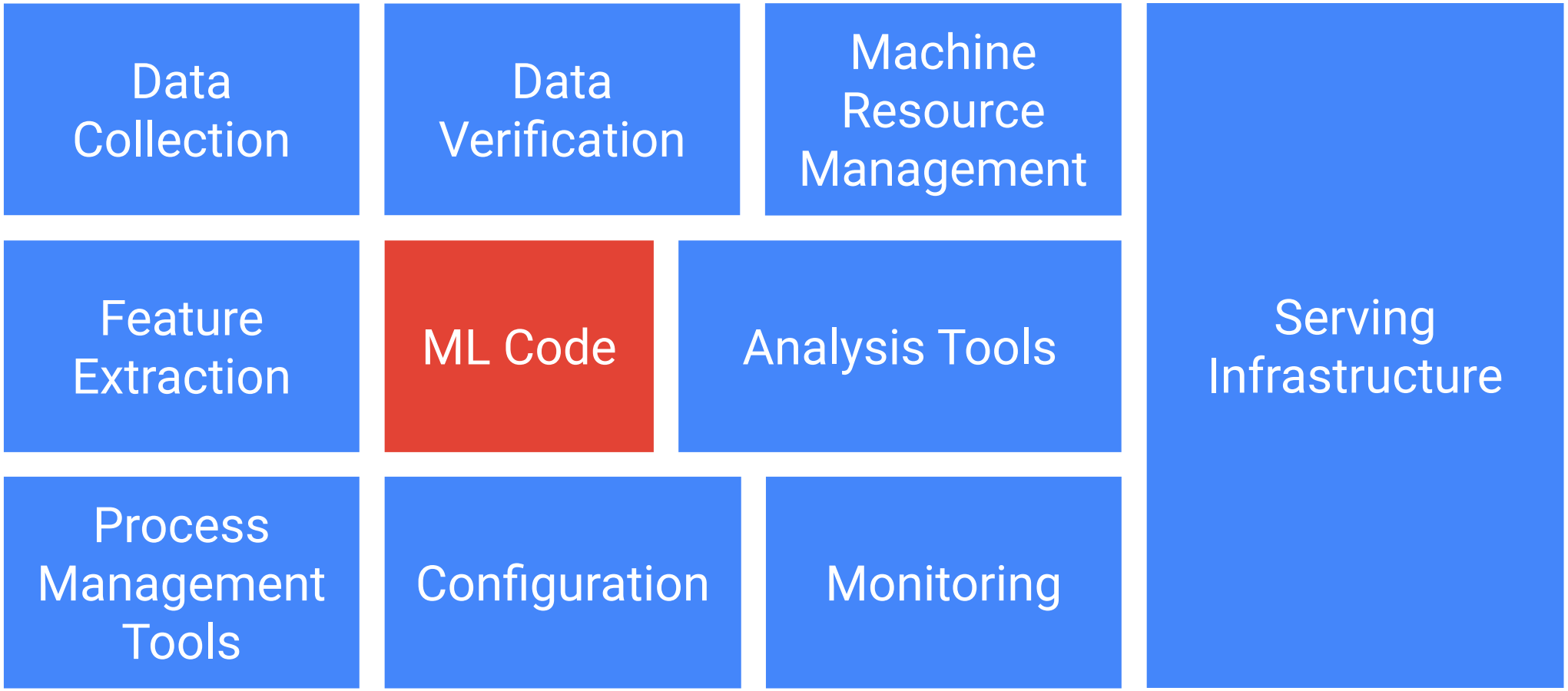
Google Cloud

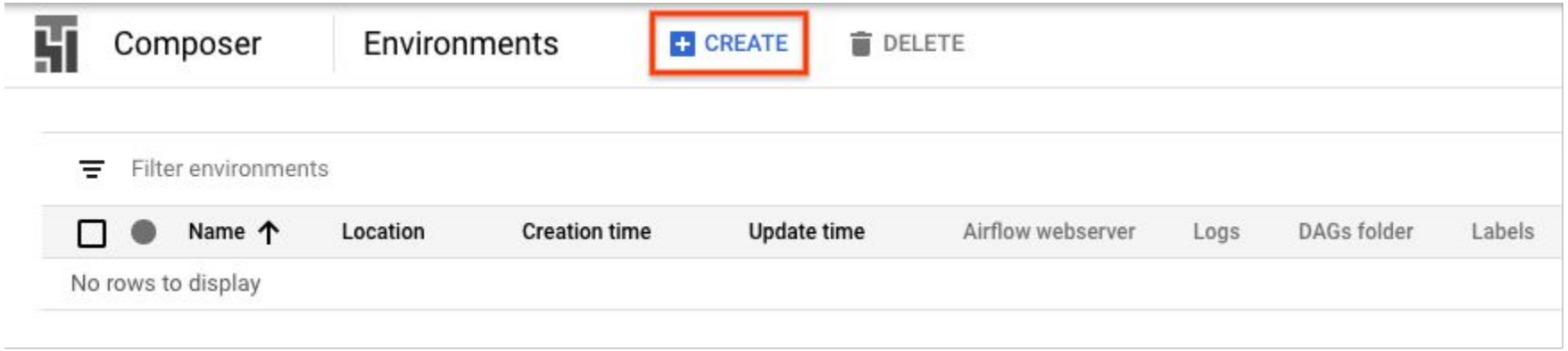
ASL Tech Talk:
Cloud Composer

ML models account for a small percentage

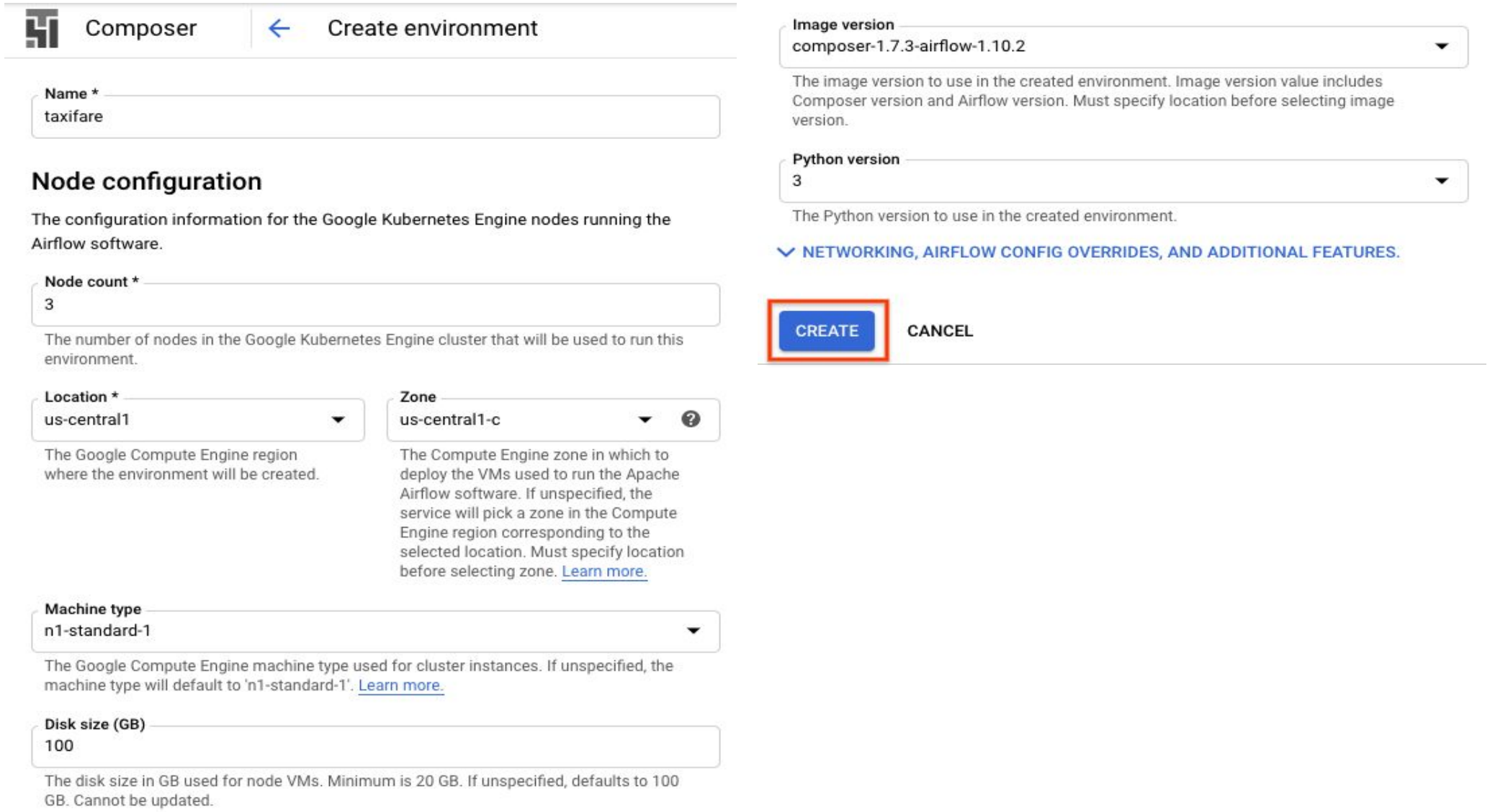
ML models account for a small percentage



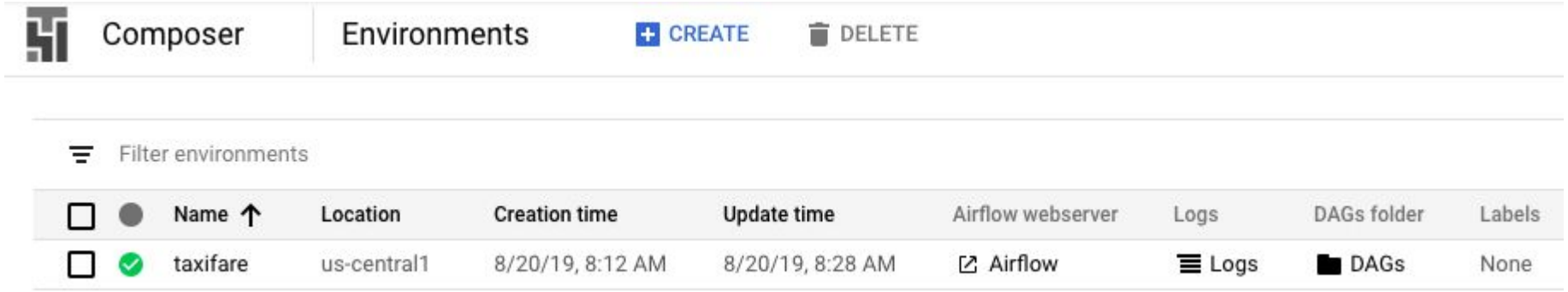
Create Composer Airflow Environment

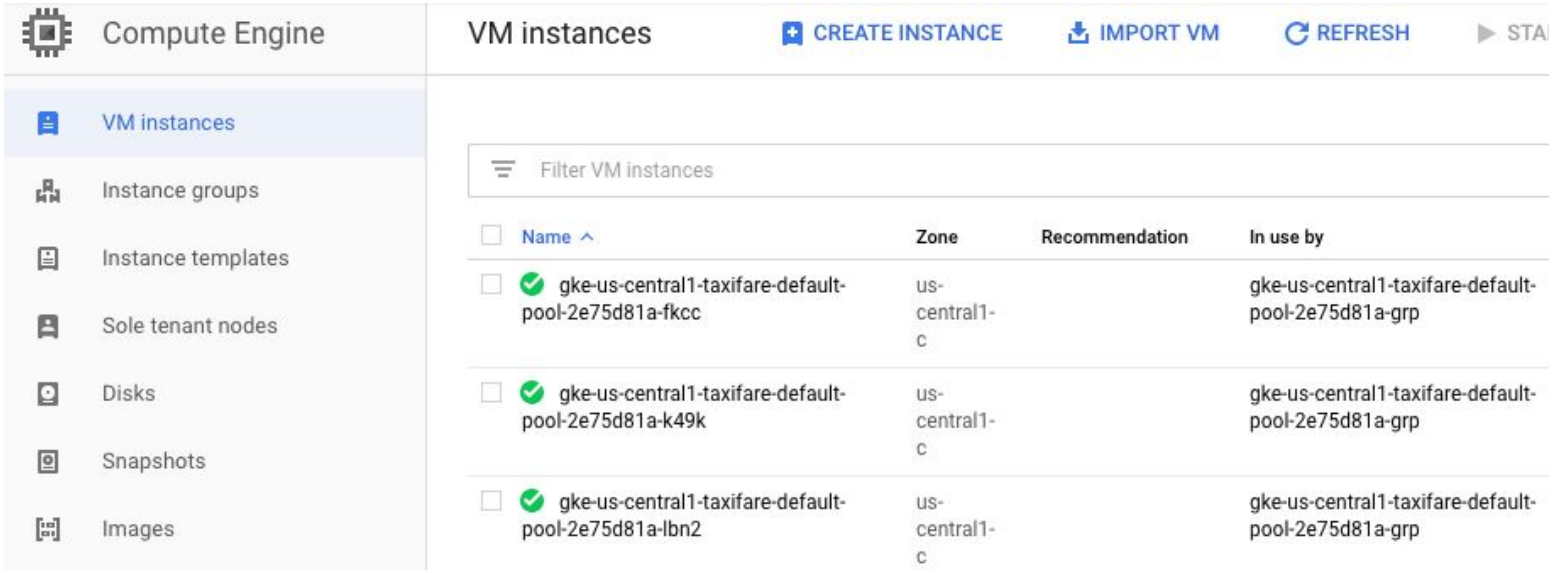


Choose a name, node count, location, etc. then create!

