriggered';
29
30   // Other constants
31   const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag`;
32   const USER_AGENT = 'gcf-event-trigger';
33   const BODY = {'conf': JSON.stringify(event.data)};
34
35   // Make the request
36   authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
37     .then(function iapAuthorizationCallback (iap) {
38       makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
39     })
40     .then(_ => callback(null))
41     .catch(callback);
42 };
43
```

Cloud Functions | Function details

ZIP from Cloud Storage
Cloud Source repository

Runtime
Node.js 6

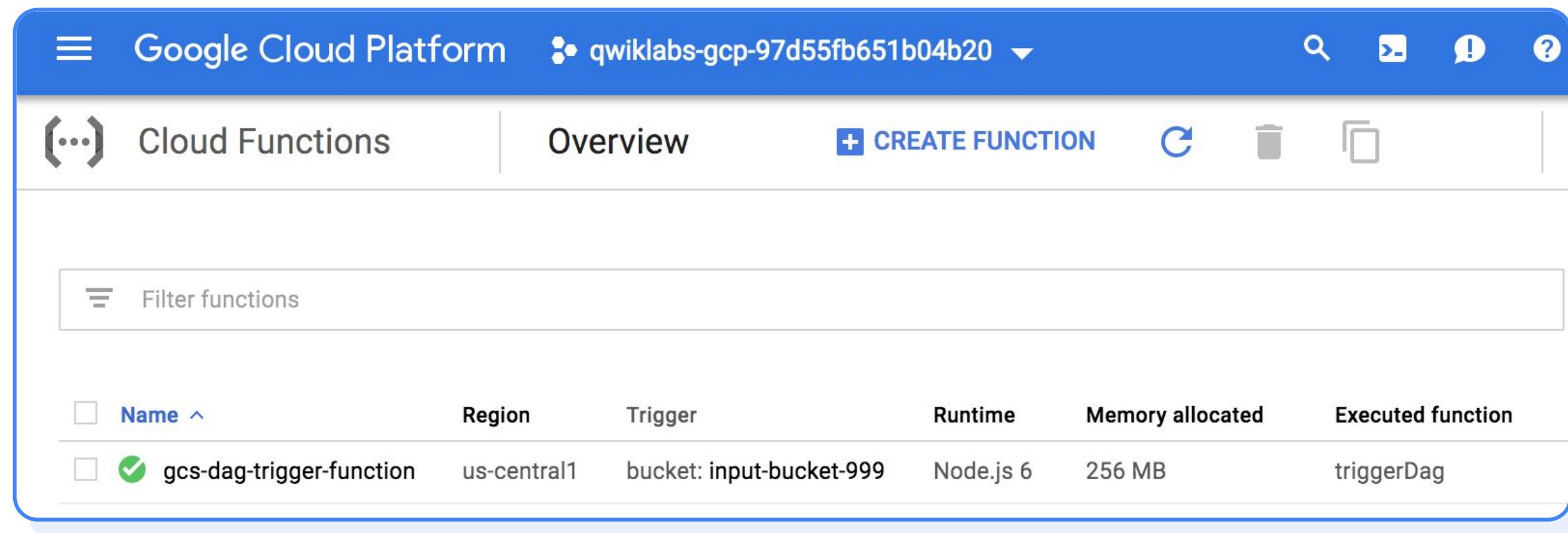
index.js package.json

```
1 'use strict';
2
3 const fetch = require('node-fetch');
4 const FormData = require('form-data');
5
6 /**
7 * Triggered from a message on a Cloud Storage bucket.
8 *
9 * IAP authorization based on:
10 * https://stackoverflow.com/questions/45787676/how-to-a
11 * and
12 * https://cloud.google.com/iap/docs/authentication-howt
13 *
14 * @param {Object} event The Cloud Functions event.
15 * @param {Function} callback The callback function.
16 */
17 exports.triggerDag = function triggerDag(event, callback) {
18 // Fill in your Composer environment information here.
