Local Airflow Development at Oden

Devon Peticolas - Oden Technologies

github.com/x/slides/airflow-nyc-2020
github.com/x/example-local-airflow

Kind-of Making Airflow Work Locally at Oden

Devon Peticolas - Oden Technologies

Devon Peticolas

Sr. Data Engineer

Devon Peticolas

Sr. Data Engineer

Not a DevOps Engineer!
Most of this very is bad!

James Maidment

Nathan Mehl

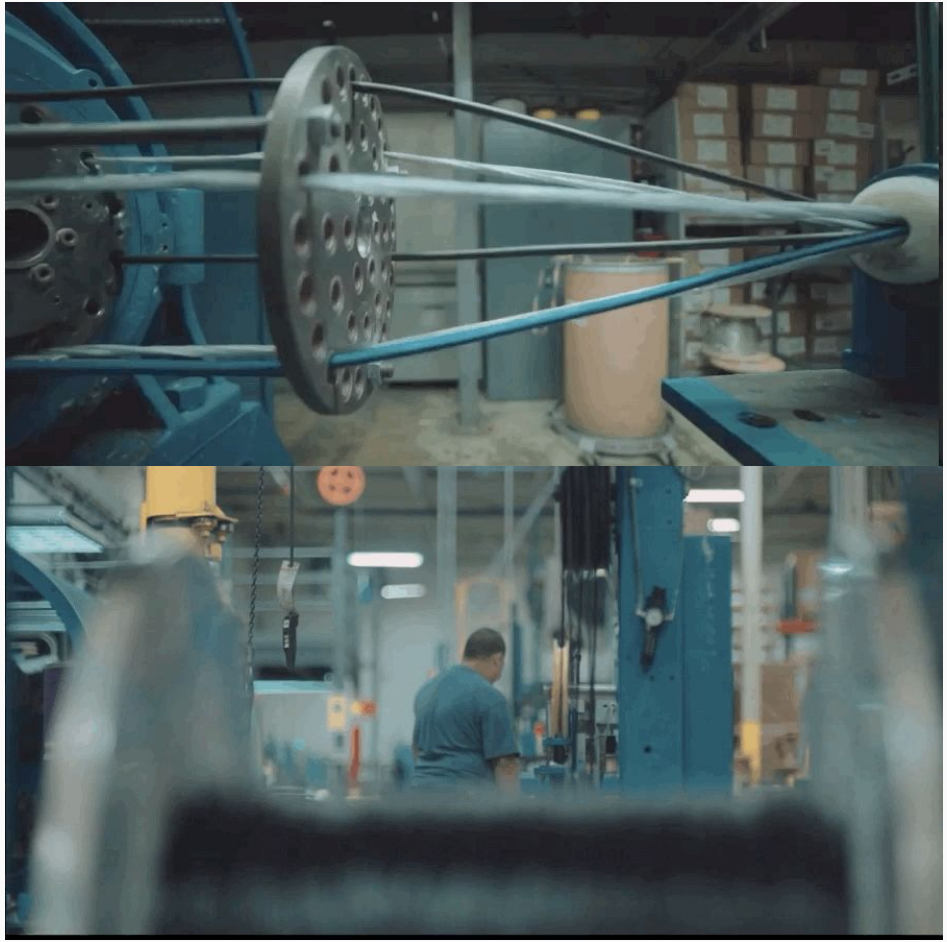
Oden

Oden's Customers

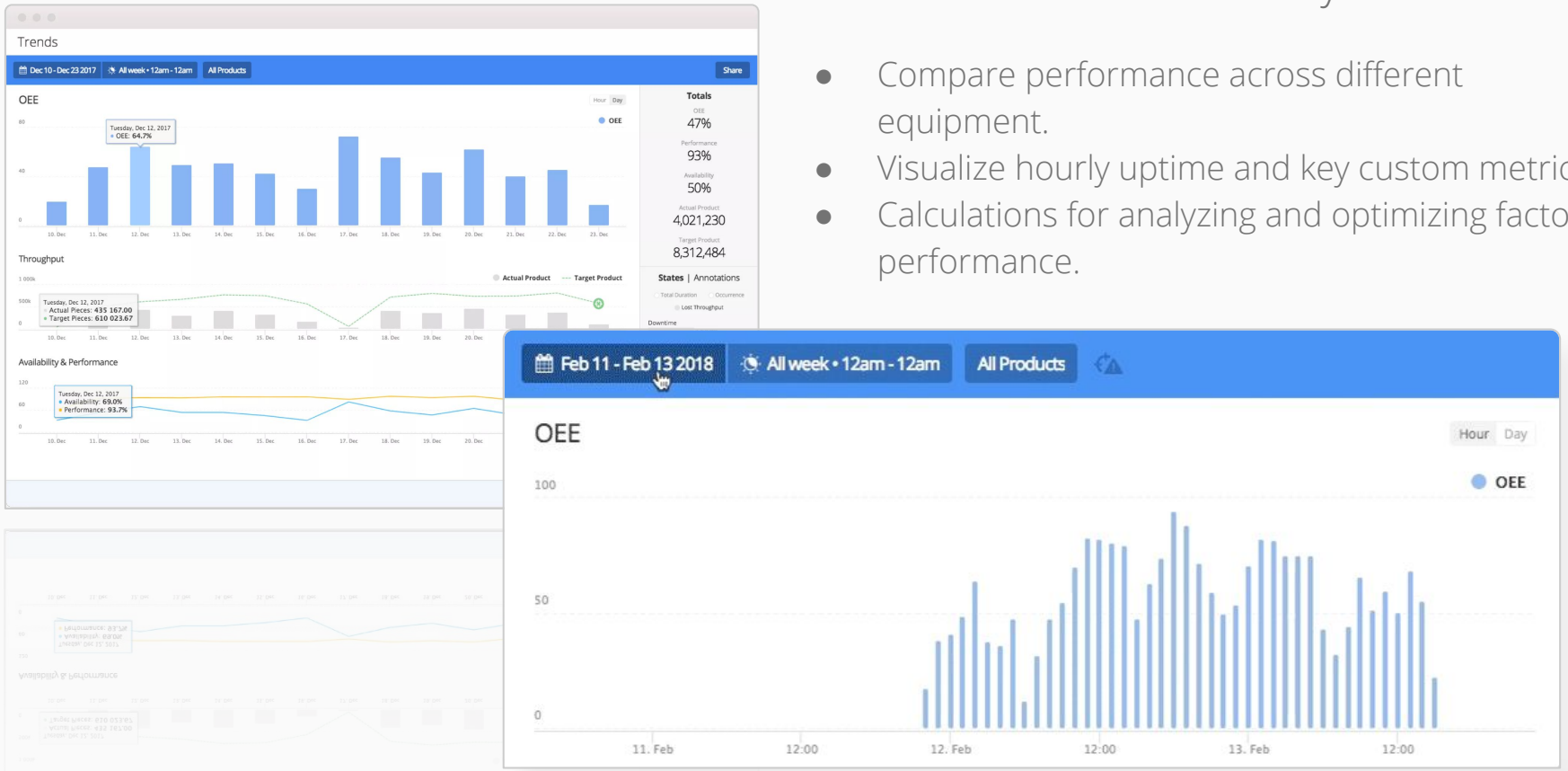
Medium to large manufacturers in plastics extrusion, injection molding, and metal stamping.