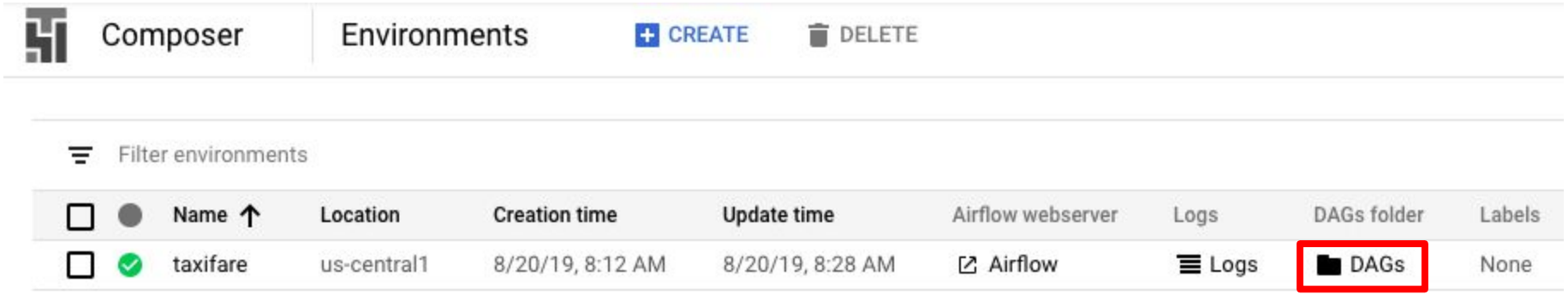


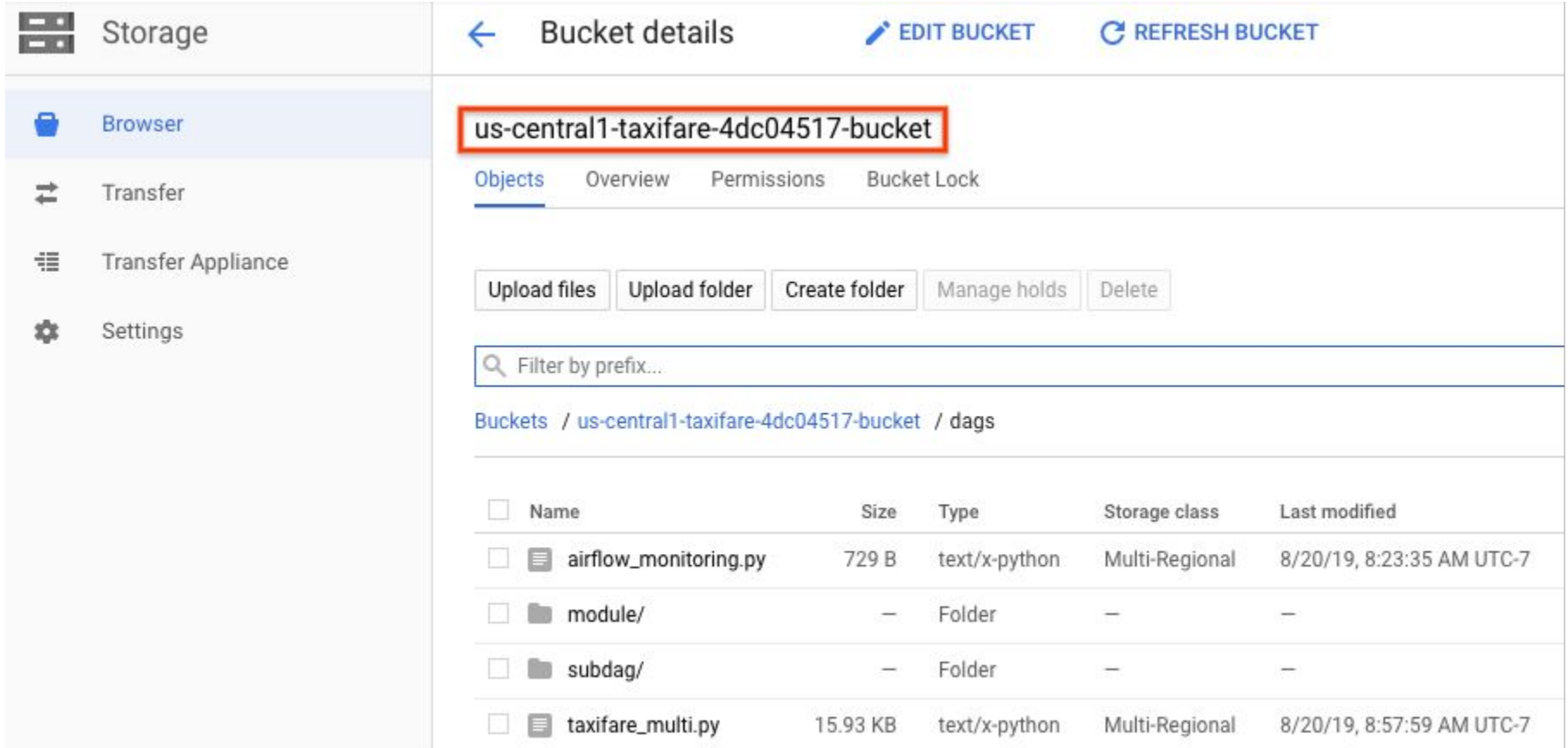
After creation, a Kubernetes cluster will be hosting the Composer Environment



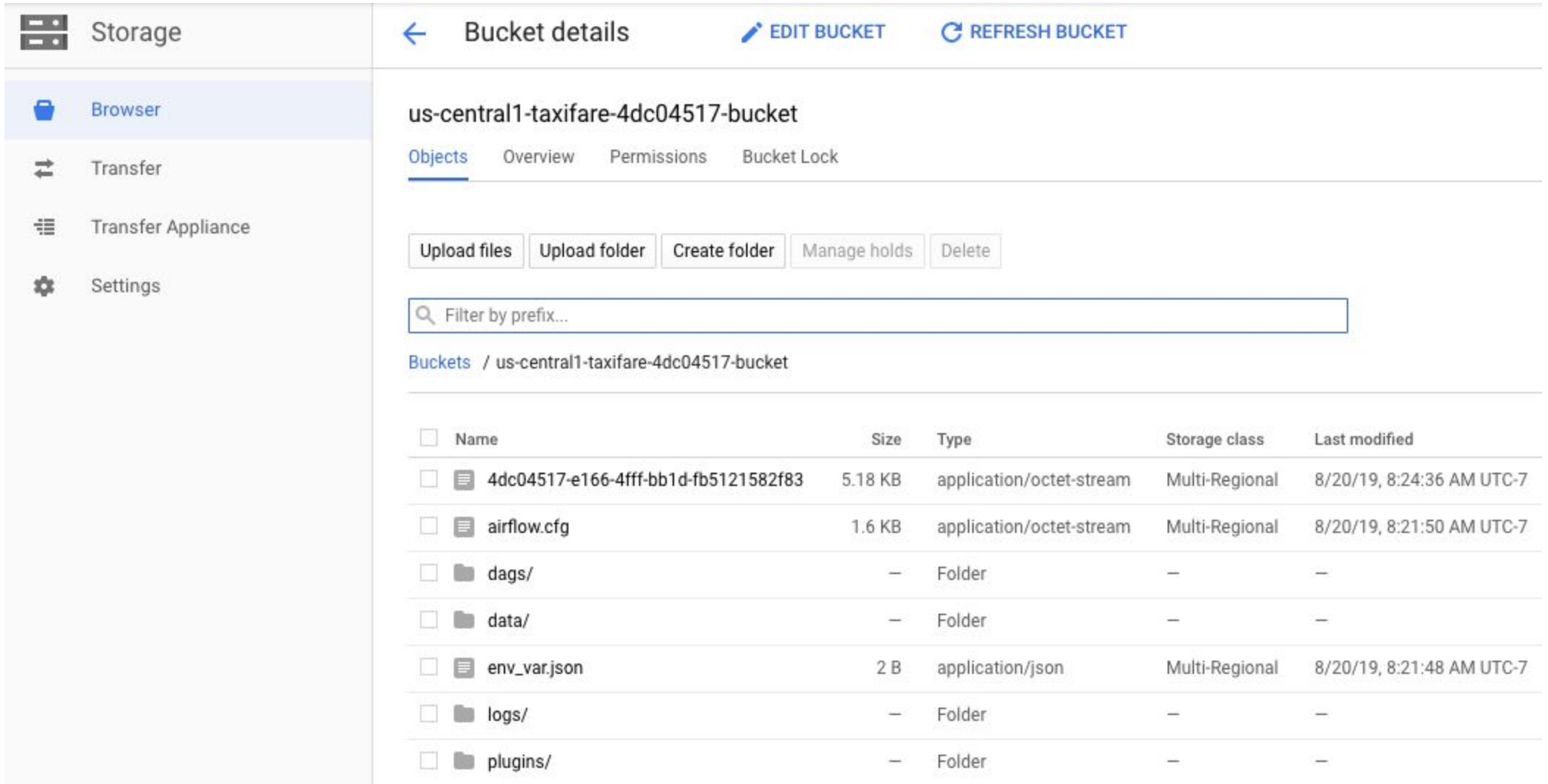


The DAGs folder is where our Python DAG files are stored

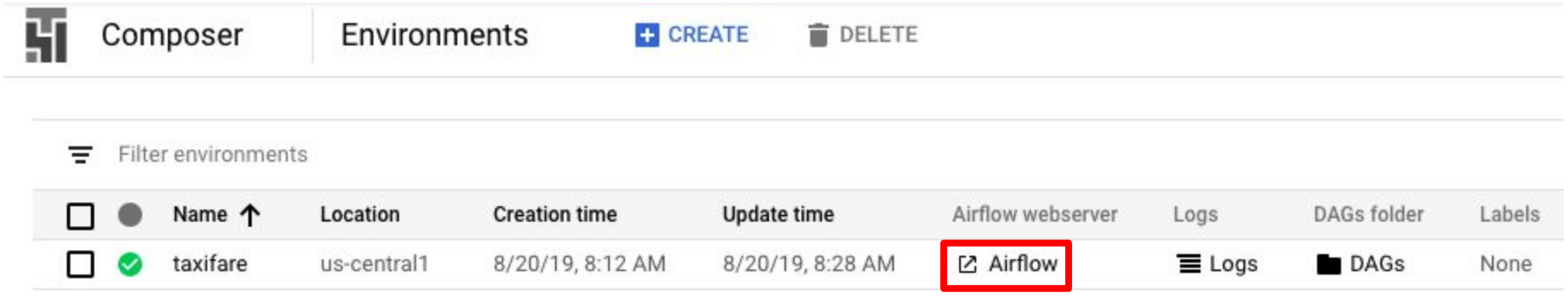


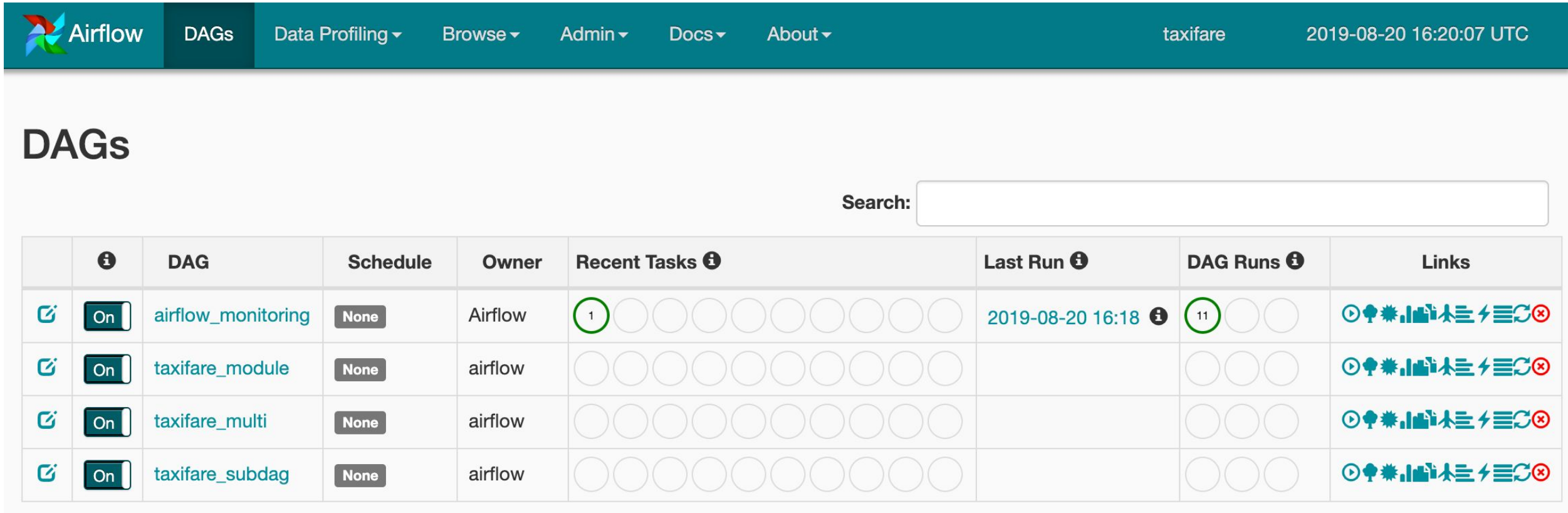


This is the root Airflow GCS Bucket



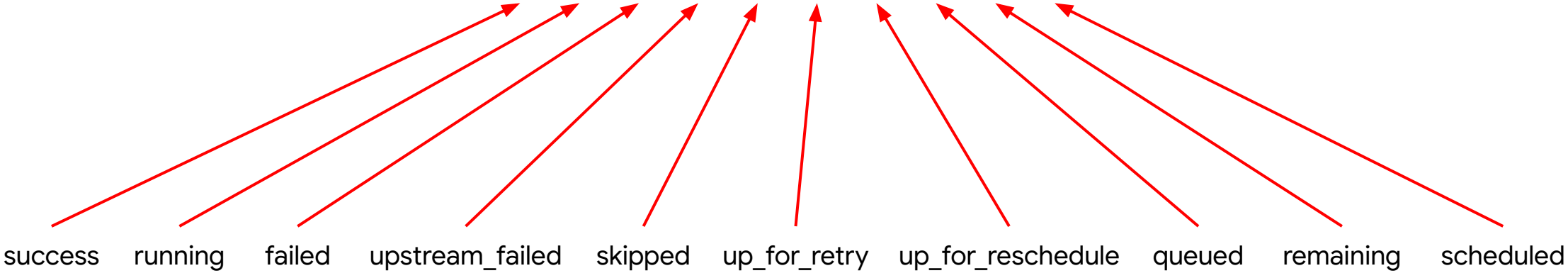
Access the Airflow Webserver





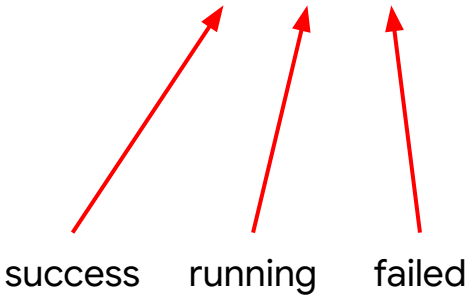
You can see the status of recent tasks

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs
airflow_monitoring	None	Airflow	1	2019-08-20 17:00	19
taxifare_module	None	airflow			
taxifare_multi	None	airflow	1 1 5 59	2019-08-20 16:53	1



You can also see the status of recent DAG runs

DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs
airflow_monitoring	None	Airflow	1	2019-08-20 17:00	19
taxifare_module	None	airflow			
taxifare_multi	None	airflow	1 1 5 59	2019-08-20 16:53	1



As DAG is running, you can see the status of each task in the graph

Common GCP Airflow Operators

- GCP: Google Cloud Platform
  - Logging
  - GoogleCloudBaseHook
  - BigQuery
  - Cloud Spanner
  - Cloud SQL
  - Cloud Bigtable
  - Cloud Build
  - Compute Engine
  - Cloud Functions
  - Cloud DataFlow
  - Cloud DataProc
  - Cloud Datastore
  - Cloud ML Engine
  - Cloud Storage
  - Transfer Service
  - Cloud Vision
    - Cloud Vision Product Search Operators
  - Cloud Text to Speech
  - Cloud Speech to Text

BigQuery

`airflow.contrib.operators.bigquery_check_operator.BigQueryCheckOperator`

Performs checks against a SQL query that will return a single row with different values.