19
20 // The project that holds your function
21 const PROJECT_ID = 'qwiklabs-gcp-97d55fb651b04b20';
22 // Navigate to your webserver's login page and get this
23 const CLIENT_ID = '954500678485-gde6id87qtdm9itl7809uj';
24 // This should be part of your webserver's URL:
25 }
```

Function to execute ?
triggerDag

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

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 Platform logo, the project name "qwiklabs-gcp-97d55fb651b04b20", and several icons for search, refresh, and help. Below the navigation bar, there are two tabs: "Cloud Functions" (selected) and "Overview". A "CREATE FUNCTION" button is located next to the tabs. On the left side, there is a "Filter functions" input field. The main area displays a table with the following data:

<input type="checkbox"/> 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

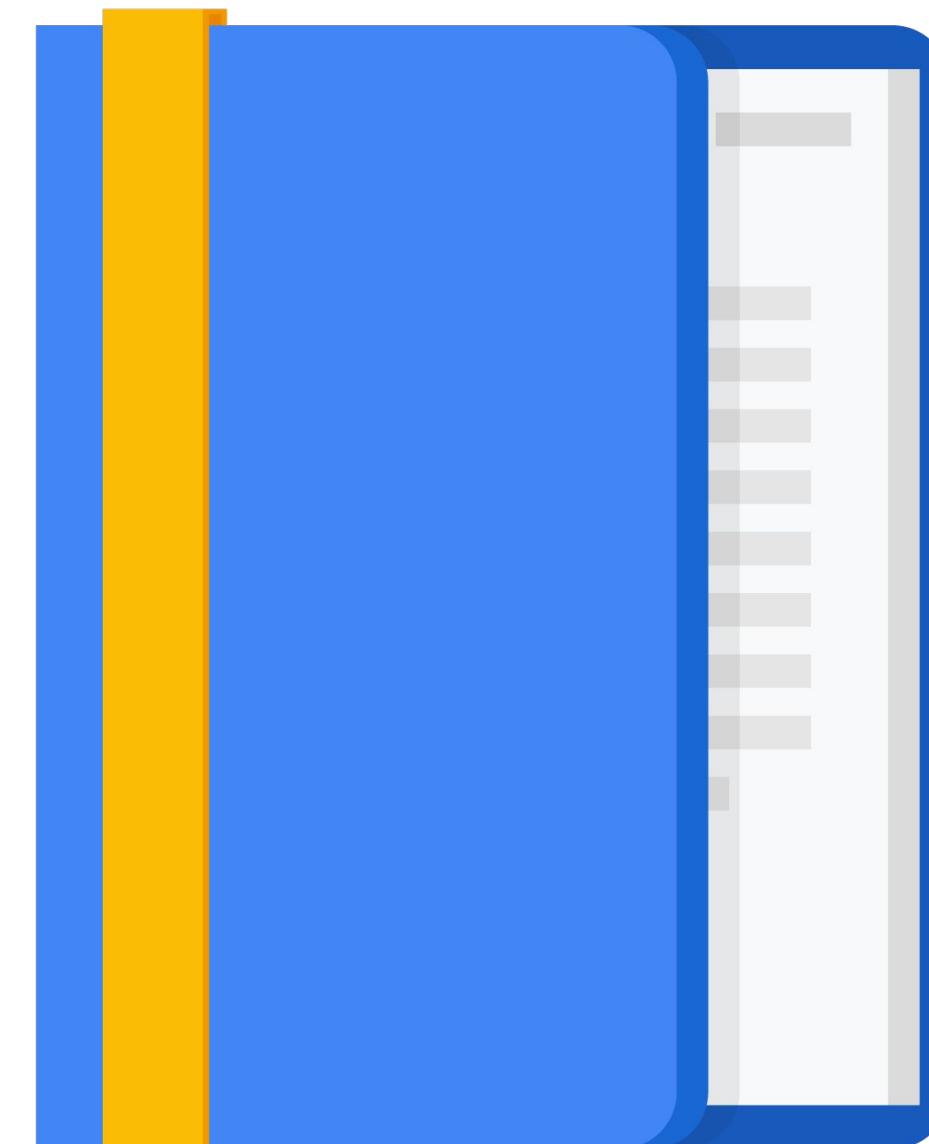
Agenda

Building Batch Data Pipelines visually with Cloud Data Fusion

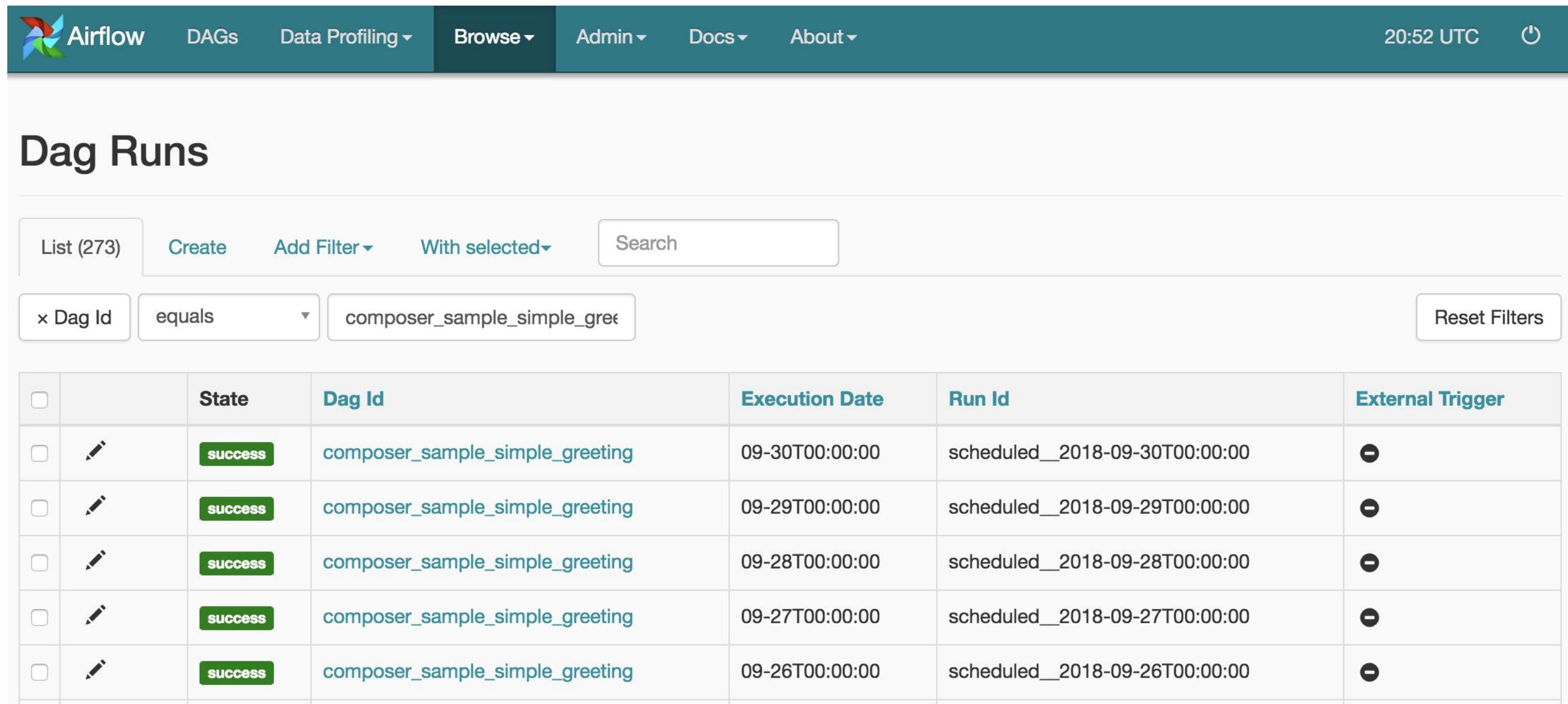
- Components
- UI Overview
- Building a Pipeline
- Exploring Data using Wrangler

Orchestrating work between GCP services with Cloud Composer

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



Monitor current and historical workflow progress



The screenshot shows the Airflow web interface with the following navigation bar:

- Airflow logo
- DAGs
- Data Profiling ▾
- Browse ▾
- Admin ▾
- Docs ▾
- About ▾
- 20:52 UTC
- Power icon