Process and Quality Engineers looking to centralize, analyze, and act on their data.

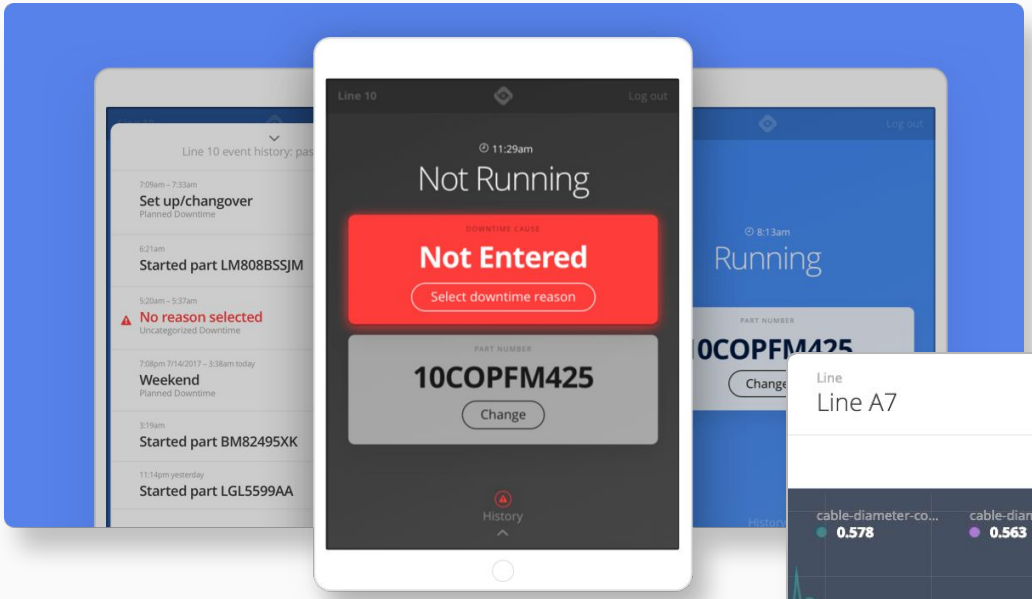
Plant managers who are looking to optimize logistics, output, and cost.



Interactive Time-series Analysis



- Compare performance across different equipment.
- Visualize hourly uptime and key custom metrics.
- Calculations for analyzing and optimizing factory performance.



Real Time Manufacturing Data

- Streaming second-by-second metrics
- Interactive app that prompts on production state changes and collects user input.

Reporting and Alerting

- Daily summaries on key process metrics from continuous intervals of production work.
- Real-time email and text alerts on target violations and concerning trends.

ALERT

Downtime violation on **Line 1**

As of 12:55pm, Line 1 has been in Downtime for more than 15 minutes.

View line

Snooze this alert for: 30m 2h 8h 24h

Powered by Oden Technologies
Is this alert useful? Let us know!

Daily Run Report

Runs completed 9:00am EST February 12, 2019 – 9:00am EST February 13, 2019
Runs sorted by worst Cpk for Cold OD Avg

SWJNG519-LQ8
Line 10 · I 0 Reels · 06:11 2/12 – 11:42 2/12 · 3h 16m uptime

View run →

METRIC	MEAN	STD DEV	TARGET	NON CON*	Cpk
Cold OD Avg	0.403	0.010	0.391 - 0.411	4.235%	**0.274**
Feet per min	274.794	194.059	-	-	-

SWHD72Y-R4
Line 10 · I 0 Reels · 10:08 2/12 – 12:34 2/13 · 1h 35m uptime

View run →

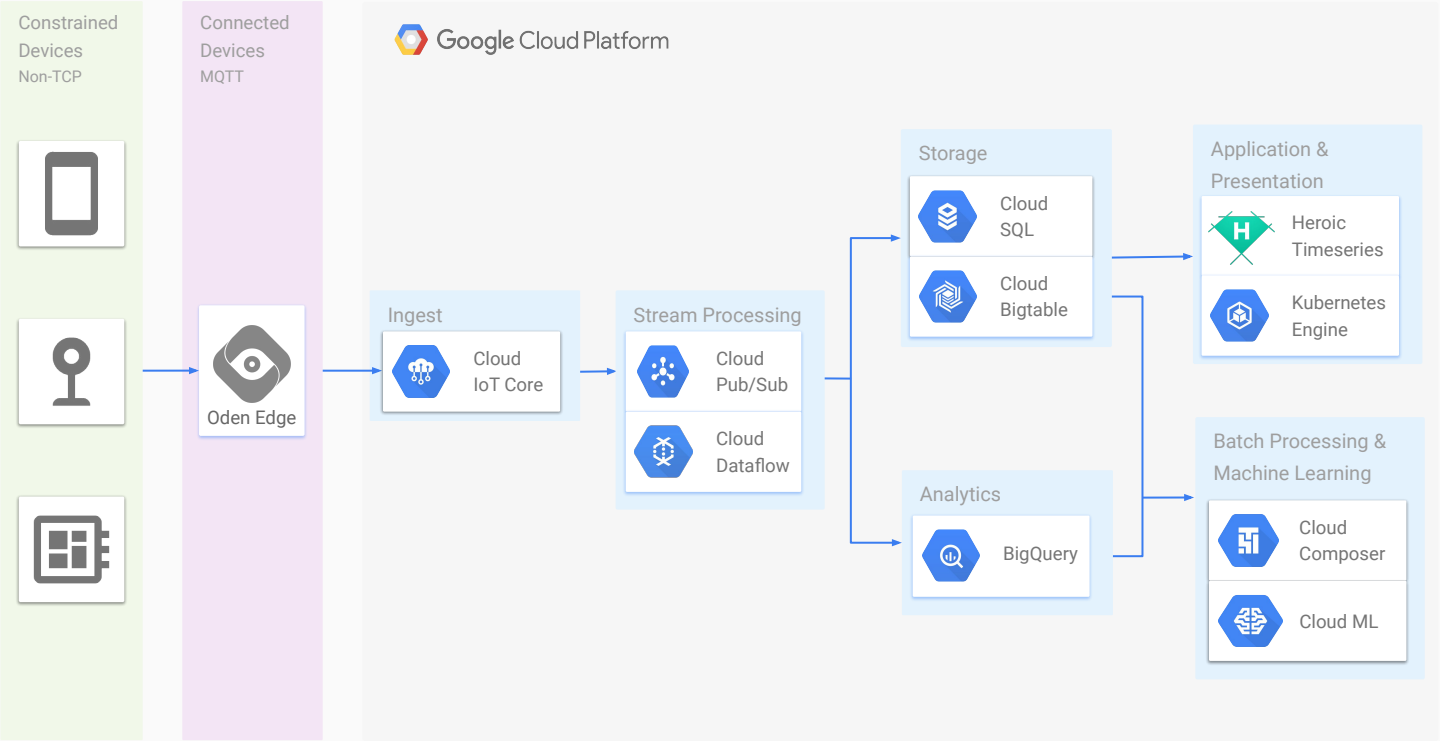
METRIC	MEAN	STD DEV	TARGET	NON CON*	Cpk
Cold OD Avg	0.141	0.002	0.135 - 0.145	0.242%	**0.782**
Feet per min	829.680	492.109	-	-	-