`airflow.contrib.operators.bigquery_check_operator.BigQueryIntervalCheckOperator`

Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from days_back before.

`airflow.contrib.operators.bigquery_check_operator.BigQueryValueCheckOperator`

Performs a simple value check using SQL code.

`airflow.contrib.operators.bigquery_get_data.BigQueryGetDataOperator`

Fetches the data from a BigQuery table and returns data in a python list

`airflow.contrib.operators.bigquery_operator.BigQueryCreateEmptyDatasetOperator`

Creates an empty BigQuery dataset.

`airflow.contrib.operators.bigquery_operator.BigQueryCreateEmptyTableOperator`

Creates a new, empty table in the specified BigQuery dataset optionally with schema.

`airflow.contrib.operators.bigquery_operator.BigQueryCreateExternalTableOperator`

Creates a new, external table in the dataset with the data in Google Cloud Storage.

`airflow.contrib.operators.bigquery_operator.BigQueryDeleteDatasetOperator`

Deletes an existing BigQuery dataset.

`airflow.contrib.operators.bigquery_operator.BigQueryOperator`

Executes BigQuery SQL queries in a specific BigQuery database.

`airflow.contrib.operators.bigquery_table_delete_operator.BigQueryTableDeleteOperator`

Deletes an existing BigQuery table.

`airflow.contrib.operators.bigquery_to_bigquery.BigQueryToBigQueryOperator`

Copy a BigQuery table to another BigQuery table.

`airflow.contrib.operators.bigquery_to_gcs.BigQueryToCloudStorageOperator`

Transfers a BigQuery table to a Google Cloud Storage bucket

Cloud ML Engine

`airflow.contrib.operators.mlengine_operator.MLEngineBatchPredictionOperator`

Start a Cloud ML Engine batch prediction job.

`airflow.contrib.operators.mlengine_operator.MLEngineModelOperator`

Manages a Cloud ML Engine model.

`airflow.contrib.operators.mlengine_operator.MLEngineTrainingOperator`

Start a Cloud ML Engine training job.

`airflow.contrib.operators.mlengine_operator.MLEngineVersionOperator`

Manages a Cloud ML Engine model version.

They also use `airflow.contrib.hooks.gcp_mlengine_hook.MLEngineHook` to communicate with Google Cloud Platform.

Cloud Storage

`airflow.contrib.operators.file_to_gcs.FileToGoogleCloudStorageOperator`

Uploads a file to Google Cloud Storage.

`airflow.contrib.operators.gcs_acl_operator.GoogleCloudStorageBucketCreateAclEntryOperator`

Creates a new ACL entry on the specified bucket.

`airflow.contrib.operators.gcs_acl_operator.GoogleCloudStorageObjectCreateAclEntryOperator`

Creates a new ACL entry on the specified object.

`airflow.contrib.operators.gcs_download_operator.GoogleCloudStorageDownloadOperator`

Downloads a file from Google Cloud Storage.

`airflow.contrib.operators.gcs_list_operator.GoogleCloudStorageListOperator`

List all objects from the bucket with the give string prefix and delimiter in name.

`airflow.contrib.operators.gcs_operator.GoogleCloudStorageCreateBucketOperator`

Creates a new cloud storage bucket.

`airflow.contrib.operators.gcs_to_bq.GoogleCloudStorageToBigQueryOperator`

Loads files from Google cloud storage into BigQuery.

`airflow.contrib.operators.gcs_to_gcs.GoogleCloudStorageToGoogleCloudStorageOperator`

Copies objects from a bucket to another, with renaming if requested.

`airflow.contrib.operators.mysql_to_gcs.MySqlToGoogleCloudStorageOperator`

Copy data from any MySQL Database to Google cloud storage in JSON format.

`airflow.contrib.operators.mssql_to_gcs.MsSqlToGoogleCloudStorageOperator`

Copy data from any Microsoft SQL Server Database to Google Cloud Storage in JSON format.

`airflow.contrib.sensors.gcs_sensor.GoogleCloudStorageObjectSensor`

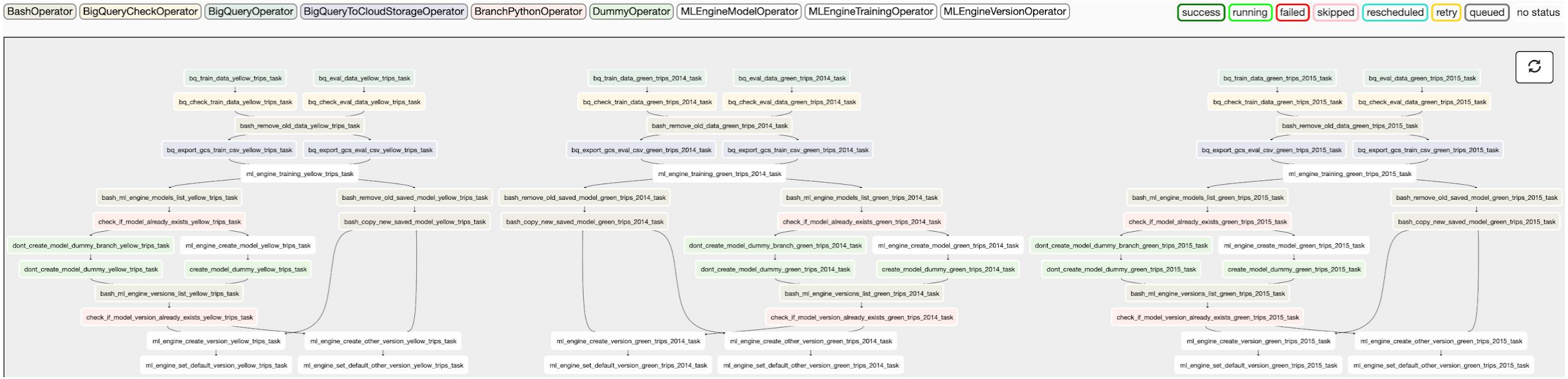
Checks for the existence of a file in Google Cloud Storage.

`airflow.contrib.sensors.gcs_sensor.GoogleCloudStorageObjectUpdatedSensor`

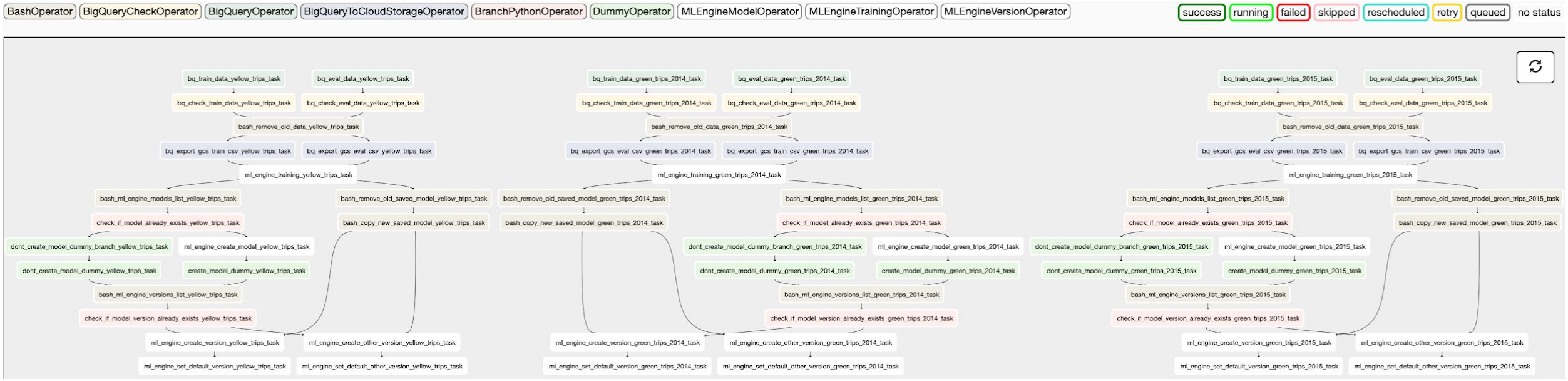
Checks if an object is updated in Google Cloud Storage.