The main page title is "Dag Runs". Below it is a toolbar with the following buttons:

- List (273)
- Create
- Add Filter ▾
- With selected ▾
- Search

Below the toolbar is a search filter section:

- x Dag Id
- equals
- composer_sample_simple_greet
- Reset Filters

The main content area displays a table of Dag Runs:

	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	

Quickly gauge pipeline health of DAG Runs

DAGs

Search:

	i	DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
	On	GcsToBigQueryTriggered	None	Airflow	1 1 1 1 1 1		268	
	On	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 1 1 1 1 4	2018-02-21 00:00 i	40 12 1	

Showing 1 to 2 of 2 entries

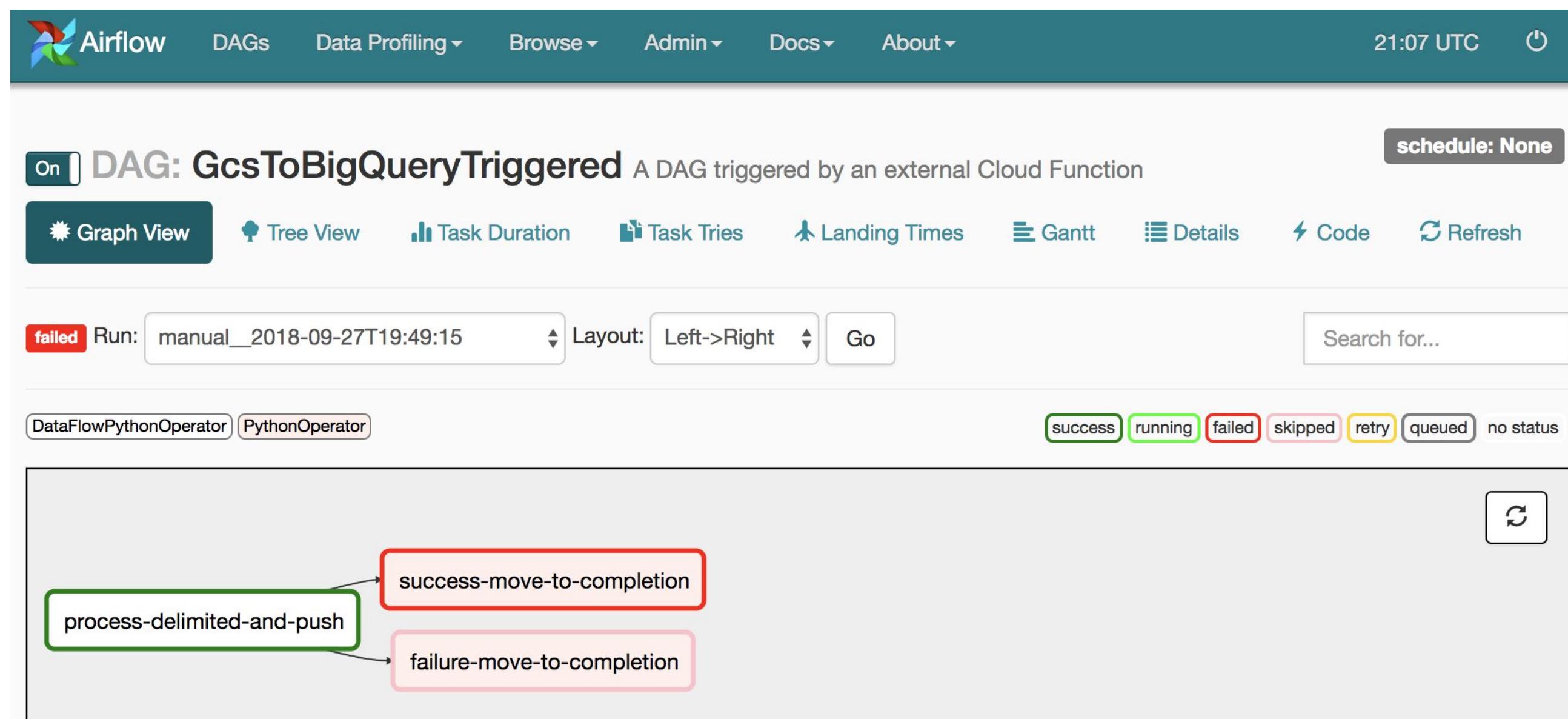
Which pipeline is healthier?

