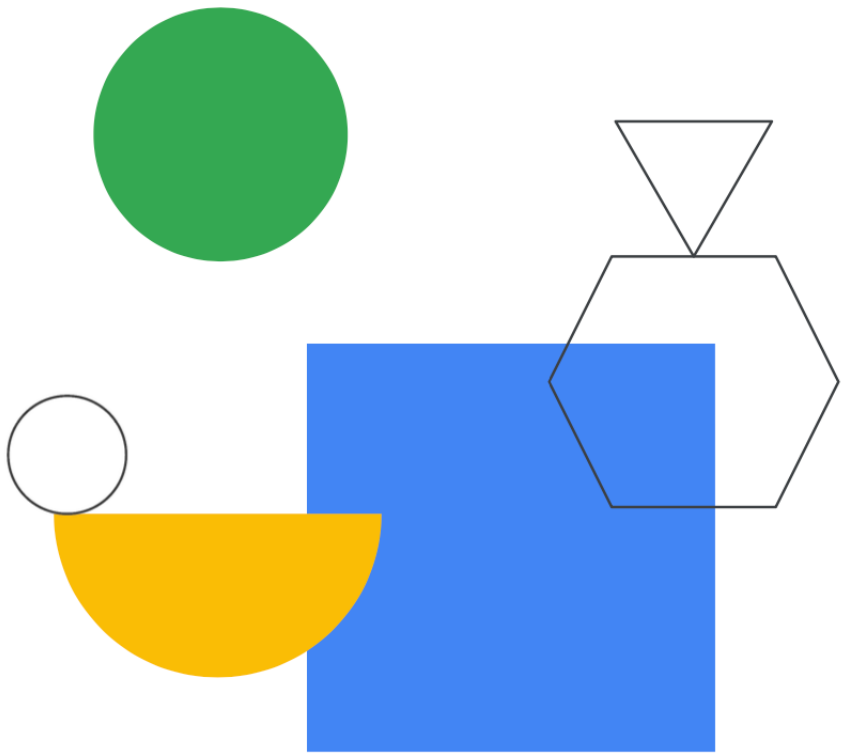
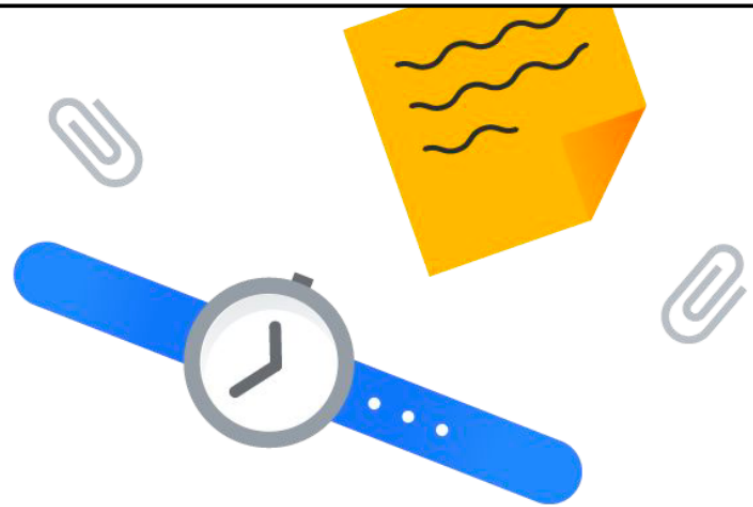
Google Cloud

Manage Data Pipelines with Cloud Data Fusion and Cloud Composer





Module agenda

01

Building Batch Data Pipelines Visually with Cloud Data Fusion

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

02

Orchestrating Work Between Google Cloud Services with Cloud Composer

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

Building Batch Data Pipelines Visually with 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.

Developer, data scientist, and business analyst

1

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

2

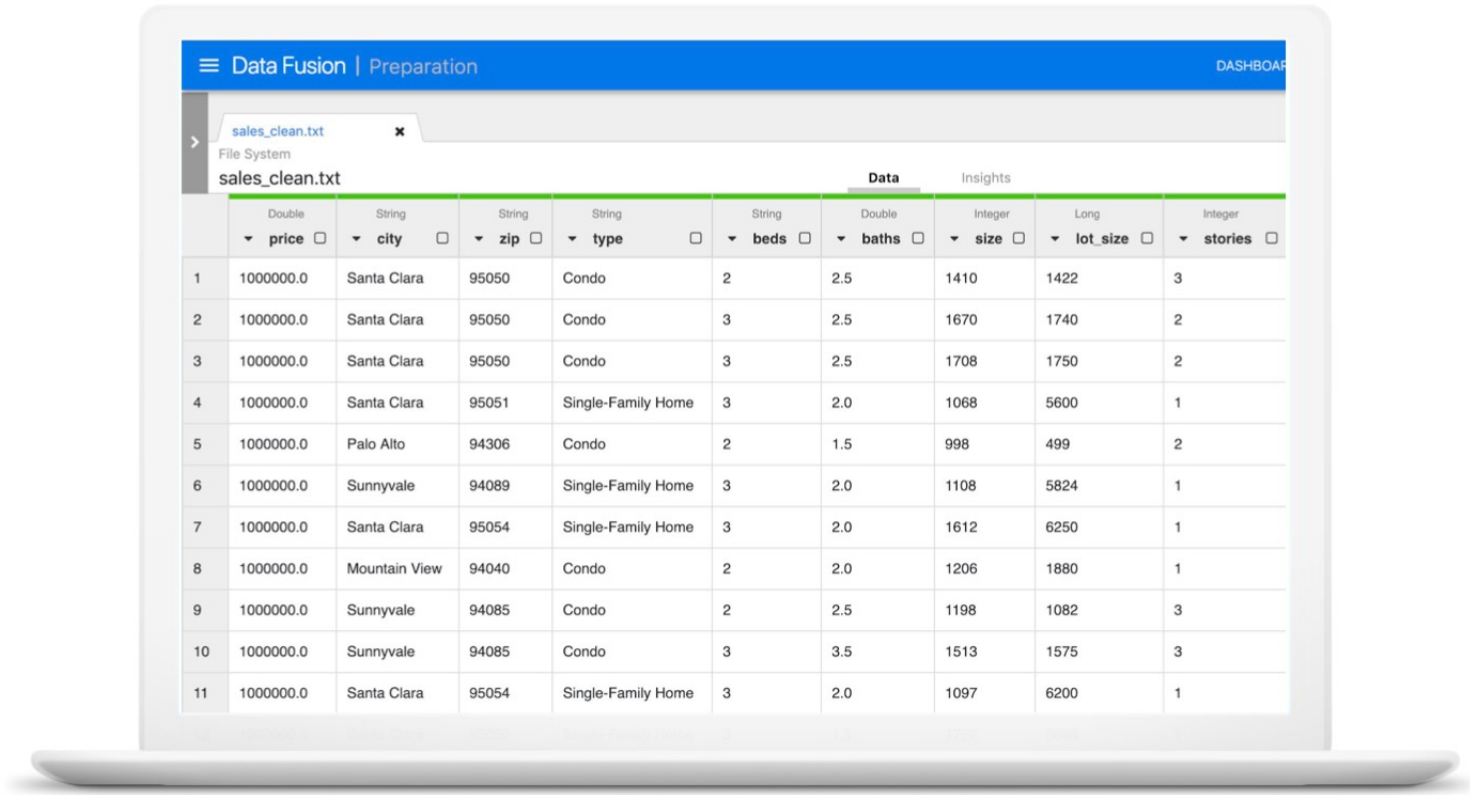
Use Cloud Data Fusion to visually build integration pipeline, test, debug and deploy.

3

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

Benefits

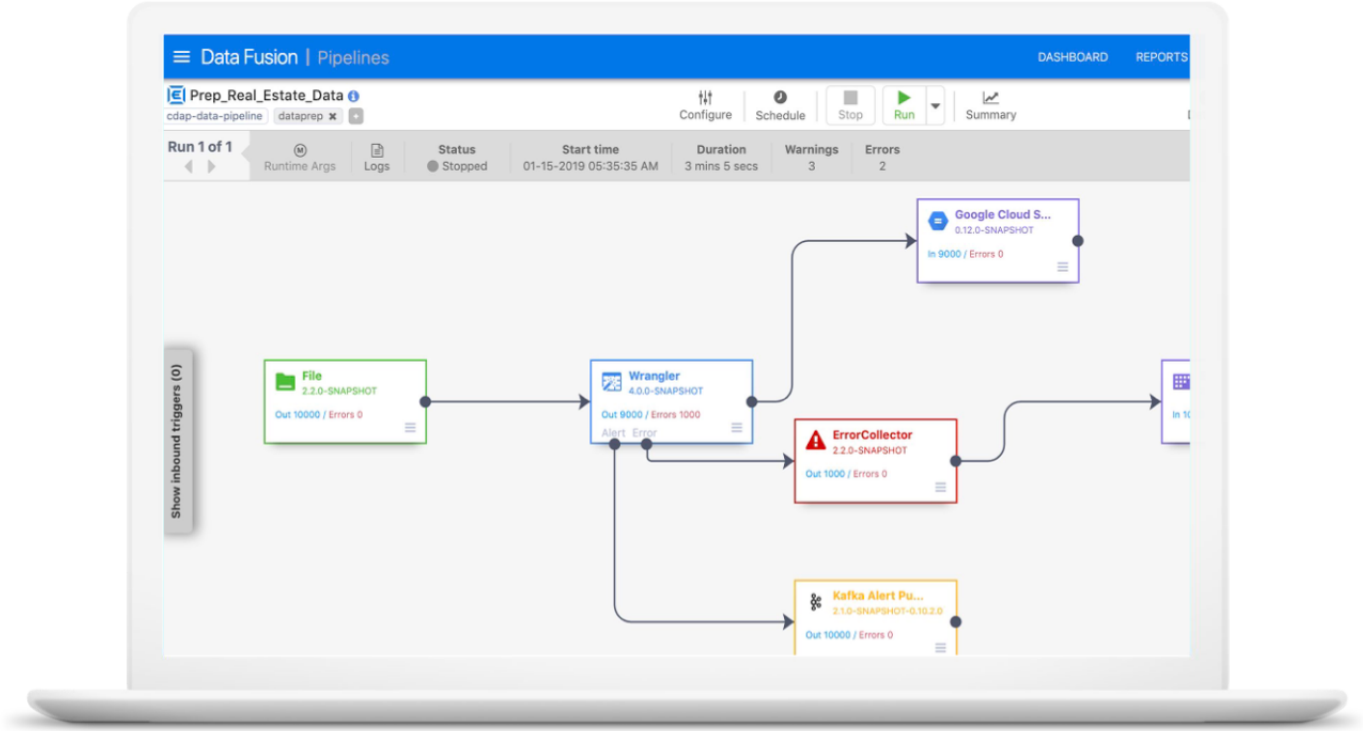
- Integrate with any data
- Increase productivity
- Reduce complexity
- Increase flexibility



	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, and actions
- Code free visual transformations
- 1000+ transforms, data quality
- Test and debug pipeline
- Pre-built pipelines
- Developer SDK



Integration metadata

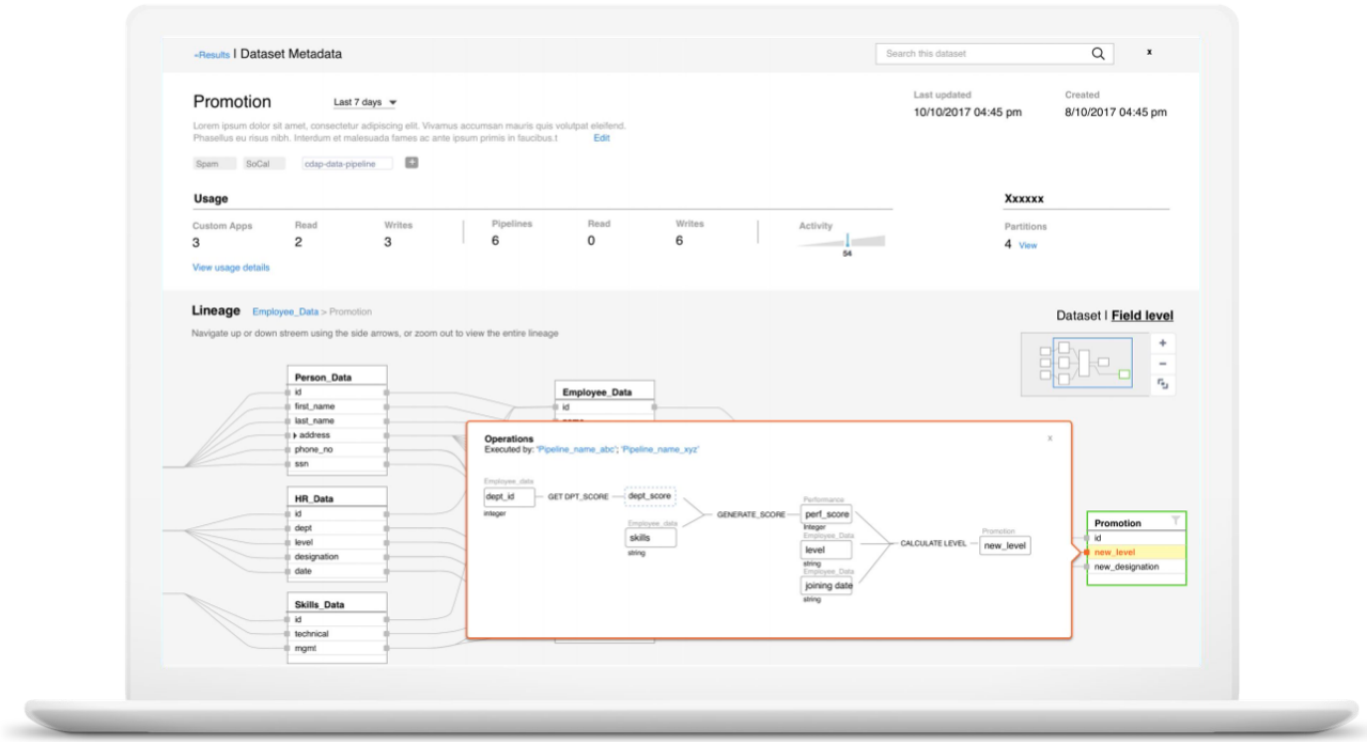
Tags and Properties support

- Pipeline
- Dataset
- Schema

Search integrated entities by

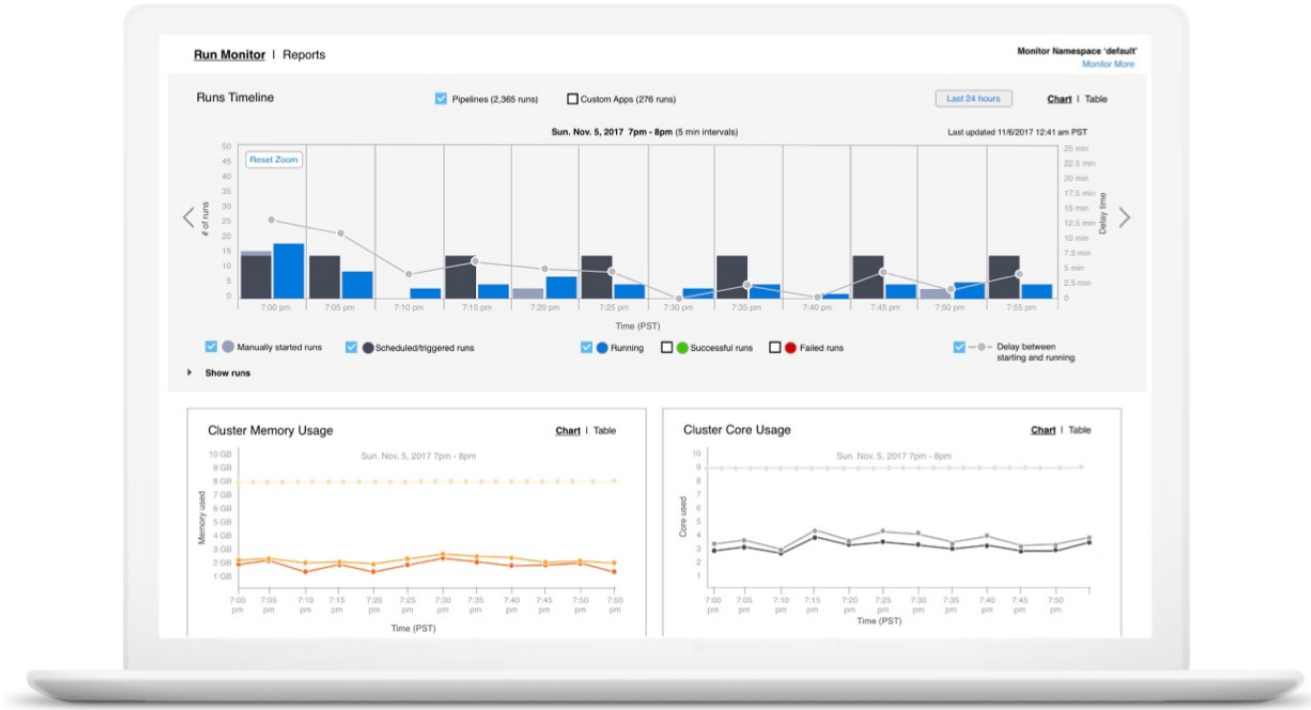
- Keyword
- Schema name and type

Dataset level and Field level lineage

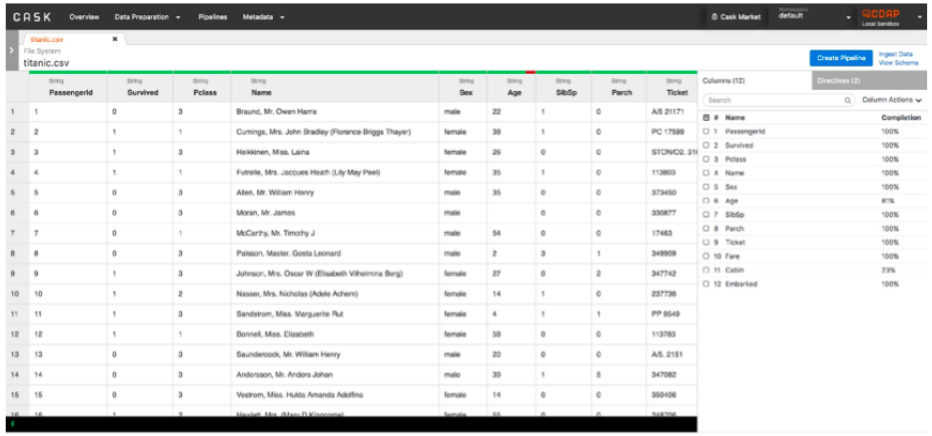


Extensible

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



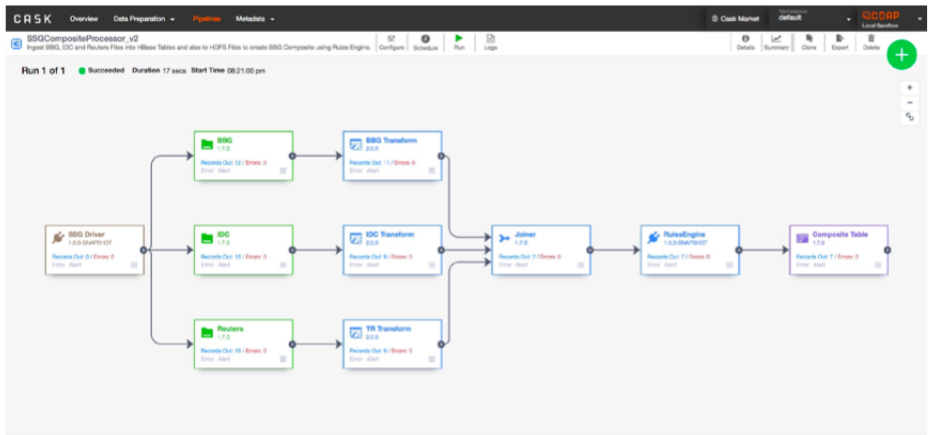
Components of Cloud Data Fusion



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.**

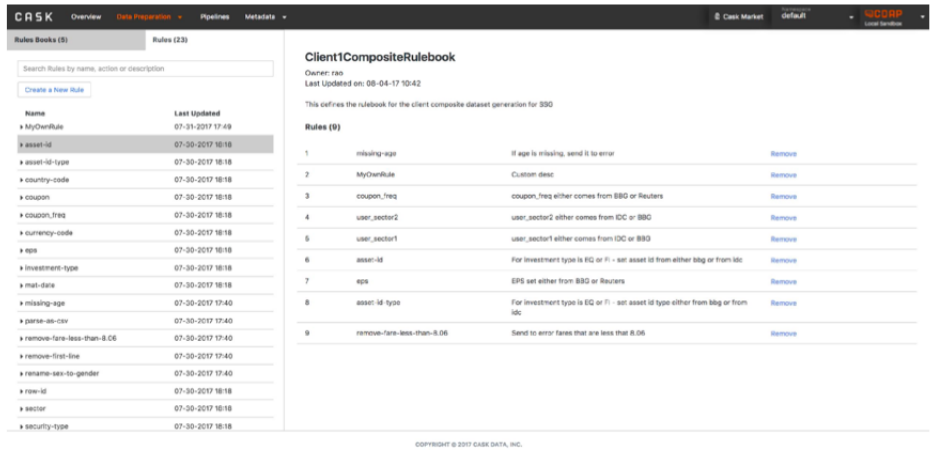


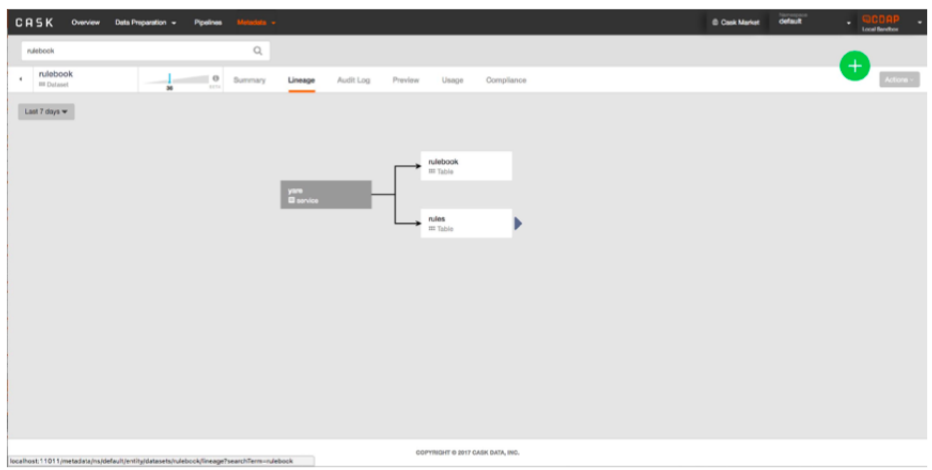
02

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.**

Components of Cloud Data Fusion





03

Rules Engine — Tool

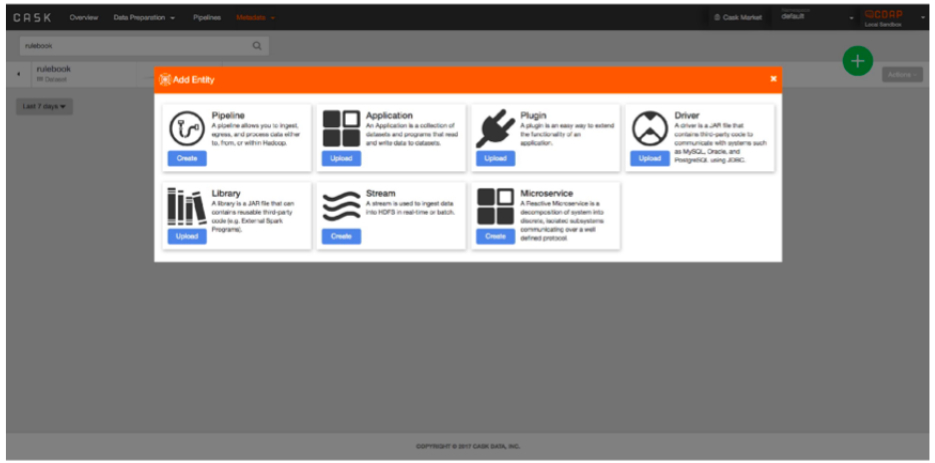
- Business Data Transformations and checks codified for business users.
- Define Complex rules using intuitive and simple to use user interface.
- Logically group Rules in Rulebook and trigger or schedule processing.
- Integrates with Data Pipeline for operationalizing Rules.

