Introduction to Airflow

Christof von Rabenau - 2022

What is Airflow?

- Open source workflow engine
  - Program with Python
  - Schedule
  - Monitor
- Scalable
- Dynamic
- Extensible

Basic components of Airflow environment

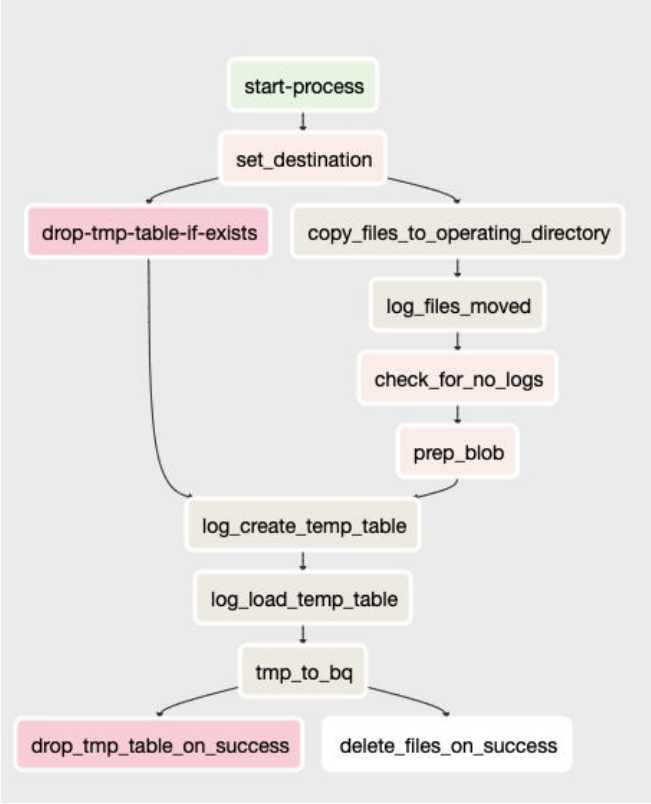
- Airflow Database
  - Simplest for a SQLite DB
  - Can be hosted in MySQL, PostgreSQL, or MSSQL
  - Used for:
    - Holds component relationship information
    - Stores state
    - All process reads and writes here.
- Airflow webserver
  - Manages web GUI for seeing workloads
  - Flask web server
- Airflow scheduler
  - Responsible for orchestrating tasks
  - Retrieves information from Airflow database
- Airflow executor
  - Responsible for actually execution of defined code

Installation:

- Add an ENV variable for AIRFLOW_HOME
  - `export AIRFLOW_HOME=~/[path for your Airflow home]`
- Install airflow with pip
  - `pip3 install apache-airflow`
- Initialize Airflow DB - this will initialize a SQLite DB
  - `airflow db init`
- Start the Airflow webserver
  - `airflow webserver -p [local port]`
- Start the Airflow scheduler
  - `airflow scheduler`
- Create an Airflow user
  - `airflow users create -e [email] -f [firstname] -l [lastname] -u [username] -r [role]

DAGS

- DAG - Directed Acyclic Graph
  - Collection of tasks defined in Python code
  - Show relationship and dependency between tasks
  - Flow in one direction



Operators are key components to DAGs

- Basic operators:
  - DummyOperator
  - BashOperator
  - PythonOperator
  - ShortCircuitOperator
- API operators:
  - Azure
    - AzureDataFactoryRunPipelineOperator
    - ADLSDeleteOperator
  - AWS
    - S3CopyObjectOperator
    - RedshiftSQLOperator
  - Google
    - BigQueryOperator
    - GoogleCloudStorageOperator

Passing information from one task to another

- XCOMs
  - Built-in functionality
  - Allows access to information from tasks
  - All tasks have a ‘result’ XCOM, but others can be set intentionally
  - Limited to 48KB
  - Use for simple passage of results
  - Results in tasks being overly dependent on upstream tasks
- External storage
  - Database
  - Object store
  - Use for passing large data between tasks