Quickly gauge pipeline health of DAG Runs

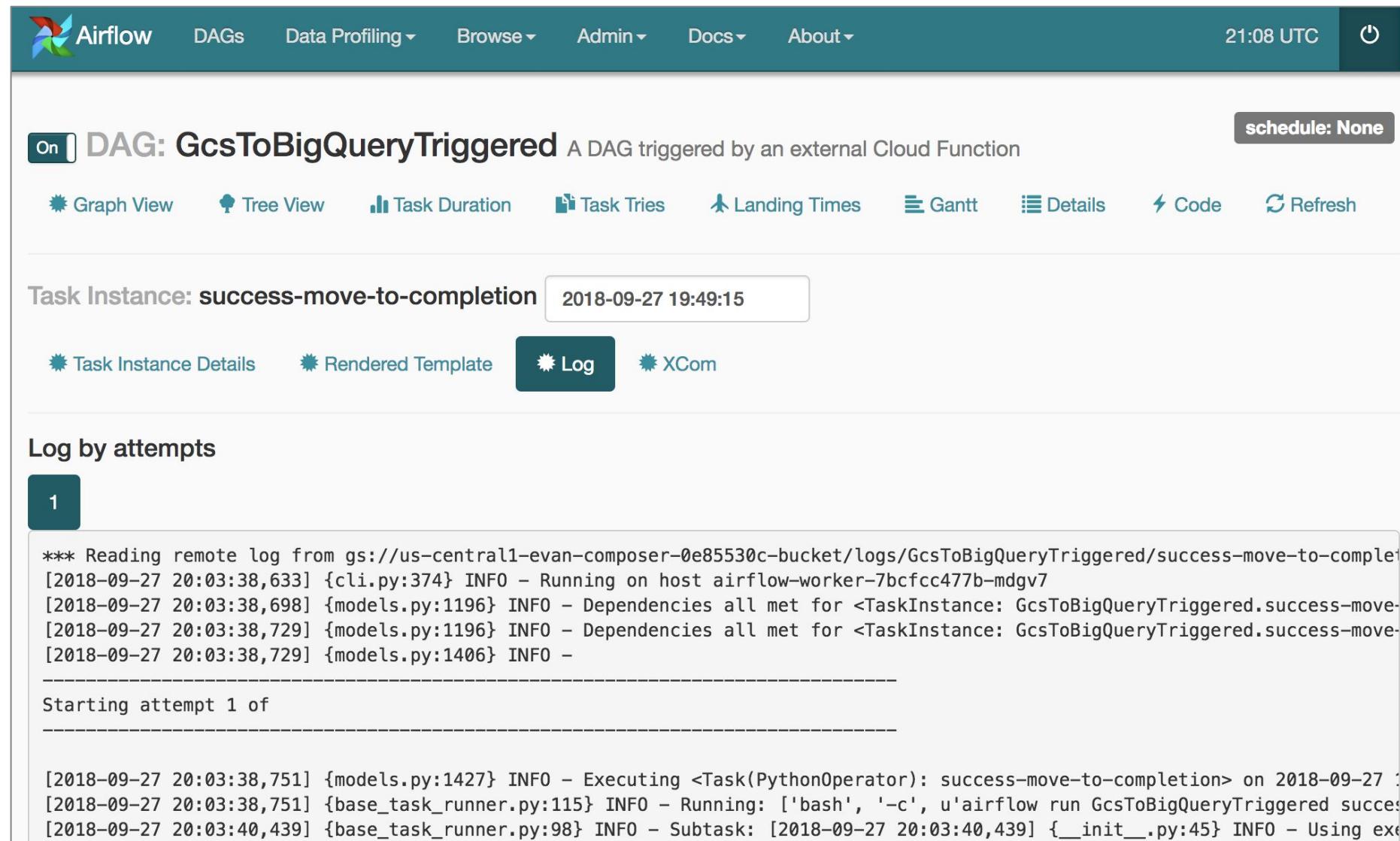
	i	DAG	Schedule	Owner	Recent Tasks i	Last Run i	DAG Runs i	Links
	On	GcsToBigQueryTriggered	None	Airflow	1 1 	2018-02-21 00:00 i	 268	
	On	composer_sample_simple_greeting	1 day, 0:00:00	Airflow	11 4 4	2018-02-21 00:00 i	40 12 	

Showing 1 to 2 of 2 entries

Quickly gauge pipeline health of DAG Runs