`airflow.contrib.sensors.gcs_sensor.GoogleCloudStoragePrefixSensor`

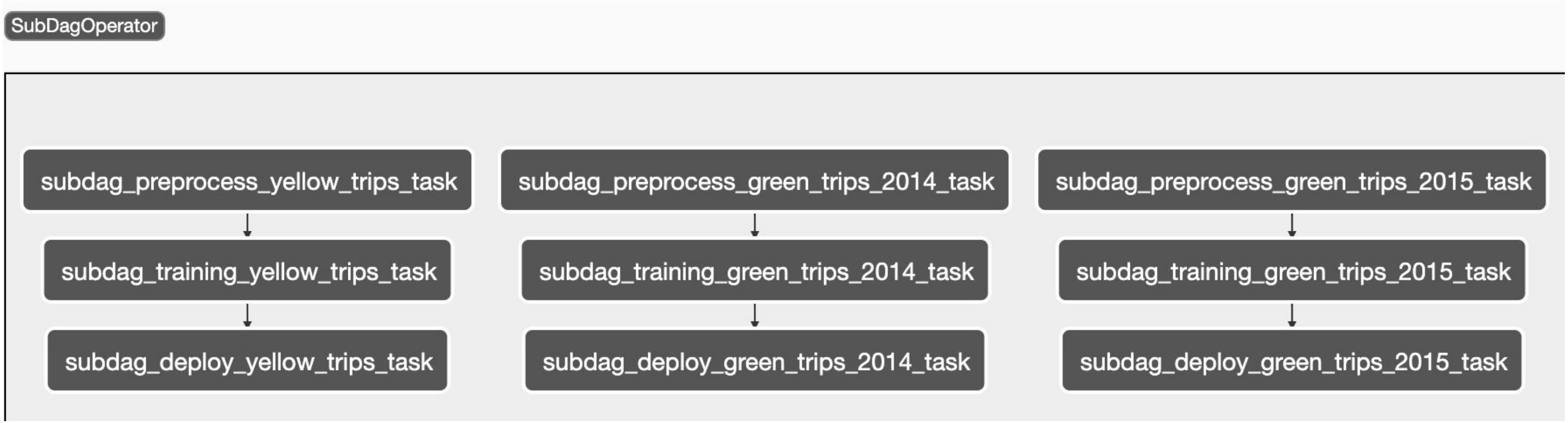
DAG: taxifare_multi



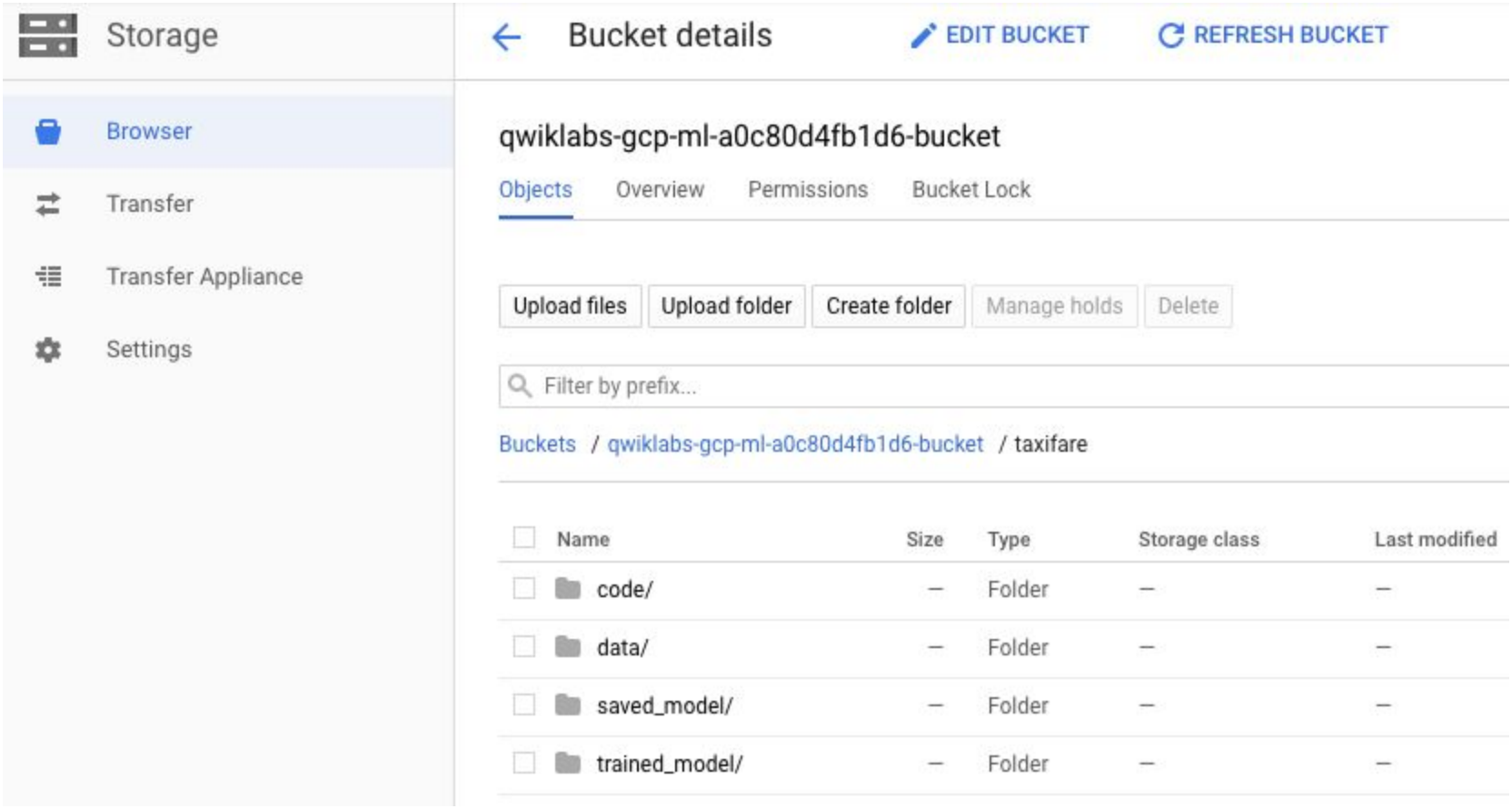
DAG: taxifare_module



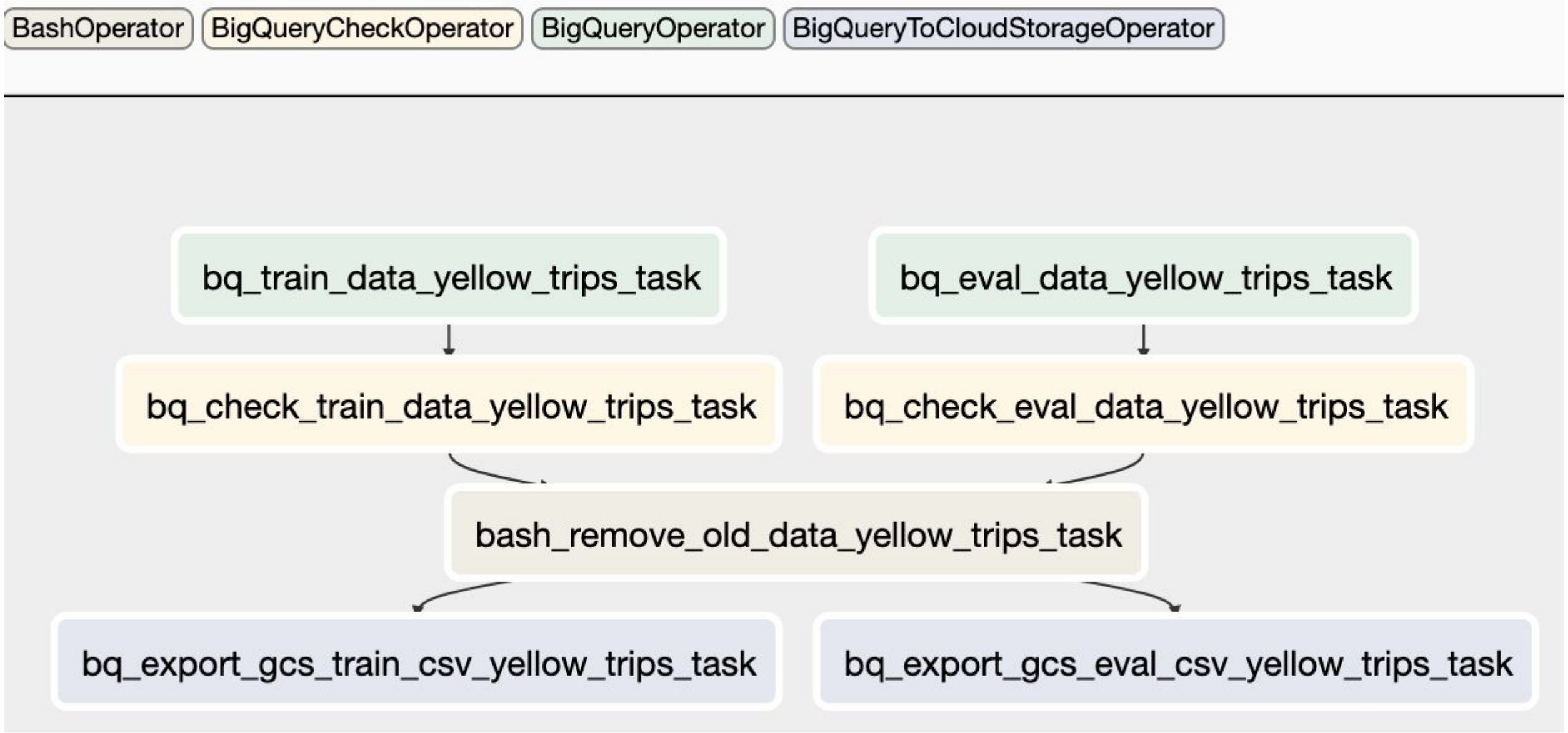
DAG: taxifare_subdag



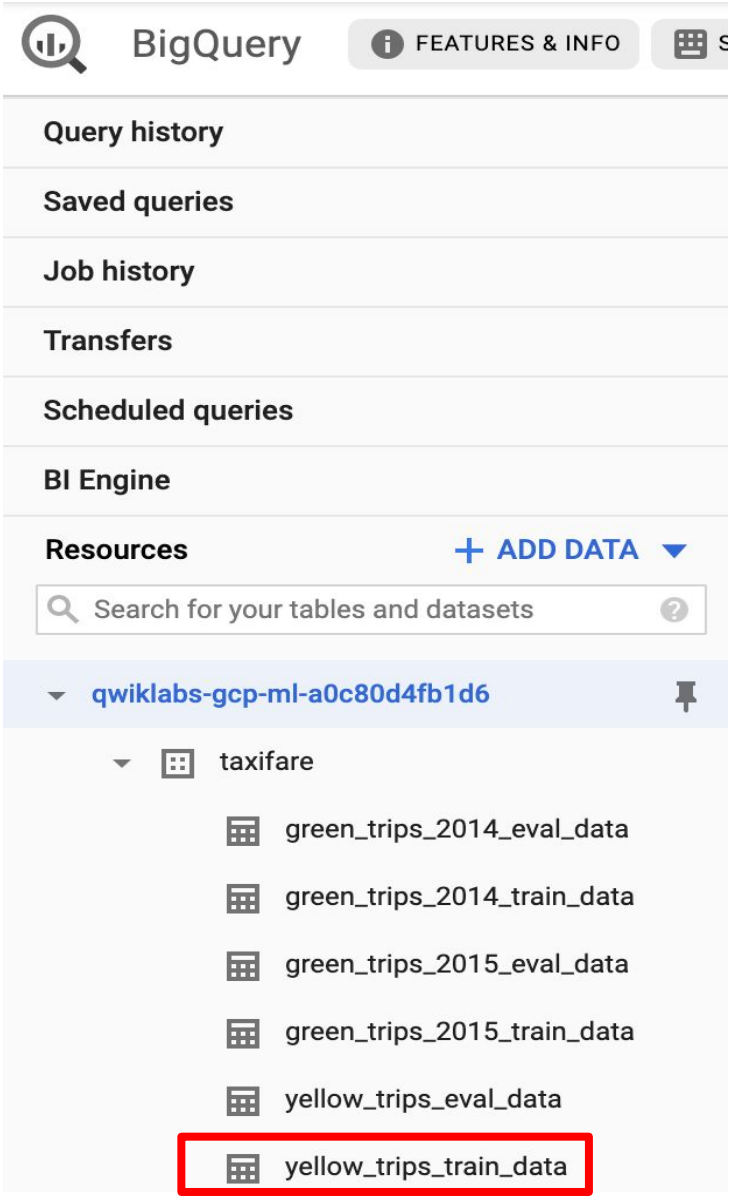
GCS staging location for entire pipeline



DAG: taxifare_multi Subdag: preprocess



Creates BigQuery training and evaluation tables



yellow_trips_train_data

Schema | Details | Preview

Field name	Type
fare_amount	FLOAT
dayofweek	FLOAT
hourofday	FLOAT
pickuplon	FLOAT
pickuplat	FLOAT
dropofflon	FLOAT
dropofflat	FLOAT
passengers	FLOAT

BigQueryOperator to create train and eval BigQuery tables

```
# BigQuery data query
bql="""
SELECT
  (tolls_amount + fare_amount) AS fare_amount,
  EXTRACT(DAYOFWEEK FROM pickup_datetime) * 1.0 AS dayofweek,
  EXTRACT(HOUR FROM pickup_datetime) * 1.0 AS hourofday,
  pickup_longitude AS pickuplon,
  pickup_latitude AS pickuplat,
  dropoff_longitude AS dropofflon,
  dropoff_latitude AS dropofflat,
  passenger_count*1.0 AS passengers,
  CONCAT(CAST(pickup_datetime AS STRING), CAST(pickup_longitude AS STRING), CAST(pickup_latitude AS STRING), CAST(dropoff_latitude AS STRING), CAST(dropoff_longitude AS STRING)) AS key
FROM
  `{0}.{1}`
WHERE
  trip_distance > 0
  AND fare_amount >= 2.5
  AND pickup_longitude > -78
  AND pickup_longitude < -70
  AND dropoff_longitude > -78
  AND dropoff_longitude < -70
  AND pickup_latitude > 37
  AND pickup_latitude < 45
  AND dropoff_latitude > 37
  AND dropoff_latitude < 45
  AND passenger_count > 0
  AND rand() < 0.00001
"""

bql = bql.format(SOURCE_BQ_PROJECT, model)

bql_train = "SELECT * EXCEPT (key) FROM({0}) WHERE MOD(ABS(FARM_FINGERPRINT(key)), 5) < 4".format(bql)
bql_eval = "SELECT * EXCEPT (key) FROM({0}) WHERE MOD(ABS(FARM_FINGERPRINT(key)), 5) = 4".format(bql)

# Complete the BigQueryOperator task to truncate the table if it already exists before writing
# Reference: https://airflow.apache.org/integration.html#bigqueryoperator
bq_train_data_op = BigQueryOperator(
  task_id="bq_train_data_{}_task".format(model.replace(".","_")),
  bql=bql_train,
  destination_dataset_table="{}.{}_train_data".format(DESTINATION_DATASET, model.replace(".","_")),
  write_disposition="WRITE_TRUNCATE", # specify to truncate on writes
  use_legacy_sql=False,
  dag=dag
)

bq_eval_data_op = BigQueryOperator(
  task_id="bq_eval_data_{}_task".format(model.replace(".","_")),
  bql=bql_eval,
  destination_dataset_table="{}.{}_eval_data".format(DESTINATION_DATASET, model.replace(".","_")),
  write_disposition="WRITE_TRUNCATE", # specify to truncate on writes
  use_legacy_sql=False,
  dag=dag
)
```

Details of training BigQuery table

yellow_trips_train_data

QUERY TABLE COPY

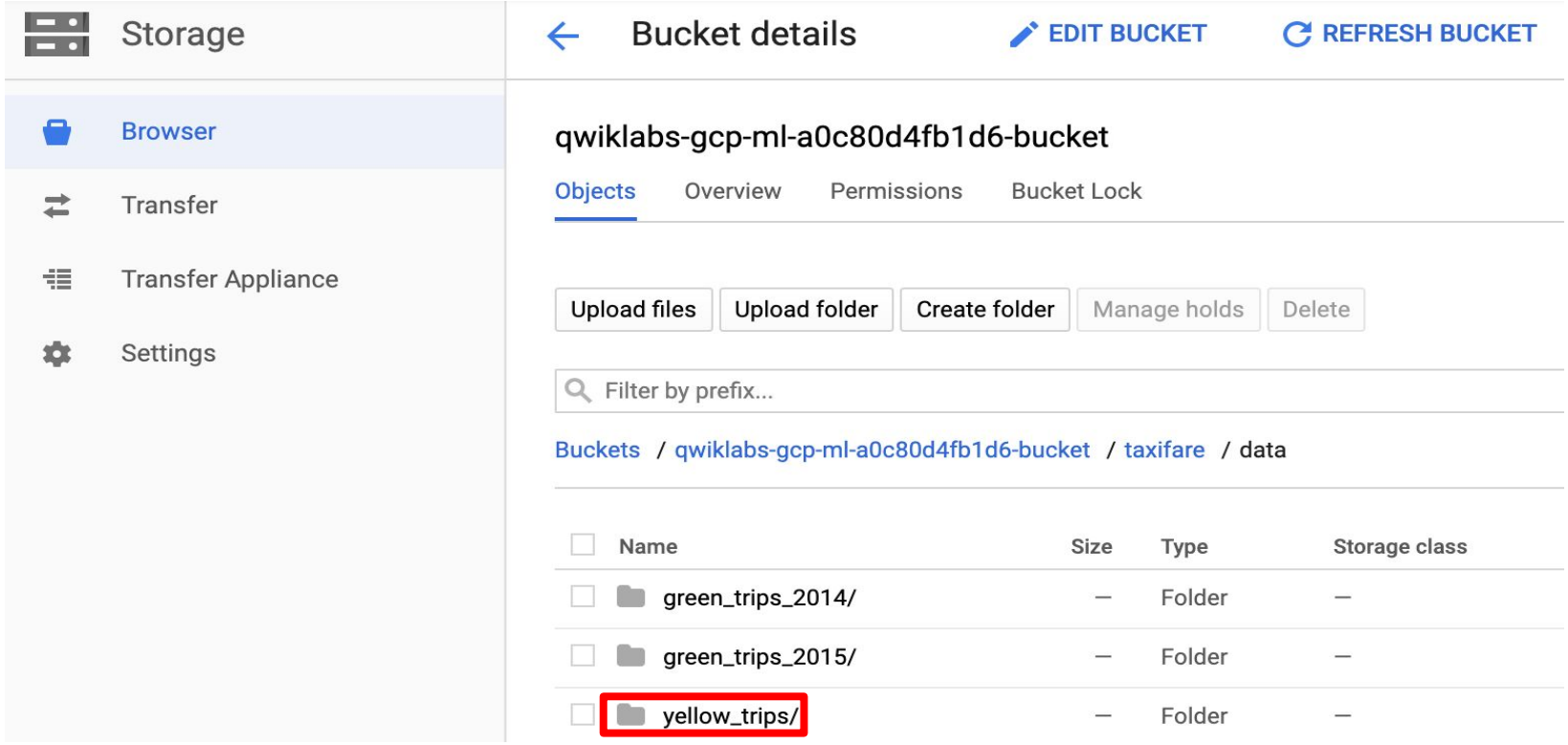
Schema Details Preview

Row	fare_amount	dayofweek	hourofday	pickuplon	pickuplat	dropofflon	dropofflat	passengers
1	15.7	2.0	0.0	-74.002895	40.720132	-73.948602	40.78798	5.0
2	10.5	2.0	0.0	-73.954148	40.778802	-73.961588	40.812907	1.0
3	14.5	2.0	0.0	-73.998647	40.740743	-73.93529	40.750498	1.0
4	11.3	2.0	0.0	-73.97982	40.726993	-73.987161	40.759539	2.0
5	8.9	2.0	0.0	-73.992568	40.714073	-73.989102	40.69129	1.0