Technology - Hardware

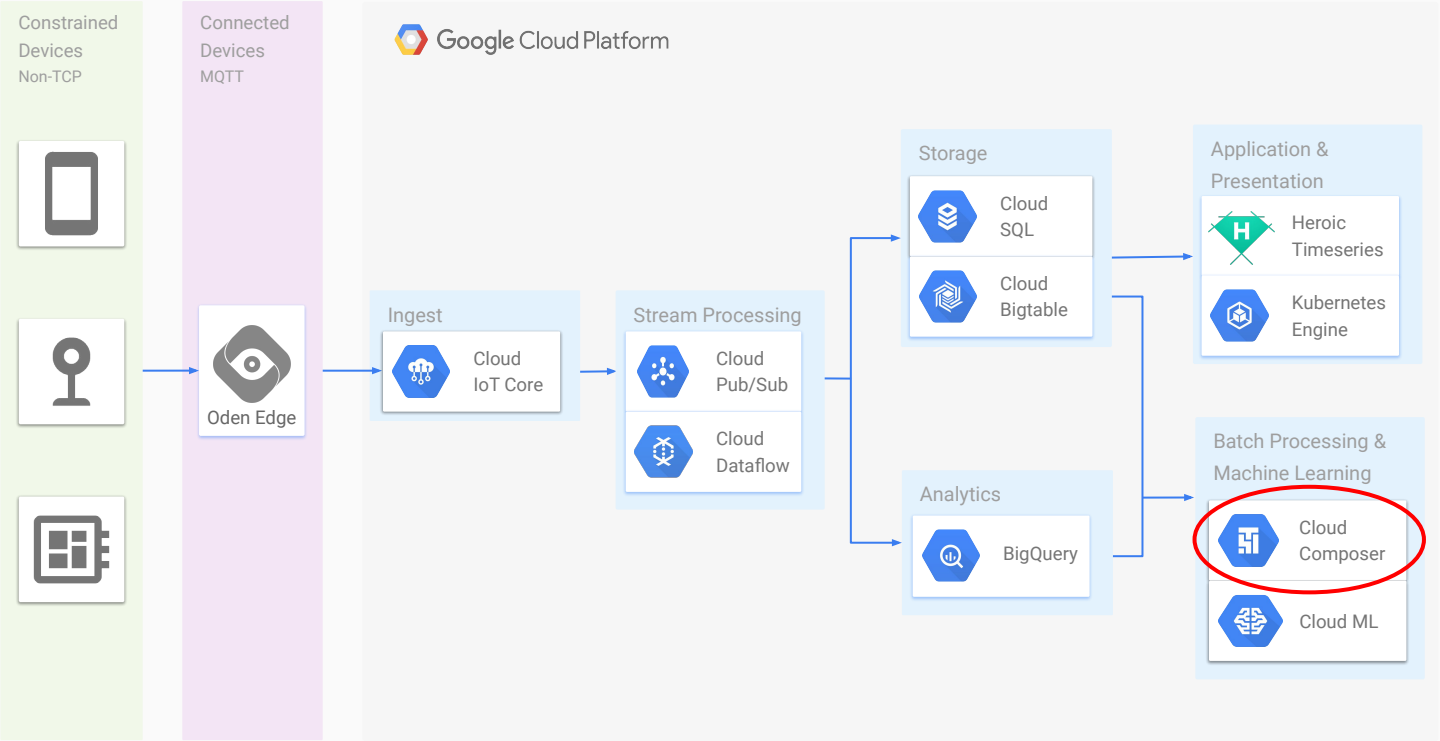
Oden Hardware

- Linux devices that connect via standard industrial protocols over serial and ethernet and speak of MQTT
- On-prem servers that a subset “edge” version of the Oden platform and speak to devices and modern PLCs via MQTT
- Connect to our services in the cloud via wired, wifi, or cellular networks.

Technology - Architecture



Technology - Architecture



Airflow At Oden

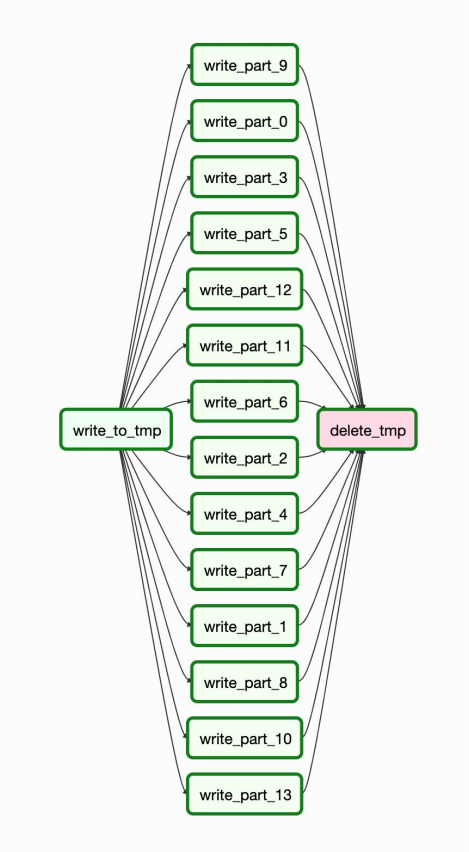
Overview

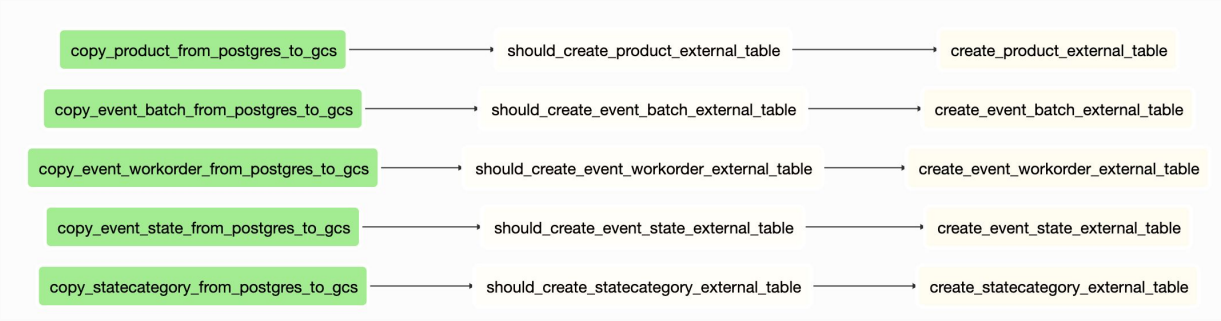
- Deployed on **Google Cloud Composer**
- Development is done locally using **puckel/docker-airflow**
- Used for all scheduled jobs, with or without dependencies
- Primarily interacts with:
  - Kubernetes
  - BigQuery
  - CloudSQL
  - Cloud Storage
  - Dataflow (Apache Beam)