Monitor and troubleshoot Airflow step errors in logs

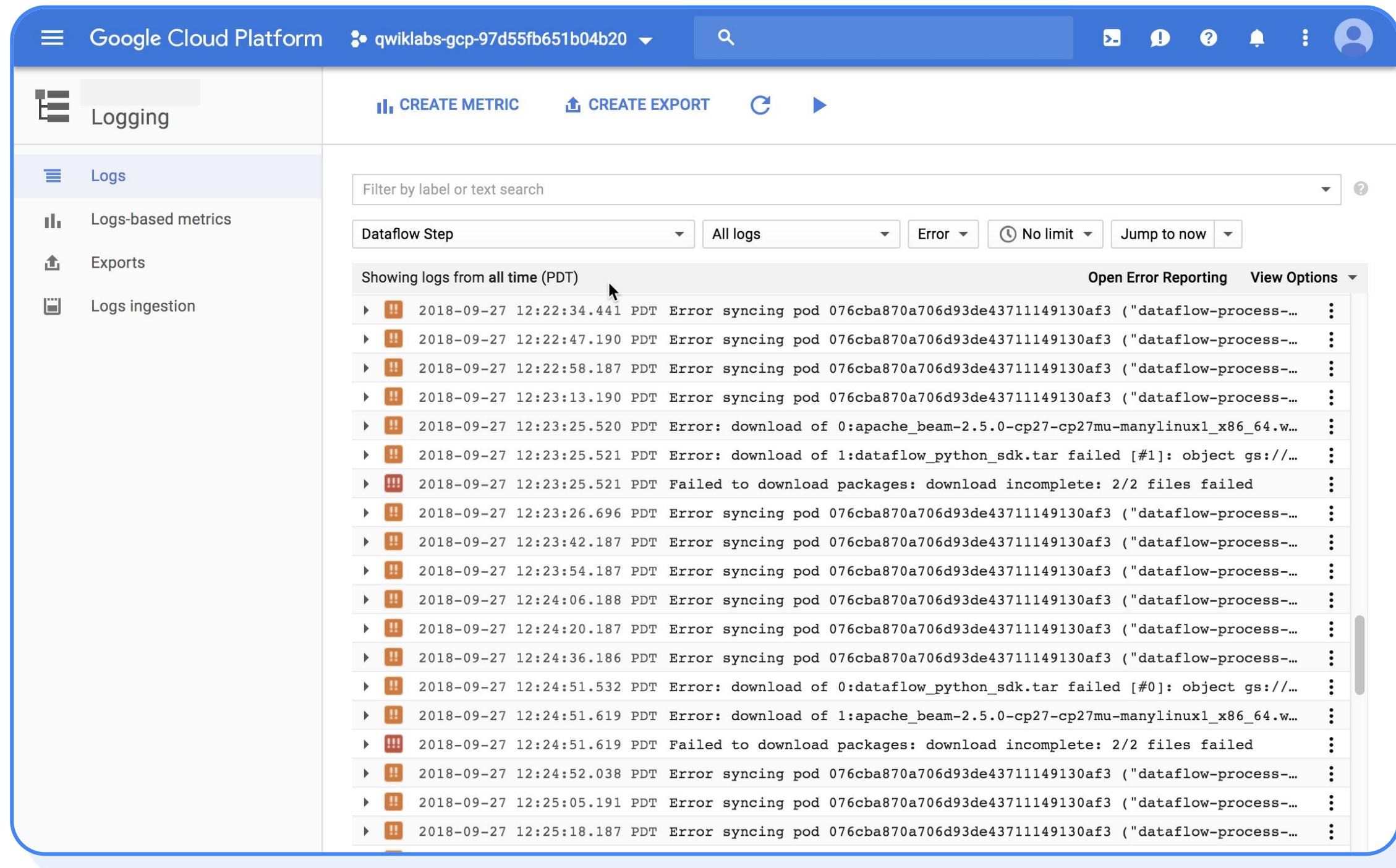


The screenshot shows the Airflow web interface for a DAG named "GcsToBigQueryTriggered". The task instance is "success-move-to-completion" from 2018-09-27 19:49:15. The "Log" tab is selected. The log output shows the task starting attempt 1, reading from a remote log, and executing the task. The final line of the log is cut off.

