Making Data Workflows a Breeze with Apache Airflow

About Us

Josh O’Rourke

Kevin Kingsbury

Scott Walters

Agenda

- Data Architecture
- Airflow Overview
- Airflow Concepts
- Airflow UI
- Demo
- Q & A

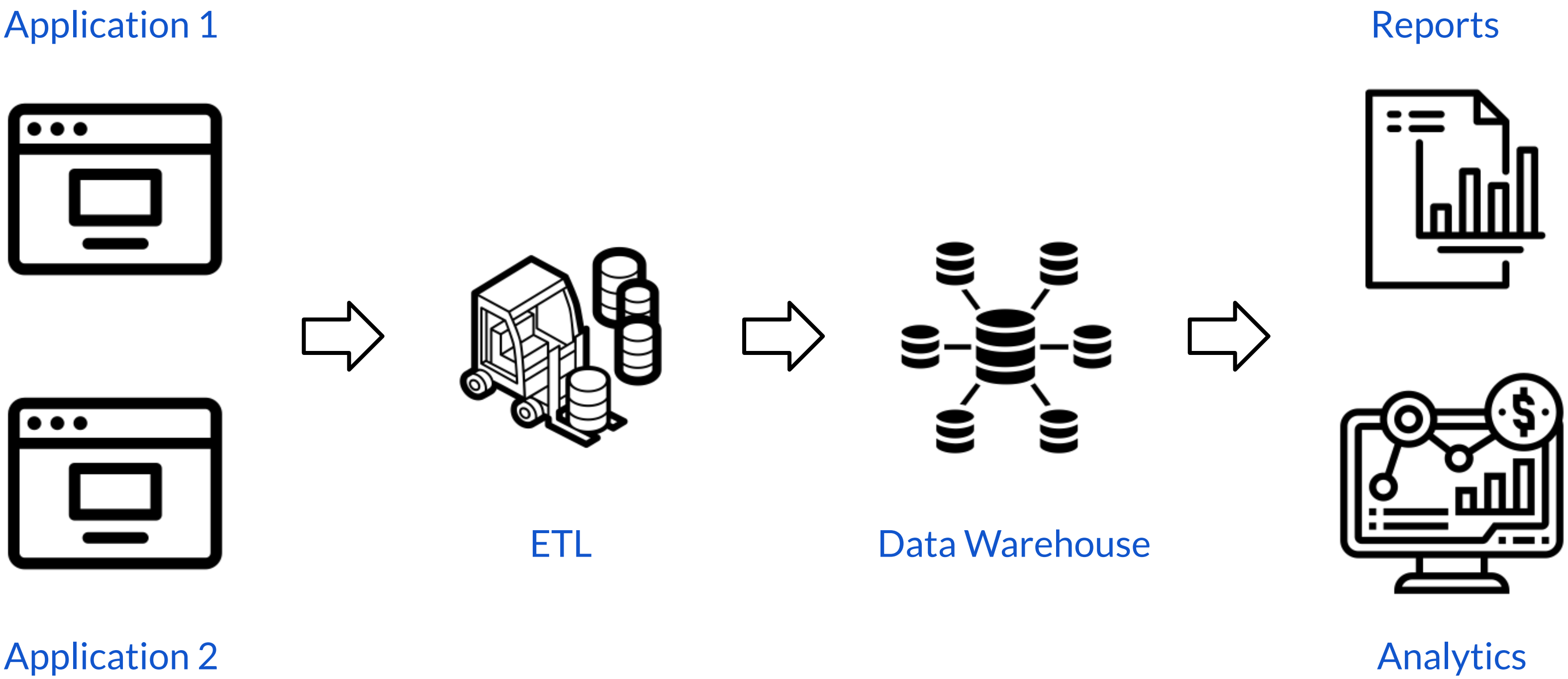
Data Architecture

A Naive Approach

> “In the beginning there was Cron.
> We had one job. It ran at 1 am.
> And it was good.”

Pete Owlett, PyData London 2016
Lessons from 6 Months of Using Luigi in Production