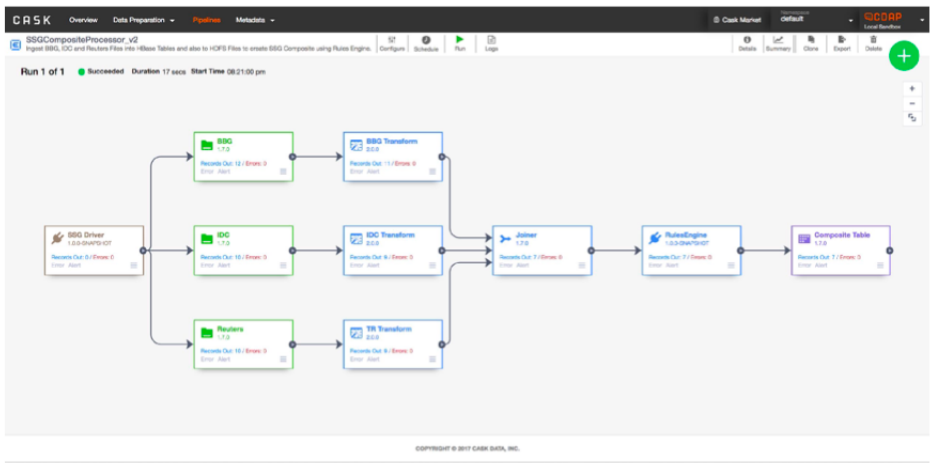
04

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.

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.

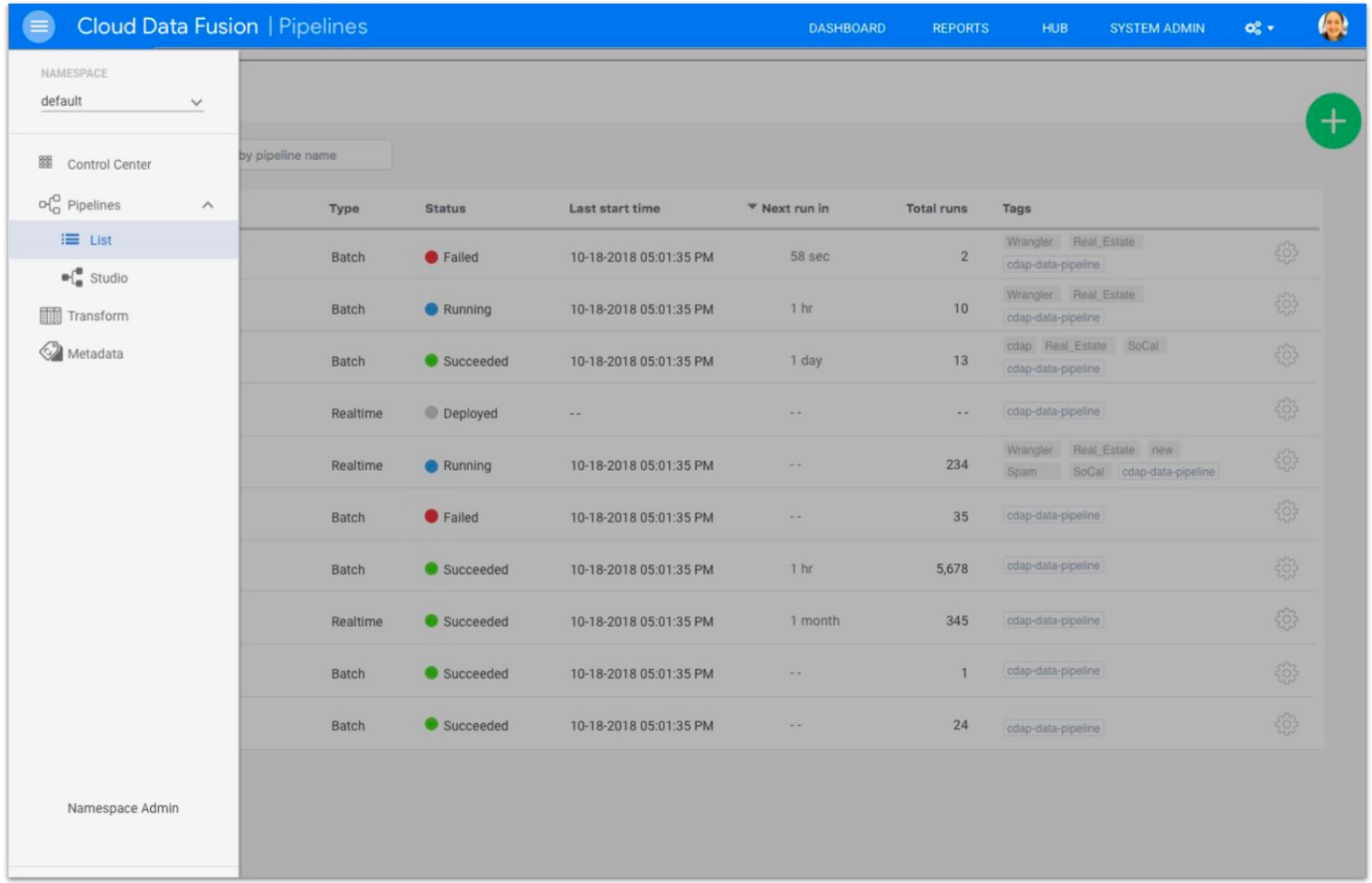
06

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.

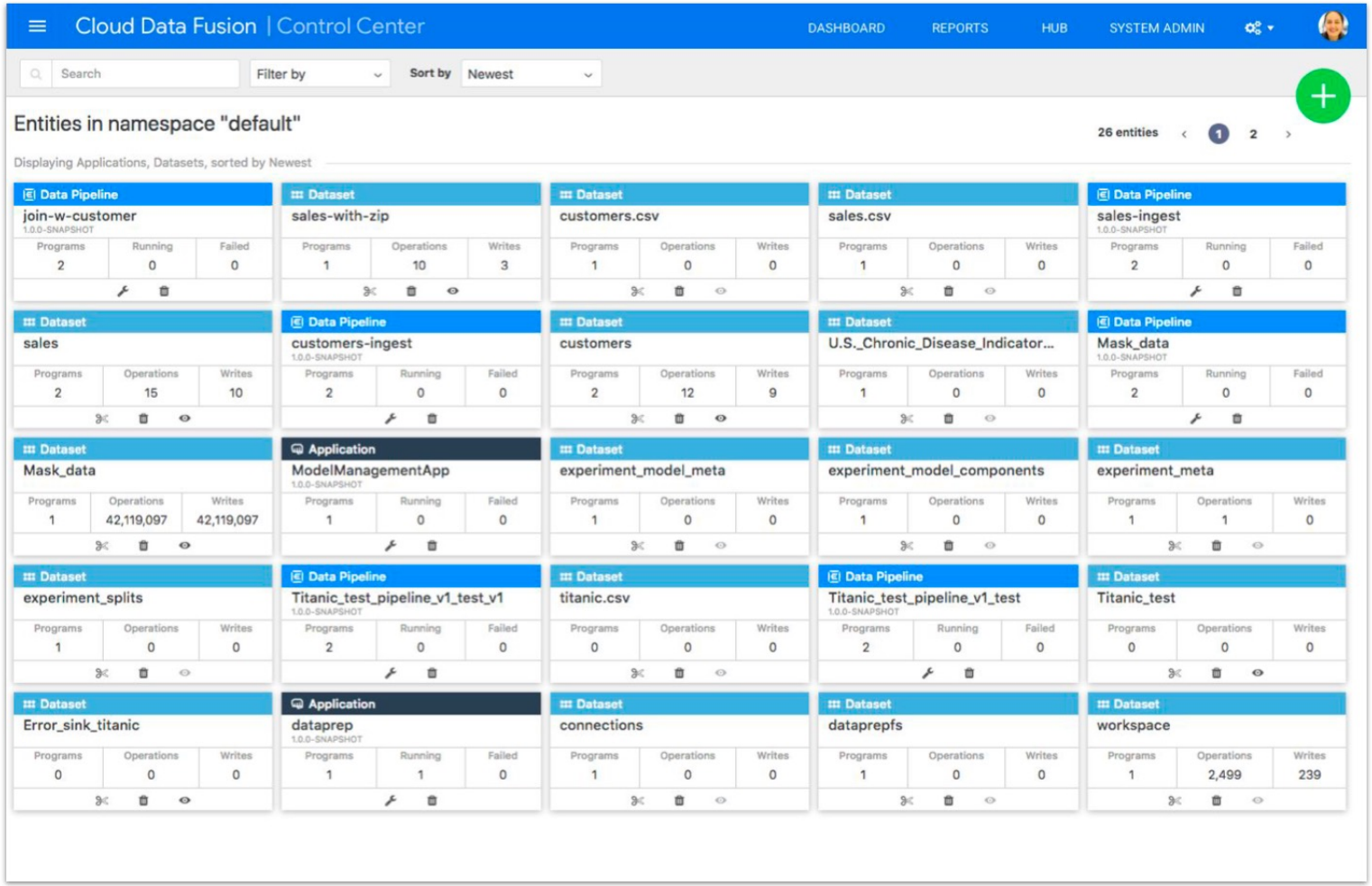
Cloud Data Fusion - UI overview

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



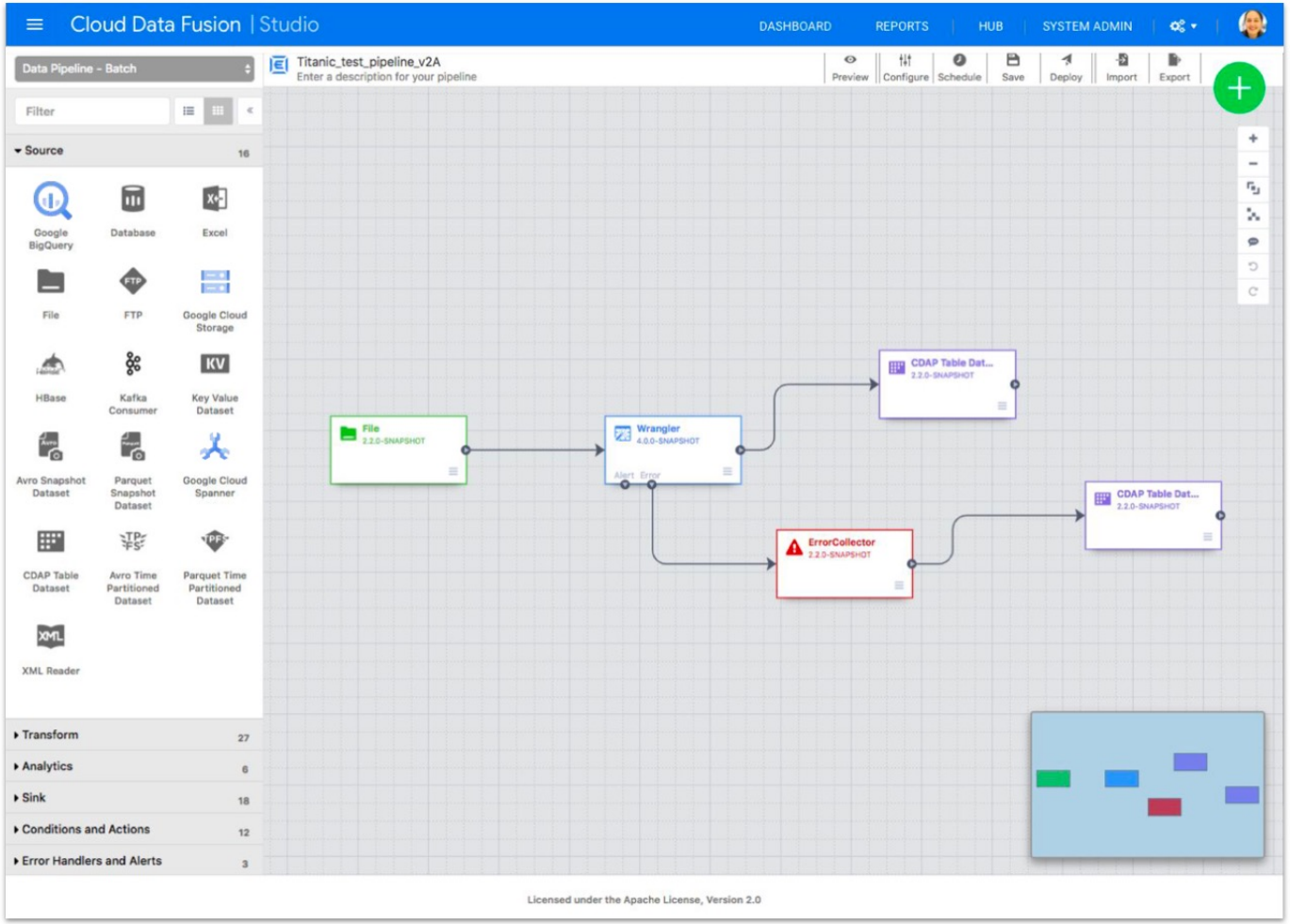
Control Center

- Application
- Artifact
- Dataset



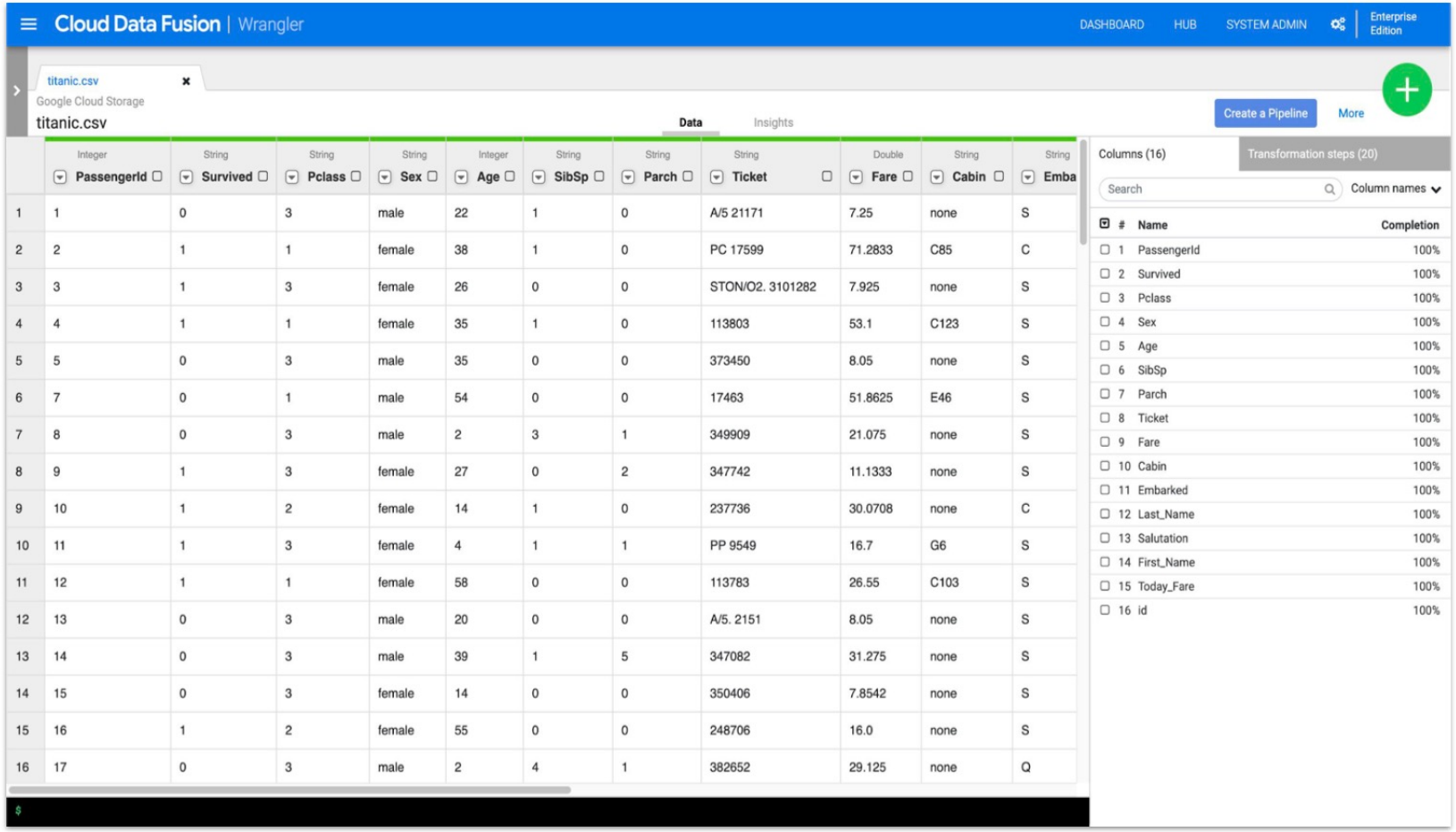
Pipelines

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



Wrangler

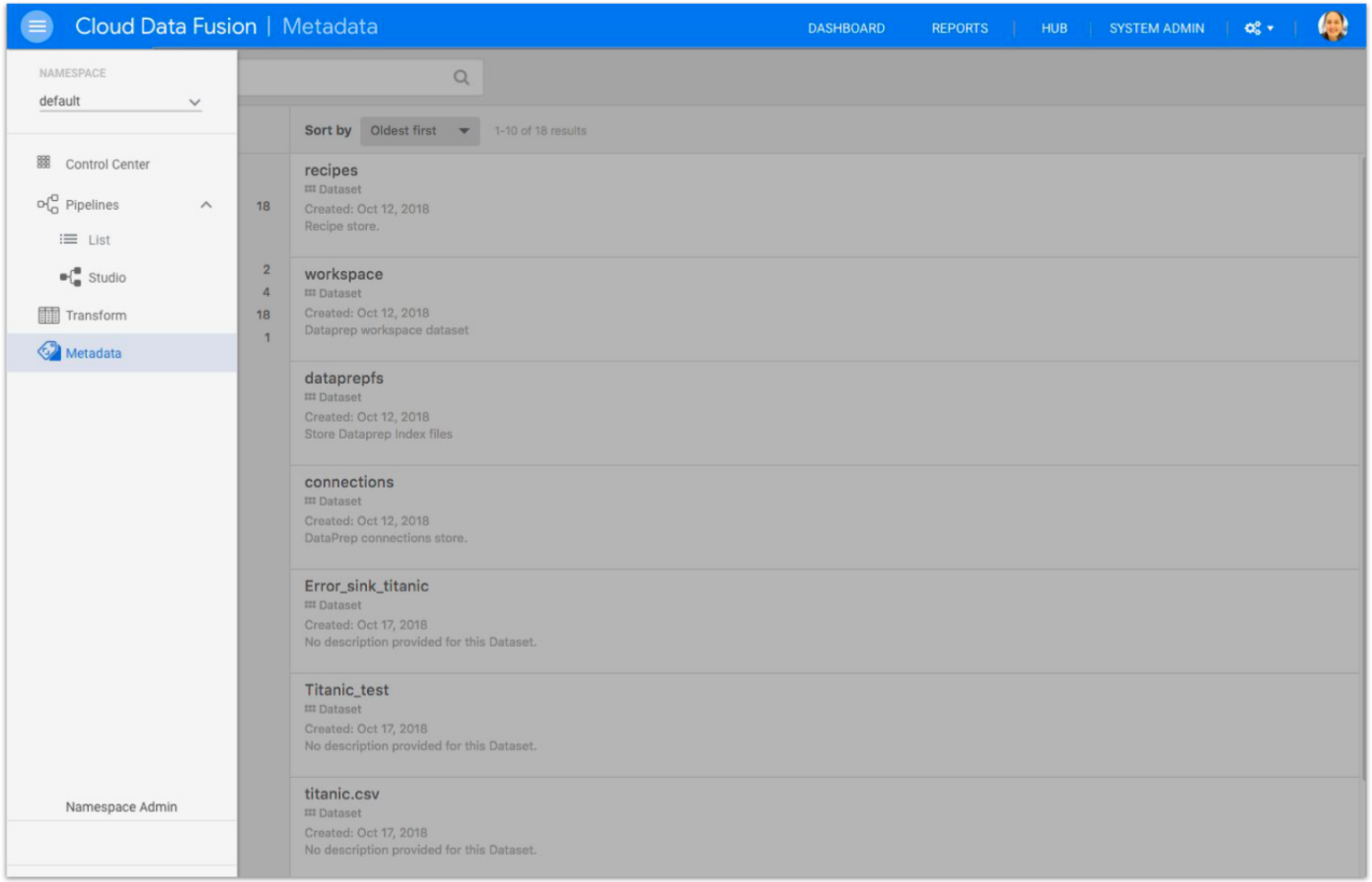
- Connections
- Transforms
- Data Quality
- Insights
- Functions



	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
	Integer	String	String	String	Integer	String	String	String	Double	String	String
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