```
*** Reading remote log from gs://us-central1-evan-composer-0e85530c-bucket/logs/GcsToBigQueryTriggered/success-move-to-complet
[2018-09-27 20:03:38,633] {cli.py:374} INFO - Running on host airflow-worker-7bcfcc477b-mdgv7
[2018-09-27 20:03:38,698] {models.py:1196} INFO - Dependencies all met for <TaskInstance: GcsToBigQueryTriggered.success-move-
[2018-09-27 20:03:38,729] {models.py:1196} INFO - Dependencies all met for <TaskInstance: GcsToBigQueryTriggered.success-move-
[2018-09-27 20:03:38,729] {models.py:1406} INFO -
Starting attempt 1 of
[2018-09-27 20:03:38,751] {models.py:1427} INFO - Executing <Task(PythonOperator): success-move-to-completion> on 2018-09-27 20:03:38,751
[2018-09-27 20:03:38,751] {base_task_runner.py:115} INFO - Running: ['bash', '-c', u'airflow run GcsToBigQueryTriggered success-
[2018-09-27 20:03:40,439] {base_task_runner.py:98} INFO - Subtask: [2018-09-27 20:03:40,439] {__init__.py:45} INFO - Using exec
```

```
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/'">
```

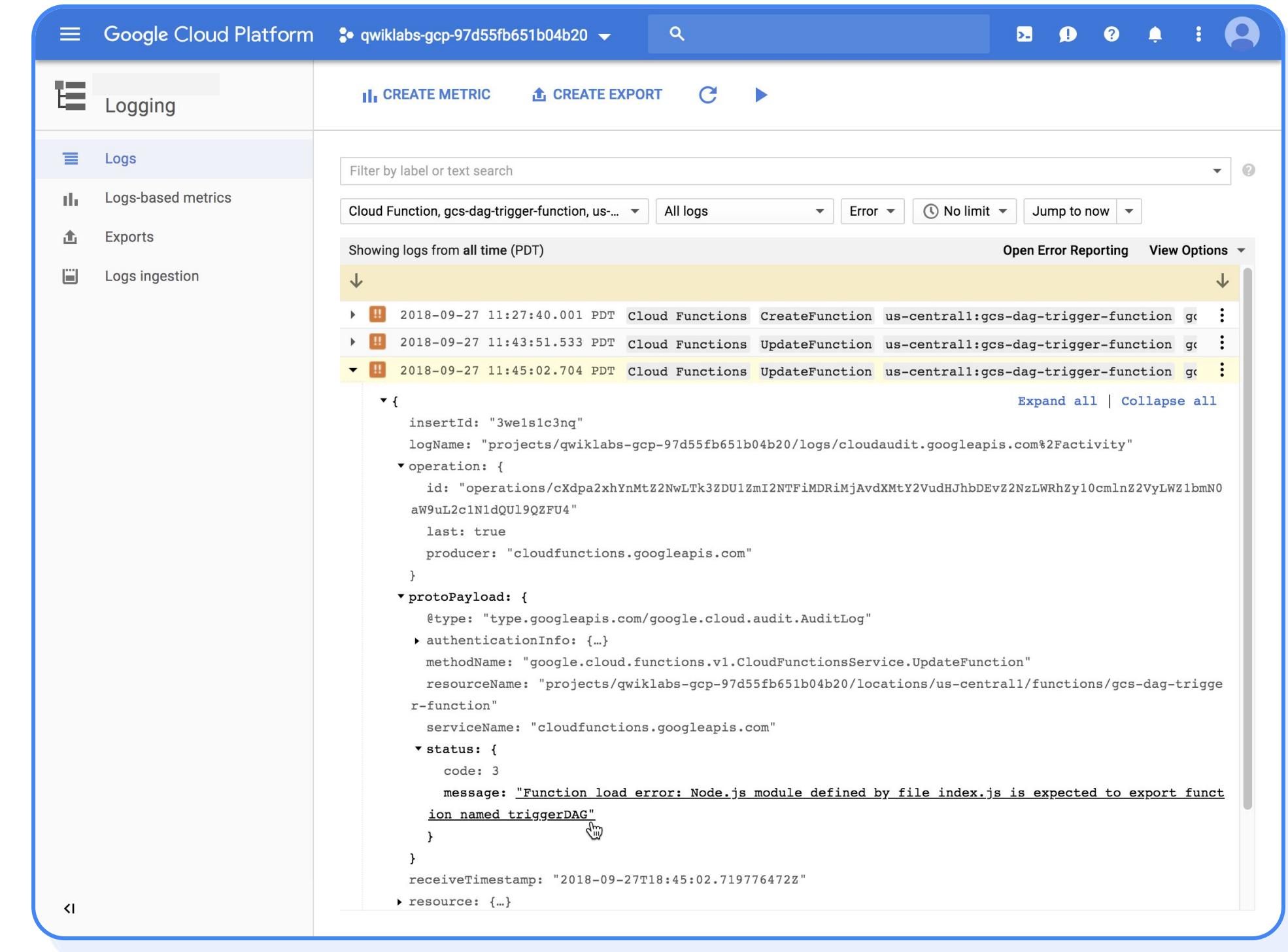
Monitor Dataflow health in Cloud Logging



The screenshot shows the Google Cloud Platform Logging interface. The left sidebar has a 'Logs' section with options for 'Logs-based metrics', 'Exports', and 'Logs ingestion'. The main area displays logs from a 'Dataflow Step'. A filter bar at the top allows setting the 'Dataflow Step' to 'All logs', 'Error', and 'No limit', with a 'Jump to now' button. The log list shows numerous error messages from September 27, 2018, all related to 'syncing pod' and 'Failed to download packages'.