Data Architecture



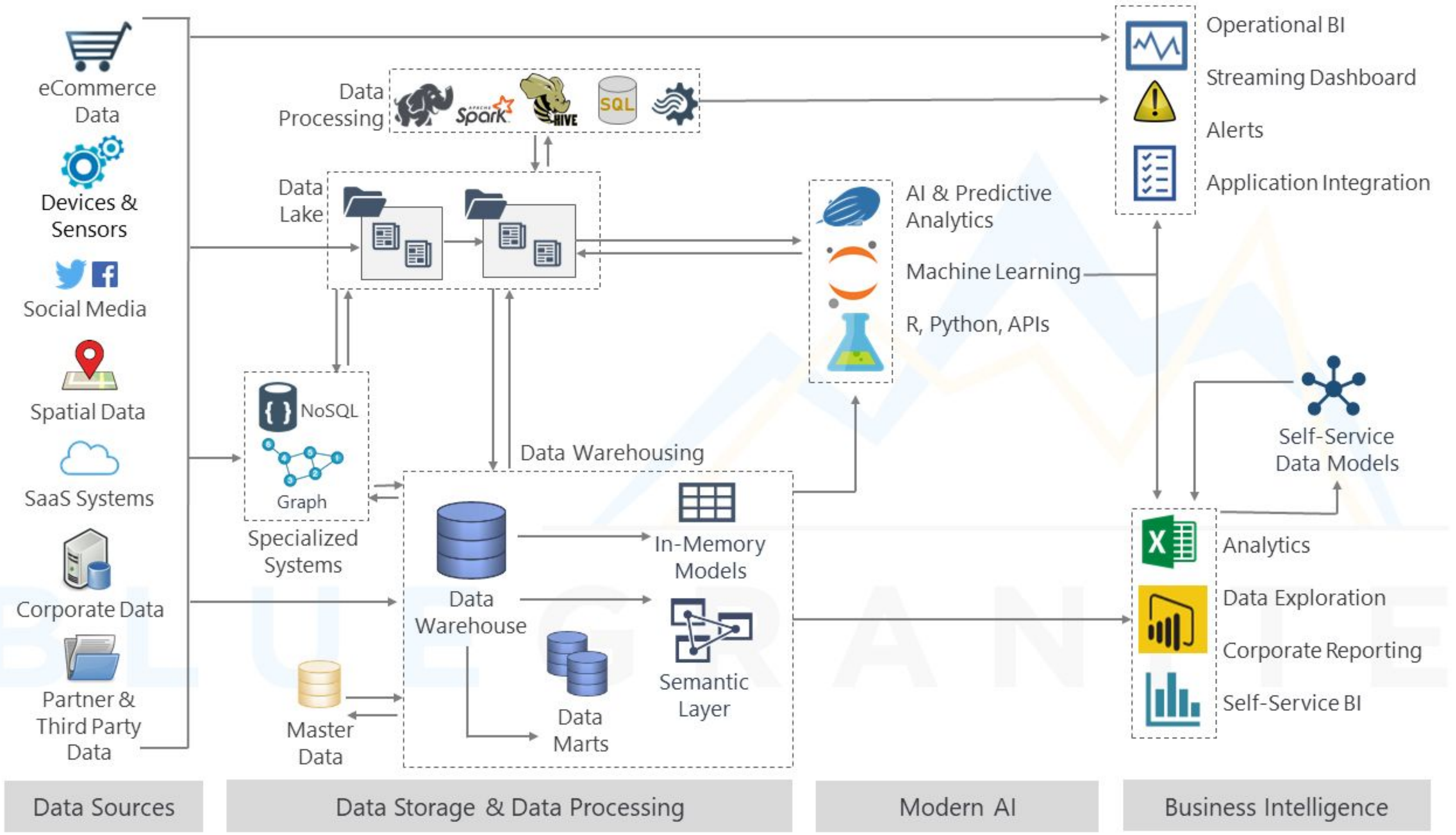
Problems

- **Failures:** How do we retry when failure happens?
- **Monitoring**: Are we notified about errors? How long does the job run?
- **Dependencies**: What happens if upstream data is missing?
- **Scalability**: How do we coordinate cron jobs on different machines?
- **Deployment**: How often do we need to deploy changes? Is it easy?
- **Historic Data**: Can we backfill historical data?
- **Logging**: Do we have unified logs?

Modern Data Architecture

Modern Architecture

Modern Architecture



Apache Airflow

What Is Apache Airflow?

- Platform to programmatically author, schedule, and monitor workflows
- Developed by Airbnb in 2015, moved to Apache Foundation in 2016
- Used by 120+ companies, including Google, ING, Lyft, Paypal, Reddit, and more
- Useful for ETL, Machine Learning, Predictive, and General Pipelines