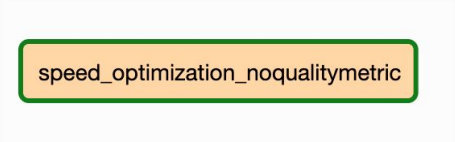
Operators we Use

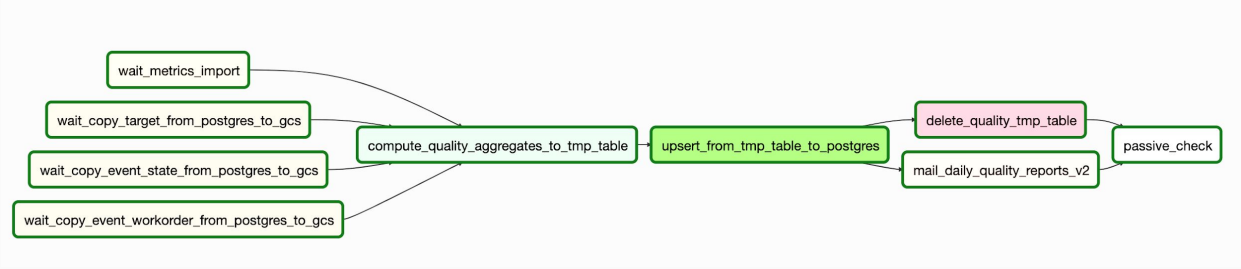
- Bigquery
  - BigqueryOperator
  - BigQueryTableDeleteOperator
  - BigQueryCreateExternalTableOperator
- Postgres
  - PostgresOperator
  - PostgresToCloudStorageOperator
  - CloudSqlInstanceExportOperator
- ML / Report Generation
  - KubernetesPodOperator
  - MLEngineTrainingOperator
  - DataflowTemplateOperator
- Other
  - PythonOperator + HTTPHook
  - *Lots of Custom Plugins*

DAGs Come in All Shapes and Sizes!

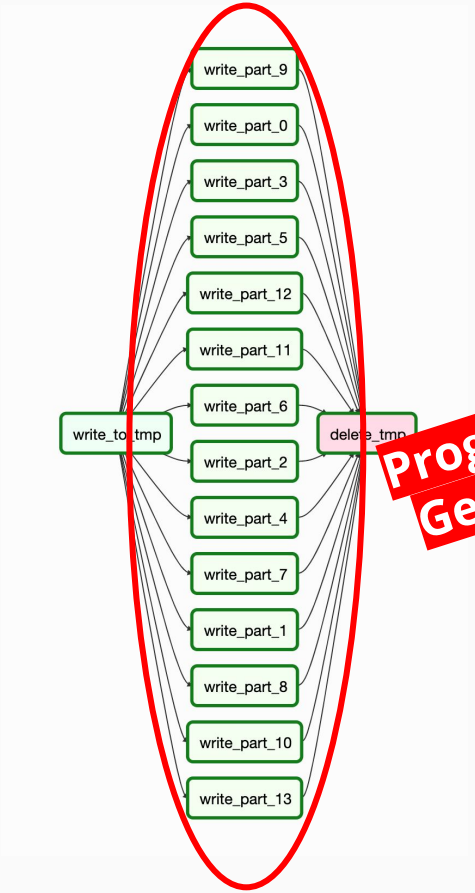


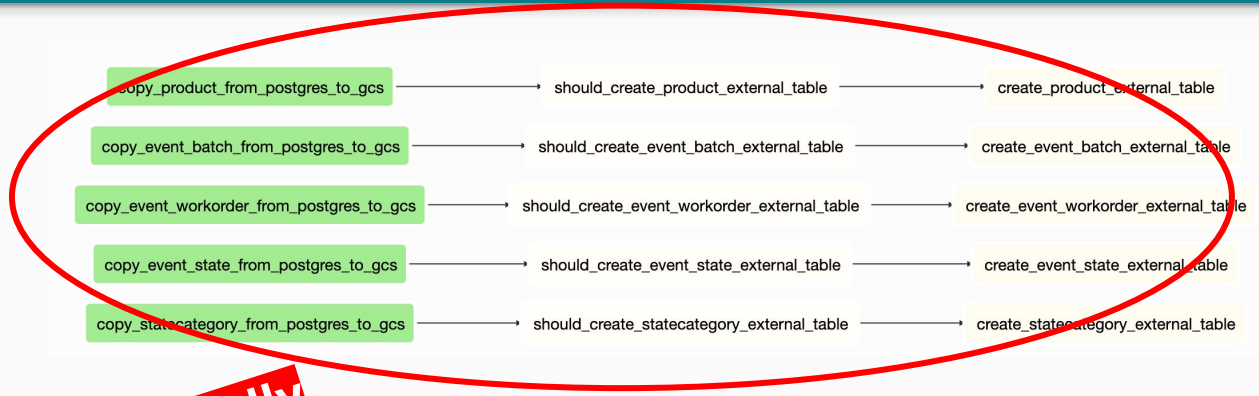




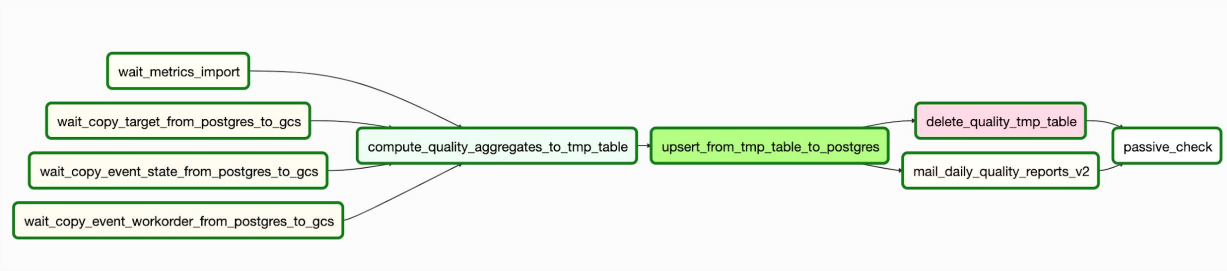


DAGs Come in All Shapes and Sizes!





Programmatically Generated DAGs



Dumping Tables From Postgres to GCS for BQ

- DAG is n separate tasks defined by a list of table names in TABLES
- Used for dumping postgres data into GCS to be queried as an external table
- Will hopefully be replaced by GCP Federated Queries soon
- Runs are independent of each other