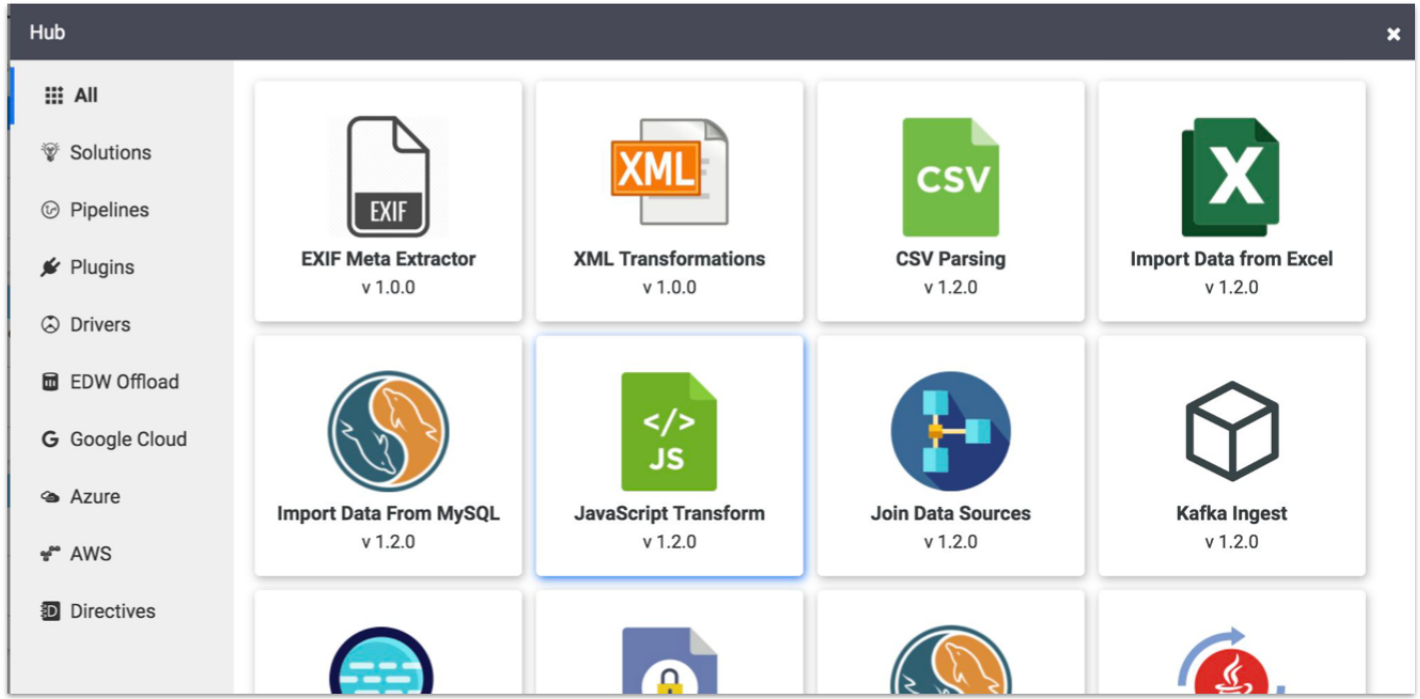
Integration metadata

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



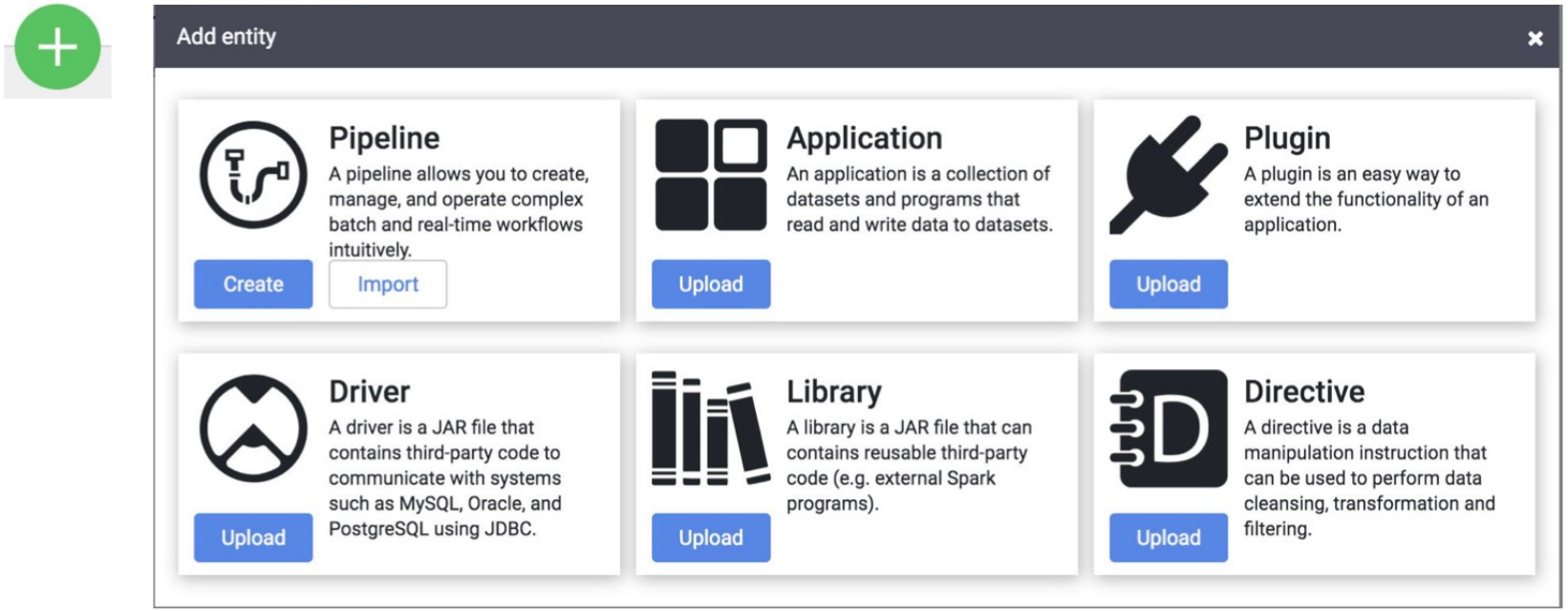
Hub

- Plugins
- Use cases
- Pre-built pipelines



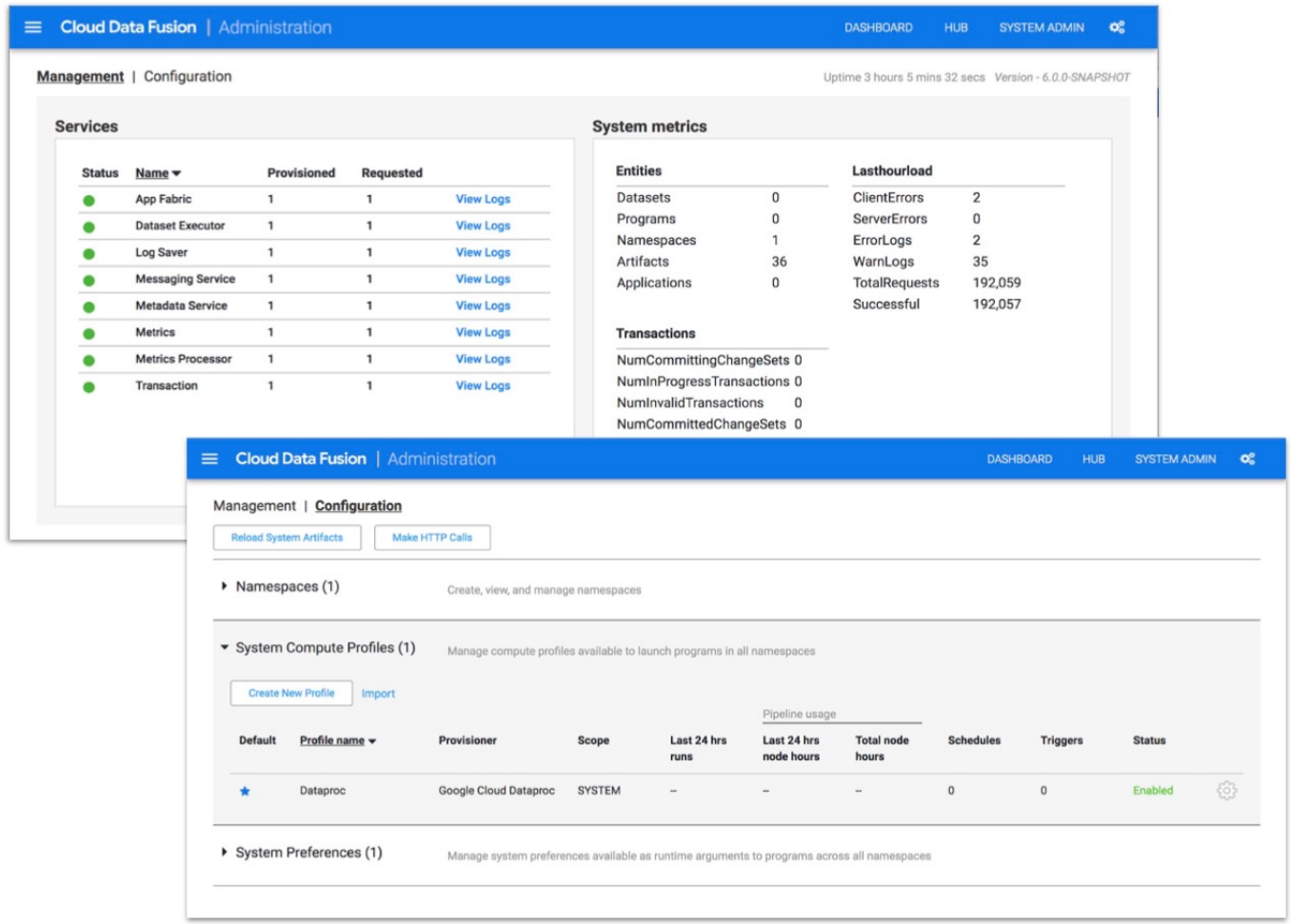
Entities

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

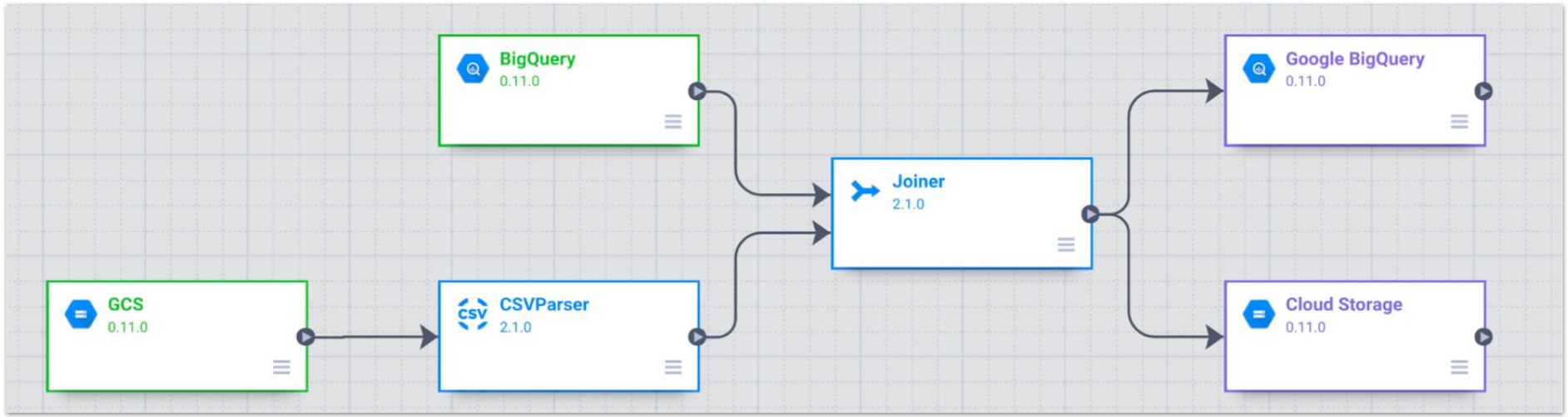


Administration

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



Data Pipeline | Directed Acyclic Graph (DAG)

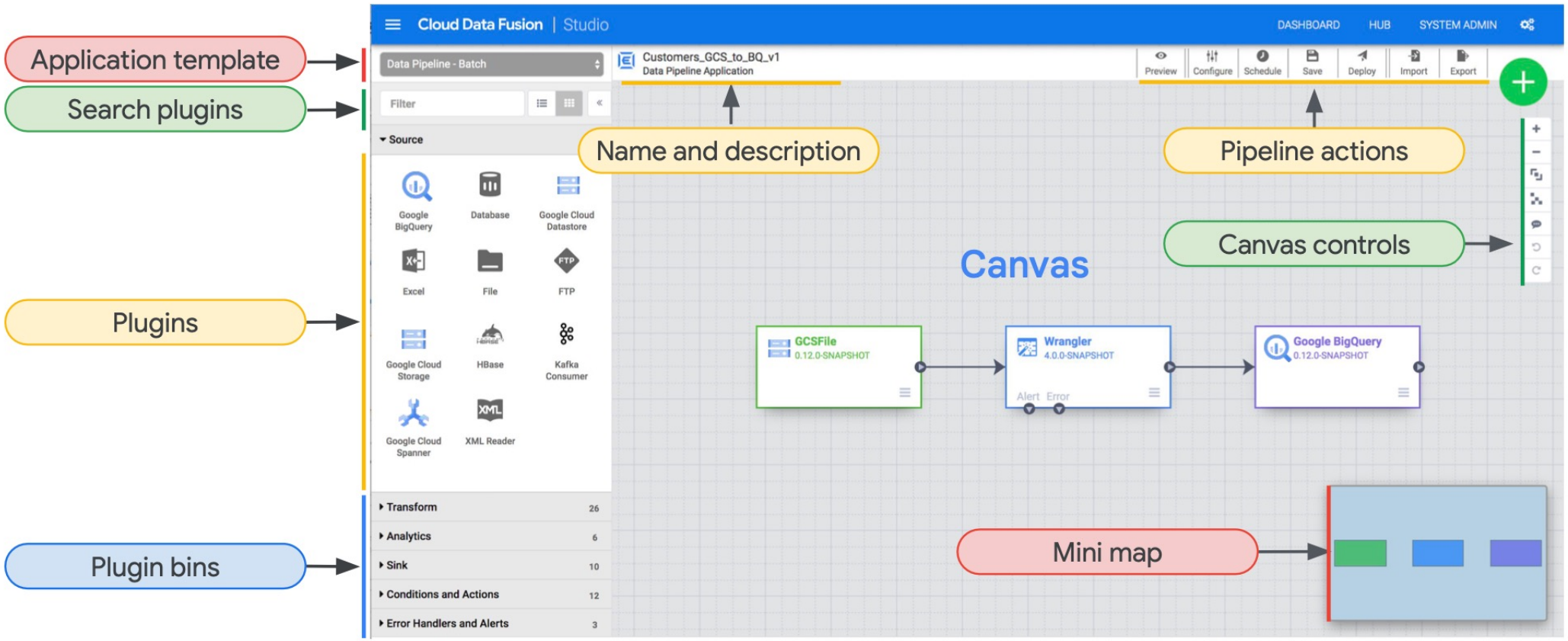


- Represented by a series of stages arranged in a DAG. This forms a one-way pipeline.
- Stages, which are the "nodes" in the pipeline graph, can be of different types.

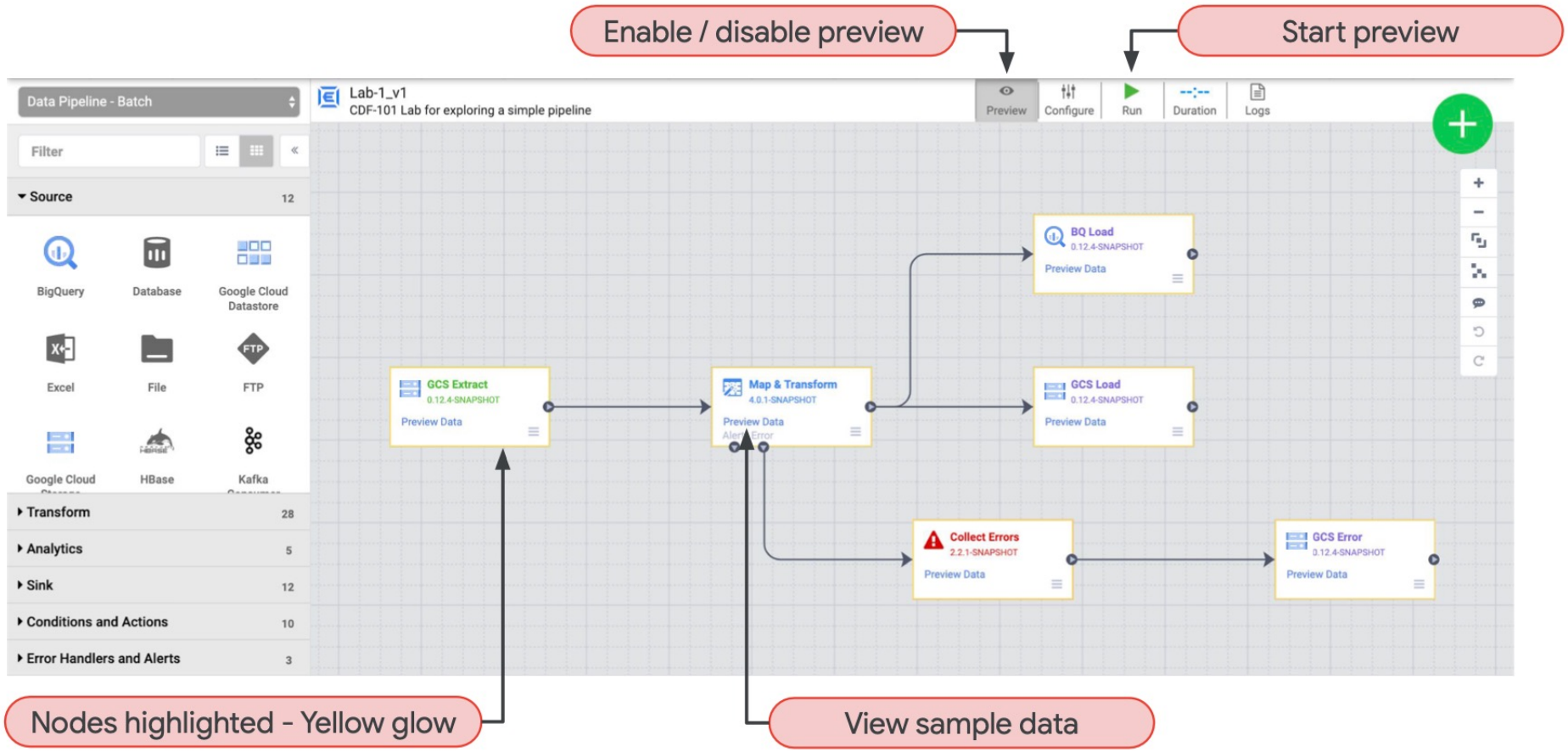
Data pipeline

- 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.

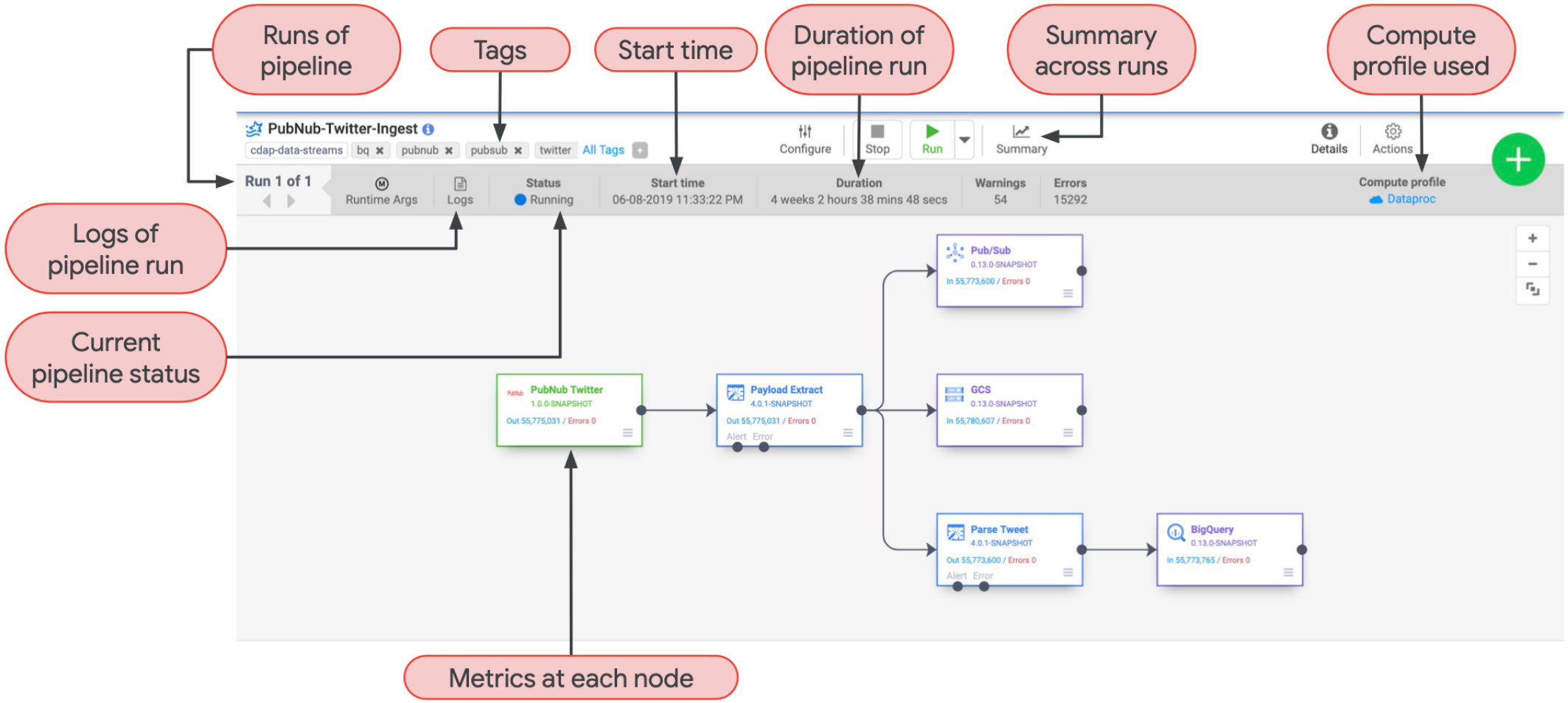
Studio is the UI where you create new pipelines



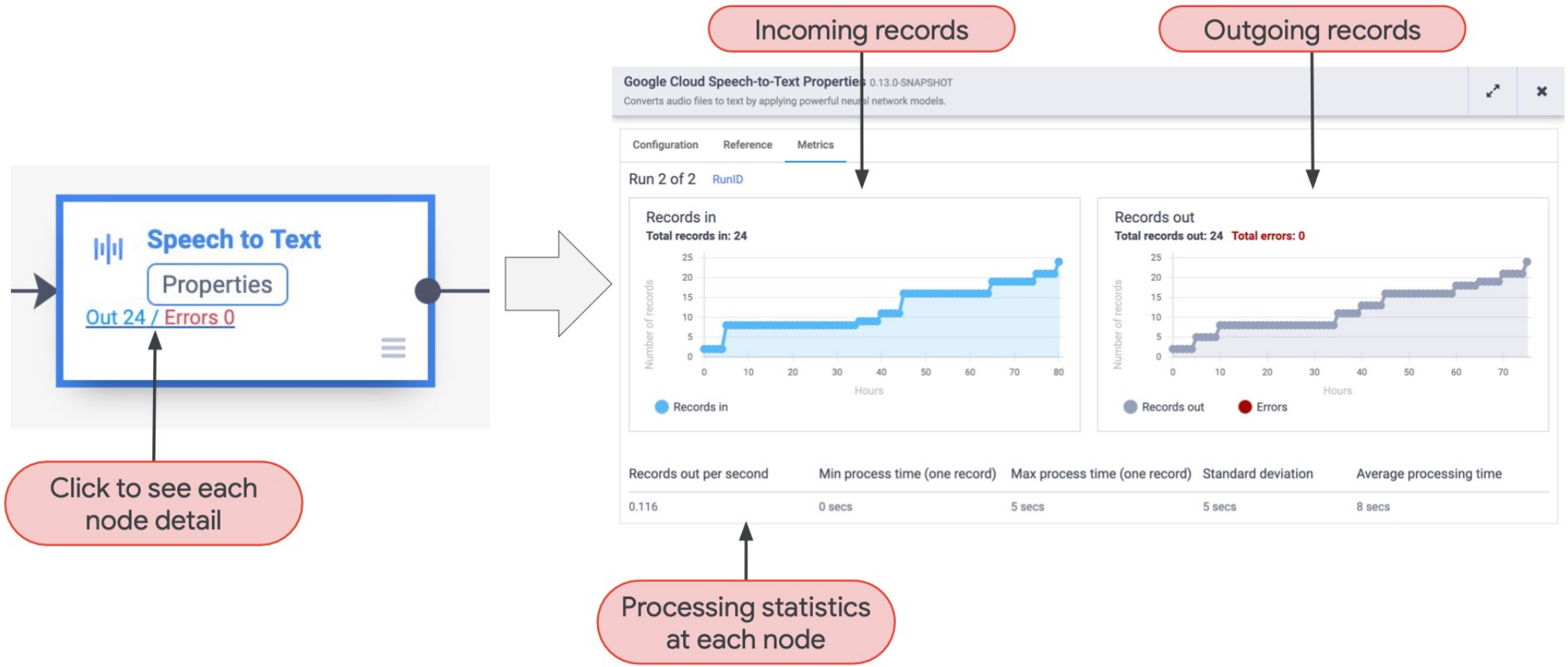
Use Preview Mode to see how the pipeline will run