BigQueryCheckOperator to validate that the tables have data

```
sql = """
SELECT
  COUNT(*)
FROM
  [{0}:{1}.{2}]
"""

# Check to make sure that the data tables won"t be empty
bq_check_train_data_op = BigQueryCheckOperator(
  task_id="bq_check_train_data_{}_task".format(model.replace(".","_")),
  sql=sql.format(PROJECT_ID, DESTINATION_DATASET, model.replace(".","_") + "_train_data"),
  dag=dag
)

bq_check_eval_data_op = BigQueryCheckOperator(
  task_id="bq_check_eval_data_{}_task".format(model.replace(".","_")),
  sql=sql.format(PROJECT_ID, DESTINATION_DATASET, model.replace(".","_") + "_eval_data"),
  dag=dag
)
```

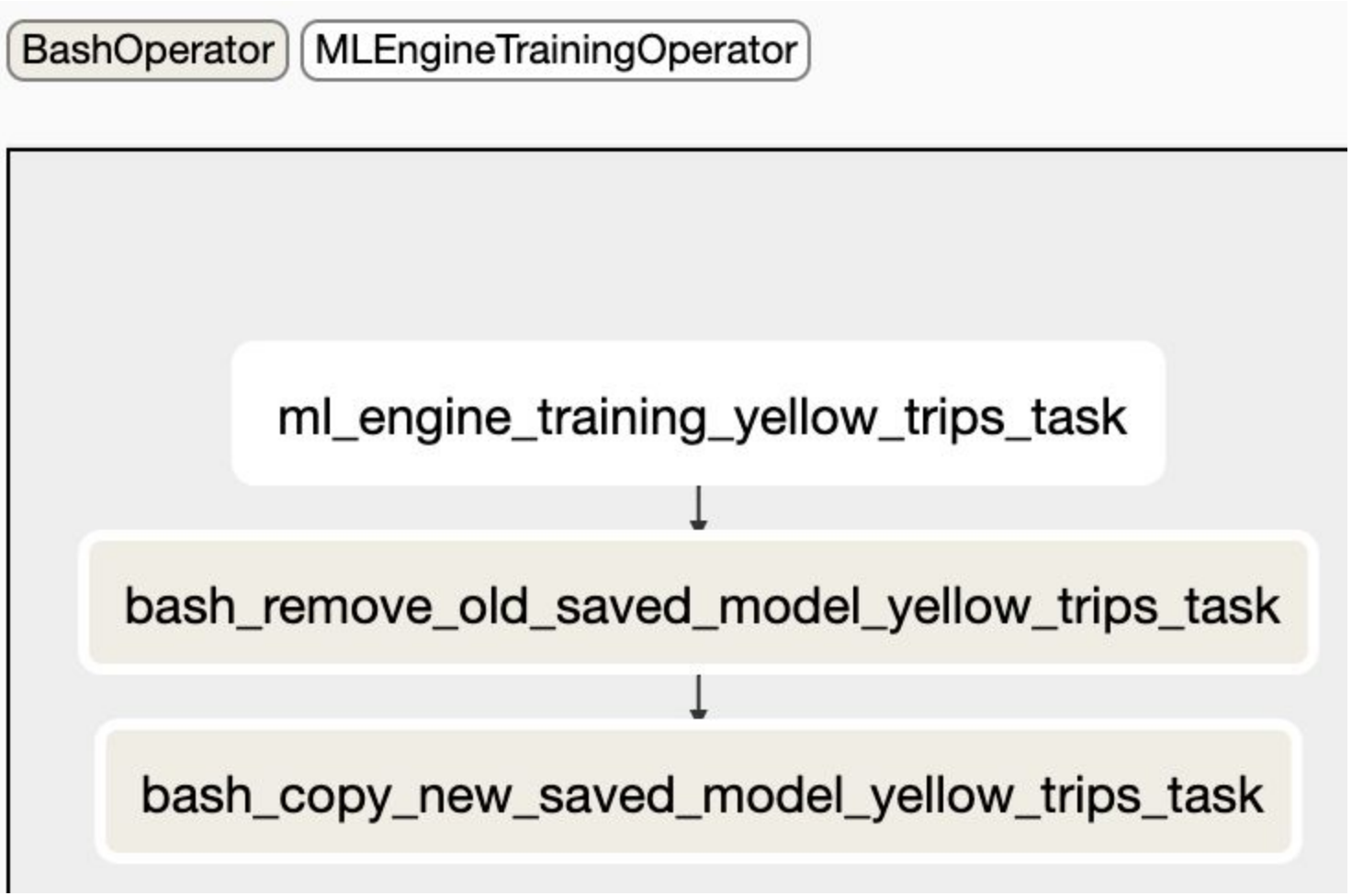
GCS location where CSV files are written from BigQuery to be read by TensorFlow



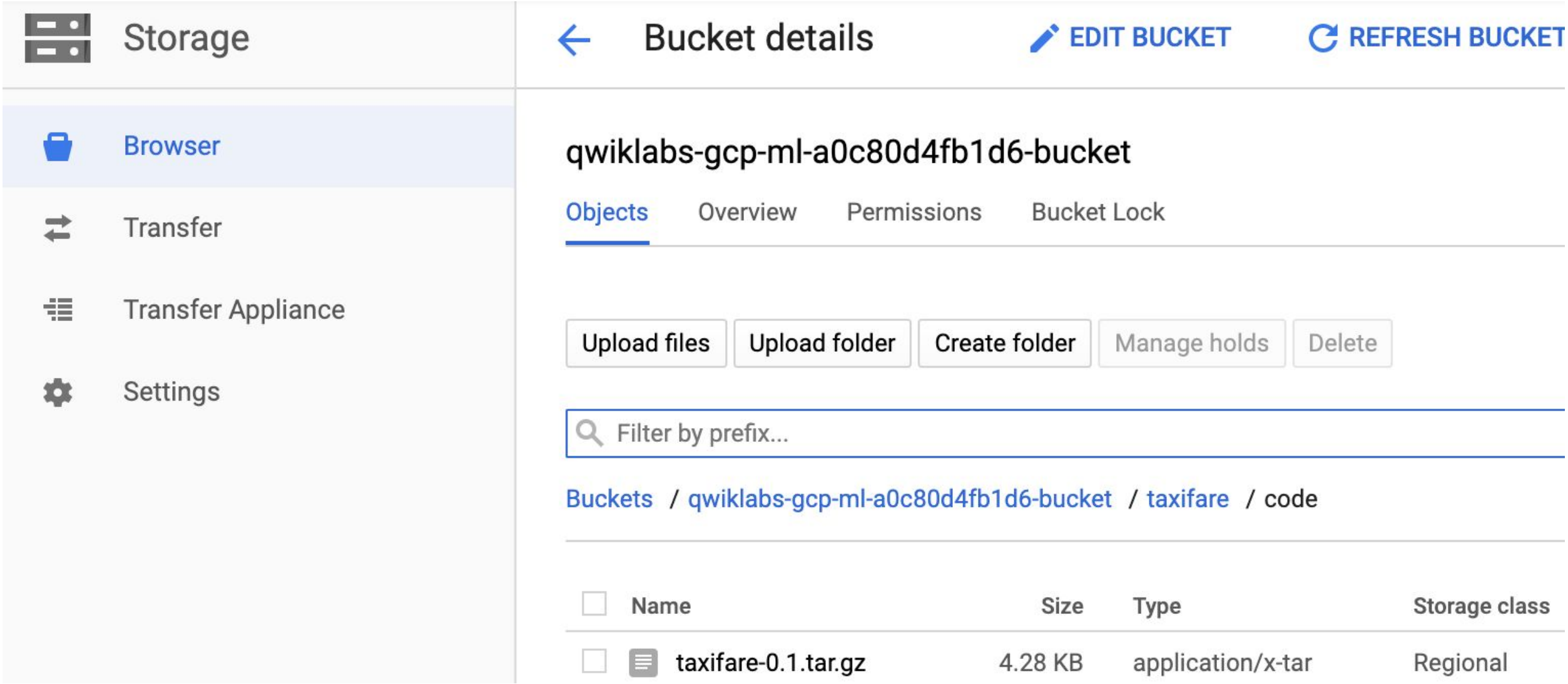
BigQueryToCloudStorageOperator to export from BigQuery to CSV

```
# BigQuery training data export to GCS
bash_remove_old_data_op = BashOperator(
  task_id="bash_remove_old_data_{}_task".format(model.replace(".","_")),
  bash_command="if gsutil ls {0}/taxifare/data/{1} 2> /dev/null; then gsutil -m rm -rf {0}/taxifare/data/{1}/*; else true; fi".format(BUCKET, model.replace(".","_")),
  dag=dag
)

# Takes a BigQuery dataset and table as input and exports it to GCS as a CSV
train_files = BUCKET + "/taxifare/data/"

bq_export_gcs_train_csv_op = BigQueryToCloudStorageOperator(
  task_id="bq_export_gcs_train_csv_{}_task".format(model.replace(".","_")),
  source_project_dataset_table="{}.{}_train_data".format(DESTINATION_DATASET, model.replace(".","_")),
  destination_cloud_storage_uris=[train_files + "{}/train-*.csv".format(model.replace(".","_"))],
  export_format="CSV",
  print_header=False,
  dag=dag
)

eval_files = BUCKET + "/taxifare/data/"

bq_export_gcs_eval_csv_op = BigQueryToCloudStorageOperator(
  task_id="bq_export_gcs_eval_csv_{}_task".format(model.replace(".","_")),
  source_project_dataset_table="{}.{}_eval_data".format(DESTINATION_DATASET, model.replace(".","_")),
  destination_cloud_storage_uris=[eval_files + "{}/eval-*.csv".format(model.replace(".","_"))],
  export_format="CSV",
  print_header=False,
  dag=dag
)
```

DAG: taxifare_multi Subdag: training



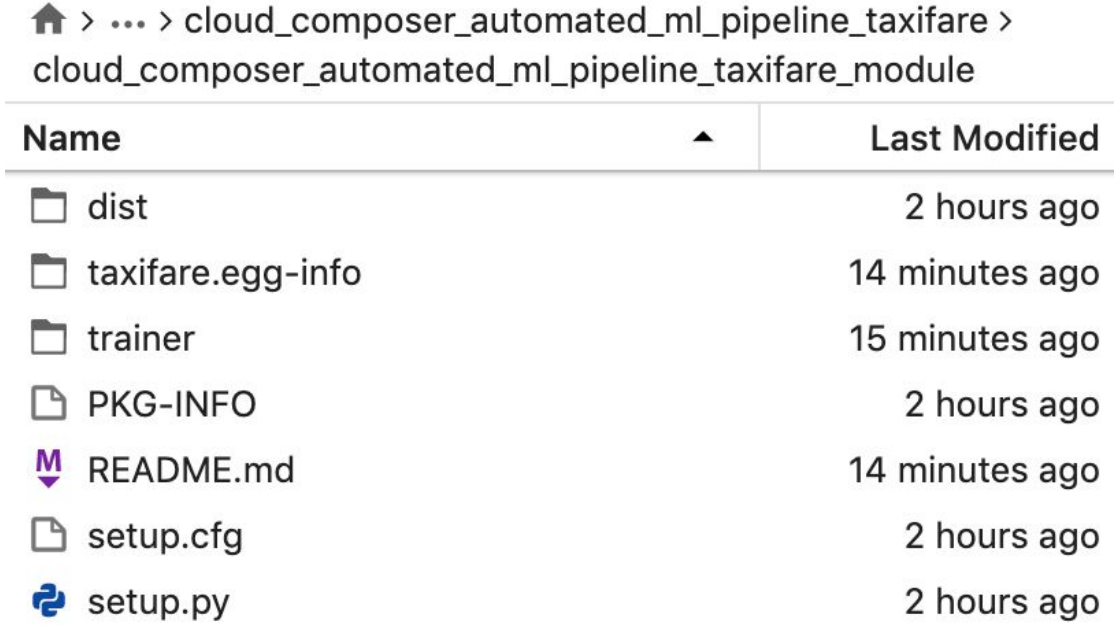
Make sure you have your trainer package in your code folder in Google Cloud Storage.

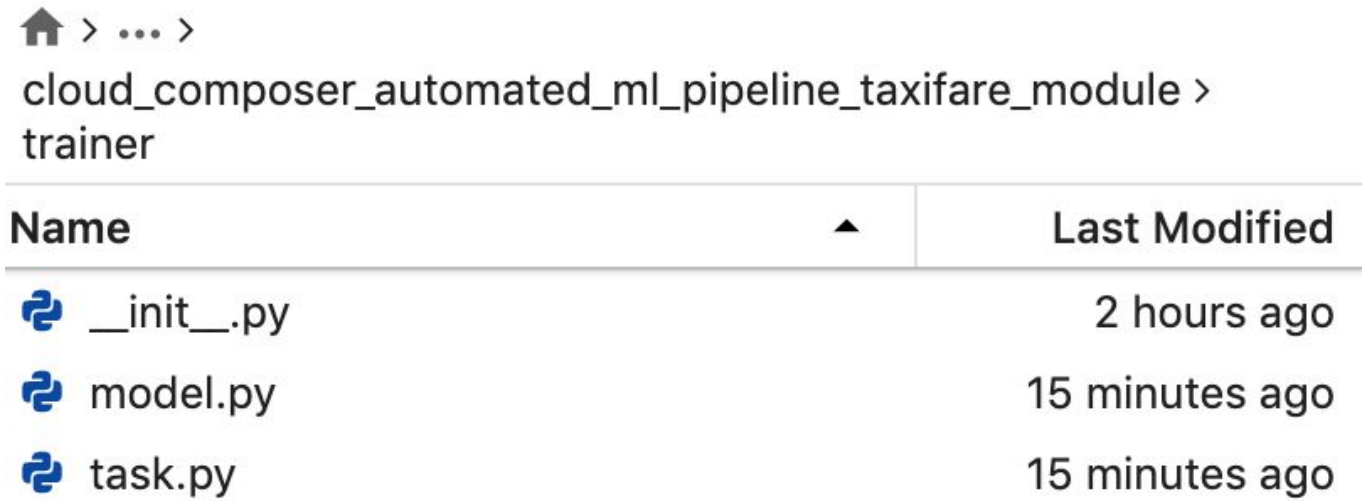


Create a source distribution and copy to GCS

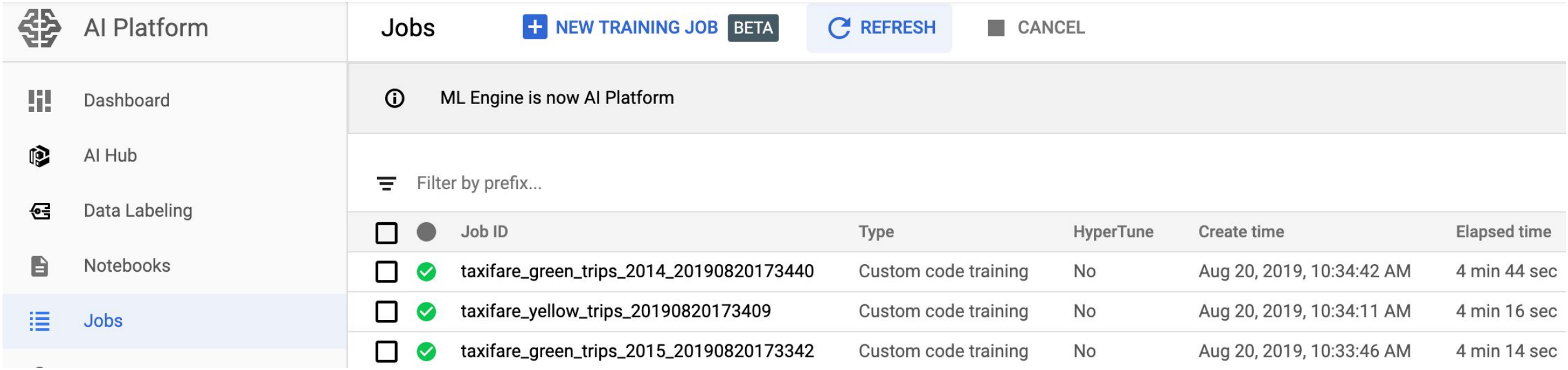
```
%%bash
cd cloud_composer_automated_ml_pipeline_taxifare_module
touch README.md
python setup.py sdist
```

```
%%bash
gsutil cp cloud_composer_automated_ml_pipeline_taxifare_module/dist/taxifare-0.1.tar.gz gs://$BUCKET/taxifare/code/
```