Date	Time	Message
2018-09-27	12:22:34.441	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:22:47.190	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:22:58.187	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:23:13.190	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:23:25.520	Error: download of 0:apache_beam-2.5.0-cp27-cp27mu-manylinux1_x86_64.w...
2018-09-27	12:23:25.521	Error: download of 1:dataflow_python_sdk.tar failed [#1]: object gs://...
2018-09-27	12:23:25.521	Failed to download packages: download incomplete: 2/2 files failed
2018-09-27	12:23:26.696	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:23:42.187	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:23:54.187	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:24:06.188	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:24:20.187	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:24:36.186	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:24:51.532	Error: download of 0:dataflow_python_sdk.tar failed [#0]: object gs://...
2018-09-27	12:24:51.619	Error: download of 1:apache_beam-2.5.0-cp27-cp27mu-manylinux1_x86_64.w...
2018-09-27	12:24:51.619	Failed to download packages: download incomplete: 2/2 files failed
2018-09-27	12:24:52.038	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:25:05.191	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")
2018-09-27	12:25:18.187	Error syncing pod 076cba870a706d93de43711149130af3 ("dataflow-process-...")

Monitor Cloud Function health in Cloud Logging



The screenshot shows the Google Cloud Platform Logging interface. The left sidebar has 'Logs' selected. The main area displays log entries for a Cloud Function named 'gcs-dag-trigger-function'. The logs are filtered by 'All logs' and 'Error' status, with no limit on time. The most recent log entry is expanded, showing detailed audit information. The log entry details the 'UpdateFunction' call from 'us-central1:gcs-dag-trigger-function' to 'operations/cXdpaxhYnMtZ2NwLTk3ZDU1ZmI2NTFiMDRiMjAvdXMtY2VudHJhbDEvZ2NzLWRhZy10cmLnZ2VyLWZ1bmN0aW9uL2c1N1dQUl9QZFU4'. The status code is 3, and the message indicates a 'Function load error: Node.js module defined by file index.js is expected to export function named triggerDAG'.

```
2018-09-27 11:45:02.704 PDT Cloud Functions UpdateFunction us-central1:gcs-dag-trigger-function gcs-dag-trigger-function
{
  insertId: "3we1s1c3nq"
  logName: "projects/qwiklabs-gcp-97d55fb651b04b20/logs/cloudaudit.googleapis.com%2Factivity"
  operation: {
    id: "operations/cXdpaxhYnMtZ2NwLTk3ZDU1ZmI2NTFiMDRiMjAvdXMtY2VudHJhbDEvZ2NzLWRhZy10cmLnZ2VyLWZ1bmN0aW9uL2c1N1dQUl9QZFU4"
    last: true
    producer: "cloudfunctions.googleapis.com"
  }
  protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    authenticationInfo: ...
    methodName: "google.cloud.functions.v1.CloudFunctionsService.UpdateFunction"
    resourceName: "projects/qwiklabs-gcp-97d55fb651b04b20/locations/us-central1/functions/gcs-dag-trigger-function"
    serviceName: "cloudfunctions.googleapis.com"
    status: {
      code: 3
      message: "Function load error: Node.js module defined by file index.js is expected to export function named triggerDAG"
    }
  }
  receiveTimestamp: "2018-09-27T18:45:02.719776472Z"
  resource: ...
}
```



Lab

An Introduction to Cloud Composer

Objectives

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

Module Summary

Cloud Data Fusion for building data pipelines.

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