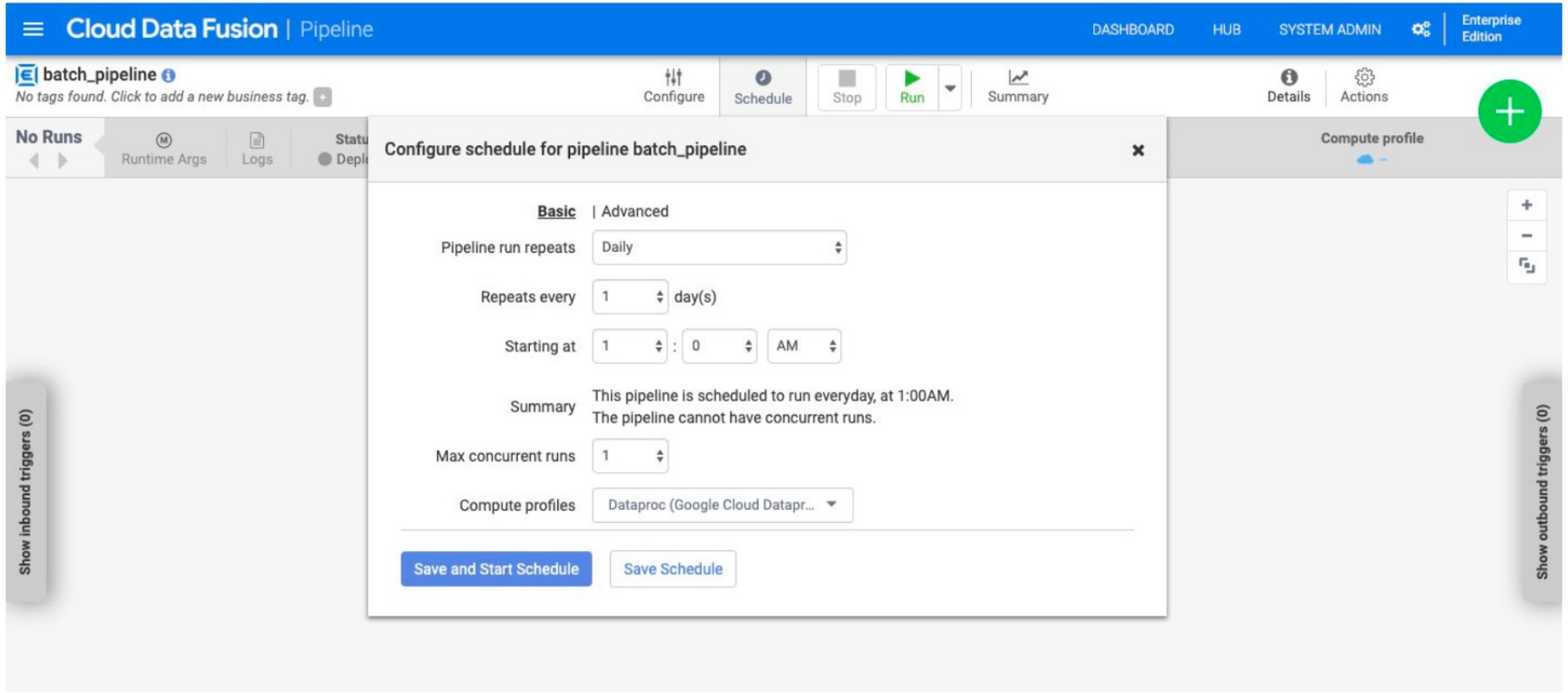
Monitor the health of the entire pipeline



Monitor the health of a single node

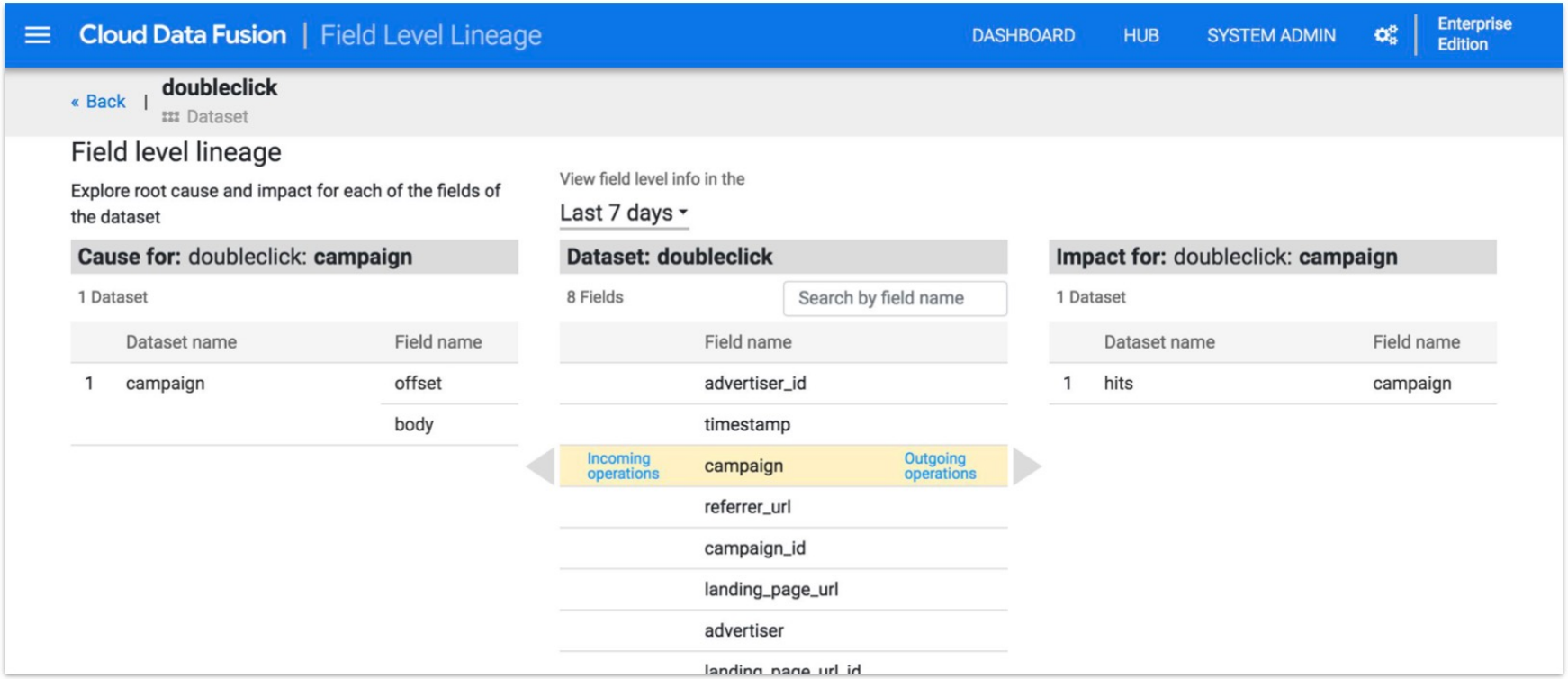


You can schedule batch pipelines



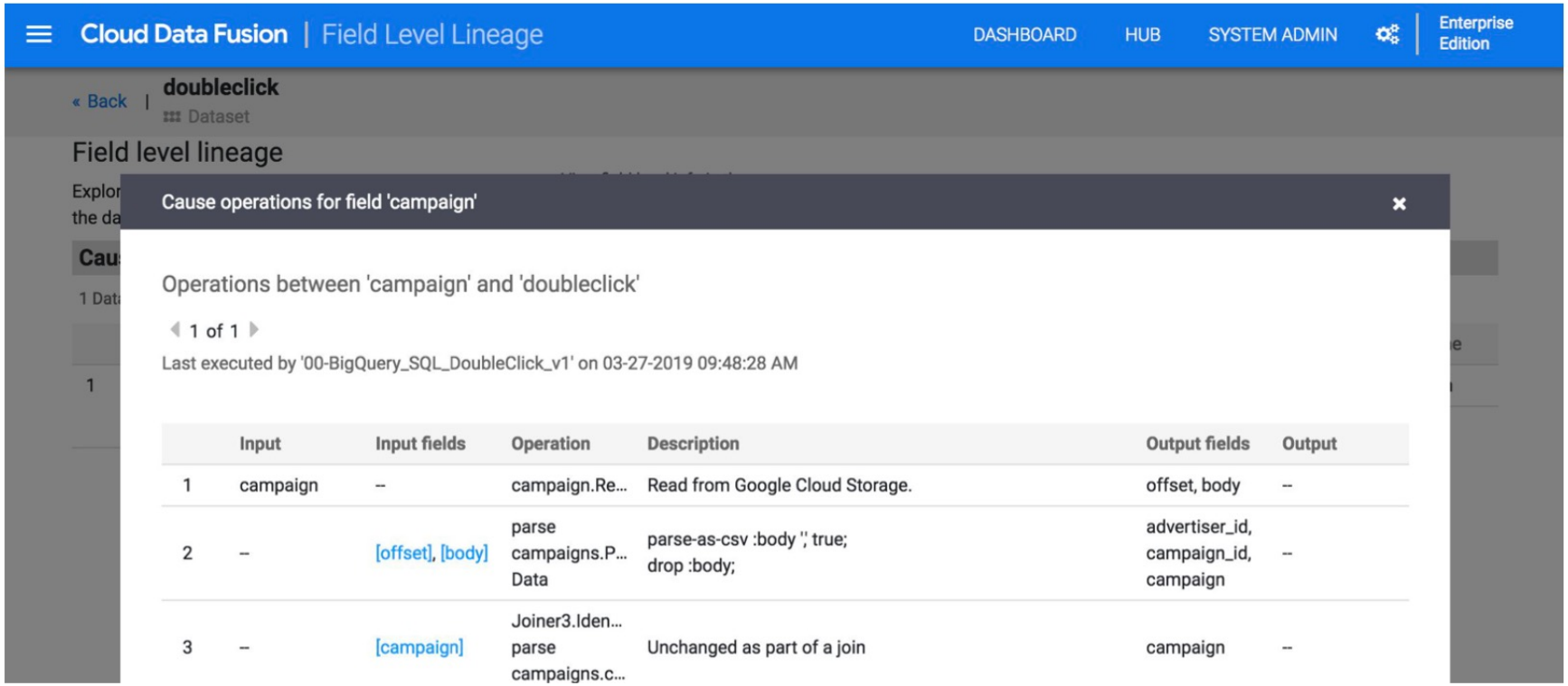
After data is transformed, you can track field-level lineage

Relationship between fields of datasets: Provenance, Impact



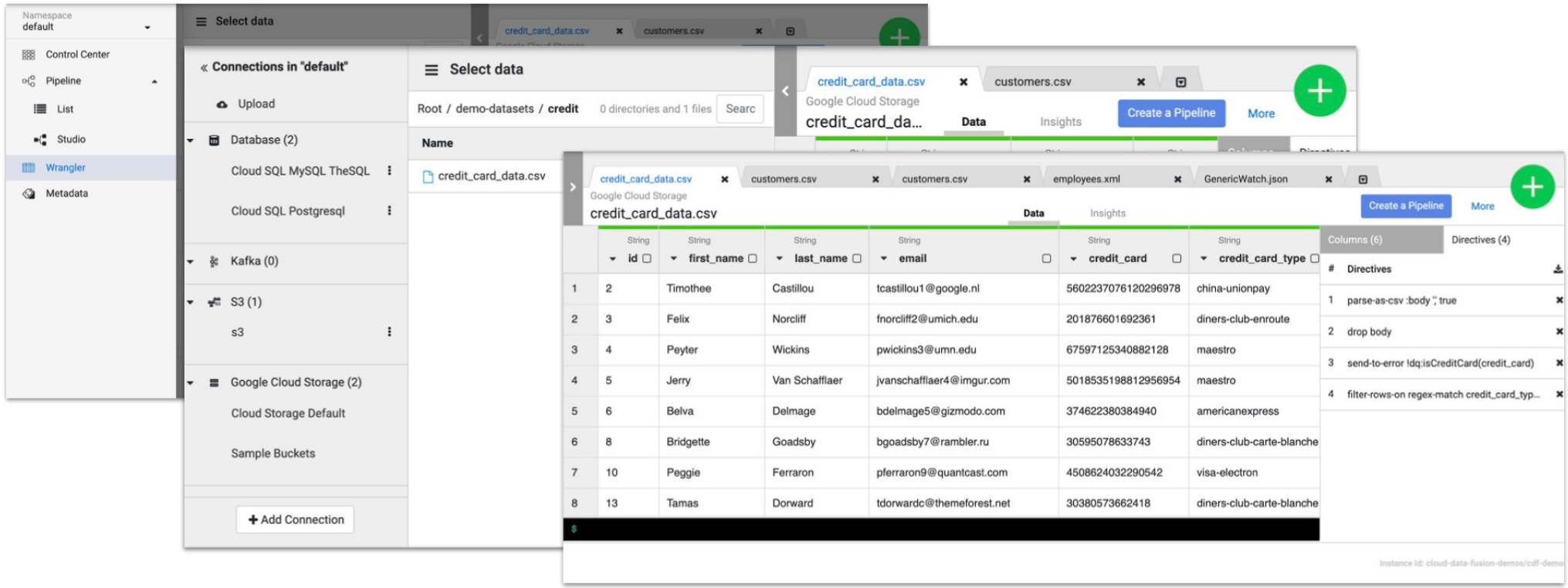
See every operation that is made on a field

Operations applied to a field

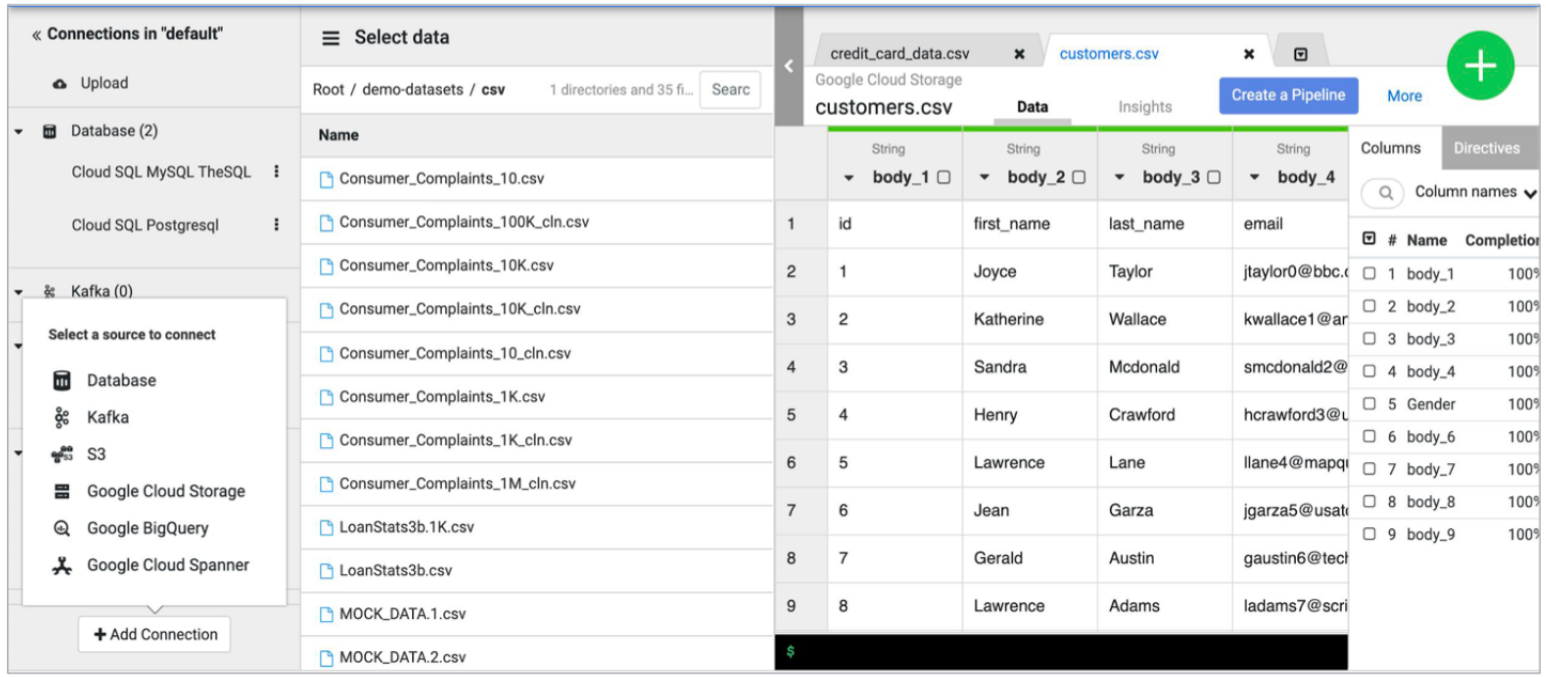


Data analysts can explore datasets in the Wrangler

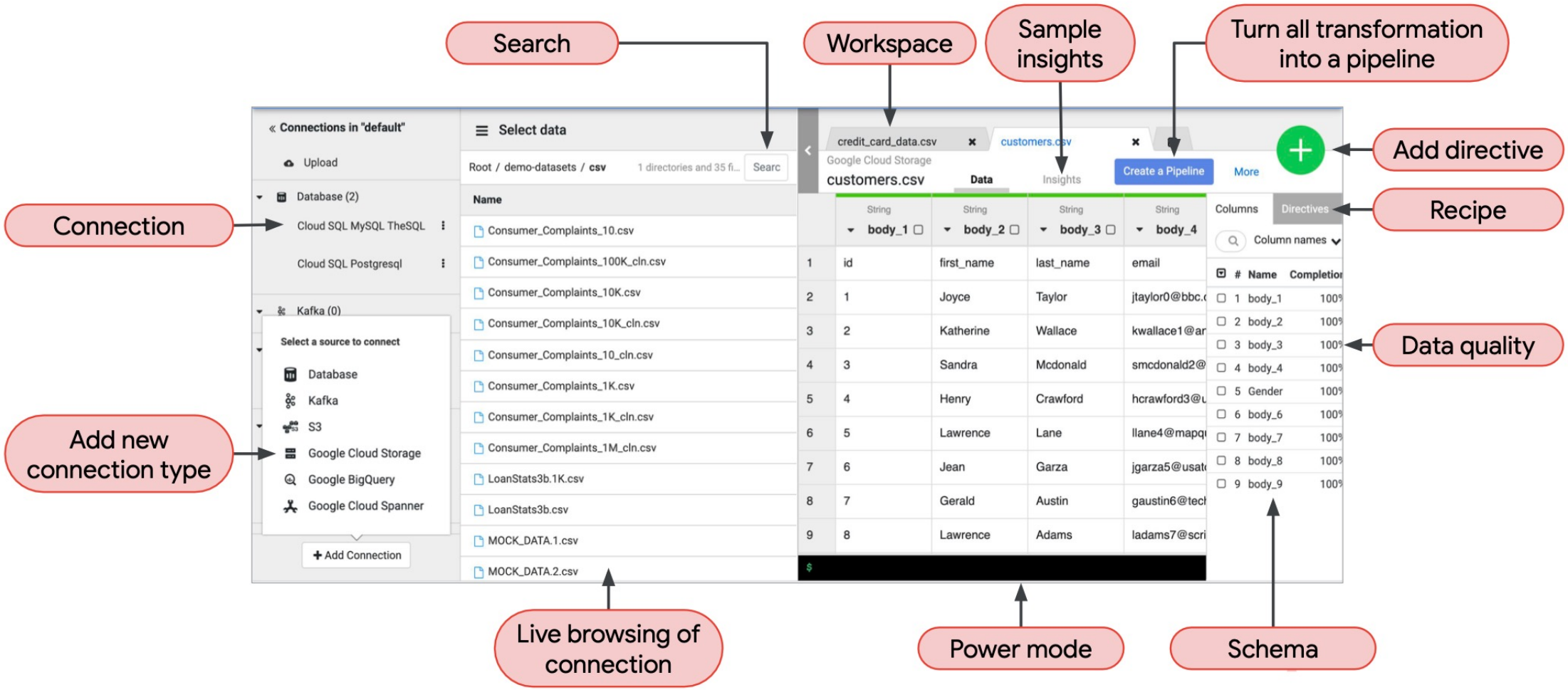
Wrangler is a code-free, visual environment for transforming data in data pipelines.