2015
Airbnb releases Airflow as open source

2016
Airflow moved to Apache Foundation (incubating status)

May 2018
Google introduces Airflow for Google Cloud Composer

December 2018
Airflow graduates from Apache Foundation incubation

Airflow Principles & Features

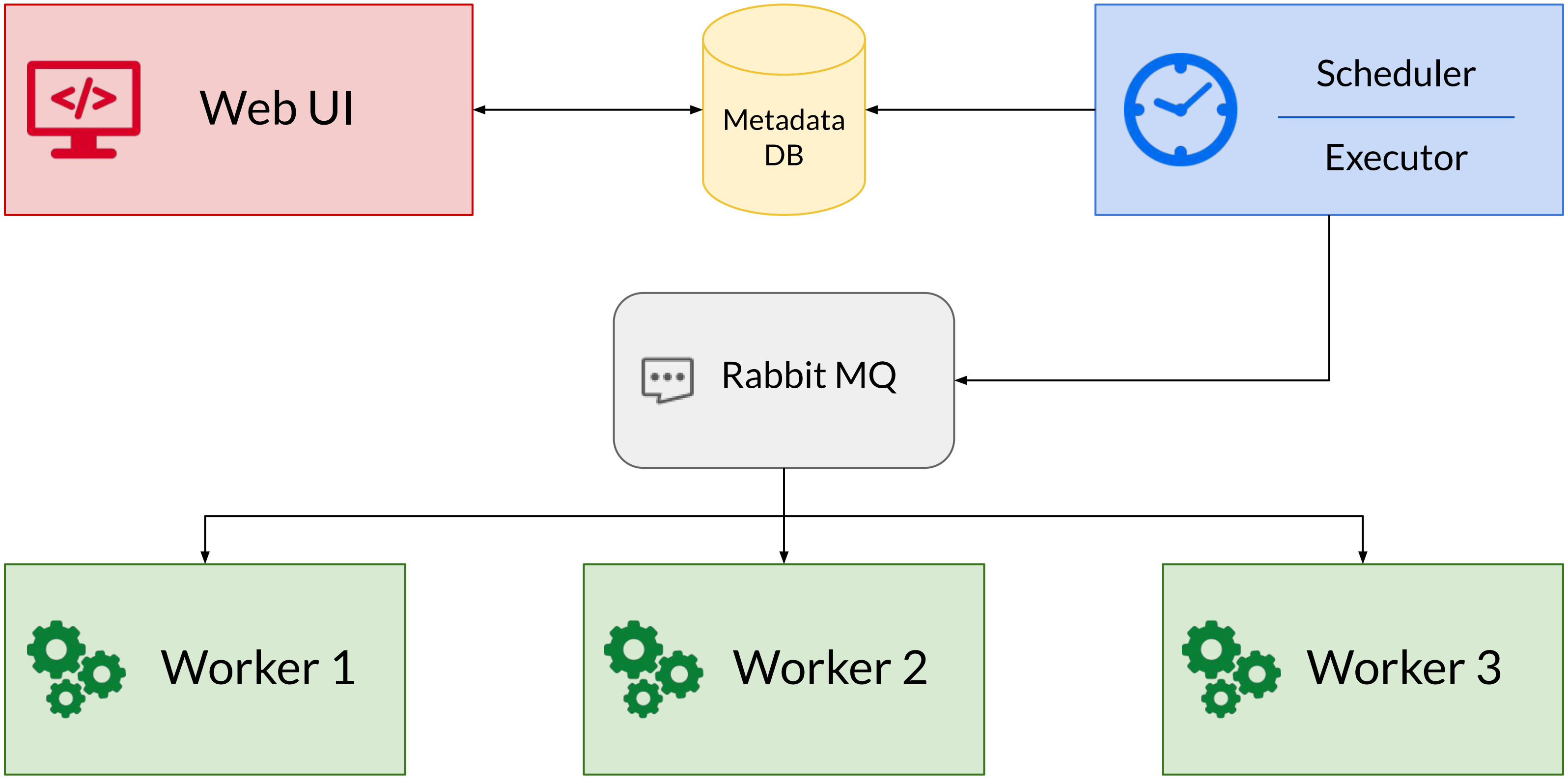
Principles

- *Reproducible*
  - Deterministic and idempotent
- *Future proof*
  - Backfilling, versioning
- *Robust Against Changes*
  - Easy to add, remove, modify DAGs
- *Clarity*
  - Transparency where data resides, what it means, and where it flows

Features

- Configuration as Code
- Usability (Web UI)
- Centralized Configuration
- Resource Pooling
- Extensibility
- Security (user, passwords)

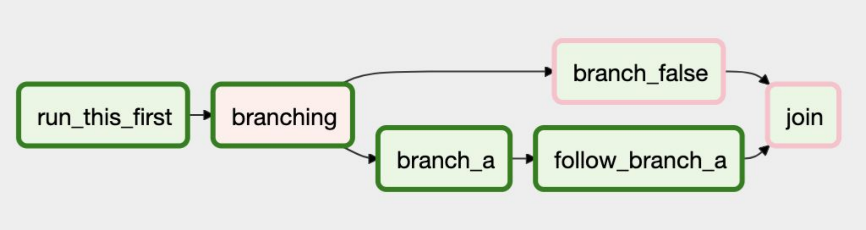
Airflow Architecture



Airflow Concepts

Airflow DAGs

“A DAG - or a Directed Acyclic Graph – is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies.”




```
default_args = {
    'owner': 'Airflow',
    'depends_on_past': False,
    'start_date': airflow.utils.dates.days_ago(2),
    'email': ['airflow@example.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
    # 'queue': 'bash_queue',
    # 'pool': 'backfill',
    # 'priority_weight': 10,
    # 'end_date': datetime(2016, 1, 1),
    # 'wait_for_downstream': False,
    # 'dag': dag,
    # 'sla': timedelta(hours=2),
    # 'execution_timeout': timedelta(seconds=300),
    # 'on_failure_callback': some_function,
    # 'on_success_callback': some_other_function,
    # 'on_retry_callback': another_function,
    # 'sla_miss_callback': yet_another_function,
    # 'trigger_rule': 'all_success'
}

dag = DAG('example',
          default_args=default_args,
          description='A simple tutorial DAG',
          schedule_interval=timedelta(days=1))
```

Airflow Operators

- **BashOperator** - executes a bash command
- **PythonOperator** - calls an arbitrary Python function
- **EmailOperator** - sends an email
- **SimpleHttpOperator** - sends an HTTP request
- **SQL Operators** - executes a SQL command
- **Sensor** - waits for a certain time, file, database row, S3 key, etc...

```
t1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag)

t2 = BashOperator(
    task_id='sleep',
    depends_on_past=False,
    bash_command='sleep 5',
    dag=dag)

templated_command = """
{% for i in range(5) %}
    echo "{{ ds }}"
    echo "{{ macros.ds_add(ds, 7)}}"
    echo "{{ params.my_param }}"
{% endfor %}
"""

t3 = BashOperator(
    task_id='templated',
    depends_on_past=False,
    bash_command=templated_command,
    params={'my_param': 'Parameter I passed in'},
    dag=dag)

t1 >> [t2, t3]
```

Airflow Operators

```
def print_context(ds, **context):
    """Print the Airflow context and ds variable from the context."""
    print(context)
    print(ds)
    return 'Whatever you return gets printed in the logs'

run_this = PythonOperator(
    task_id='print_the_context',
    provide_context=True,
    python_callable=print_context
    dag=dag)

def my_sleeping_function(random_base):
    """This is a function that will run within the DAG execution"""
    time.sleep(random_base)

# Generate 5 sleeping tasks, sleeping from 0.0 to 0.4 seconds respectively
for i in range(5):
    task = PythonOperator(
        task_id='sleep_for_' + str(i),
        python_callable=my_sleeping_function,
        op_kwargs={'random_base': float(i) / 10},
        dag=dag)

    run_this >> task
```