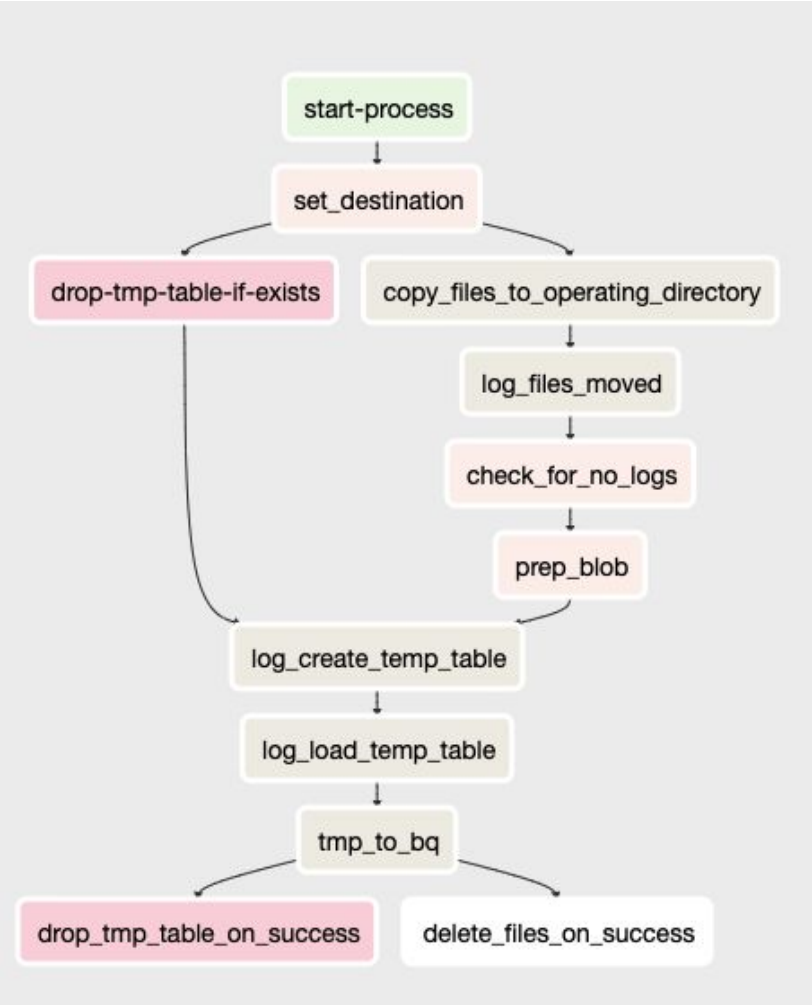
Demo

http://localhost:8080

Sample dags are in my git repo here: https://github.com/cgapperi/airflow_demo

More complex Code

```
import sys
import os

from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.python_operator import PythonOperator, ShortCircuitOperator
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.contrib.operators.gcs_to_gcs import GoogleCloudStorageToGoogleCloudStorageOperator
from airflow.contrib.operators.bigquery_operator import BigQueryCreateEmptyTableOperator
from airflow.contrib.operators.gcs_list_operator import GoogleCloudStorageListOperator
from airflow.contrib.operators.bigquery_operator import BigQueryOperator
from airflow.utils.trigger_rule import TriggerRule
from airflow.contrib.operators.bigquery_table_delete_operator import BigQueryTableDeleteOperator
from airflow.contrib.operators.gcs_delete_operator import GoogleCloudStorageDeleteOperator

from dag_lib.superlog_common import SuperlogProcessor

script_dir = os.path.dirname(os.path.realpath(__file__))
sys.path.append(os.path.join(script_dir, 'dag_lib'))

dag_id = os.path.basename(__file__)[:-3]
log_type = "gateway.impression.log"
schema_file = "full_superlog_tmp.json"

# Define success and failure flag to have consistency.
SUCCESS_LABEL = "success"
FAILURE_LABEL = "failure"

# Set Dag operator parameters
DS_TAG = '{{ ds }}'

sp = SuperlogProcessor(log_type)
default_args = sp.set_default_args()

# creating a Dag object
with DAG(dag_id=dag_id, schedule_interval='*/5 * * * *',
         catchup=False,
         default_args=default_args) as dag:

    # step1: a dummy task to indicate the start of the process
    start_process = DummyOperator(task_id="start-process")

    # step2: initialize XCOMs for use through DAG
    set_destination = PythonOperator(
        task_id="set_destination",
        python_callable=sp.set_destination,
        op_kwargs={"log_type": log_type},
        provide_context=True,
        retries=3,
        dag=dag
    )

    # step3a: copy files from sync bucket to operating directory
    copy_files_to_operating_directory = GoogleCloudStorageToGoogleCloudStorageOperator(
        task_id="copy_files_to_operating_directory",
        source_bucket=sp.source_bucket,
        source_object=f"{log_type}*.gz",
        destination_bucket=sp.quarantine_bucket,
```



Resources

- Official Airflow documentation
- Operator documentation
- Mark Lamberti!
- Medium author: Kaxil Naik
- Astronomer.io Resources
  - Blog:
    - 7 Common Errors to Check When Debugging Airflow DAGs
    - Useful SQL queries for Apache Airflow
  - Guides
- Airflow: Tips, Tricks & Pitfalls
- Testing DAGS (in Google Composer)
-

Questions?

Airflow web interface

Airflow DAGs Security Browse Admin Docs 18:43 UTC AU

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. **Click here** for more information.

Do not use **SequentialExecutor** in production. **Click here** for more information.