Wrangler UI overview for exploring datasets

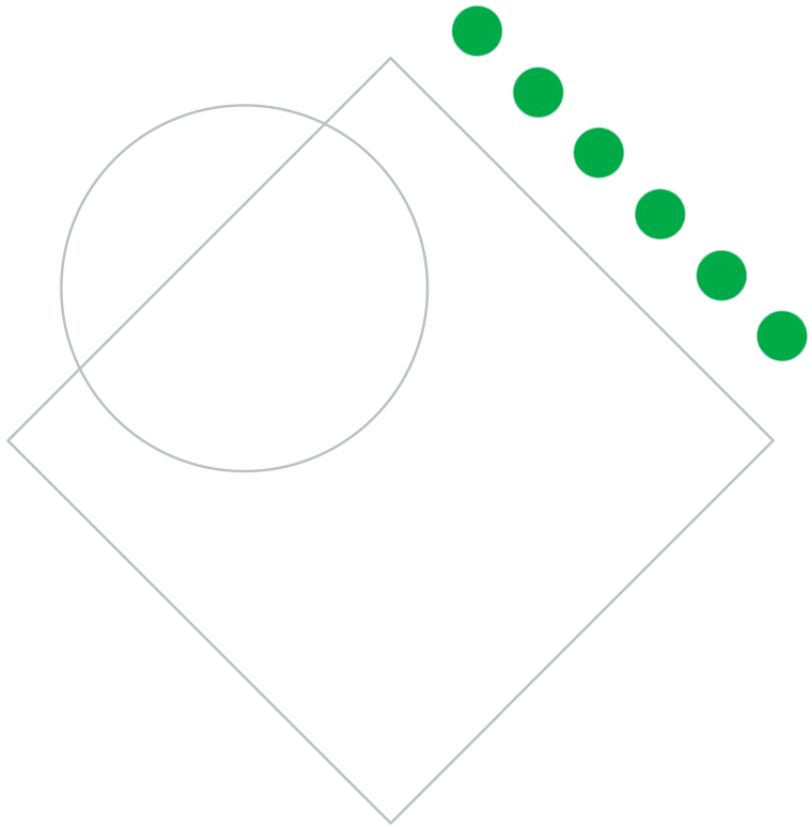


Wrangler UI overview for exploring datasets



Lab Intro

Building and Executing a Pipeline Graph in Cloud Data Fusion



Lab objectives

01 Connect Cloud Data Fusion to a couple of data sources

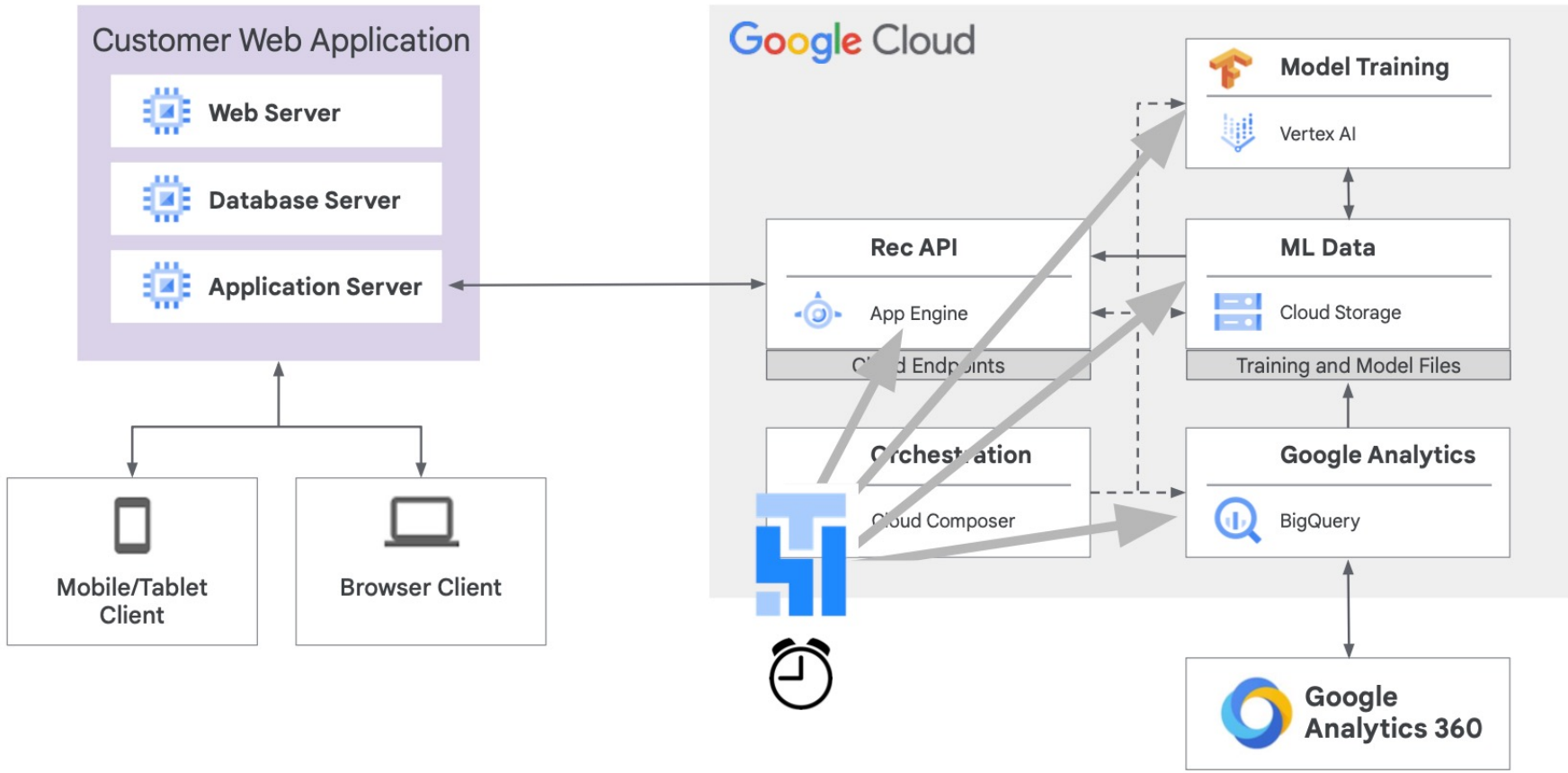
02 Apply basic transformations

03 Join two data sources

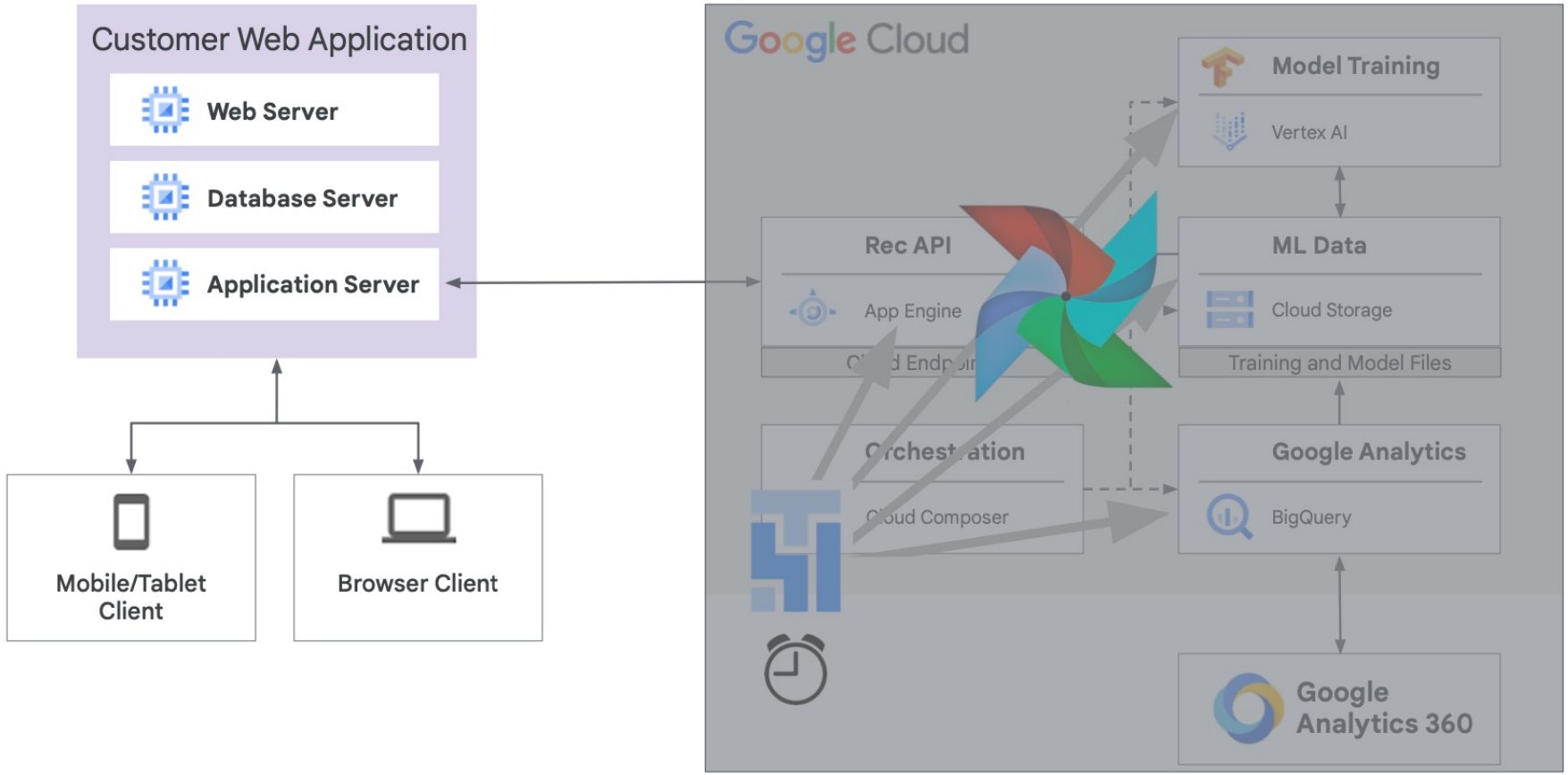
04 Write data to a sink

Orchestrating Work Between Google Cloud Services with Cloud Composer

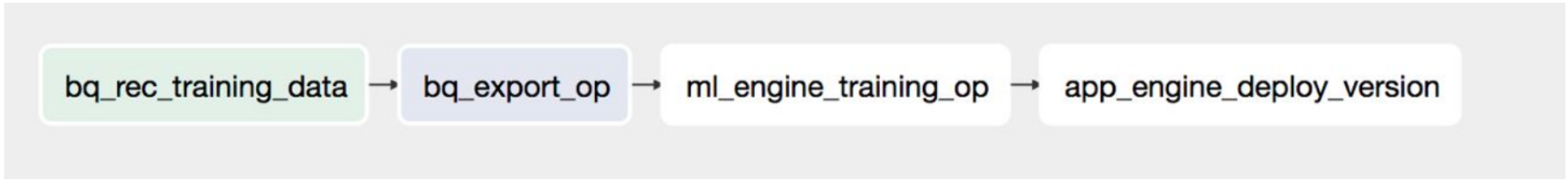
Cloud Composer orchestrates automatic workflows



Cloud Composer is managed Apache Airflow

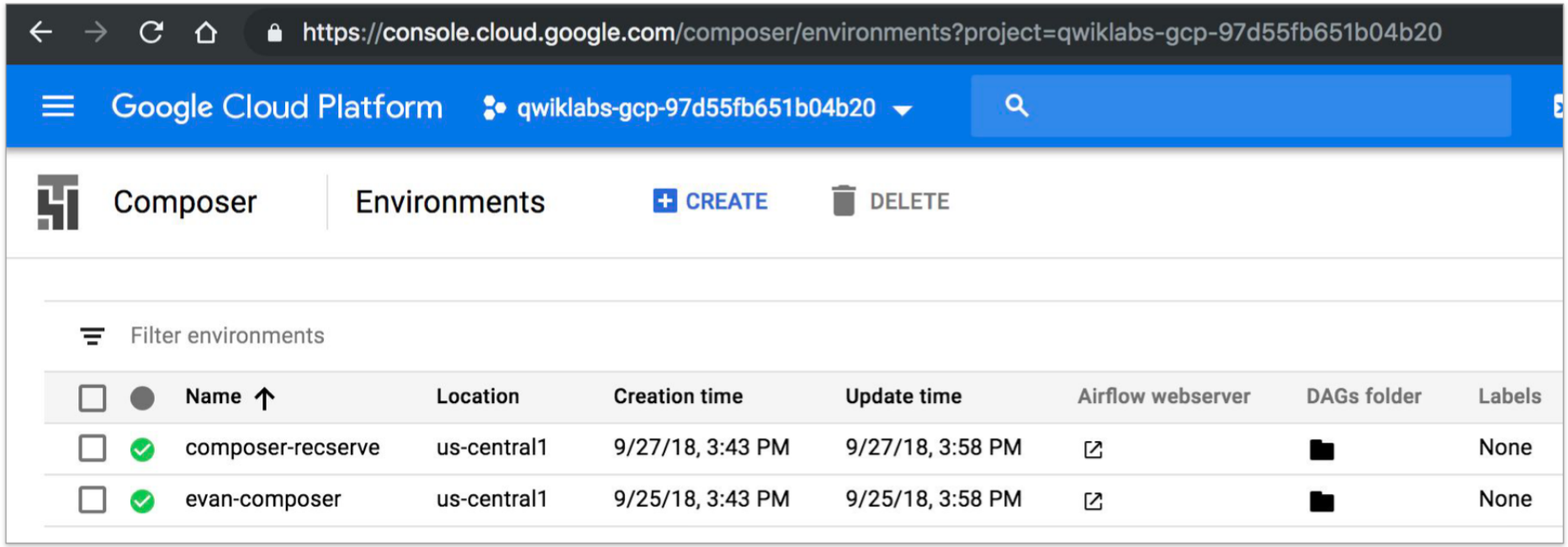


Use Apache Airflow DAGs to orchestrate Google Cloud services

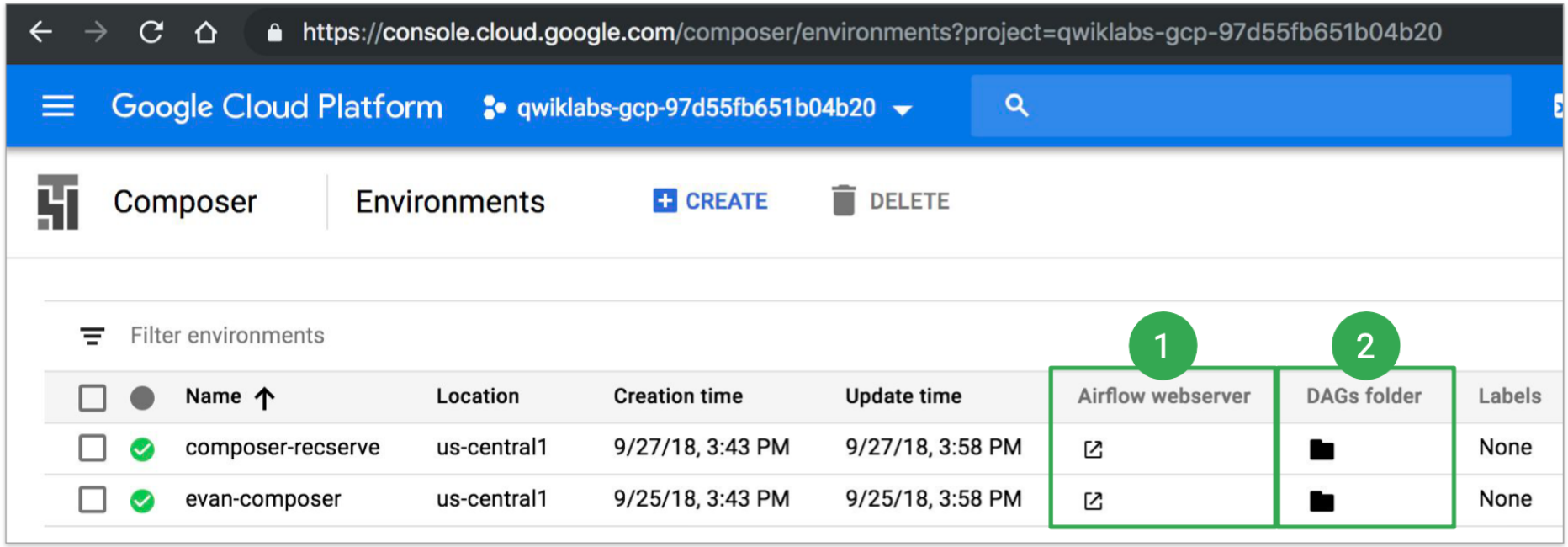


DAG = Directed Acyclic Graph

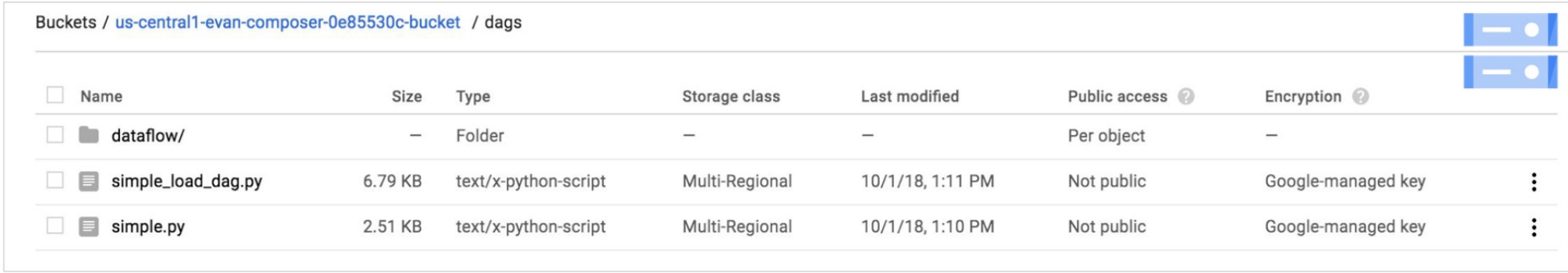
Cloud Composer creates managed Apache Airflow environments



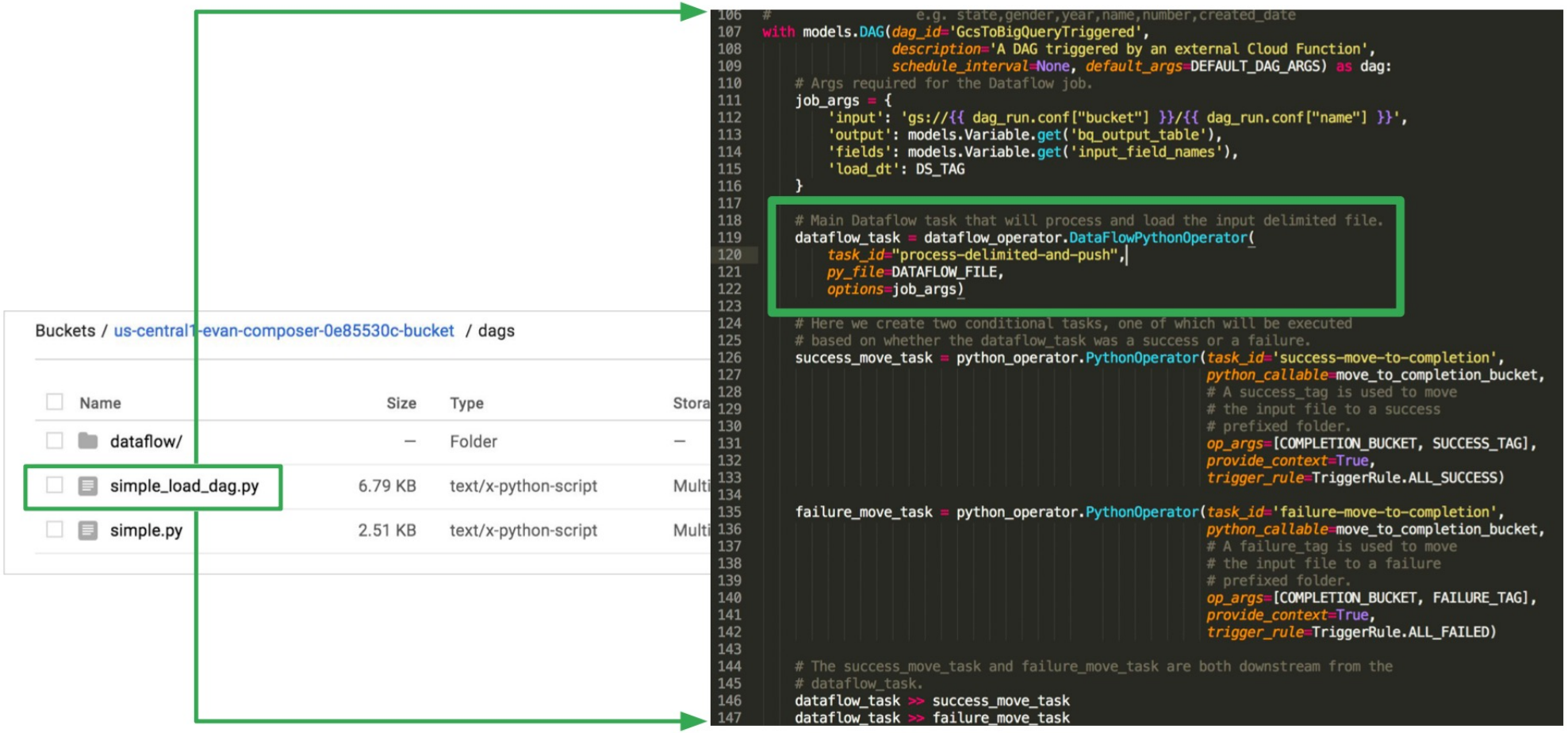
Each Airflow environment has a separate web server and folder in Cloud Storage for pipeline DAGs



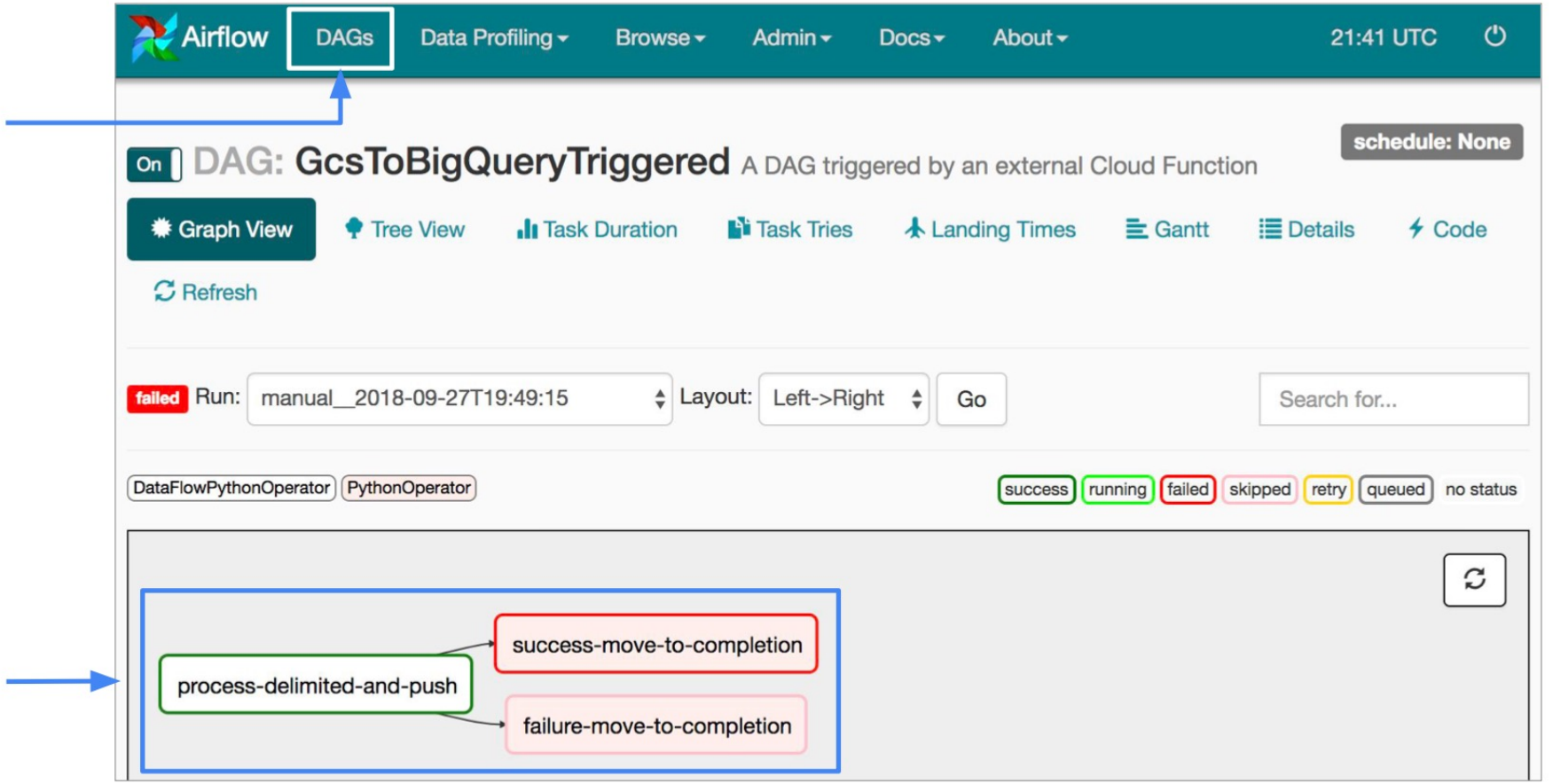
The DAGs folder is simply a Cloud Storage bucket where you will load your pipeline code