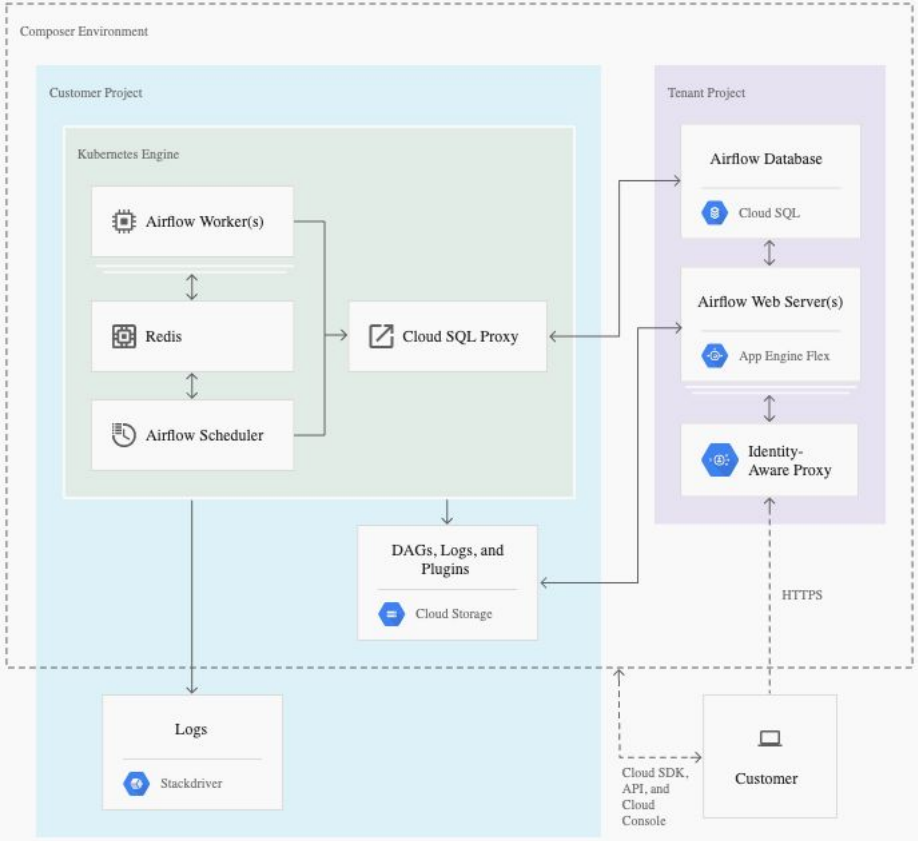
```
TABLES = [
    ...
]

dag = DAG(
    "postgres_dumps_v1",
    default_args=default_args(retries=3),
    concurrency=4,  # prevents LocalExecutor psycopg2 error
    schedule_interval="0 * * * *",  # Every hour
    catchup=False,
    max_active_runs=1,  # no need for more than this
)

for table in TABLES:
    PostgresToGoogleCloudStorageOperator(
        sql="SELECT * FROM " + table + ";",
        task_id="copy_" + table + "_from_postgres_to_gcs",
        postgres_conn_id="oden_postgres_default",
        filename=EXPORT_PATH_FMTSTR.format(table=table,
number="{}"),
        bucket="bigquery-external-tables.{{
macros.oden_apex_domain() }}",

schema_filename=SCHEMA_PATH_FMTSTR.format(table=table),
        dag=dag,
    )
```

Deployment - Cloud Composer



- Managed Airflow
- Upgrades are difficult
- No way to directly access (and therefore backup) the Airflow DB
- Webserver does not have a static address, can't use Airflow API
- No way to direct issue Airflow CLI commands
- General sense that no one who at Google has ever used the product

Our Local Dev Setup

Overview

- We use the Puckel Docker Image
- Run docker-compose with local DBs and proxies to staging
- All GCP actions done with user's credentials
- Shell scripts for maintaining "connections" and "variables"
- Shell scripts for issuing commands to the container

github.com/puckel/docker-airflow

Search or jump to... / Pull requests Issues Marketplace Explore

puckel / docker-airflow

Watch 103 Star 2k Fork 1.6k

Code Issues 145 Pull requests 31 Actions Projects 0 Security Insights

Docker Apache Airflow

docker-airflow airflow docker scheduler workflow task management

218 commits 1 branch 0 packages 36 releases 32 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

kevoriordan and puckel Fix bug with kubernetes executor (#470) Latest commit 70892af on Nov 22, 2019

.github/workflows	Create ci.yml	2 months ago
config	Fix bug with kubernetes executor (#470)	2 months ago
dags	Bump to 1.9.0-4	2 years ago
script	starting scheduler for SequentialExecutor (#359)	5 months ago
.dockerignore	Initial commit	5 years ago
.gitignore	Update to Apache Airflow 1.9	2 years ago
Dockerfile	Bump to Airflow 1.10.6	2 months ago

Docker Image

1. Use the puckel docker-airflow image
2. Install the gcloud sdk
3. Start custom entrypoint.sh script (overwrites original) that sets up connections, variables, and packages.

Dockerfile

```
FROM puckel/docker-airflow:1.10.3

USER root

ARG CLOUD_SDK_VERSION=274.0.1
ENV CLOUD_SDK_VERSION=$CLOUD_SDK_VERSION
ARG INSTALL_COMPONENTS
RUN apt-get update -qqy && apt-get install -qqy \
  curl gcc python-dev python-setuptools apt-transport-https \
  lsb-release openssh-client git gnupg \
  && easy_install -U pip && pip install -U crcmod && \
  export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && \
  echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" >
/etc/apt/sources.list.d/google-cloud-sdk.list && \
  curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \
  apt-get update && apt-get install -y google-cloud-sdk=${CLOUD_SDK_VERSION}-0
$INSTALL_COMPONENTS && \
  gcloud config set core/disable_usage_reporting true && \
  gcloud config set component_manager/disable_update_check true && \
  gcloud config set metrics/environment github_docker_image && \
  gcloud --version

USER airflow

COPY config/airflow.cfg /usr/local/airflow/airflow.cfg
COPY script/connections.sh /usr/local/airflow/connections.sh
COPY script/entrypoint.sh /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

Docker Image

1. **Use the puckel docker-airflow image**
2. Install the gcloud sdk
3. Start custom entrypoint.sh script (overwrites original) that sets up connections, variables, and packages.

Dockerfile

```
FROM puckel/docker-airflow:1.10.3

USER root

ARG CLOUD_SDK_VERSION=274.0.1
ENV CLOUD_SDK_VERSION=$CLOUD_SDK_VERSION
ARG INSTALL_COMPONENTS
RUN apt-get update -qqy && apt-get install -qqy \
  curl gcc python-dev python-setuptools apt-transport-https \
  lsb-release openssh-client git gnupg \
  && easy_install -U pip && pip install -U crcmod && \
  export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && \
  echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" >
/etc/apt/sources.list.d/google-cloud-sdk.list && \
  curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \
  apt-get update && apt-get install -y google-cloud-sdk=${CLOUD_SDK_VERSION}-0
$INSTALL_COMPONENTS && \
  gcloud config set core/disable_usage_reporting true && \
  gcloud config set component_manager/disable_update_check true && \
  gcloud config set metrics/environment github_docker_image && \
  gcloud --version

USER airflow

COPY config/airflow.cfg /usr/local/airflow/airflow.cfg
COPY script/connections.sh /usr/local/airflow/connections.sh
COPY script/entrypoint.sh /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

Docker Image

1. Use the puckel docker-airflow image
2. **Install the gcloud sdk**
3. Start custom entrypoint.sh script (overwrites original) that sets up connections, variables, and packages.