Airflow Operators

```
start = DummyOperator(task_id='start')

github = GithubToS3Operator(task_id='github_to_s3'
                           github_conn_id=GITHUB_CONN_ID,
                           github_repo=GITHUB_REPO,
                           github_object=GITHUB_OBJECT,
                           s3_conn_id=S3_CONN_ID,
                           s3_bucket=S3_BUCKET,
                           s3_key=S3_KEY)

redshift = S3ToRedshiftOperator(task_id='s3_to_redshift',
                               s3_conn_id=S3_CONN_ID,
                               s3_bucket=S3_BUCKET,
                               s3_key=S3_KEY,
                               ...
                               redshift_schema=REDSHIFT_SCHEMA,
                               redshift_conn_id=REDSHIFT_CONN_ID,
                               table='my_table')

start >> github >> redshift
```

Airflow Hooks

- Interfaces to external platforms and DBs
- Building block for operators
- Keep authentication code out of pipelines, centralized in the metadata DB

SQL (MSSQL, Oracle, Postgres)	AWS (S3, Lambda, Dynamo)
FS, FTP, SFTP, Samba, SSH	Azure (Cosmos, Data Lake, Fileshare)
HTTP, IMAP	Google Cloud (Big Table, ML, NLP)
Docker	Spark, Vertica
Redis, Mongo	Datadog, Salesforce, Slack

```
# Specify a connection
hook = SqliteHook('sqlite_connection_id')

# Get some records
records = hook.get_records('select * from accounts')
hook.run('delete from accounts')

# Bulk dump / load some data
hook.bulk_dump('accounts', '/tmp/accounts.csv')
hook.bulk_load('accounts', '/tmp/accounts.csv')

# Get a Pandas dataframe
df = hook.get_pandas_df('select * from users')

# Get a SqlAlchemy Engine
engine = hook.get_sqlalchemy_engine(...)
```

Airflow XComs

- XCom = Cross Communication
- Communication between tasks
- Can be “pushed” or “pulled” by all task instances

```
# Pushes an XCom without a specific target
def pusher1(**context):
    context['ti'].xcom_push(key='value 1',
                            value=value1)


# Pushes an XCom without a specific target,
# just by returning it
def pusher2():
    return 'value 2'


# Pull all previously pushed XComs
def puller(**context):
    ti = context['ti']

    # get value1
    value1 = ti.xcom_pull(key='value 1',
                          task_ids='pusher1')

    # get value2
    value2 = ti.xcom_pull(key='return_value',
                          task_ids='pusher2')

    # get both value1 and value2
    value1, value2 = ti.xcom_pull(
           key=None,
           task_ids=['pusher1', 'pusher2'])
```

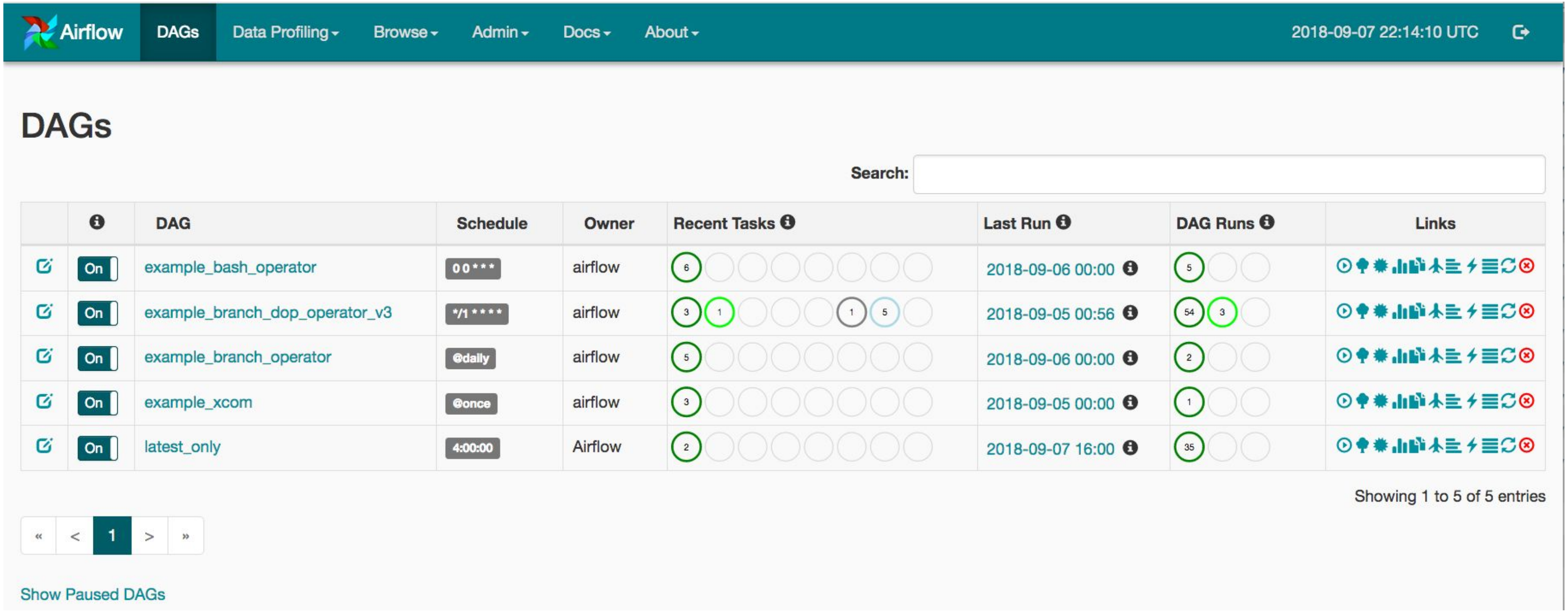
Airflow Variables

- Generic way to store and retrieve content as a simple key value store within Airflow
- Useful for configuration items that must be accessible and modifiable through a UI