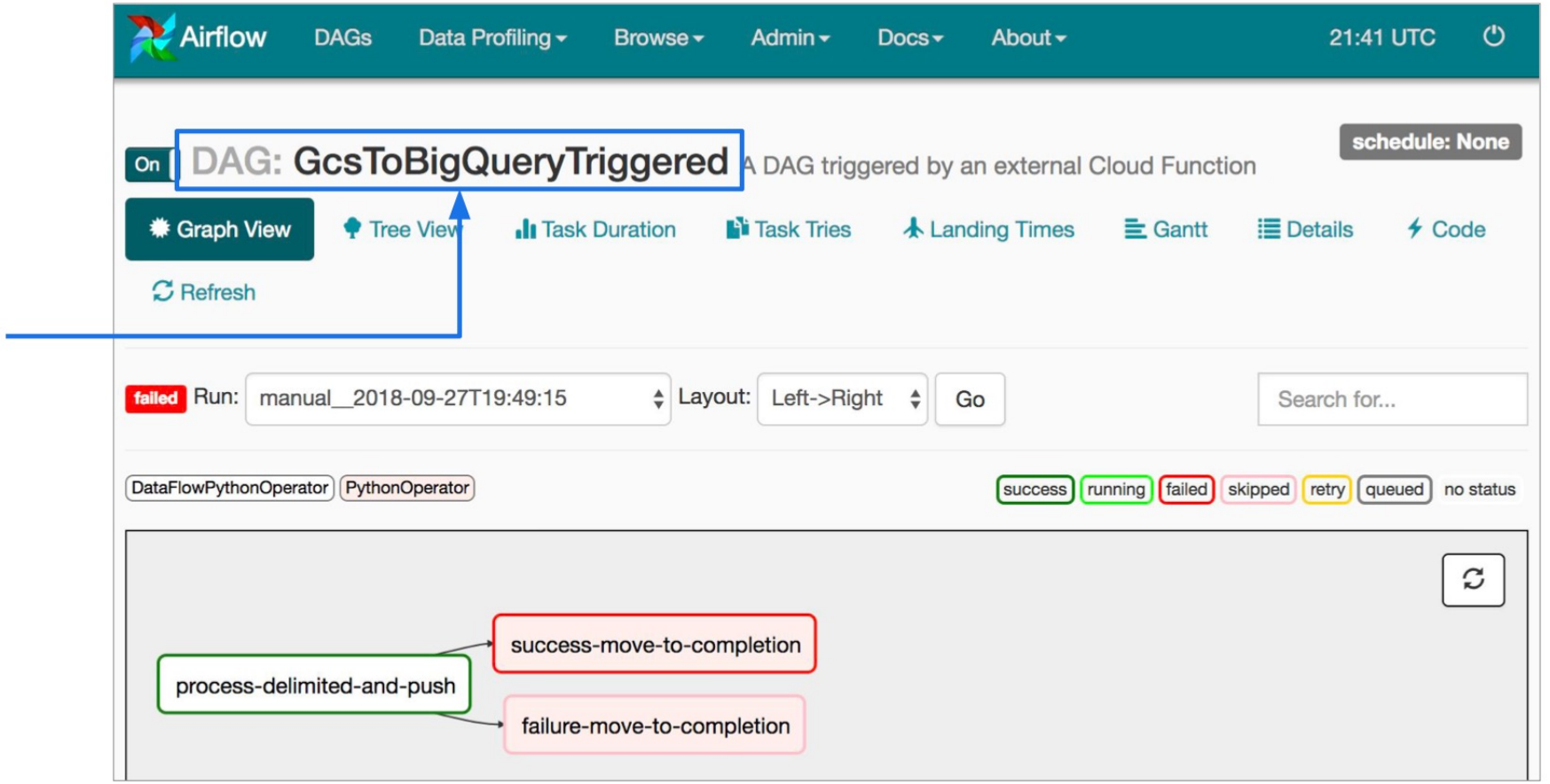
Airflow workflows are written in Python



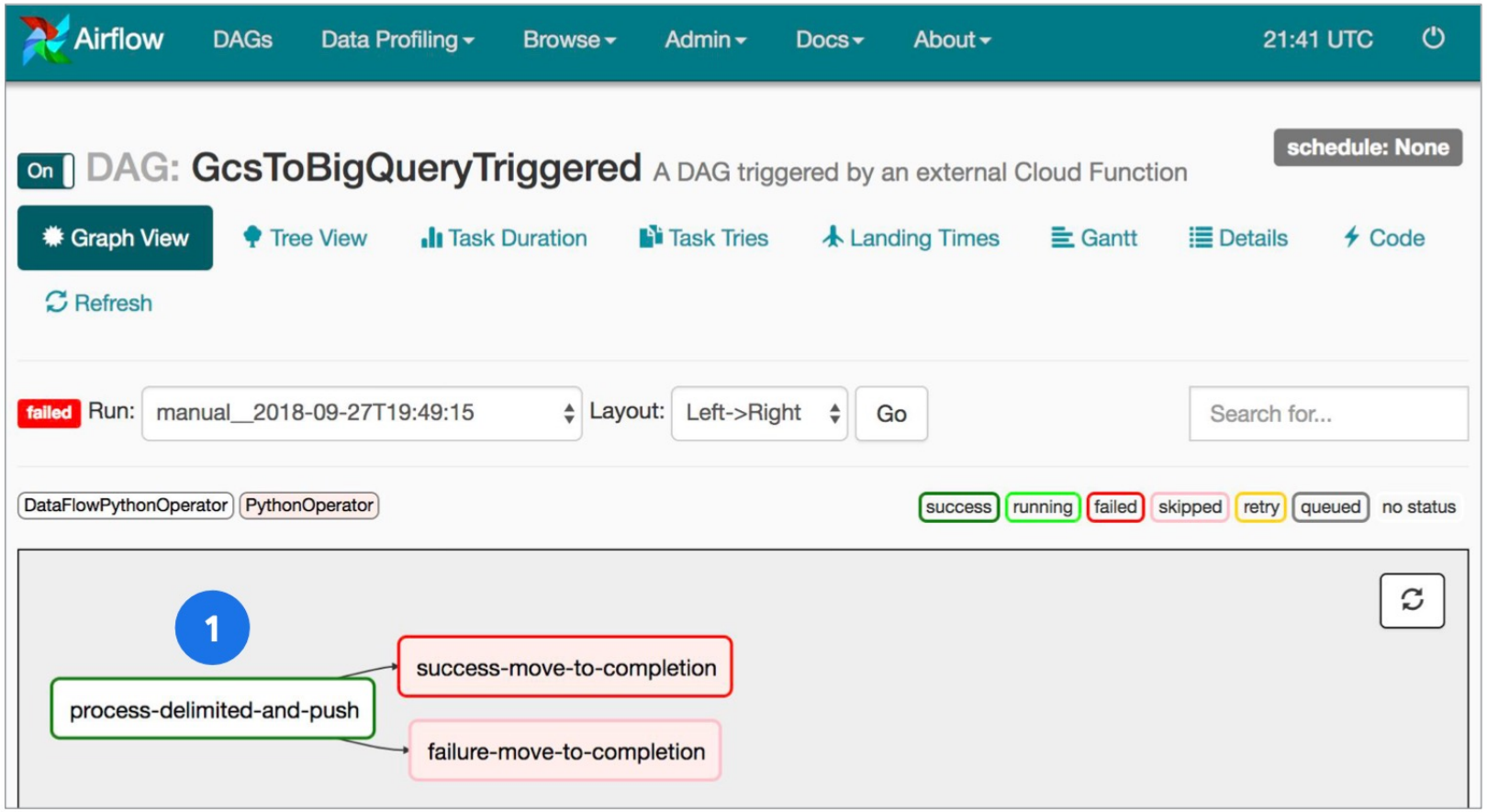
The Python file creates a DAG



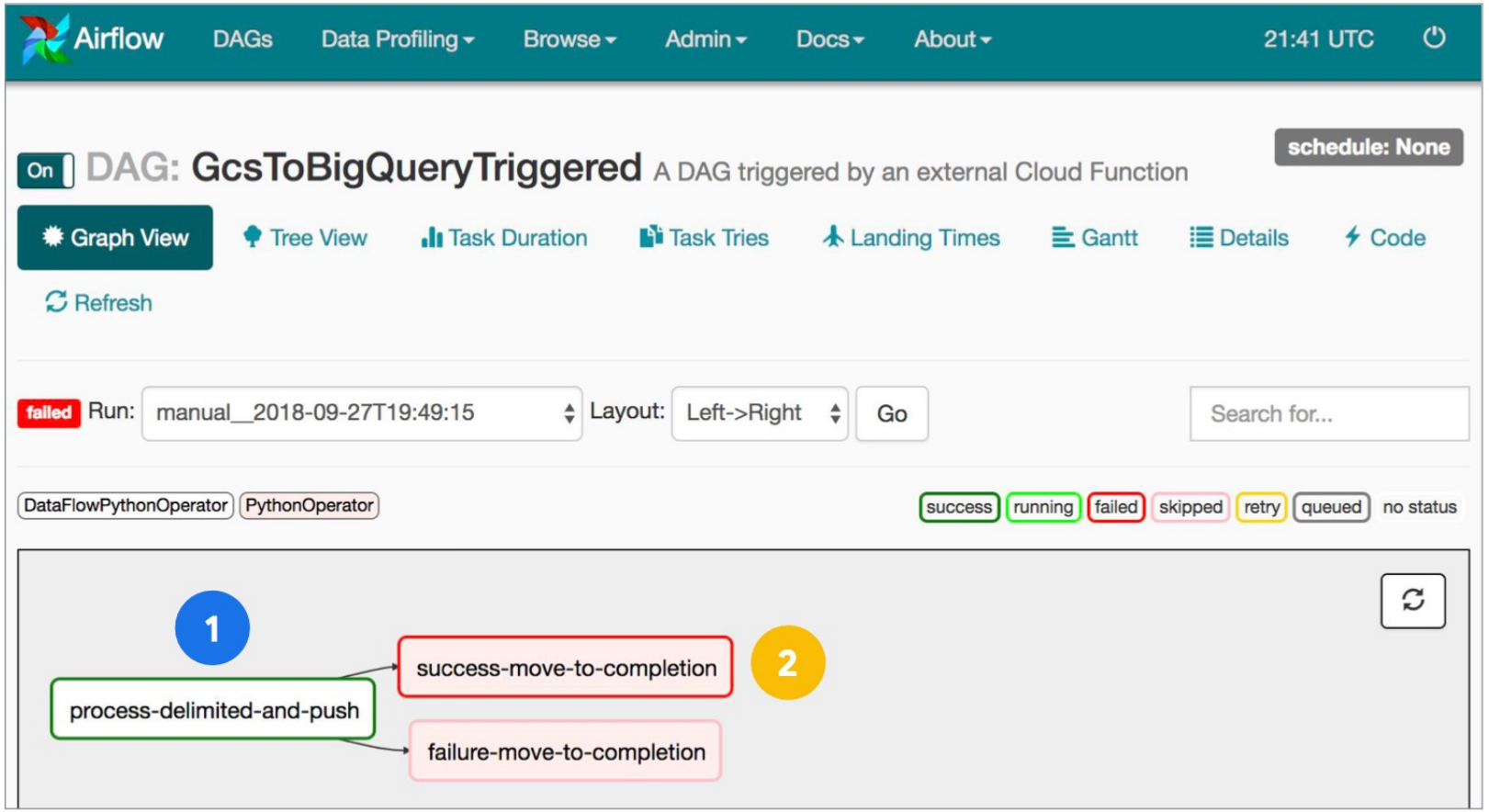
Airflow web server UI overview



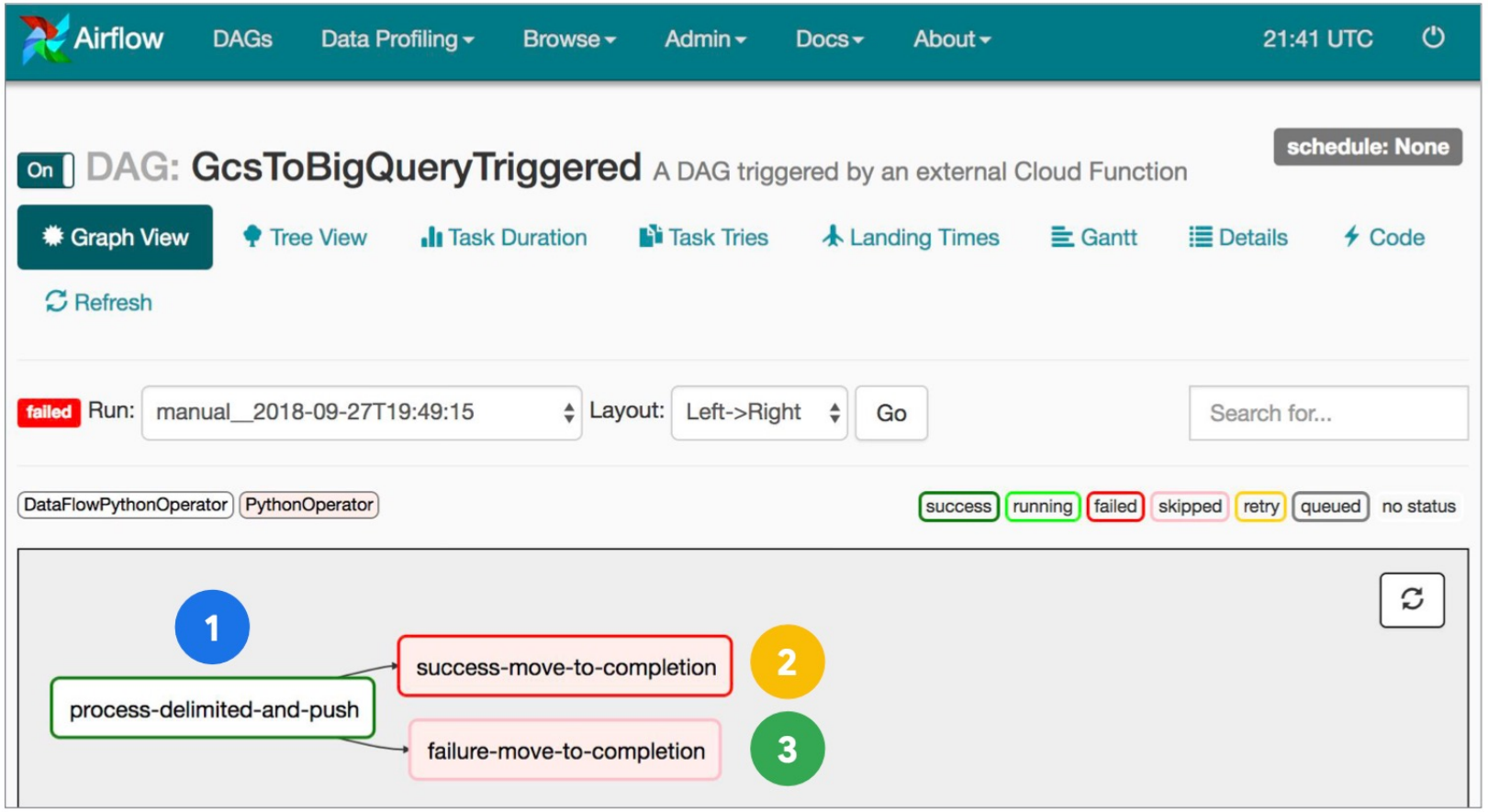
The Python file creates a DAG



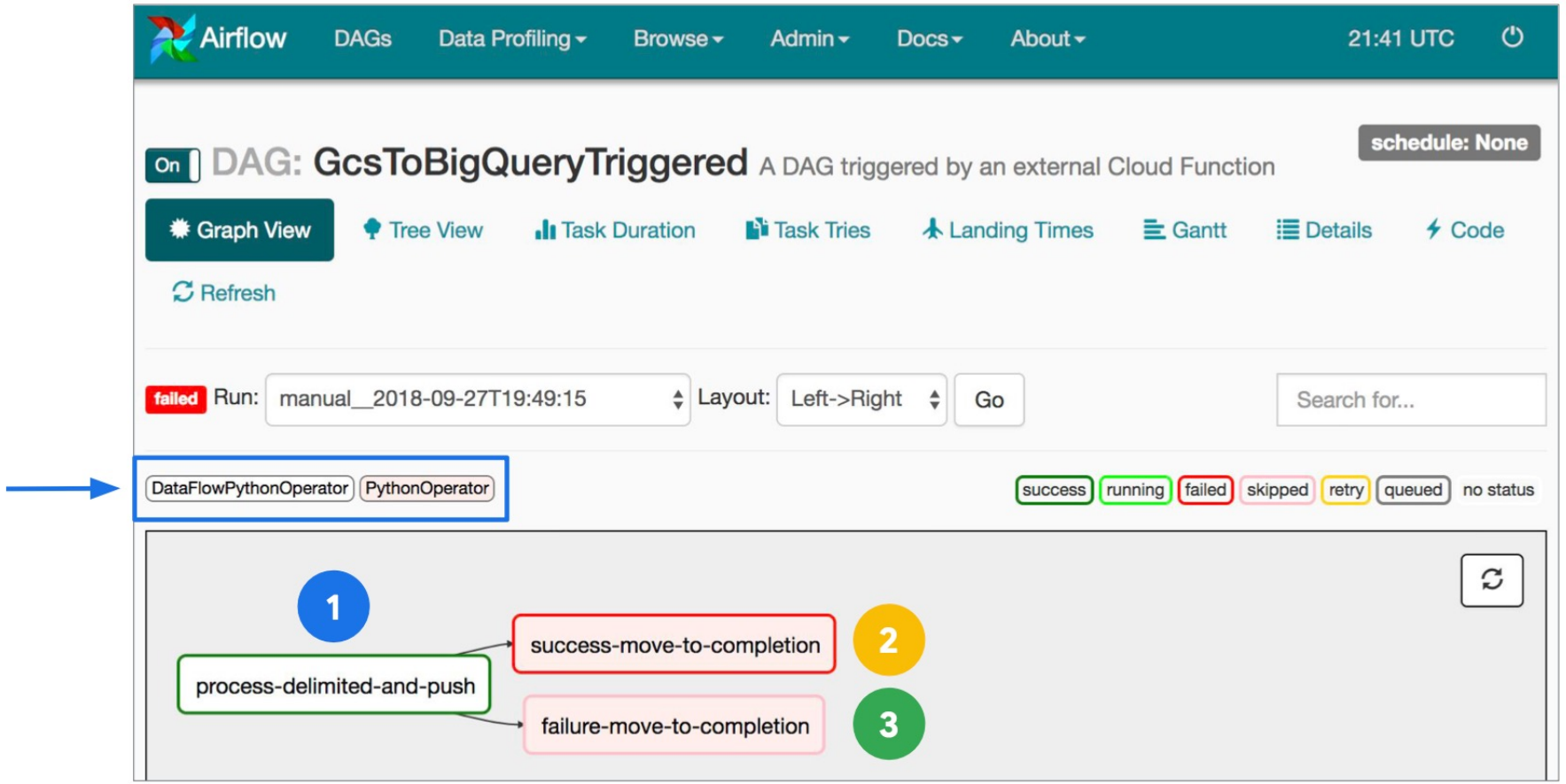
The Python file creates a DAG



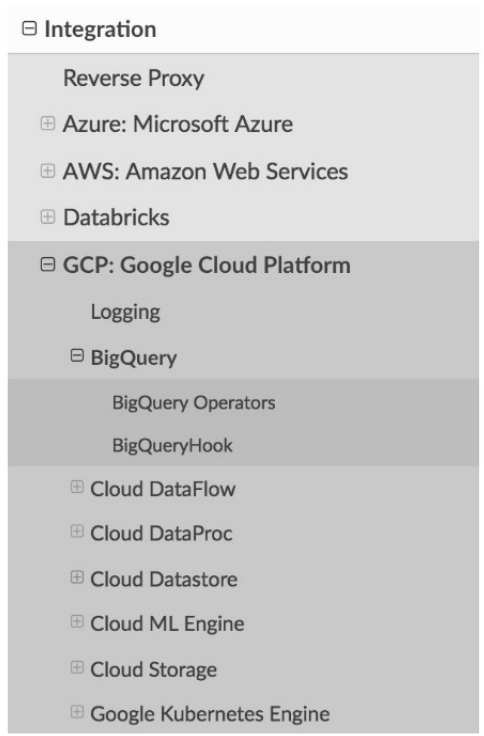
The Python file creates a DAG



The Python file creates a DAG



Airflow uses operators in your DAG to orchestrate other Google Cloud services



View all Google Cloud services that Airflow can orchestrate:

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

\+ sample code

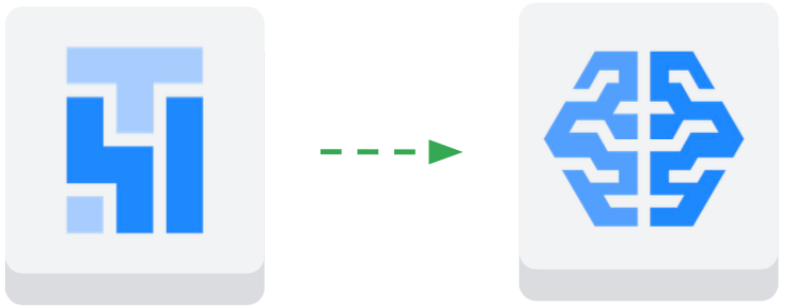
BigQuery Operators are popular for updating your ML training dataset

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

```
BigQueryOperator
BigQueryCheckOperator
BigQueryIntervalCheckOperator
BigQueryCreateEmptyTableOperator
BigQueryCreateExternalTableOperator
BigQueryDeleteDatasetOperator
BigQueryToBigQueryOperator
BigQueryToCloudStorageOperator
```

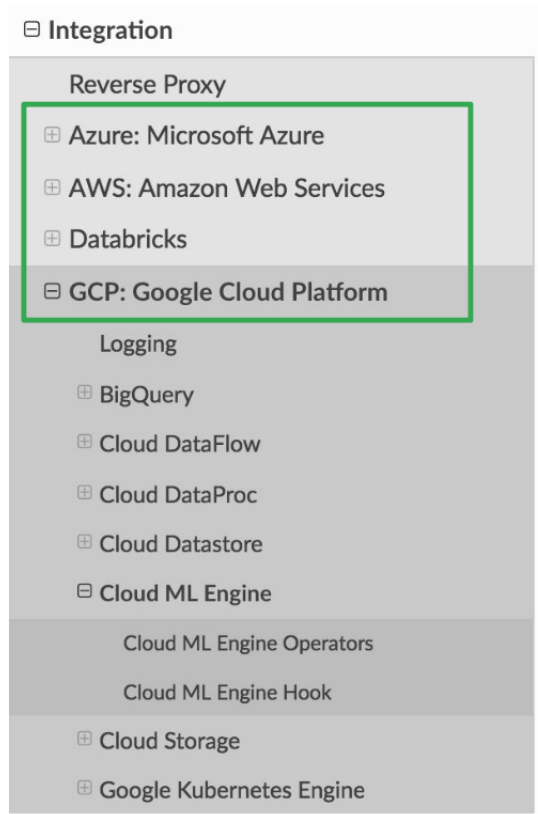
Vertex AI (formerly Cloud ML Engine) operators can launch training and deployment jobs

- Integration
  - 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



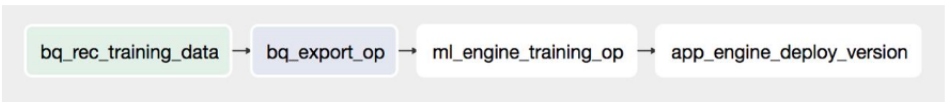
```
MLEngineBatchPredictionOperator
MLEngineModelOperator
MLEngineTrainingOperator
MLEngineVersionOperator
```

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



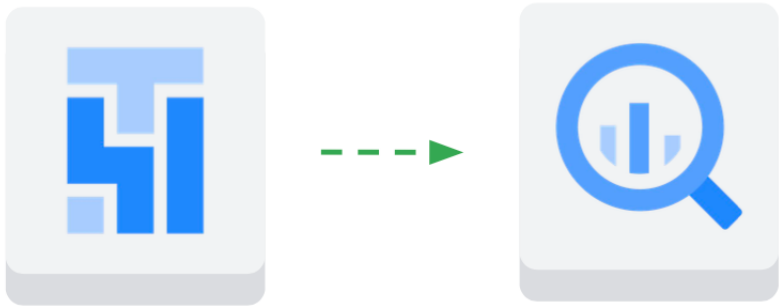
Example: Common Machine Learning workflow

DAG Operators




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

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

# AI Platform training job                                3
t3 = MLEngineTrainingOperator(
)

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

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

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',

        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)
```

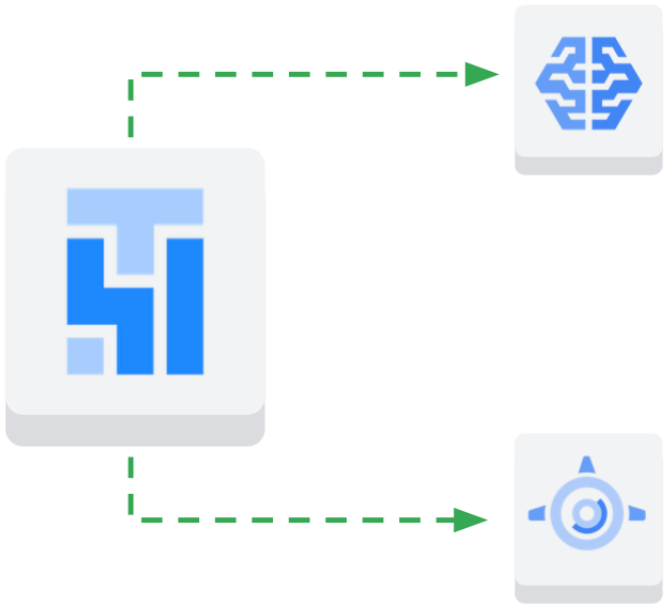
Note the Python constants for a date range here