DAGs

All 4 | Active 3 | Paused 1

Filter DAGs by tag

Search DAGs

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
dummy_dag	airflow	3	1 day, 0:00:00	2022-02-03, 17:53:07	2022-02-03, 00:00:00	5		
more_complex	airflow	3	1 day, 0:00:00	2022-02-03, 18:01:26	2022-02-03, 00:00:00	11		
simple_operators	airflow	2	1 day, 0:00:00	2022-02-02, 00:00:00	2022-02-03, 00:00:00	5		
tutorial example	airflow		1 day, 0:00:00		2022-02-02, 18:26:03			

« ‹ 1 › »

Showing **1-4** of **4** DAGs

DAG Tree view

Airflow DAGs Security Browse Admin Docs 18:44 UTC AU

DAG: dummy_dag

Schedule: 1 day, 0:00:00 Next Run: 2022-02-03, 00:00:00

Tree Graph Calendar Task Duration Task Tries Landing Times Gantt Details

Code

2022-02-03T17:53:07Z Runs 25 Update

DummyOperator

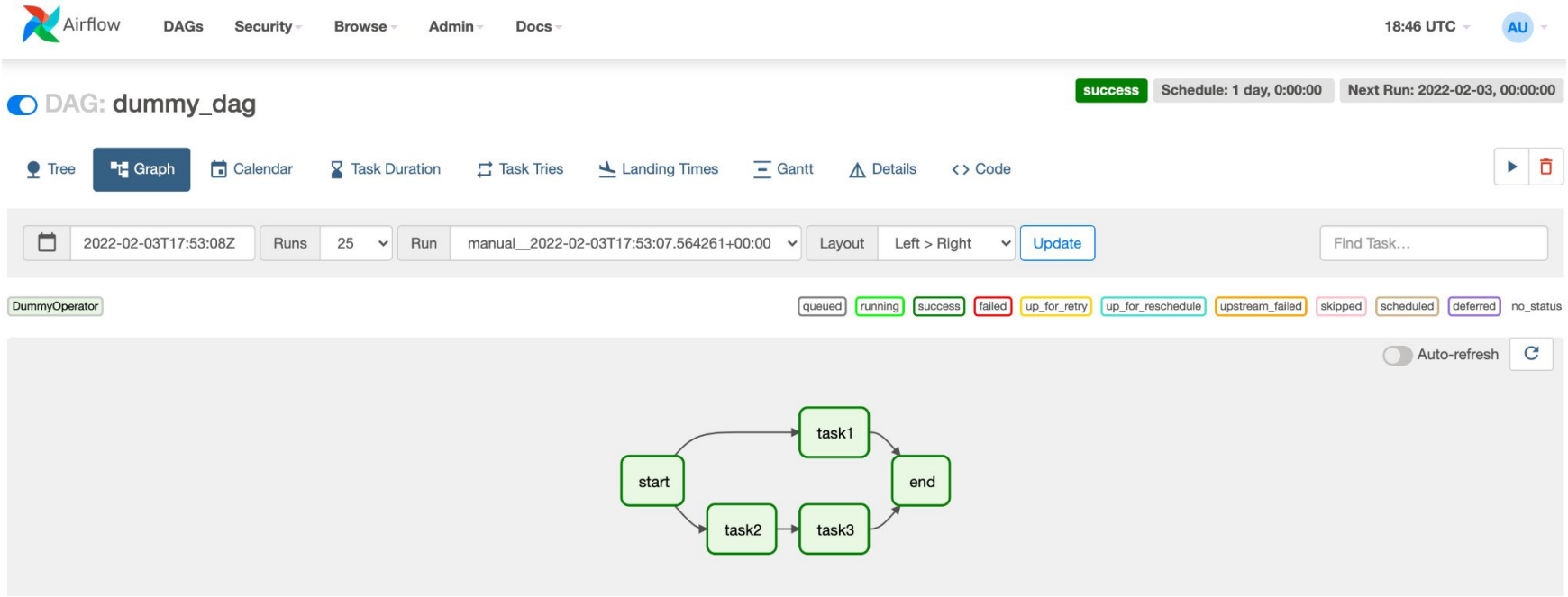
queued running success failed up_for_retry up_for_reschedule upstream_failed skipped scheduled deferred no_status

Auto-refresh

Jan 31, 19:00
Feb 03, 12:53

[DAG]
start
task1
end
task2
task3
end

DAG graph view



Airflow code view

Airflow Security Browse Admin Docs

DAG: more_complex

Tree Graph Calendar Task Duration Task Tries Landing Times Gantt Details Code

```
1  from datetime import datetime, timedelta
2  from airflow import DAG
3  from airflow.operators.dummy import DummyOperator
4  from airflow.operators.python import PythonOperator
5  from airflow.operators.bash import BashOperator
6  from random import randint
7
8  params = {
9      "schedule_interval":"@once",
10     "catchup":False
11 }
12
13 def hello_world(**kwargs):
14     print("Hello World!")
15
16 def add_ints(int1, int2):
17     return int1 + int2
18
19 def sub_ints(int1, int2):
20     return int1 - int2
21
22 def mult_ints(int1, int2):
23     return int1 * int2
24
25 def div_ints(int1, int2):
26     return int1/int2
27
28 def do_math(**kwargs):
29     ti = kwargs['ti']
30     print(ti)
31     operator = kwargs['operator']
```