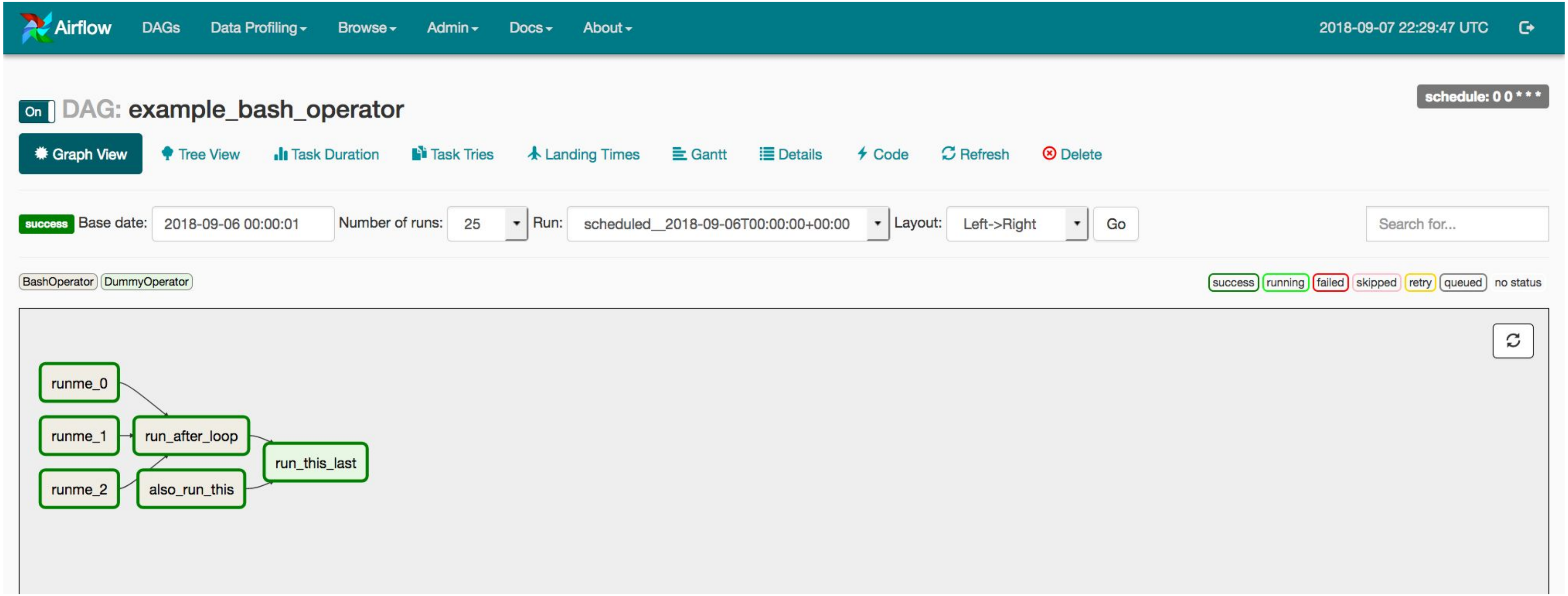
```
# Variables can be in Python ...
from airflow.models import Variable
foo = Variable.get("foo")
bar = Variable.get("bar", deserialize_json=True)
baz = Variable.get("baz", default_var=None)

# ... or Jinja templates
echo {{ var.value.<variable_name> }}
echo {{ var.json.<variable_name> }}
```