```
from airflow.contrib.operators import bigquery_operator

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

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

        task_id='bq_recent_questions_query',

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

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

You could even set a rolling window with macros

```
from airflow.contrib.operators import bigquery_operator

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

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

        task_id='bq_recent_questions_query',

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

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

Vertex AI and App Engine operators retrain and redeploy our model



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

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

# AI Platform training job                                3
t3 = MLEngineTrainingOperator(
)

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

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

Manage pipelines and dependencies as code

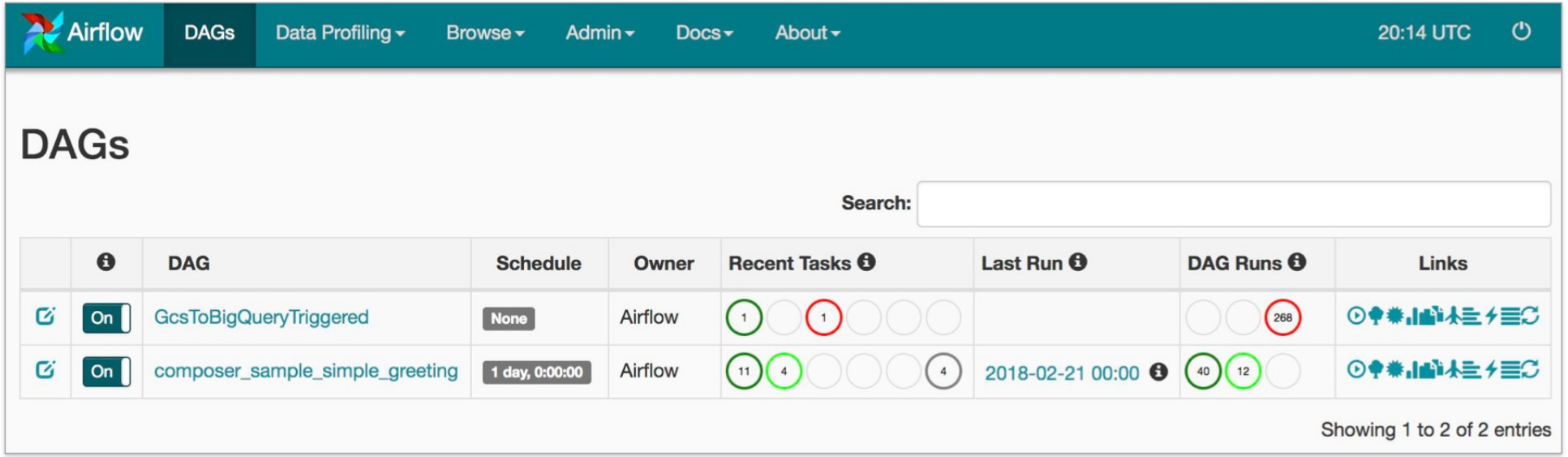
```
# update training data
t1 = BigQueryOperator(
)

# 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)
```

Two scheduling options for Cloud Composer workflows

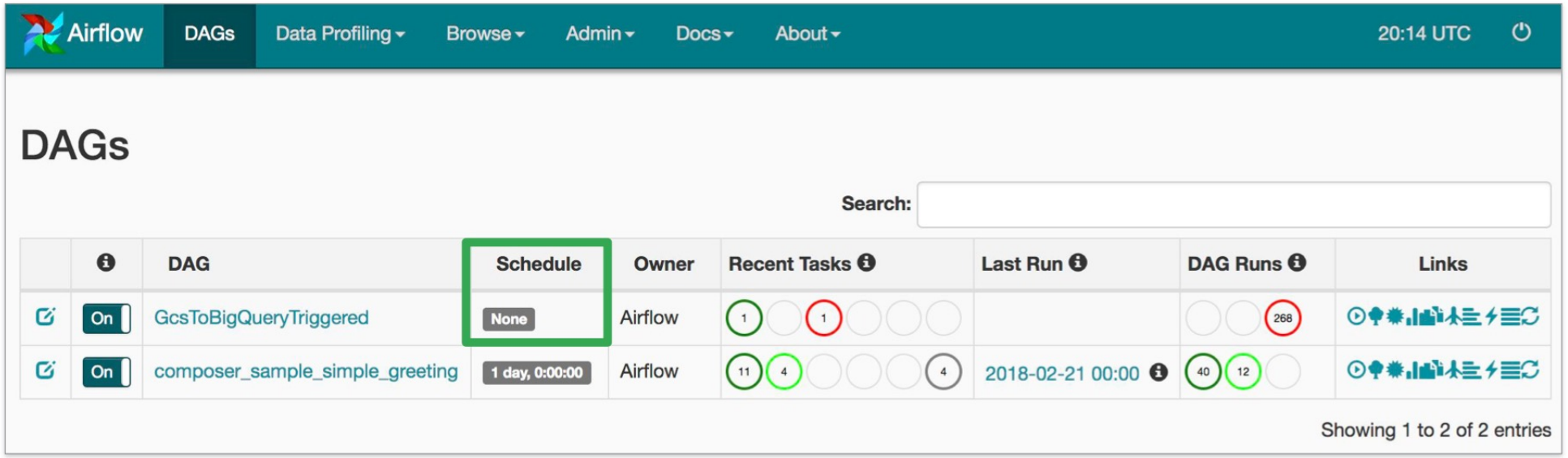
Periodic

Event-driven

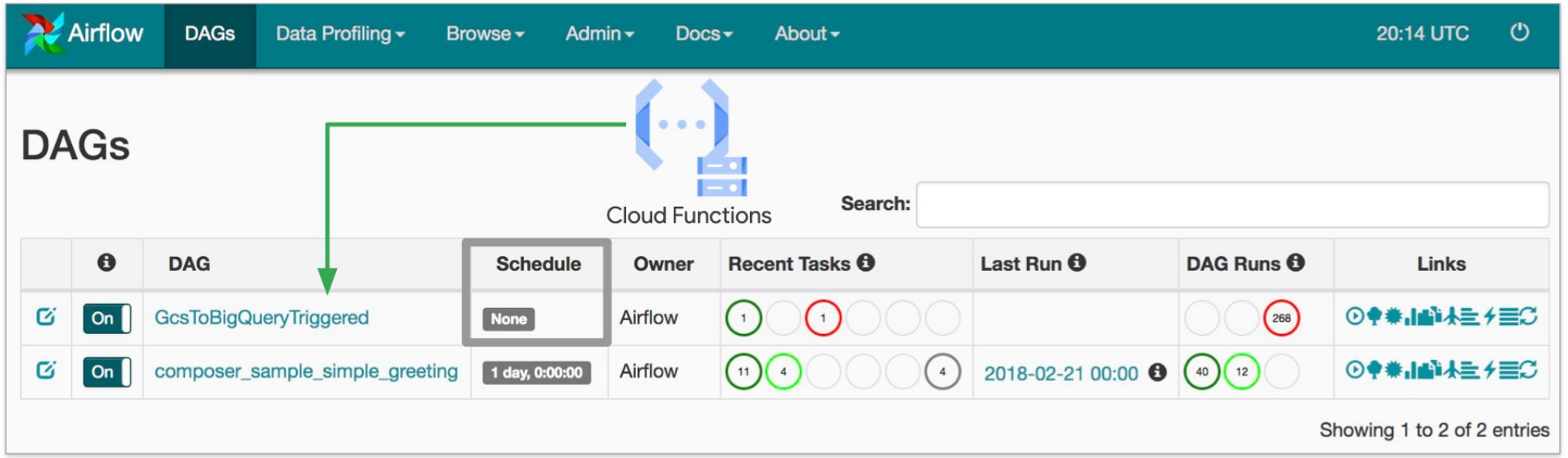
Airflow scheduling basics



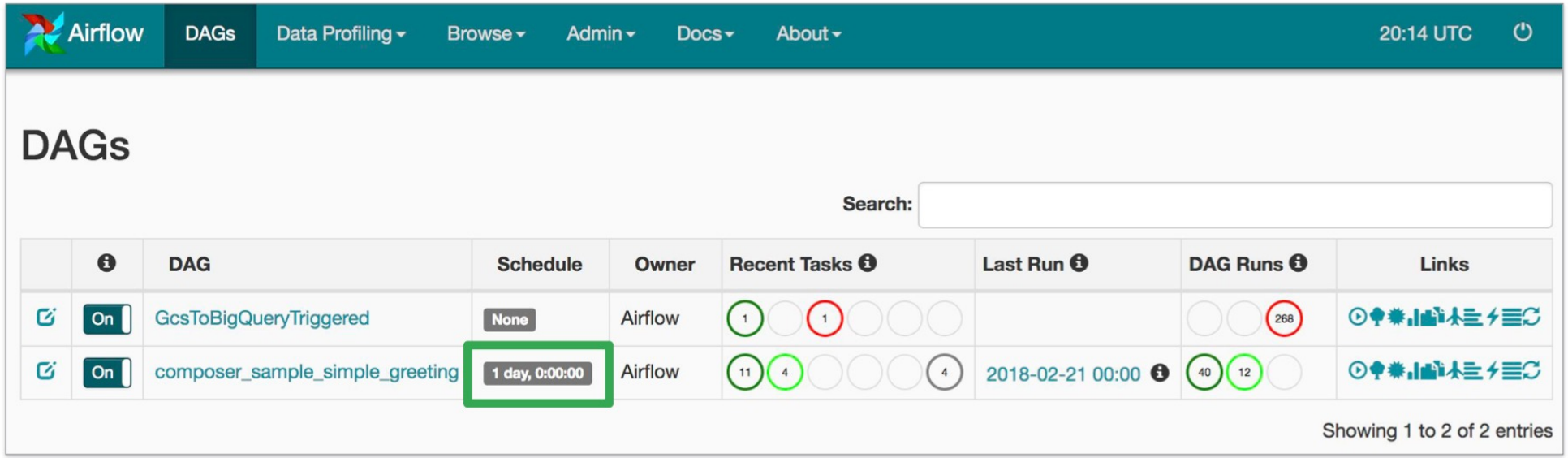
Why is this DAG missing a schedule?



Option 1: Event-driven scheduling with Cloud Functions



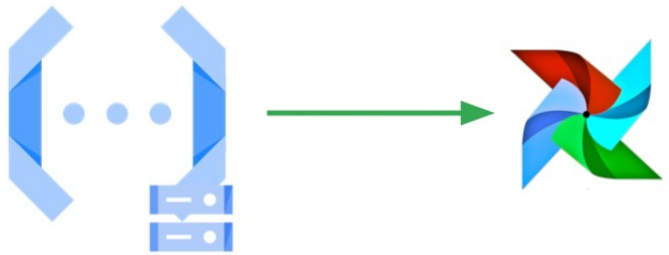
Option 2: Specify pipeline schedule_interval in your DAG




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

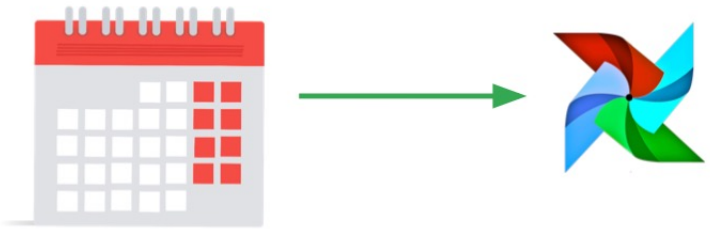
Two types of workflow ETL patterns

Push (event-triggered)

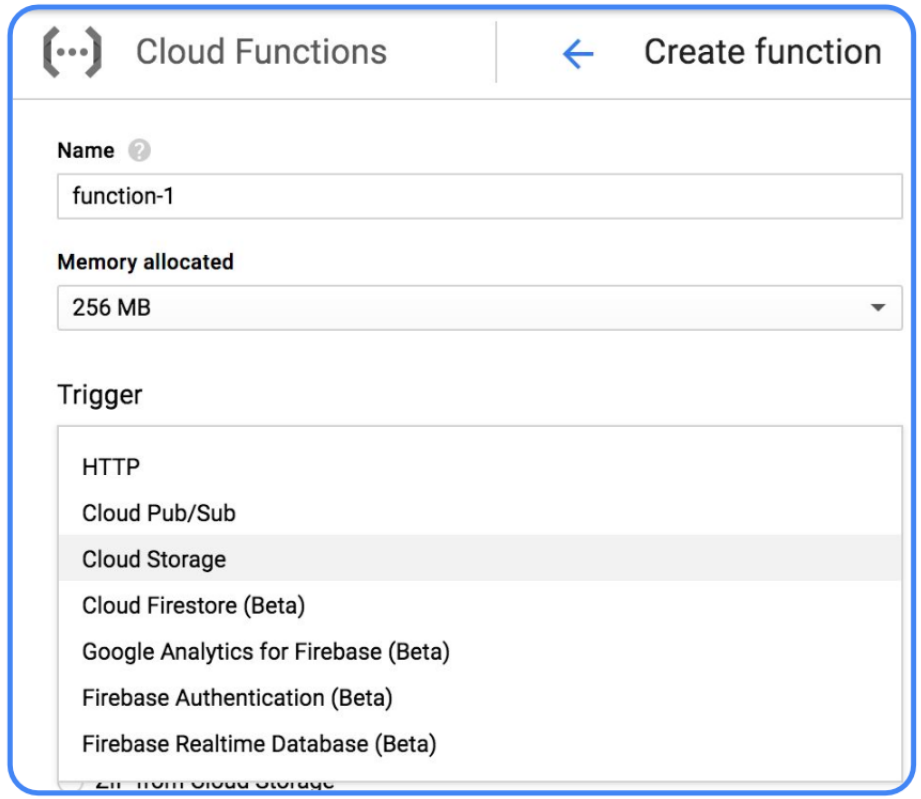


Cloud Functions

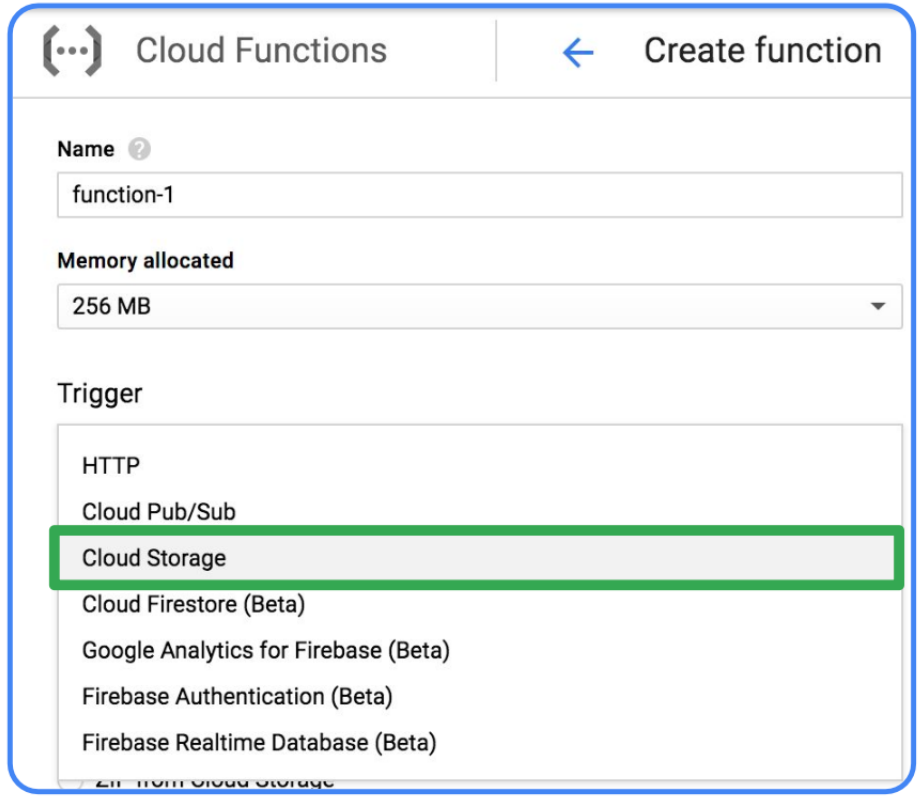
Pull (scheduled workflow run)



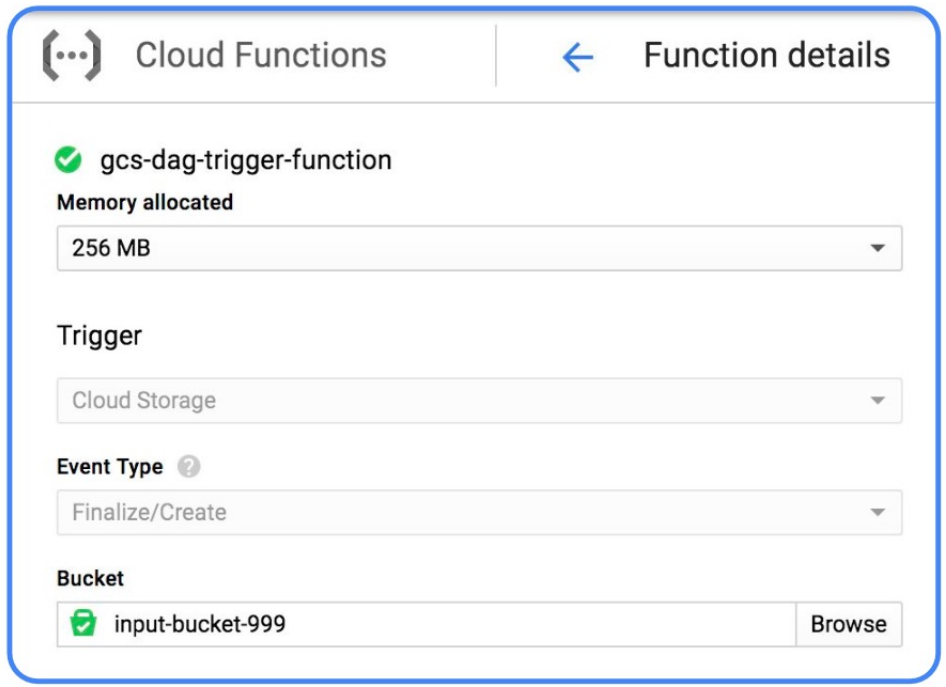
Use Cloud Functions to create an event-based push architecture



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 Cloud Storage 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 = '                                                    .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
```

1

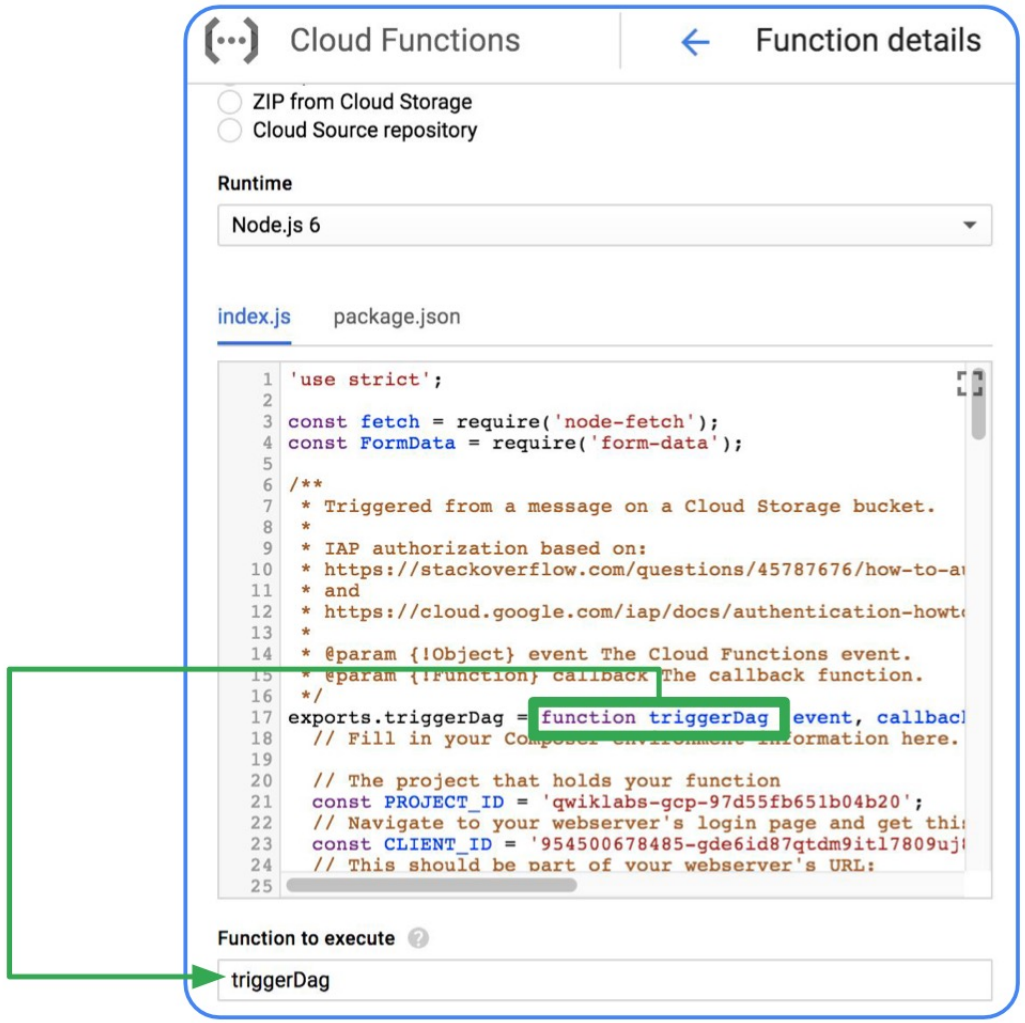
2

3

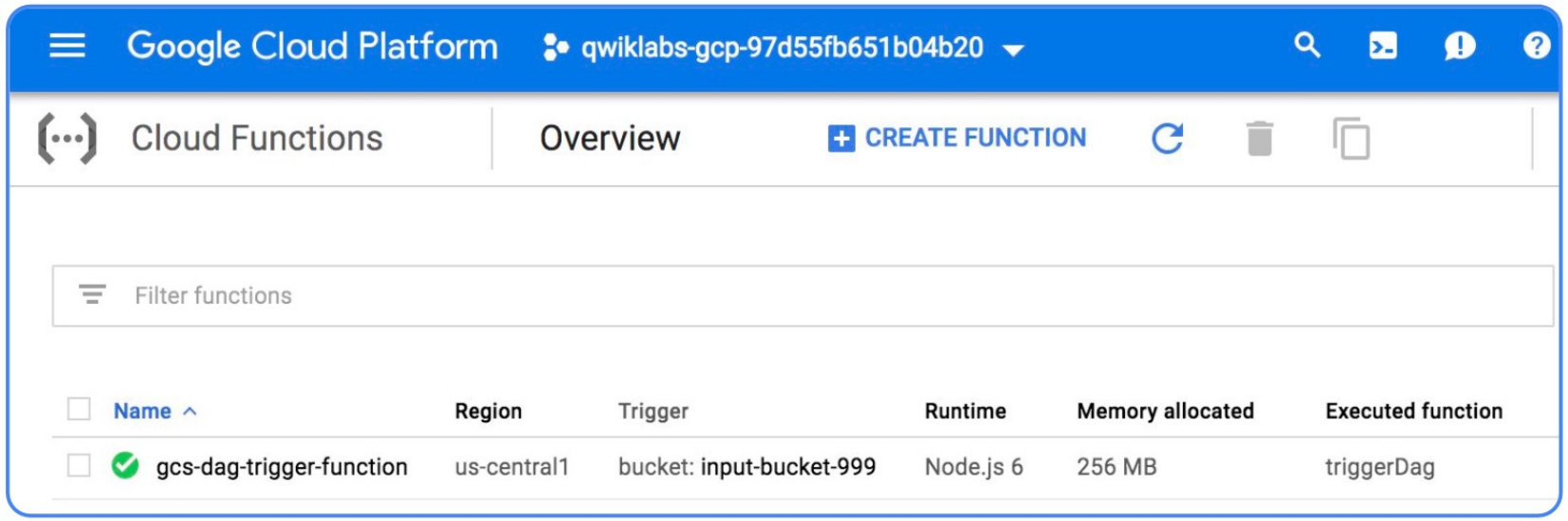
4

5

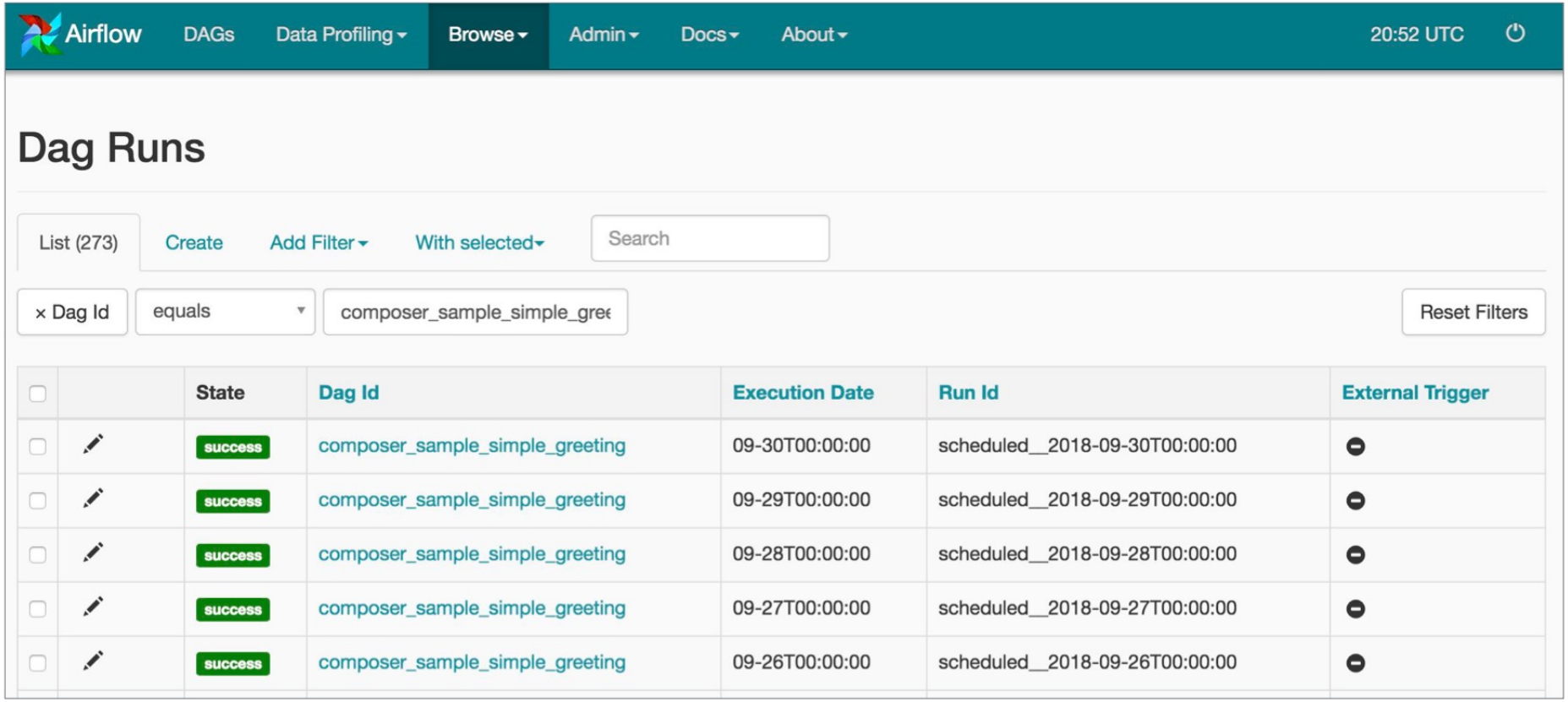
Be sure to specify the function you want Cloud Functions to call in your script