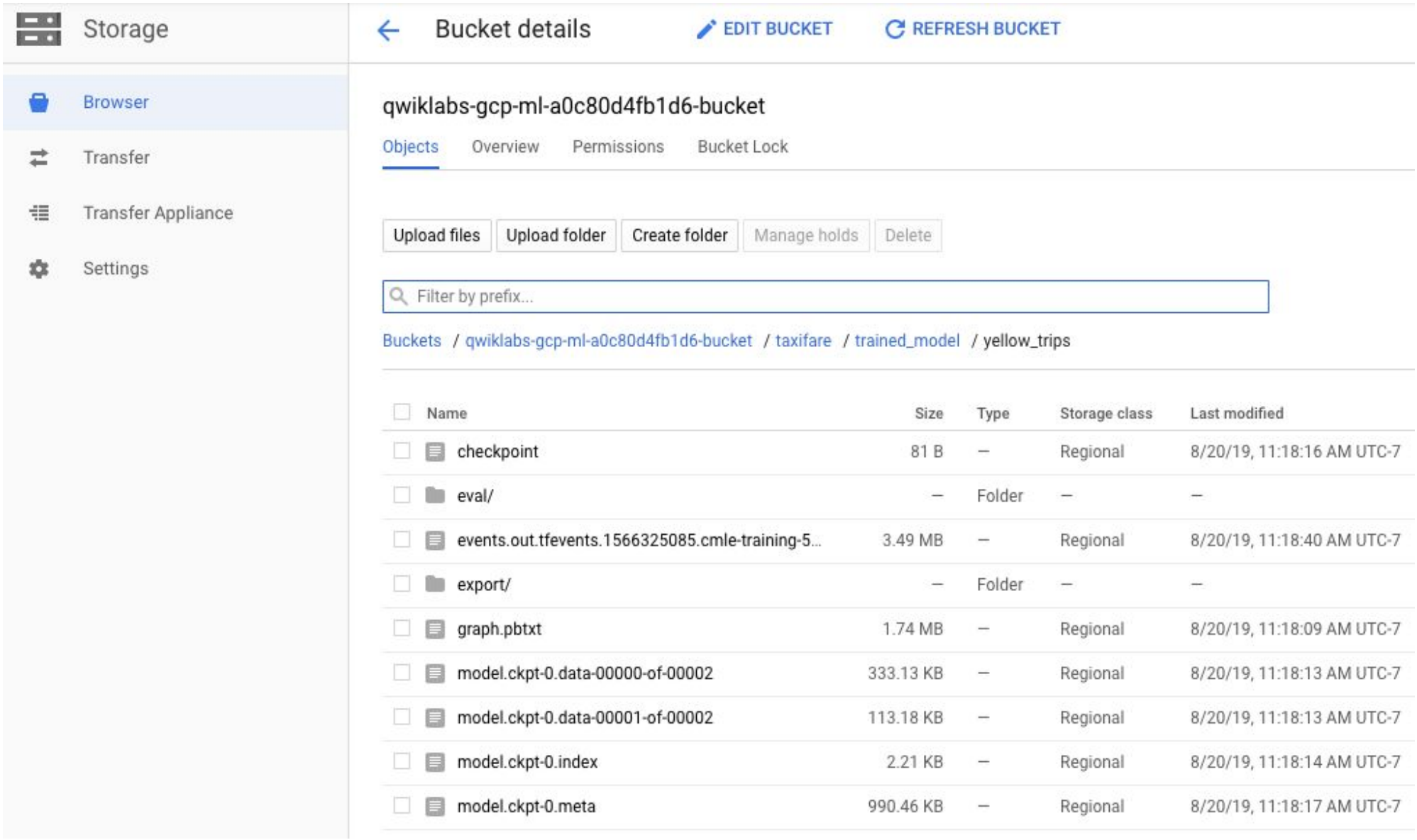
Cloud AI Platform training jobs kicked off in parallel



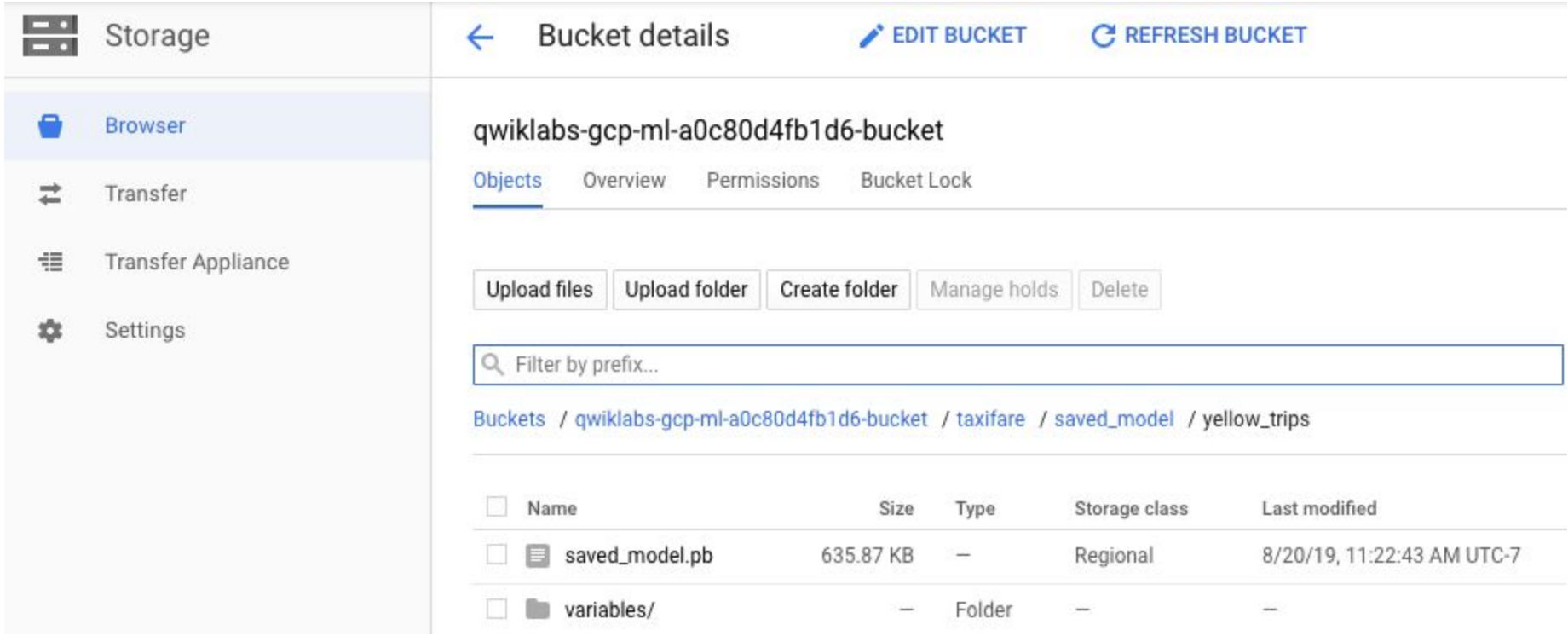
MLEngineTrainingOperator to create training jobs

```
# ML Engine training job
job_id = "taxifare_{}_{}".format(model.replace(".","_"), datetime.datetime.now().strftime("%Y%m%d%H%M%S"))
output_dir = BUCKET + "/taxifare/trained_model/{}".format(model.replace(".","_"))
job_dir = JOB_DIR + "/" + job_id
training_args = [
  "--job-dir", job_dir,
  "--train_data_paths", train_files,
  "--eval_data_paths", eval_files,
  "--output_dir", output_dir,
  "--train_steps", str(500),
  "--train_batch_size", str(32),
  "--eval_steps", str(500),
  "--eval_batch_size", str(32),
  "--nbuckets", str(8),
  "--hidden_units", "128,32,4"
]

# Reference: https://airflow.apache.org/integration.html#cloud-ml-engine
ml_engine_training_op = MLEngineTrainingOperator(
  task_id="ml_engine_training_{}_task".format(model.replace(".","_")),
  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="BASIC",
  runtime_version="1.13",
  python_version="3.5",
  dag=dag
)
```

GCS location where model training files are written



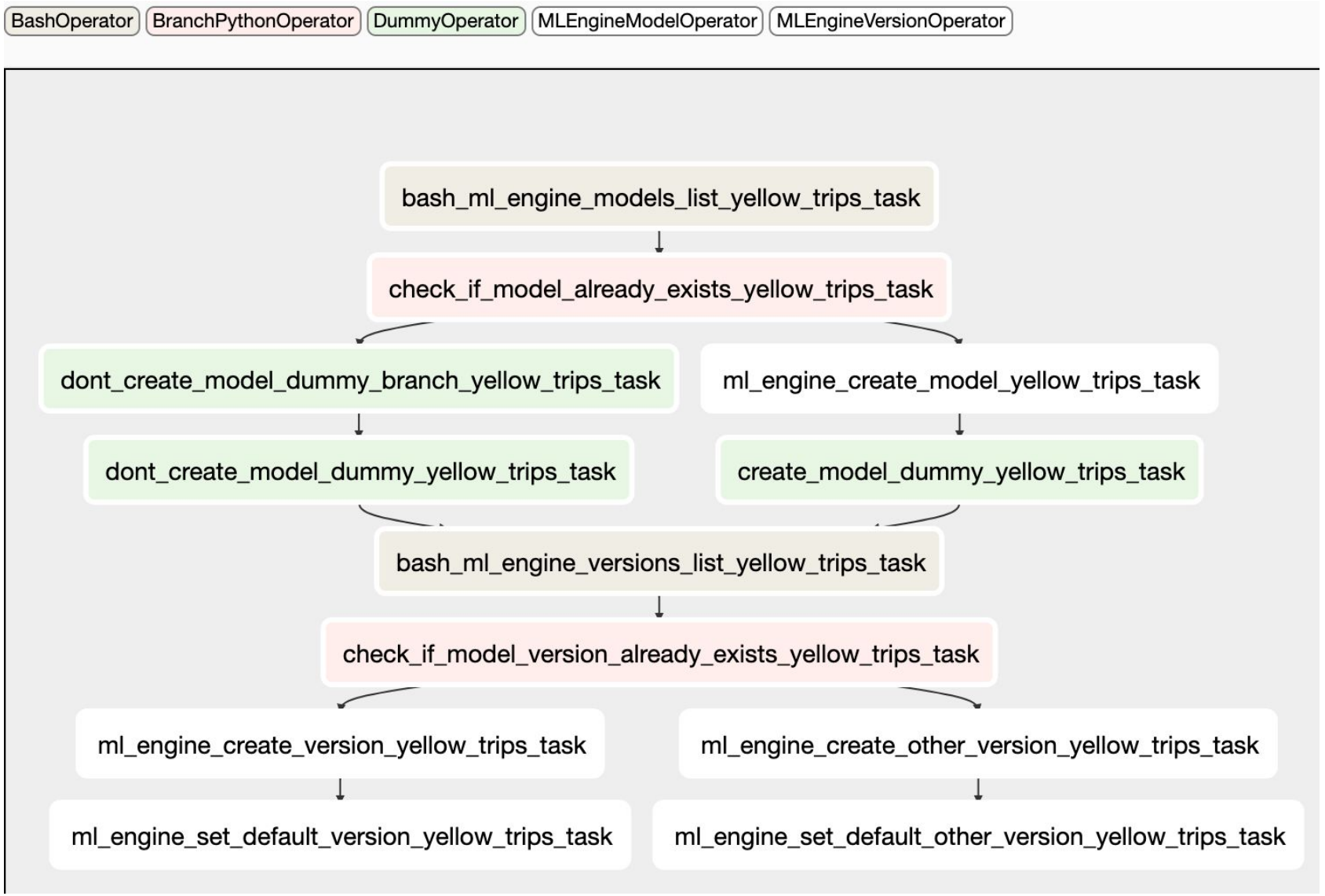
GCS location where saved model files are moving for deployment