UI Walkthrough

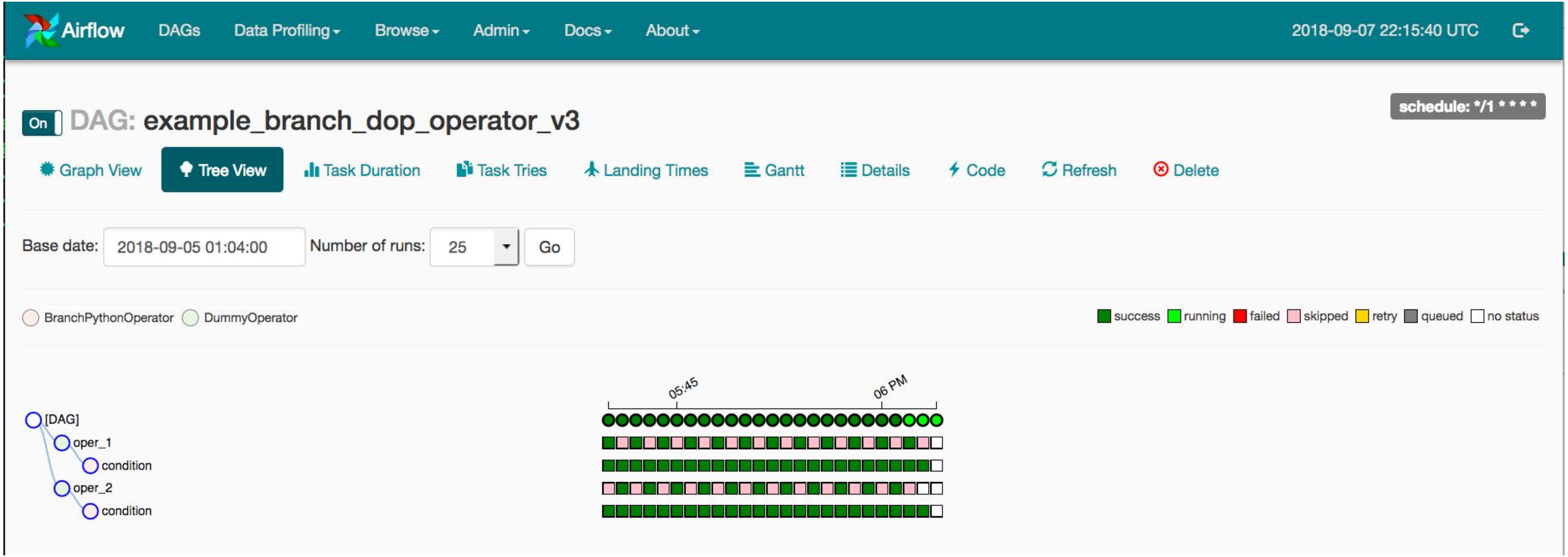
DAGS View


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2018-09-07 22:14:10 UTC
DAGs
Search:
DAG
Schedule
Owner
Recent Tasks
Last Run
DAG Runs
Links
On
example_bash_operator
0 0 * * *
airflow
2018-09-06 00:00
On
example_branch_dop_operator_v3
*/1 * * * *
airflow
2018-09-05 00:56
On
example_branch_operator
@daily
airflow
2018-09-06 00:00
On
example_xcom
@once
airflow
2018-09-05 00:00
On
latest_only
4:00:00
Airflow
2018-09-07 16:00
Showing 1 to 5 of 5 entries
Show Paused DAGs

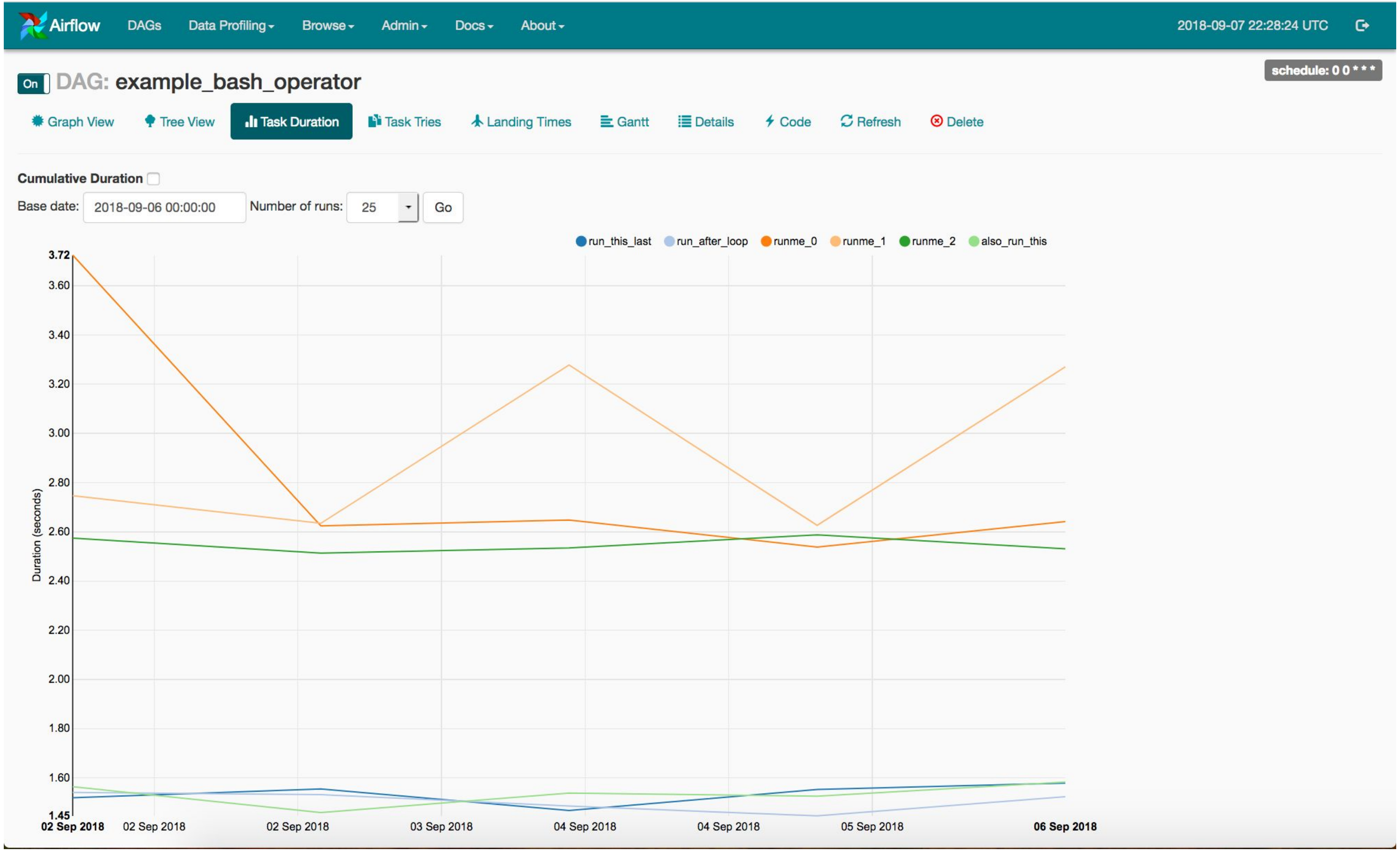
Graph View


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2018-09-07 22:29:47 UTC
On
DAG: example_bash_operator
schedule: 0 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Delete
success
Base date:
2018-09-06 00:00:01
Number of runs:
25
Run:
scheduled__2018-09-06T00:00:00+00:00
Layout:
Left->Right
Go
Search for...
BashOperator
DummyOperator
success
running
failed
skipped
retry
queued
no status
runme_0
runme_1
runme_2
run_after_loop
also_run_this
run_this_last