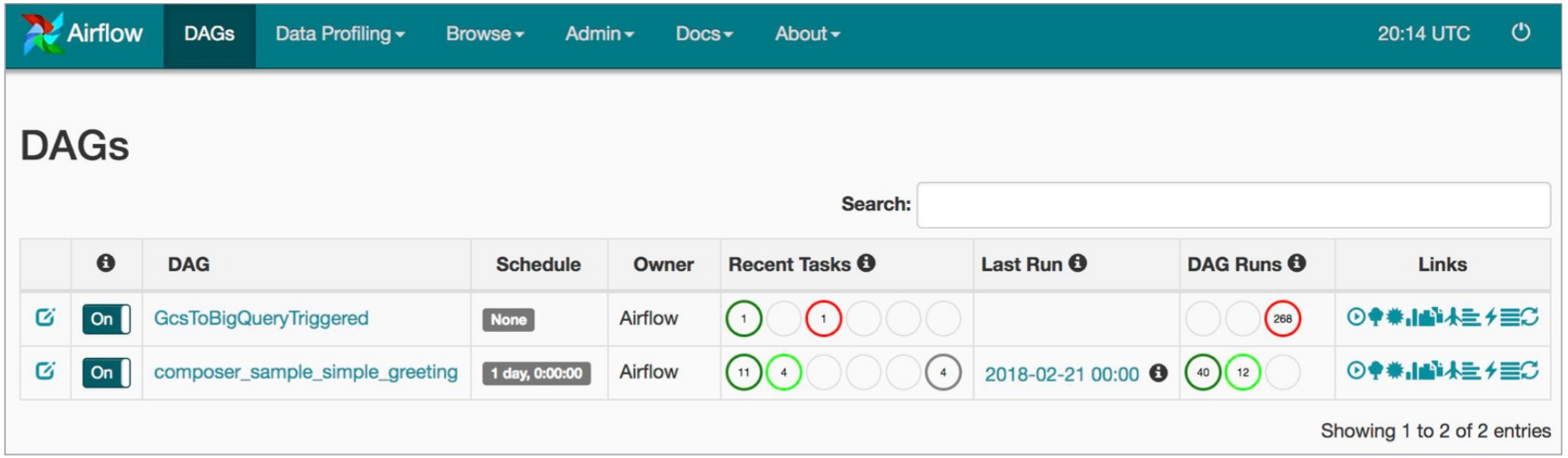
Cloud Function is triggered whenever a new file is loaded to a specific Cloud Storage bucket



Monitor current and historical workflow progress

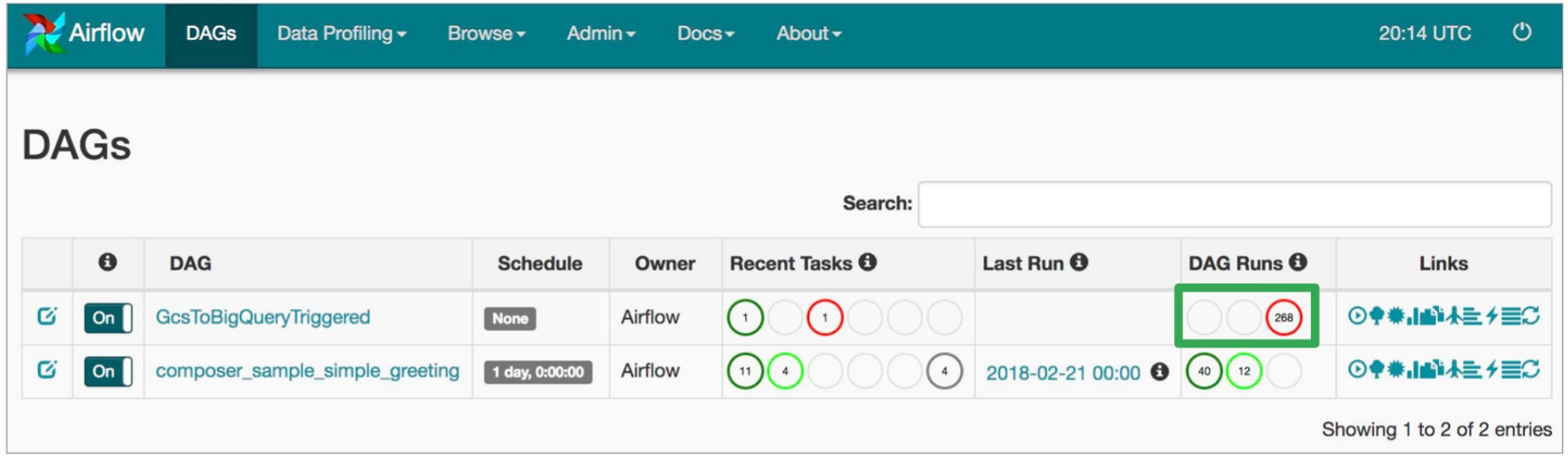


Quickly gauge pipeline health of DAG runs

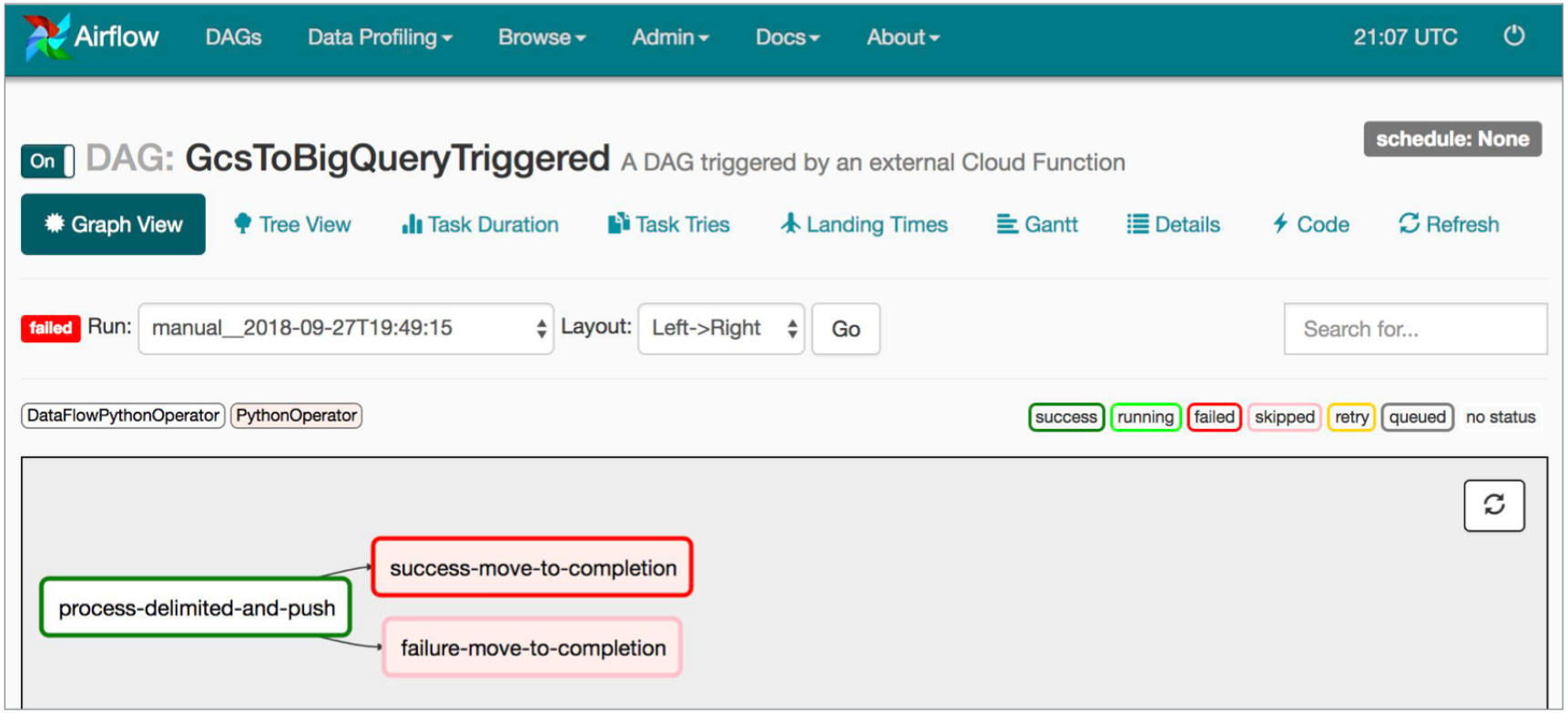


Which pipeline is healthier?

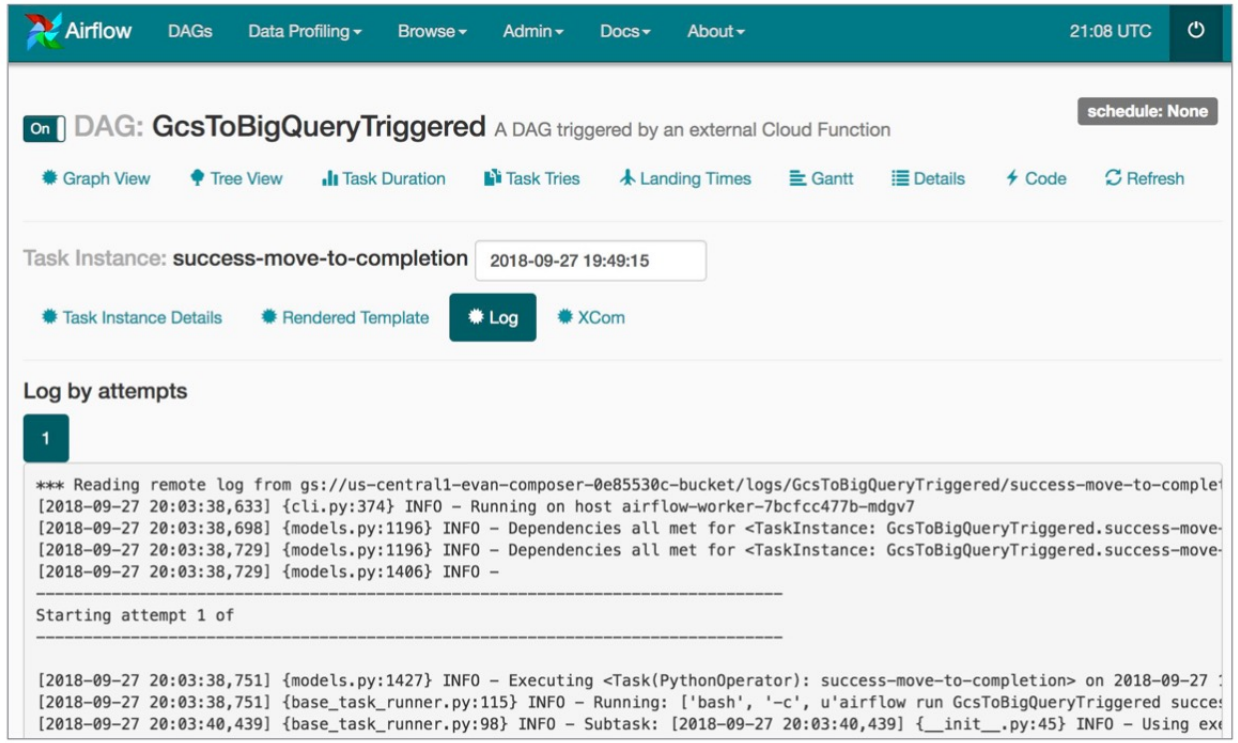
Quickly gauge pipeline health of DAG runs



Quickly gauge pipeline health of DAG runs

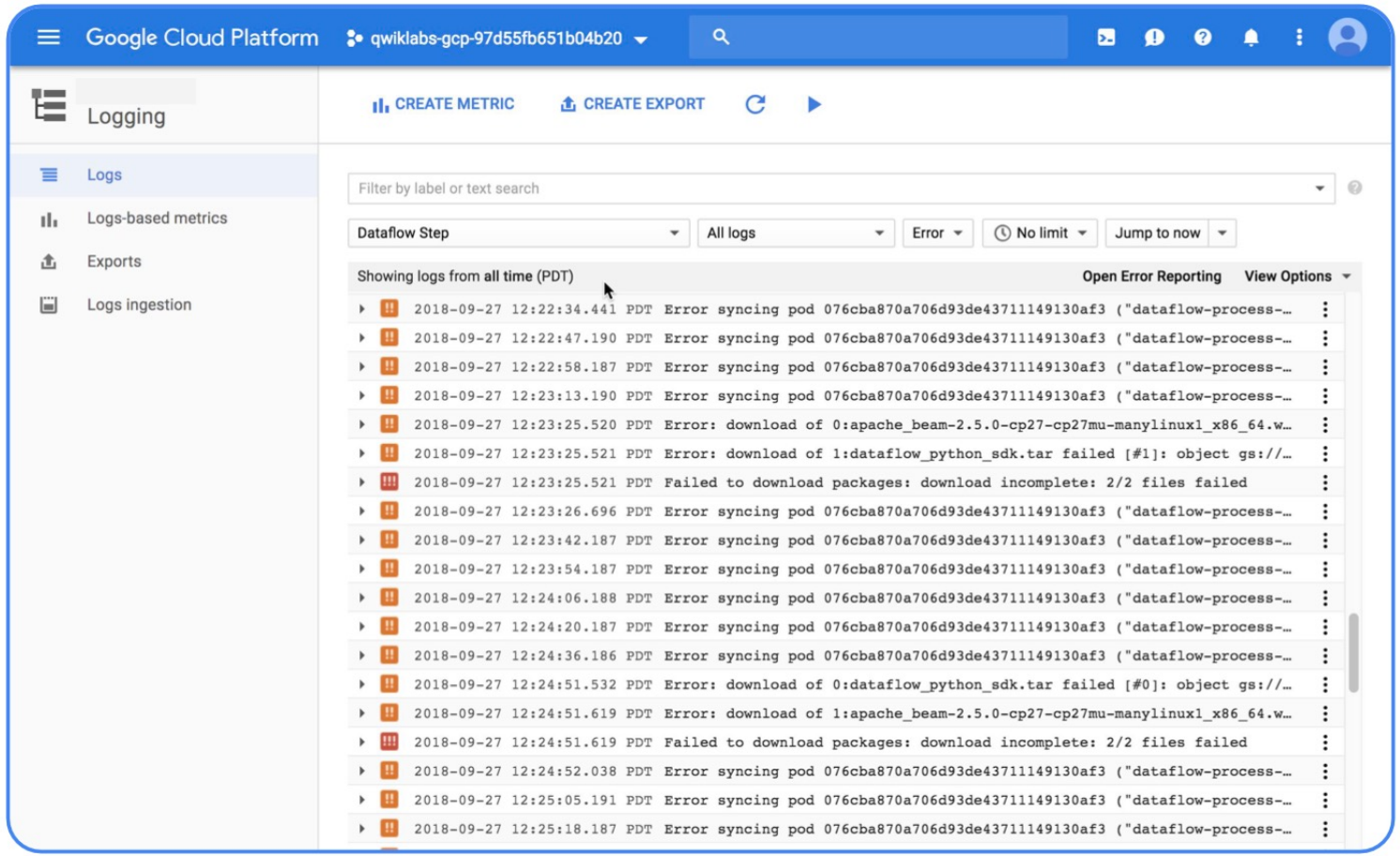


Monitor and troubleshoot Airflow step errors in logs

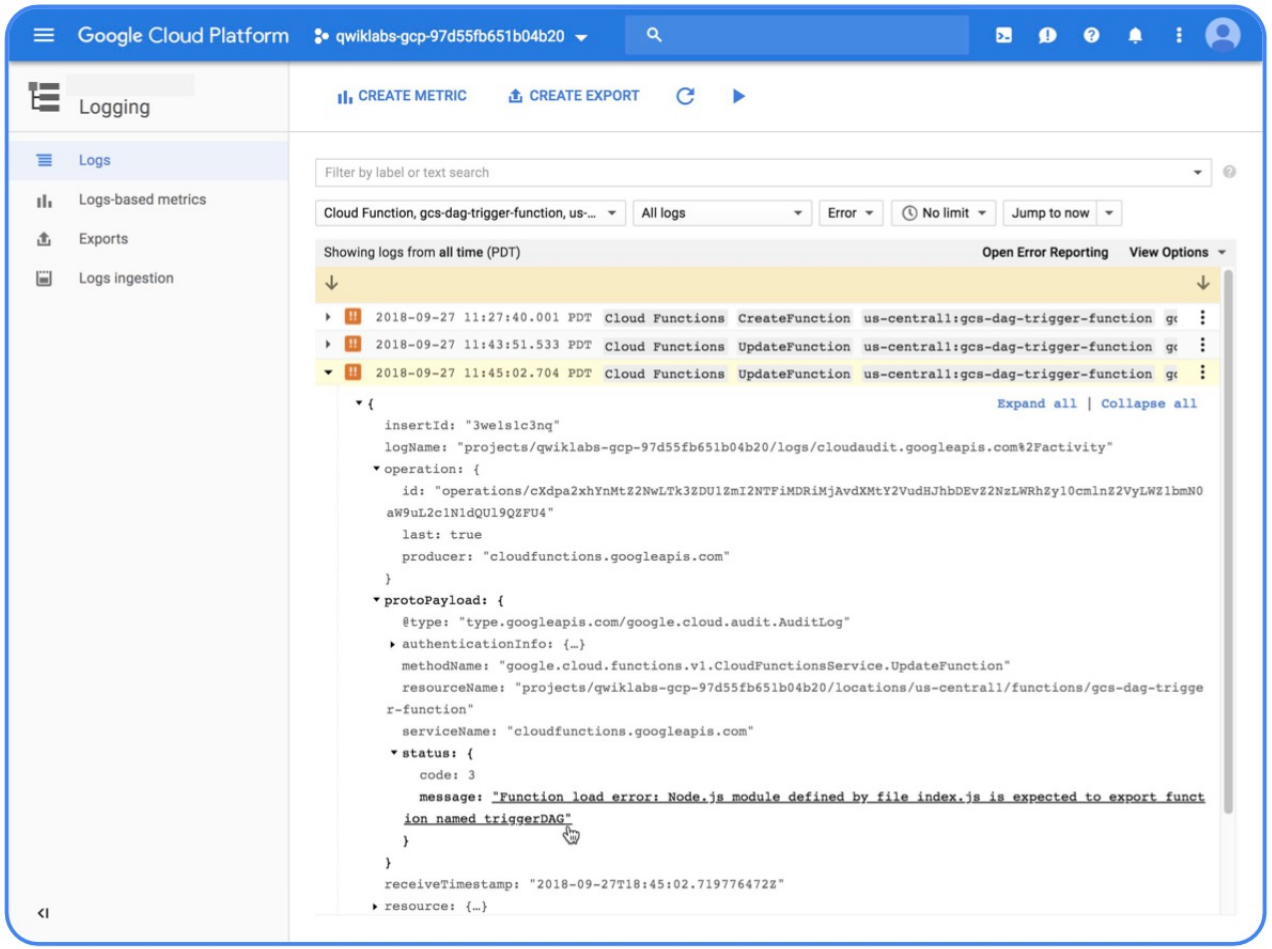


errors.HttpError: <HttpError 400 when requesting https://www.googleapis.com/storage/v1/b/input-bucket-999/o/usa_names.csv/copyTo/b/gs%3A%2F%2Foutput-bucket-999%2F/o/success%2F2018-09-27%2Fusa_names.csv?alt=json returned "**Invalid bucket name:** 'gs://output-bucket-999/'">

Monitor Dataflow health in Cloud Logging

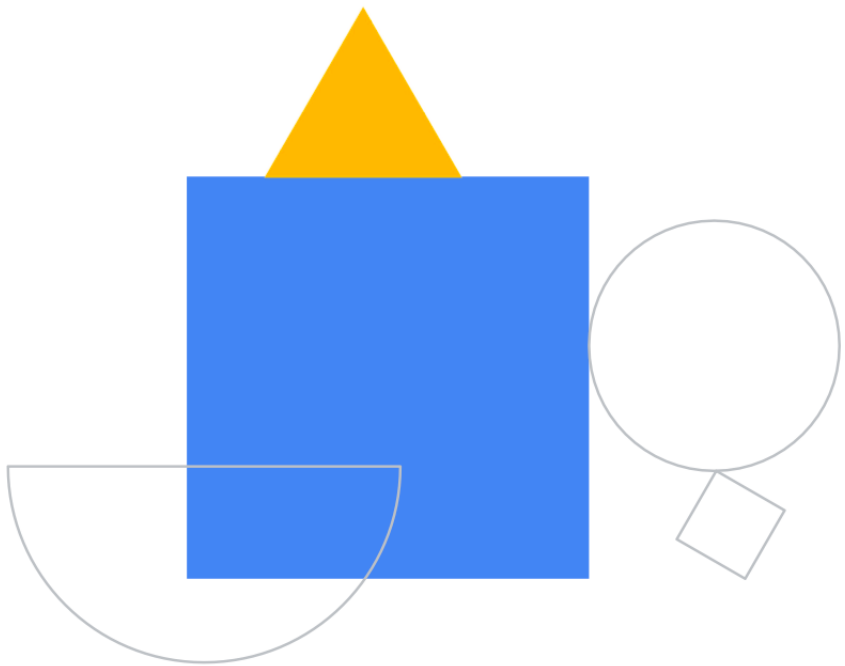


Monitor Cloud Function health in Cloud Logging



Lab Intro

An Introduction to
Cloud Composer



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