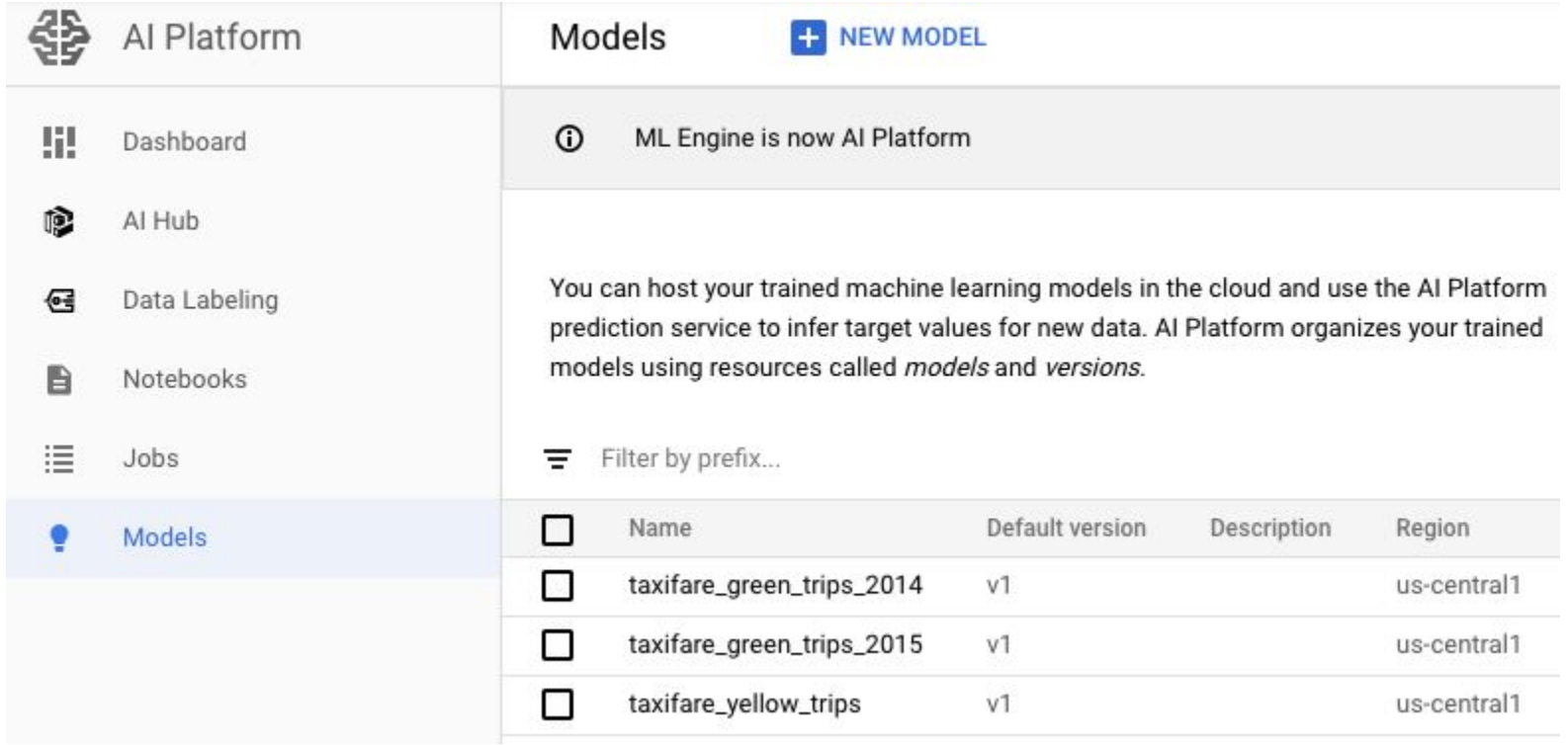
BashOperators to replace old saved model files with new ones

```
MODEL_NAME = "taxifare_"
MODEL_VERSION = "v1"
MODEL_LOCATION = BUCKET + "/taxifare/saved_model/"

bash_remove_old_saved_model_op = BashOperator(
  task_id="bash_remove_old_saved_model_{}_task".format(model.replace(".","_")),
  bash_command="if gsutil ls {0} 2> /dev/null; then gsutil -m rm -rf {0}/*; else true; fi".format(MODEL_LOCATION + model.replace(".","_")),
  dag=dag
)

bash_copy_new_saved_model_op = BashOperator(
  task_id="bash_copy_new_saved_model_{}_task".format(model.replace(".","_")),
  bash_command="gsutil -m rsync -d -r `gsutil ls {0}/export/exporter/ | tail -1` {1}".format(output_dir, MODEL_LOCATION + model.replace(".","_")),
  dag=dag
)
```

DAG: taxifare_multi Subdag: deploy



Models will end up deployed on Cloud AI Platform



First check to if model already exists on Cloud AI Platform

```
# Create model on ML-Engine
bash_ml_engine_models_list_op = BashOperator(
  task_id="bash_ml_engine_models_list_{}_task".format(model.replace(".","_")),
  xcom_push=True,
  bash_command="gcloud ml-engine models list --filter='name:{0}'".format(MODEL_NAME + model.replace(".","_")),
  dag=dag
)

def check_if_model_already_exists(templates_dict, **kwargs):
  cur_model = templates_dict["model"].replace(".","_")
  ml_engine_models_list = kwargs["ti"].xcom_pull(task_ids="bash_ml_engine_models_list_{}_task".format(cur_model))
  logging.info("check_if_model_already_exists: {}: ml_engine_models_list = \n{}".format(cur_model, ml_engine_models_list))
  create_model_task = "ml_engine_create_model_{}_task".format(cur_model)
  dont_create_model_task = "dont_create_model_dummy_branch_{}_task".format(cur_model)
  if len(ml_engine_models_list) == 0 or ml_engine_models_list == "Listed 0 items.":
    return create_model_task
  return dont_create_model_task

check_if_model_already_exists_op = BranchPythonOperator(
  task_id="check_if_model_already_exists_{}_task".format(model.replace(".","_")),
  templates_dict={"model": model.replace(".","_")},
  python_callable=check_if_model_already_exists,
  provide_context=True,
  dag=dag
)
```

Create model if it doesn't exist, else skip

```
ml_engine_create_model_op = MLEngineModelOperator(
  task_id="ml_engine_create_model_{}_task".format(model.replace(".","_")),
  project_id=PROJECT_ID,
  model={"name": MODEL_NAME + model.replace(".","_")},
  operation="create",
  dag=dag
)

create_model_dummy_op = DummyOperator(
  task_id="create_model_dummy_{}_task".format(model.replace(".","_")),
  trigger_rule="all_done",
  dag=dag
)

dont_create_model_dummy_branch_op = DummyOperator(
  task_id="dont_create_model_dummy_branch_{}_task".format(model.replace(".","_")),
  dag=dag
)

dont_create_model_dummy_op = DummyOperator(
  task_id="dont_create_model_dummy_{}_task".format(model.replace(".","_")),
  trigger_rule="all_done",
  dag=dag
)
```

Check to if model_version already exists within model

```
# Create version of model on ML-Engine
bash_ml_engine_versions_list_op = BashOperator(
  task_id="bash_ml_engine_versions_list_{}_task".format(model.replace(".","_")),
  xcom_push=True,
  bash_command="gcloud ml-engine versions list --model {0} --filter='name:{1}'".format(MODEL_NAME + model.replace(".","_"), MODEL_VERSION),
  dag=dag
)

def check_if_model_version_already_exists(templates_dict, **kwargs):
  cur_model = templates_dict["model"].replace(".","_")
  ml_engine_versions_list = kwargs["ti"].xcom_pull(task_ids="bash_ml_engine_versions_list_{}_task".format(cur_model))
  logging.info("check_if_model_version_already_exists: {}: ml_engine_versions_list = \n{}".format(cur_model, ml_engine_versions_list))
  create_version_task = "ml_engine_create_version_{}_task".format(cur_model)
  create_other_version_task = "ml_engine_create_other_version_{}_task".format(cur_model)
  if len(ml_engine_versions_list) == 0 or ml_engine_versions_list == "Listed 0 items.":
    return create_version_task
  return create_other_version_task

check_if_model_version_already_exists_op = BranchPythonOperator(
  task_id="check_if_model_version_already_exists_{}_task".format(model.replace(".","_")),
  templates_dict={"model": model.replace(".","_")},
  python_callable=check_if_model_version_already_exists,
  provide_context=True,
  dag=dag
)
```

Create model_version if it doesn't exist, else create other version

```
OTHER_VERSION_NAME = "v_{0}".format(datetime.datetime.now().strftime("%Y%m%d%H%M%S")[0:12]

ml_engine_create_version_op = MLEngineVersionOperator(
  task_id="ml_engine_create_version_{}_task".format(model.replace(".","_")),
  project_id=PROJECT_ID,
  model_name=MODEL_NAME + model.replace(".","_"),
  version_name=MODEL_VERSION,
  version={
    "name": MODEL_VERSION,
    "deploymentUri": MODEL_LOCATION + model.replace(".","_"),
    "runtimeVersion": "1.13",
    "framework": "TENSORFLOW",
    "pythonVersion": "3.5",
  },
  operation="create",
  dag=dag
)

ml_engine_create_other_version_op = MLEngineVersionOperator(
  task_id="ml_engine_create_other_version_{}_task".format(model.replace(".","_")),
  project_id=PROJECT_ID,
  model_name=MODEL_NAME + model.replace(".","_"),
  version_name=OTHER_VERSION_NAME,
  version={
    "name": OTHER_VERSION_NAME,
    "deploymentUri": MODEL_LOCATION + model.replace(".","_"),
    "runtimeVersion": "1.13",
    "framework": "TENSORFLOW",
    "pythonVersion": "3.5",
  },
  operation="create",
  dag=dag
)
```

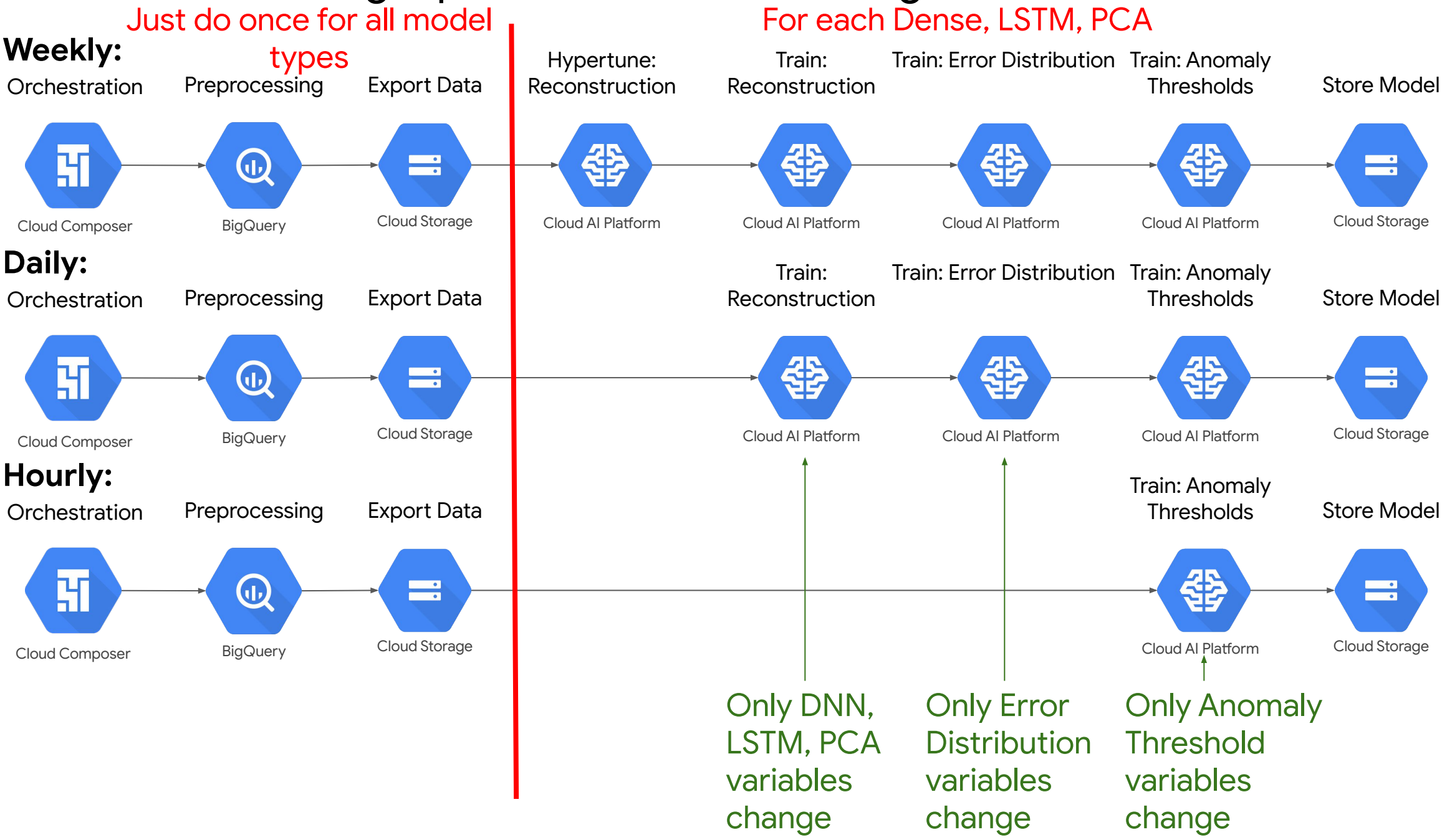
Set default version

```
ml_engine_set_default_version_op = MLEngineVersionOperator(
  task_id="ml_engine_set_default_version_{}_task".format(model.replace(".","_")),
  project_id=PROJECT_ID,
  model_name=MODEL_NAME + model.replace(".","_"),
  version_name=MODEL_VERSION,
  version={"name": MODEL_VERSION},
  operation="set_default",
  dag=dag
)

ml_engine_set_default_other_version_op = MLEngineVersionOperator(
  task_id="ml_engine_set_default_other_version_{}_task".format(model.replace(".","_")),
  project_id=PROJECT_ID,
  model_name=MODEL_NAME + model.replace(".","_"),
  version_name=OTHER_VERSION_NAME,
  version={"name": OTHER_VERSION_NAME},
  operation="set_default",
  dag=dag
)
```