Dockerfile

```
FROM puckel/docker-airflow:1.10.3

USER root

ARG CLOUD_SDK_VERSION=274.0.1
ENV CLOUD_SDK_VERSION=$CLOUD_SDK_VERSION
ARG INSTALL_COMPONENTS
RUN apt-get update -qqy && apt-get install -qqy \
  curl gcc python-dev python-setuptools apt-transport-https \
  lsb-release openssh-client git gnupg \
  && easy_install -U pip && pip install -U crcmod && \
  export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && \
  echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" >
/etc/apt/sources.list.d/google-cloud-sdk.list && \
  curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \
  apt-get update && apt-get install -y google-cloud-sdk=${CLOUD_SDK_VERSION}-0
$INSTALL_COMPONENTS && \
  gcloud config set core/disable_usage_reporting true && \
  gcloud config set component_manager/disable_update_check true && \
  gcloud config set metrics/environment github_docker_image && \
  gcloud --version

USER airflow

COPY config/airflow.cfg /usr/local/airflow/airflow.cfg
COPY script/connections.sh /usr/local/airflow/connections.sh
COPY script/entrypoint.sh /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

Docker Image

1. Use the puckel docker-airflow image
2. Install the gcloud sdk
3. **Start custom entrypoint.sh script (overwrites original) that sets up connections, variables, and packages.**

Dockerfile

```
FROM puckel/docker-airflow:1.10.3

USER root

ARG CLOUD_SDK_VERSION=274.0.1
ENV CLOUD_SDK_VERSION=$CLOUD_SDK_VERSION
ARG INSTALL_COMPONENTS
RUN apt-get update -qqy && apt-get install -qqy \
  curl gcc python-dev python-setuptools apt-transport-https \
  lsb-release openssh-client git gnupg \
  && easy_install -U pip && pip install -U crcmod && \
  export CLOUD_SDK_REPO="cloud-sdk-$(lsb_release -c -s)" && \
  echo "deb https://packages.cloud.google.com/apt $CLOUD_SDK_REPO main" >
/etc/apt/sources.list.d/google-cloud-sdk.list && \
  curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add - && \
  apt-get update && apt-get install -y google-cloud-sdk=${CLOUD_SDK_VERSION}-0
$INSTALL_COMPONENTS && \
  gcloud config set core/disable_usage_reporting true && \
  gcloud config set component_manager/disable_update_check true && \
  gcloud config set metrics/environment github_docker_image && \
  gcloud --version

USER airflow

COPY config/airflow.cfg /usr/local/airflow/airflow.cfg
COPY script/connections.sh /usr/local/airflow/connections.sh
COPY script/entrypoint.sh /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

docker-compose.yml

- Minimum manages:
  - Web Server
  - Airflow DB
- Mounts necessary resources for development
- FERNET_KEY allows us to keep the same configured credentials between builds

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
    env_file:
      - dev.env
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./plugins:/usr/local/airflow/plugins
      - ./requirements.txt:/requirements.txt
      - ./variables.json:/usr/local/airflow/variables.json
      - ~/.config:/usr/local/airflow/.config
      - ~/.docker:/usr/local/airflow/.docker
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
    command: webserver
  airflow-postgres:
    image: postgres:9.6
    environment:
      - POSTGRES_PASSWORD=airflow
```

docker-compose.yml

- **Minimum manages:**
  - **Web Server**
  - **Airflow DB**
- Mounts necessary resources for development
- FERNET_KEY allows us to keep the same configured credentials between builds

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
    env_file:
      - dev.env
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./plugins:/usr/local/airflow/plugins
      - ./requirements.txt:/requirements.txt
      - ./variables.json:/usr/local/airflow/variables.json
      - ~/.config:/usr/local/airflow/.config
      - ~/.docker:/usr/local/airflow/.docker
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
    command: webserver
  airflow-postgres:
    image: postgres:9.6
    environment:
      - POSTGRES_PASSWORD=airflow
```

docker-compose.yml

- Minimum manages:
  - Web Server
  - Airflow DB
- **Mounts necessary resources for development**
- FERNET_KEY allows us to keep the same configured credentials between builds

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
    env_file:
      - dev.env
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./plugins:/usr/local/airflow/plugins
      - ./requirements.txt:/requirements.txt
      - ./variables.json:/usr/local/airflow/variables.json
      - ~/.config:/usr/local/airflow/.config
      - ~/.docker:/usr/local/airflow/.docker
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
    command: webserver
  airflow-postgres:
    image: postgres:9.6
    environment:
      - POSTGRES_PASSWORD=airflow
```

docker-compose.yml

- Minimum manages:
  - Web Server
  - Airflow DB
- Mounts necessary resources for development
- **FERNET_KEY allows us to keep the same configured credentials between builds**

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
    env_file:
      - dev.env
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./plugins:/usr/local/airflow/plugins
      - ./requirements.txt:/requirements.txt
      - ./variables.json:/usr/local/airflow/variables.json
      - ~/.config:/usr/local/airflow/.config
      - ~/.docker:/usr/local/airflow/.docker
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
    command: webserver
  airflow-postgres:
    image: postgres:9.6
    environment:
      - POSTGRES_PASSWORD=airflow
```

Example Project Structure

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Example Project Structure

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Example Project Structure

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Example Project Structure

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Web Server

Example Project Structure

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Web Server

Building and Testing DAG Construction

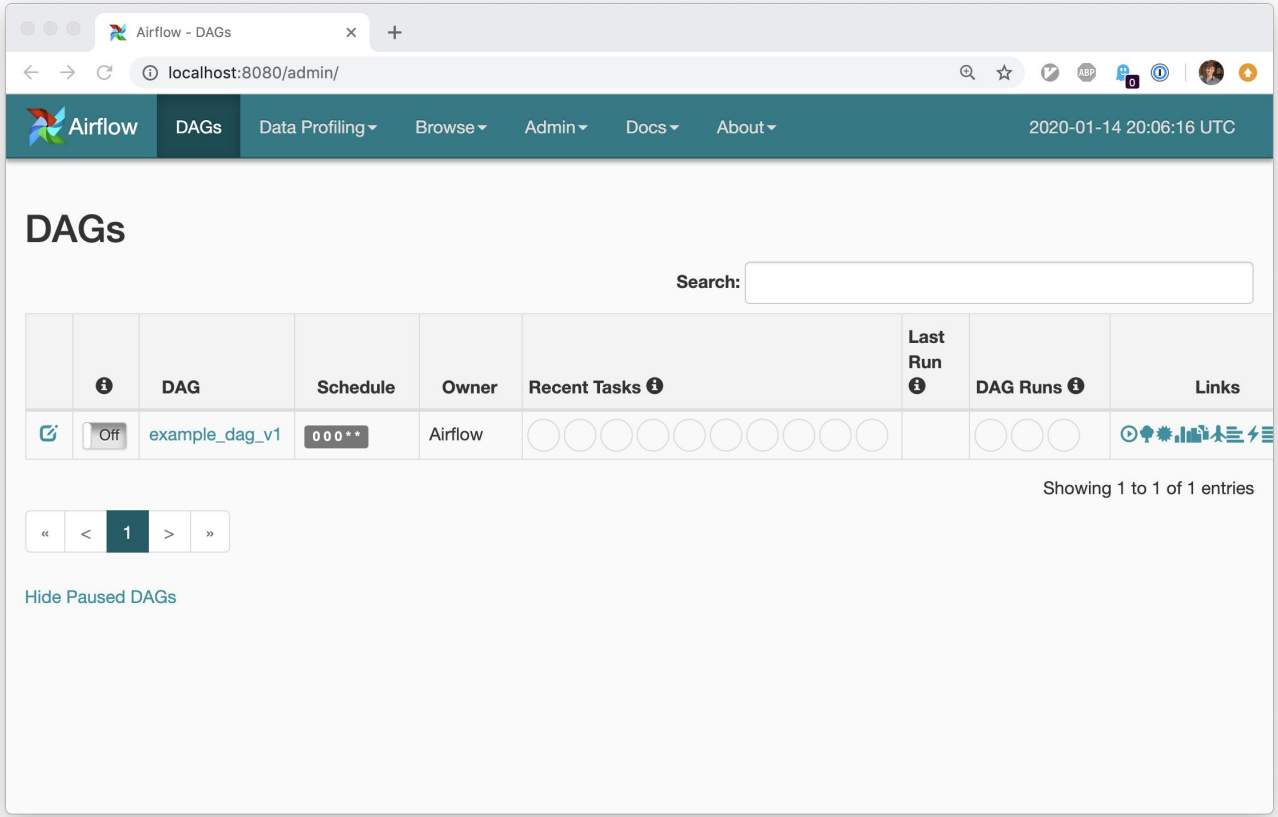
```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime

dag = DAG(
    "example_dag_v1",
    schedule_interval="0 0 0 * *",
    start_date=datetime(2020, 1, 12, 4, 20, 0),
)

t1 = PythonOperator(
    task_id="example_task",
    python_callable=lambda: print("hello world!"),
    dag=dag,
)
```