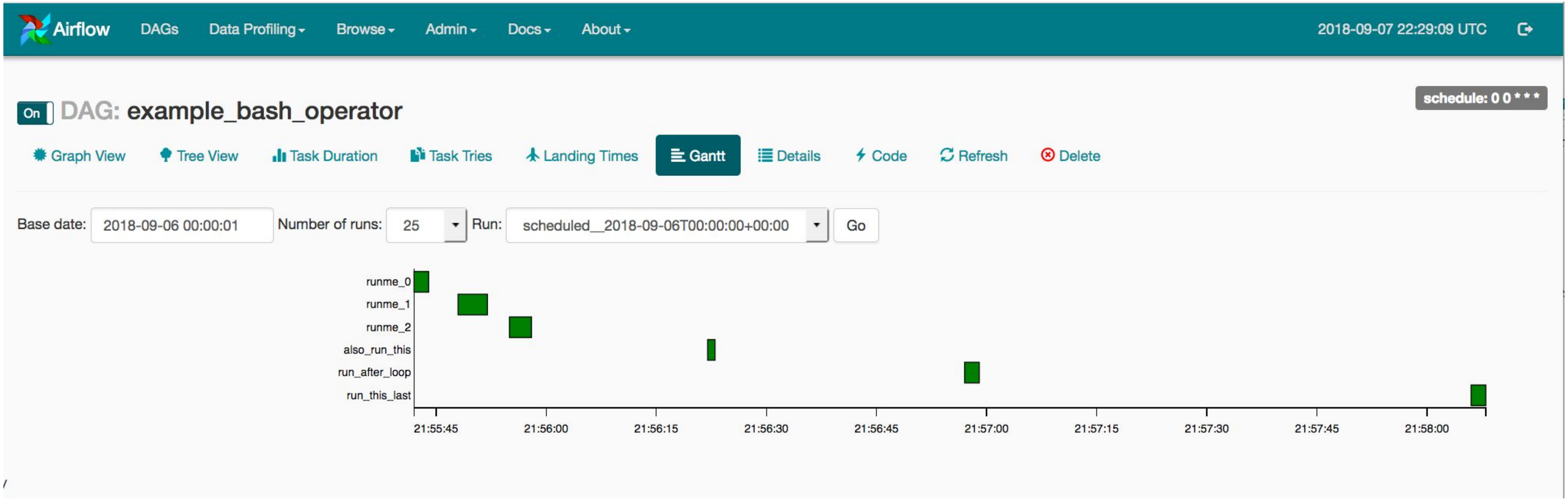
Tree View


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2018-09-07 22:15:40 UTC
schedule: */1 * * * *
On
DAG: example_branch_dop_operator_v3
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Delete
Base date:
2018-09-05 01:04:00
Number of runs:
25
Go
BranchPythonOperator
DummyOperator
success
running
failed
skipped
retry
queued
no status
05:45
06 PM
[DAG]
oper_1
condition
oper_2
condition