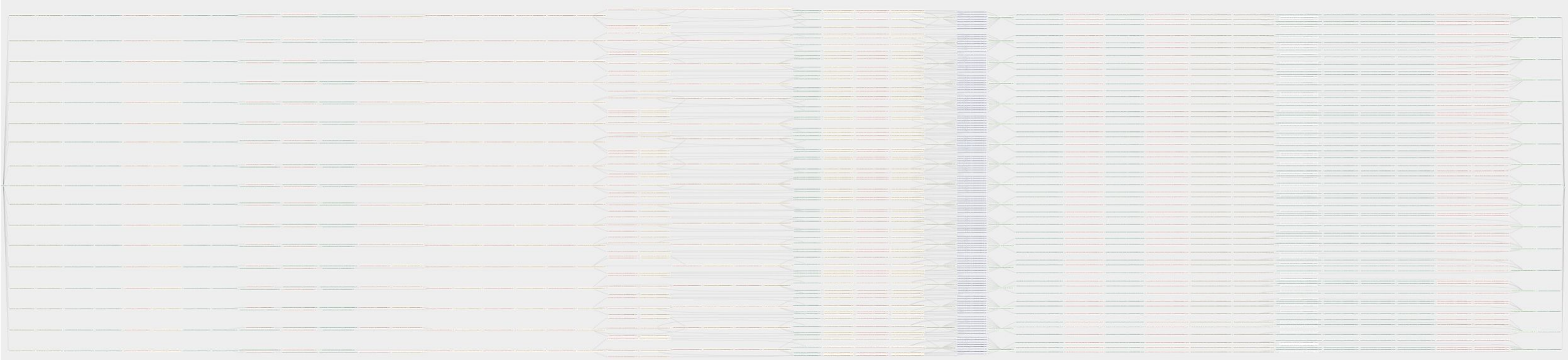
Finally build dependency graph

```
# Build dependency graph, set_upstream dependencies for all tasks
bq_check_train_data_op.set_upstream(bq_train_data_op)
bq_check_eval_data_op.set_upstream(bq_eval_data_op)

bash_remove_old_data_op.set_upstream([bq_check_train_data_op, bq_check_eval_data_op])

bq_export_gcs_train_csv_op.set_upstream([bash_remove_old_data_op])
bq_export_gcs_eval_csv_op.set_upstream([bash_remove_old_data_op])

ml_engine_training_op.set_upstream([bq_export_gcs_train_csv_op, bq_export_gcs_eval_csv_op])

bash_remove_old_saved_model_op.set_upstream(ml_engine_training_op)
bash_copy_new_saved_model_op.set_upstream(bash_remove_old_saved_model_op)

bash_ml_engine_models_list_op.set_upstream(ml_engine_training_op)
check_if_model_already_exists_op.set_upstream(bash_ml_engine_models_list_op)

ml_engine_create_model_op.set_upstream(check_if_model_already_exists_op)
create_model_dummy_op.set_upstream(ml_engine_create_model_op)
dont_create_model_dummy_branch_op.set_upstream(check_if_model_already_exists_op)
dont_create_model_dummy_op.set_upstream(dont_create_model_dummy_branch_op)

bash_ml_engine_versions_list_op.set_upstream([dont_create_model_dummy_op, create_model_dummy_op])
check_if_model_version_already_exists_op.set_upstream(bash_ml_engine_versions_list_op)

ml_engine_create_version_op.set_upstream([bash_copy_new_saved_model_op, check_if_model_version_already_exists_op])
ml_engine_create_other_version_op.set_upstream([bash_copy_new_saved_model_op, check_if_model_version_already_exists_op])

ml_engine_set_default_version_op.set_upstream(ml_engine_create_version_op)
ml_engine_set_default_other_version_op.set_upstream(ml_engine_create_other_version_op)
```

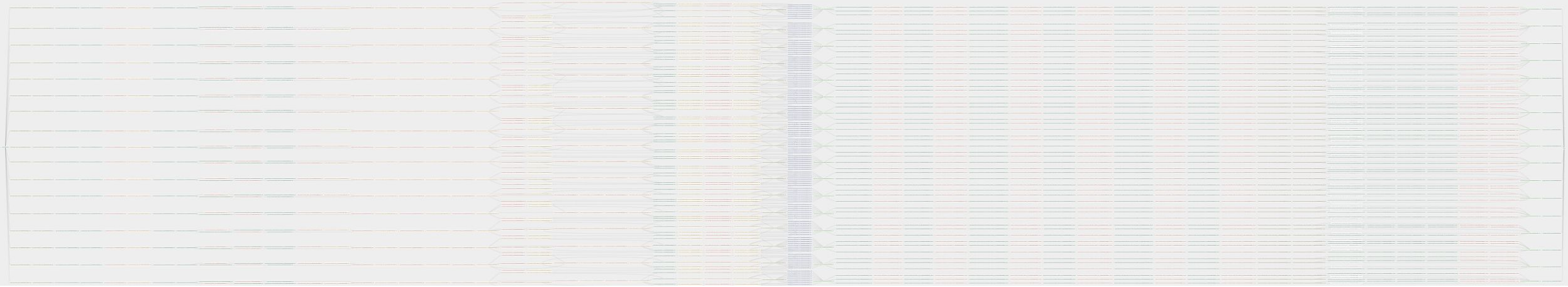
Training Pipelines Based on Configurable Schedule



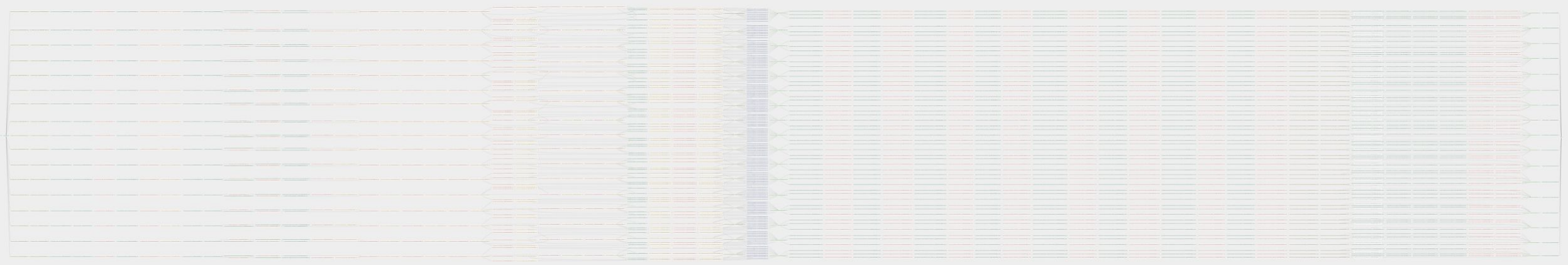
Customer Hourly DAG Example



Customer Daily DAG Example

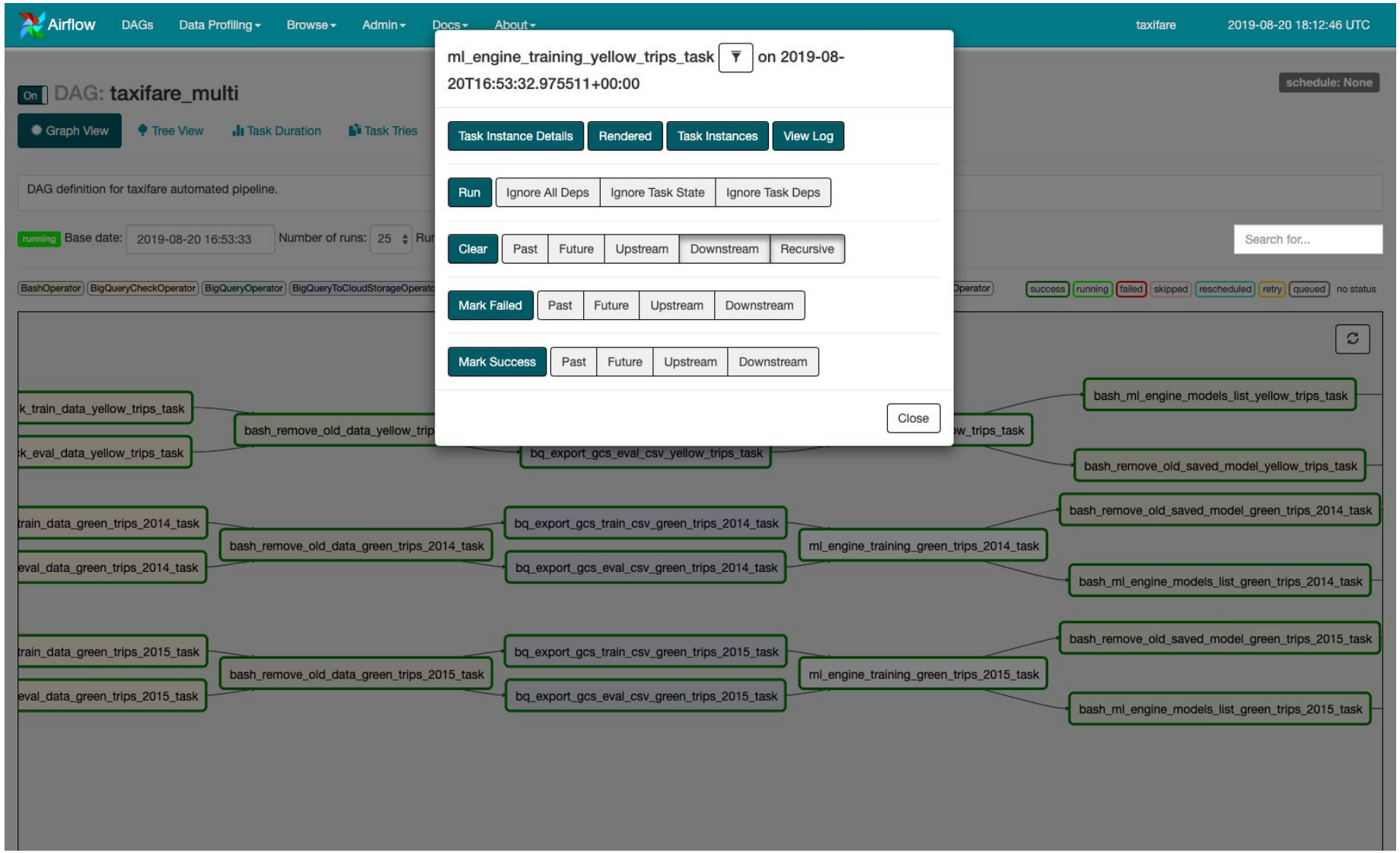


Customer Weekly DAG Example



Customer Single Branch DAG Example

Clicking on a task gives you multiple options



You can investigate task logs

```
*** Reading remote log from gs://us-central1-taxifare-4dc04517-bucket/logs/taxifare_multi/ml_engine_training_yellow_trips_task/2019-08-20T16:53:32.975511+00:00/3.log.
[2019-08-20 17:34:02,515] {models.py:1359} INFO - Dependencies all met for <TaskInstance: taxifare_multi.ml_engine_training_yellow_trips_task 2019-08-20T16:53:32.975511+00:00 [queued]>
[2019-08-20 17:34:02,691] {models.py:1359} INFO - Dependencies all met for <TaskInstance: taxifare_multi.ml_engine_training_yellow_trips_task 2019-08-20T16:53:32.975511+00:00 [queued]>
[2019-08-20 17:34:02,703] {models.py:1577} INFO -
--------------------------------------------------------------------------------
Starting attempt 3 of
--------------------------------------------------------------------------------

[2019-08-20 17:34:02,875] {models.py:1599} INFO - Executing <Task(MLEngineTrainingOperator): ml_engine_training_yellow_trips_task> on 2019-08-20T16:53:32.975511+00:00
[2019-08-20 17:34:02,889] {base_task_runner.py:118} INFO - Running: ['bash', '-c', 'airflow run taxifare_multi ml_engine_training_yellow_trips_task 2019-08-20T16:53:32.975511+00:00 --job_id 84 --raw -
[2019-08-20 17:34:08,034] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:08,032] {settings.py:176} INFO - settings.configure_orm(): Using pool
[2019-08-20 17:34:08,942] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:08,941] {default_celery.py:90} WARNING - You have configured a result_
[2019-08-20 17:34:08,945] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:08,945] {__init__.py:51} INFO - Using executor CeleryExecutor
[2019-08-20 17:34:09,374] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:09,374] {app.py:52} WARNING - Using default Composer Environment Varia
[2019-08-20 17:34:09,398] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:09,392] {configuration.py:522} INFO - Reading the config from /etc/air
[2019-08-20 17:34:09,432] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:09,432] {configuration.py:522} INFO - Reading the config from /etc/air
[2019-08-20 17:34:09,714] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:09,713] {models.py:273} INFO - Filling up the DagBag from /home/airflo
[2019-08-20 17:34:09,885] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,886] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:09,887] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,887] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:09,890] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,892] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:09,892] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,894] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:09,898] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,898] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:09,900] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task /usr/local/lib/airflow/airflow/contrib/operators/bigquery_operator.py:172: DeprecationWa
[2019-08-20 17:34:09,901] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task   category=DeprecationWarning)
[2019-08-20 17:34:10,979] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:10,979] {cli.py:520} INFO - Running <TaskInstance: taxifare_multi.ml_e
[2019-08-20 17:34:11,156] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:11,155] {gcp_api_base_hook.py:94} INFO - Getting connection using `goo
[2019-08-20 17:34:11,185] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:11,184] {discovery.py:272} INFO - URL being requested: GET https://www
[2019-08-20 17:34:11,316] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:11,316] {discovery.py:873} INFO - URL being requested: POST https://ml
[2019-08-20 17:34:12,026] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:12,026] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:34:42,123] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:34:42,122] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:35:12,253] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:35:12,252] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:35:42,340] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:35:42,339] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:36:12,490] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:36:12,488] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:36:42,690] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:36:42,682] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:37:12,896] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:37:12,895] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:37:42,974] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:37:42,974] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:38:13,087] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:38:13,085] {discovery.py:873} INFO - URL being requested: GET https://ml.
[2019-08-20 17:38:43,176] {base_task_runner.py:101} INFO - Job 84: Subtask ml_engine_training_yellow_trips_task [2019-08-20 17:38:43,175] {discovery.py:873} INFO - URL being requested: GET https://ml.
```

Clearing task instances can save you from running the entire DAG again

Wait a minute.

Here's the list of task instances you are about to clear:

```
<TaskInstance: taxifare_multi.bash_copy_new_saved_model_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [failed]>
<TaskInstance: taxifare_multi.bash_ml_engine_models_list_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.bash_ml_engine_versions_list_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.bash_remove_old_saved_model_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.check_if_model_already_exists_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.check_if_model_version_already_exists_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.create_model_dummy_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.dont_create_model_dummy_branch_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [skipped]>
<TaskInstance: taxifare_multi.dont_create_model_dummy_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.ml_engine_create_model_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
<TaskInstance: taxifare_multi.ml_engine_create_other_version_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [skipped]>
<TaskInstance: taxifare_multi.ml_engine_create_version_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [upstream_failed]>
<TaskInstance: taxifare_multi.ml_engine_set_default_other_version_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [skipped]>
<TaskInstance: taxifare_multi.ml_engine_set_default_version_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [upstream_failed]>
<TaskInstance: taxifare_multi.ml_engine_training_yellow_trips_task 2019-08-20 16:53:32.975511+00:00 [success]>
```

OK! bail.

cloud.google.com