Building and Testing DAG Construction

Building and Testing DAG Construction



Cloud Credentials (GCP Specific)

- Mounted from the home directory of the user into the home directory of the airflow user within the container
- The connections.sh script, run at image build time, configures the credentials to be keyless causing the google client libraries to default to the credentials in the user's (airflow user) home directory.

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    ...
    volumes:
      - ./dags:/usr/local/airflow/dags
      - ./plugins:/usr/local/airflow/plugins
      - ./requirements.txt:/requirements.txt
      - ./variables.json:/usr/local/airflow/variables.json
      - ~/.config:/usr/local/airflow/.config
      - ~/.docker:/usr/local/airflow/.docker
      - /var/run/docker.sock:/var/run/docker.sock
    ...
```

scripts/connections.sh

```bash
#!/bin/bash -e

# Connection for google cloud services, GCS, bigquery, cloudsql.
# By not setting a key_path we default to the user's credentials mounted
# into /usr/local/airflow/.config
airflow connections --delete --conn_id google_cloud_default
airflow connections --add \
    --conn_id=google_cloud_default \
    --conn_type=google_cloud_platform \
    --conn_extra='{
        "extra__google_cloud_platform__key_path": "",
        "extra__google_cloud_platform__project": "oden-qa",
        "extra__google_cloud_platform__scope": ""
    }'
```

Proxying to CloudSQL in Staging (GCP Specific)

- For jobs that need to read or write to a staging cloudsql instance we include a cloudsql-proxy.
- In destructive query development we run a regular postgres instance like we did the airflow-postgres and load in dumps of data.

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
      - CLOUDSQL_POSTGRES_HOST=cloudsql-proxy
      - CLOUDSQL_POSTGRES_PORT=5432
    env_file:
      - dev.env
    ...

  cloudsql-proxy:
    image: gcr.io/cloudsql-docker/gce-proxy:1.13
    command: sh -c "/cloud_sql_proxy -dir=/cloudsql
-instances=$${GCLOUD_PROJECT}:$${GCLOUD_REGION}:$${CLOUDSQL_INSTANCE_NAME}=tcp:0.
0.0.0:5432"
    env_file:
      - dev.env
    volumes:
      - ~/.config:/root/.config
```

Proxying to CloudSQL in Staging (GCP Specific)

- Connections to the remote instance can be made by embedding them into the connection.sh file.

scripts/connections.sh

```
#!/bin/bash -e

...

# Regular postgres connection
airflow connections --delete --conn_id postgres_default
airflow connections --add \
    --conn_id=postgres_default \
    --conn_uri
"postgresql://${CLOUDSQL_POSTGRES_USERNAME}:${CLOUDSQL_POSTGRES_PASSWORD}
@cloudsql-proxy/${CLOUDSQL_POSTGRES_DBNAME}"


# CloudSQL connection (used for google first-party operators)
airflow connections --delete --conn_id google_cloud_sql
airflow connections --add \
  --conn_id=google_cloud_sql
--conn_uri="gcpcloudsql://${CLOUDSQL_POSTGRES_USERNAME}:${CLOUDSQL_POSTGR
ES_PASSWORD}@1.1.1.1:3306/${CLOUDSQL_POSTGRES_DBNAME}?database_type=postg
res&project_id=${GCLOUD_PROJECT}&location=${GCLOUD_REGION}&instance=${CLO
UDSQL_POSTGRES_DBNAME}&use_proxy=True&sql_proxy_use_tcp=False"
```

Proxying Kubernetes Services Using Kubefwd

- Use kubefwd to connect to services running in staging or a locally running kubernetes cluster

docker-compose.yml

```yaml
version: '3'
services:
  airflow-webserver:
    build: .
    depends_on:
      - airflow-postgres
    environment:
      - EXECUTOR=Local
      - LOAD_EX=n
      - FERNET_KEY=x19IeYL1ZJwyYQ-z76N1kaPbZm1nJzDyRMnU2txQhro=
      - POSTGRES_HOST=airflow-postgres
      - POSTGRES_PASSWORD=airflow
      - FOO_SERVICE_HOST=foo-kubernetes-service-proxy
    env_file:
      - dev.env
    ...

foo-kubernetes-service-proxy:
    image: google/cloud-sdk:237.0.0
    command: kubectl port-forward svc/foo 80 --address 0.0.0.0
    volumes:
      - ~/.config:/root/.config
      - ~/.kube:/root/.kube
      -
/usr/local/Caskroom/google-cloud-sdk:/usr/local/Caskroom/google-cloud-sdk
```

Testing DAGs with Airflow CLI

```
~/s/example-local-airflow 🔥 docker-compose exec airflow-webserver bash -c \
     "airflow test example_dag_v1 example_task 2020-01-10"
```

Testing DAGs with Airflow CLI

```
~/s/example-local-airflow 🔥 docker-compose exec airflow-webserver bash -c \
     "airflow test example_dag_v1 example_task 2020-01-10"

[2020-01-14 20:45:23,890] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2020-01-14 20:45:24,159] {{models.py:273}} INFO - Filling up the DagBag from /usr/local/airflow/dags
[2020-01-14 20:45:24,250] {{models.py:1359}} INFO - Dependencies all met for <TaskInstance:
example_dag_v1.example_task 2020-01-10T00:00:00+00:00 [None]>
[2020-01-14 20:45:24,255] {{models.py:1359}} INFO - Dependencies all met for <TaskInstance:
example_dag_v1.example_task 2020-01-10T00:00:00+00:00 [None]>
[2020-01-14 20:45:24,255] {{models.py:1571}} INFO -
--------------------------------------------------------------------------------
Starting attempt 1 of 1
--------------------------------------------------------------------------------

[2020-01-14 20:45:24,256] {{models.py:1593}} INFO - Executing <Task(PythonOperator): example_task> on
2020-01-10T00:00:00+00:00
[2020-01-14 20:45:24,270] {{python_operator.py:95}} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_ID=example_dag_v1
AIRFLOW_CTX_TASK_ID=example_task
AIRFLOW_CTX_EXECUTION_DATE=2020-01-10T00:00:00+00:00
hello world!
[2020-01-14 20:45:24,271] {{python_operator.py:104}} INFO - Done. Returned value was: None
```