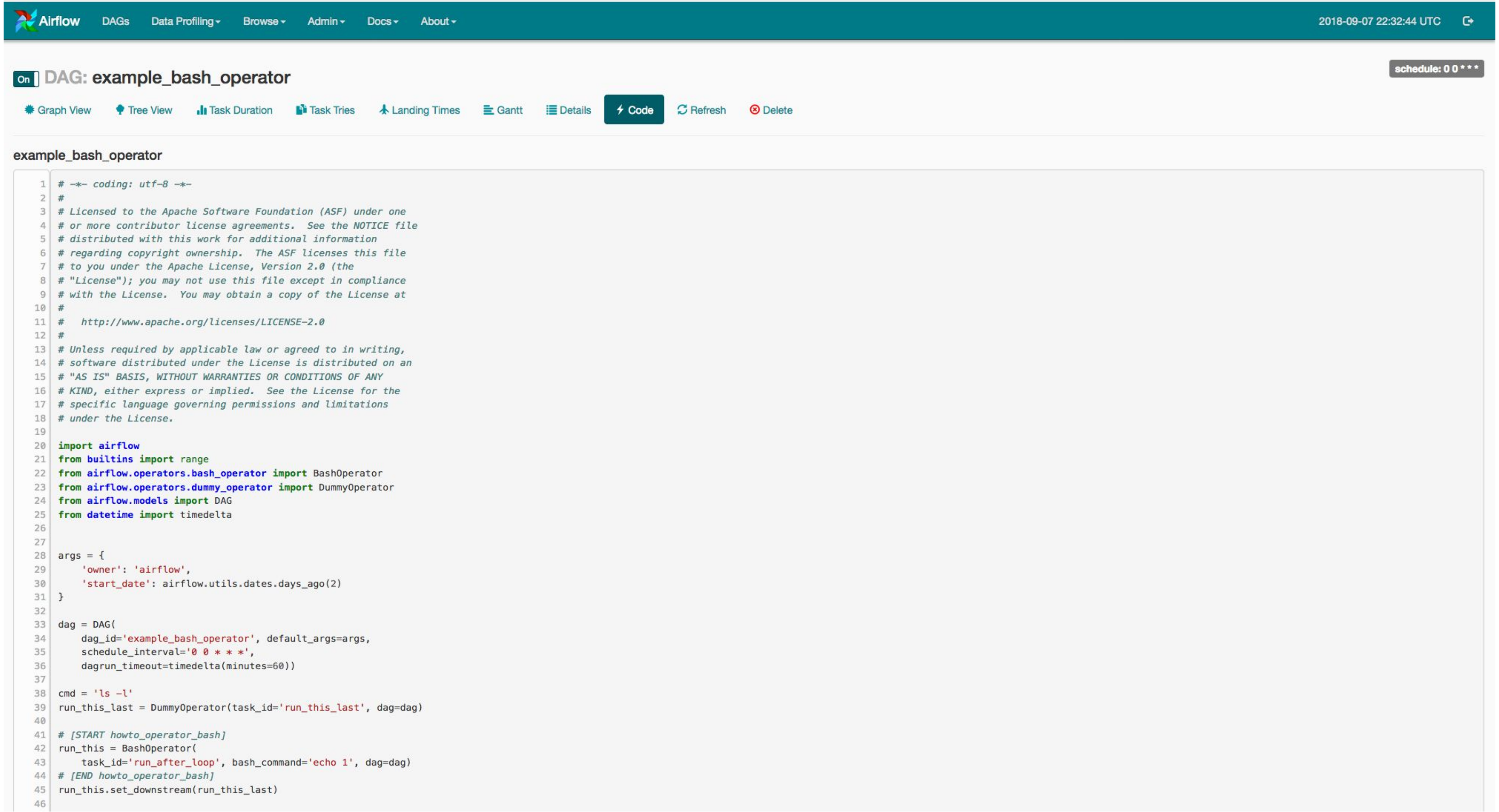
Task Duration



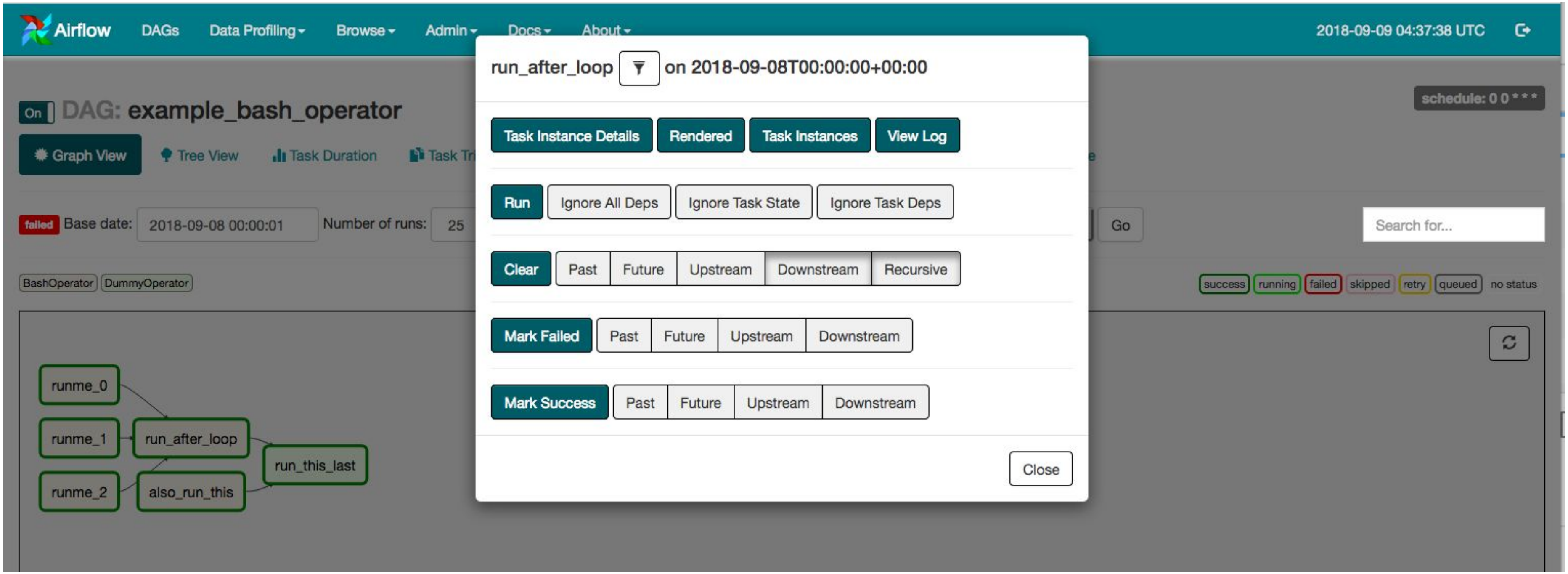
Gantt View


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2018-09-07 22:29:09 UTC
On
DAG: example_bash_operator
schedule: 0 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Delete
Base date: 2018-09-06 00:00:01
Number of runs: 25
Run: scheduled__2018-09-06T00:00:00+00:00
Go
runme_0
runme_1
runme_2
also_run_this
run_after_loop
run_this_last
21:55:45
21:56:00
21:56:15
21:56:30
21:56:45
21:57:00
21:57:15
21:57:30
21:57:45
21:58:00

Code View


Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2018-09-07 22:32:44 UTC
On
DAG: example_bash_operator
schedule: 0 0 * * *
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Delete
example_bash_operator
-*- coding: utf-8 -*-
#
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements. See the NOTICE file
distributed with this work for additional information
regarding copyright ownership. The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License. You may obtain a copy of the License at
#
http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied. See the License for the
# specific language governing permissions and limitations
# under the License.
import airflow
from builtins import range
from airflow.operators.bash_operator import BashOperator
from airflow.operators.dummy_operator import DummyOperator
from airflow.models import DAG
from datetime import timedelta
args = {
'owner': 'airflow',
'start_date': airflow.utils.dates.days_ago(2)
}
dag = DAG(
dag_id='example_bash_operator', default_args=args,
schedule_interval='0 0 * * *',
dagrun_timeout=timedelta(minutes=60))
cmd = 'ls -l'
run_this_last = DummyOperator(task_id='run_this_last', dag=dag)
# [START howto_operator_bash]
run_this = BashOperator(
task_id='run_after_loop', bash_command='echo 1', dag=dag)
# [END howto_operator_bash]
run_this.set_downstream(run_this_last)

Task Instance View



Demos

Resources

Resources

Apache Airflow
https://airflow.apache.org/

Docker Apache Airflow
https://github.com/puckel/docker-airflow

Awesome Apache Airflow
https://github.com/jghoman/awesome-apache-airflow

Questions?