Testing DAGs with Airflow CLI

scripts/test_task.sh

```bash
#!/bin/bash

docker-compose exec airflow-webserver bash -c "airflow test $1 $2 $3"
```

Testing DAGs with Airflow CLI

```
~/s/example-local-airflow 🔥 ./scripts/test_task.sh example_dag_v1 example_task 2020-01-10"


[2020-01-14 20:45:23,890] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2020-01-14 20:45:24,159] {{models.py:273}} INFO - Filling up the DagBag from /usr/local/airflow/dags
[2020-01-14 20:45:24,250] {{models.py:1359}} INFO - Dependencies all met for <TaskInstance:
example_dag_v1.example_task 2020-01-10T00:00:00+00:00 [None]>
[2020-01-14 20:45:24,255] {{models.py:1359}} INFO - Dependencies all met for <TaskInstance:
example_dag_v1.example_task 2020-01-10T00:00:00+00:00 [None]>
[2020-01-14 20:45:24,255] {{models.py:1571}} INFO -
--------------------------------------------------------------------------------
Starting attempt 1 of 1
--------------------------------------------------------------------------------

[2020-01-14 20:45:24,256] {{models.py:1593}} INFO - Executing <Task(PythonOperator): example_task> on
2020-01-10T00:00:00+00:00
[2020-01-14 20:45:24,270] {{python_operator.py:95}} INFO - Exporting the following env vars:
AIRFLOW_CTX_DAG_ID=example_dag_v1
AIRFLOW_CTX_TASK_ID=example_task
AIRFLOW_CTX_EXECUTION_DATE=2020-01-10T00:00:00+00:00
hello world!
[2020-01-14 20:45:24,271] {{python_operator.py:104}} INFO - Done. Returned value was: None
```

Testing DAGs with Airflow CLI - Rendering SQL

🔥 **./script/render_task.sh** example_sql_dag_v1 example_sql_task 2020-01-13

```
[2020-01-14 20:54:56,259] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2020-01-14 20:54:56,591] {{models.py:273}} INFO - Filling up the DagBag from /usr/local/airflow/dags
  """)
    # ------------------------------------------------------------
    # property: sql
    # ------------------------------------------------------------

SELECT *
  FROM my_table
 WHERE timestamp > '2020-01-13'
```

Testing DAGs with Airflow CLI - Rendering SQL

scripts/render_task.sh

```bash
#!/bin/bash

docker-compose exec airflow-webserver bash -c "airflow render $1 $2 $3"
```

Developing Plugins

- Plugins can be developed locally
- Airflow server will, by default, refresh the dagbag periodically
- You can also force a trigger of the DAG bag

```
example-local-airflow 🔥 tree

.
├── Dockerfile
├── docker-compose.yml

├── dev.env
├── requirements.txt
├── script
│   ├── connections.sh
│   └── entrypoint.sh
├── variables.json

├── config
│   └── airflow.cfg
├── dags
│   └── example_dag_v1.py
└── plugins
    ├── hooks
    ├── macros
    └── operators
        └── example_operator.py
```

Example Plugin Operator

plugins/operators/example_operator.py

```python
from airflow.operators import BaseOperator
from airflow.operators.python_operator import PythonOperator
from airflow.plugins_manager import AirflowPlugin
from airflow.utils.decorators import apply_defaults


class ExampleOperator(PythonOperator):
    @apply_defaults
    def __init__(self, python_callable=None, **kwargs):
        def _new_callable():
            print("hello")
            python_callable()
            print("outer!")
        super(ExampleOperator, self).__init__(python_callable=new_callable, **kwargs)


class ExampleOperatorPlugin(AirflowPlugin):
    name = "example_operator"
    operators = [ExampleOperator]
```

Example Plugin DAG

dags/example_plugin_dag.py

```python
from airflow import DAG
from airflow.operators.example_operator import ExampleOperator
from datetime import datetime

dag = DAG(
    "example_plugin_dag_v1",
    schedule_interval="0 0 0 * *",
    start_date=datetime(2020, 1, 12, 4, 20, 0),
)

t1 = ExampleOperator(
    task_id="example_task",
    python_callable=lambda: print("(hello inner!)"),
    dag=dag,
)
```

Testing DAGs with Airflow CLI - Testing Plugin Rendering

```
~/s/example-local-airflow 🔥 docker-compose exec airflow-webserver bash -c "python -c \"from airflow.models import DagBag; d = DagBag();\""

[2020-01-14 21:21:54,790] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2020-01-14 21:21:54,791] {{models.py:273}} INFO - Filling up the DagBag from /usr/local/airflow/dags
[2020-01-14 21:21:54,885] {{models.py:377}} ERROR - Failed to import:
/usr/local/airflow/dags/example_plugin_dag_v1.py
Traceback (most recent call last):
  File "/usr/local/airflow/.local/lib/python3.6/site-packages/airflow/models.py", line 374, in process_file
    m = imp.load_source(mod_name, filepath)
  File "/usr/local/lib/python3.6/imp.py", line 172, in load_source
    module = _load(spec)
  File "<frozen importlib._bootstrap>", line 684, in _load
  File "<frozen importlib._bootstrap>", line 665, in _load_unlocked
  File "<frozen importlib._bootstrap_external>", line 678, in exec_module
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "/usr/local/airflow/dags/example_plugin_dag_v1.py", line 14, in <module>
    dag=dag,
  File "/usr/local/airflow/.local/lib/python3.6/site-packages/airflow/utils/decorators.py", line 98, in wrapper
    result = func(*args, **kwargs)
  File "/usr/local/airflow/plugins/operators/example_operator.py", line 14, in __init__
    super(ExampleOperator, self).__init__(python_callable=new_callable, **kwargs)
NameError: name 'new_callable' is not defined
```

Error in plugin

Testing DAGs with Airflow CLI - Testing Plugin Rendering

```
~/s/example-local-airflow 🔥 docker-compose exec airflow-webserver bash -c "python -c \"from airflow.models import DagBag; d = DagBag();\""

[2020-01-14 21:25:41,849] {{__init__.py:51}} INFO - Using executor SequentialExecutor
[2020-01-14 21:25:41,850] {{models.py:273}} INFO - Filling up the DagBag from /usr/local/airflow/dags

~/s/example-local-airflow 🔥
```

Testing DAGs with Airflow CLI - Testing Plugin Rendering

scripts/refresh_dagbag.sh

```
#!/bin/bash

docker-compose exec airflow-webserver bash -c "python -c \"from airflow.models import
DagBag; d = DagBag();\"
"
```

In Summary

- Extend existing docker images to suit your needs
- Use airflow CLI commands and bash to set connections and variables
- Use docker-compose to manage local databases and proxies
- Use airflow CLI to test and render tasks and debug DAGs and plugins
- The only way to get a good local Airflow environment is bash, sweat, and tears

github.com/x/slides/airflow-nyc-2020
github.com/x/example-local-airflow

Thank You

We're hiring! **oden.io/jobs**

Data Scientist
Software Engineer (Front-end)
Software Engineer (Edge)
Head of Product Engineering