Manageable data pipelines with Airflow (and Kubernetes)
[AIRFLOW-3275] Add Google Cloud SQL Query operator (#4170)
potiuk authored and kaxil committed 17 days ago ✓

[AIRFLOW-3345] Add Google Cloud Storage (GCS) operators for ACL (#4192)
sprzedwojski authored and kaxil committed 15 days ago ✓

GDG DevFest Warsaw 2018
Airflow is a platform to programmatically author, schedule and monitor workflows.

- Dynamic/Elegant
- Extensible
- Scalable
Workflows

Extract Ad Network Revenue → Transform Spreadsheet Data → Combine & Append Data Sources → Check Weekly Historical Data

Extract App Store A Revenue → Transform JSON Data → Perform Currency Conversions → Make Next Day Prediction

Extract App Store B Revenue

Extract App Store C Revenue

Wait for Conversion Rates → Get Daily Conversion Rates

Source: https://medium.com/@dustinstansbury/understanding-apache-airflows-key-concepts-a96efed52b1a

GDG DevFest Warsaw 2018

@higrys, @sprzedwojski
Companies using Airflow
(>200 officially)
Data Pipeline

https://xkcd.com/2054/
Airflow vs. other workflow platforms

- Programming workflows
  - writing code not XML
  - versioning as usual
  - automated testing as usual
  - complex dependencies between tasks

- Managing workflows
  - aggregate logs in one UI
  - tracking execution
  - re-running, backfilling (run all missed runs)
Airflow use cases

- ETL jobs
- ML pipelines
- Regular operations:
  - Delivering data
  - Performing backups
- ...

GDG DevFest Warsaw 2018  
@higrys, @sprzedwojski
Core concepts - Directed Acyclic Graph (DAG)

Source: https://github.com/apache/incubator-airflow/blob/master/airflow/contrib/example_dags/example_twitter_README.md
Core concepts - Operators

Source: https://blog.usejournal.com/testing-in-airflow-part-1-dag-validation-tests-dag-definition-tests-and-unit-tests-2a5f4970570c
Operator types

- Action Operators
  - Python, Bash, Docker, GCEInstanceStart, ...
- Sensor Operators
  - S3KeySensor, HivePartitionSensor, BigtableTableWaitForReplicationOperator, ...
- Transfer Operators
  - MsSqlToHiveTransfer, RedshiftToS3Transfer, ...
class ExampleOperator(BaseOperator):
    def execute(self, context):
        # Do something
        pass
Operator and Sensor

class ExampleOperator(BaseOperator):
    def execute(self, context):
        # Do something
        pass

class ExampleSensorOperator(BaseSensorOperator):
    def poke(self, context):
        # Check if the condition occurred
        return True
Operator good practices

- Idempotent
- Atomic
- No direct data sharing
  - Small portions of data between tasks: XCOMs
  - Large amounts of data: S3, GCS, etc.
Core concepts - Tasks, TaskInstances, DagRuns

Source: https://medium.com/@dustinstansbury/understanding-apache-airflows-key-concepts-a96efed52b1a

@higrys, @sprzedwojski
The solution

Sources:
https://services.garmin.cn/appsLibraryBusinessServices_v0/rest/apps/9b5dabf3-925b
https://malloc.fi/static/images/slack-memory-management.png
https://i.gifer.com/9GXs.gif
Solution components

- Generic
  - BashOperator
  - PythonOperator
- Specific
  - EmailOperator
The DAG

gcp_service_list_instances_spanner

gcp_service_list_instances_sql

prepare_email_task

send_email

send_slack_msg_task

gcp_service_list_instances_compute

gcp_service_list_instances_bigtable
Initialize DAG

dag = DAG(dag_id='gcp_spy',
          ...
        )
Initialize DAG

dag = DAG(dag_id='gcp_spy',
          default_args={
              'start_date': utils.dates.days_ago(1),
              'retries': 1
          },
          ...
Initialize DAG

dag = DAG(dag_id='gcp_spy',
    default_args={
        'start_date': utils.dates.days_ago(1),
        'retries': 1
    },
    schedule_interval='0 16 * * *'
)
List of instances

```python
bash_task = BashOperator(
    task_id="gcp_service_list_instances_sql",
    ...
)
```
bash_task = BashOperator(
    task_id="gcp_service_list_instances_sql",
    bash_command=
    "gcloud sql instances list | tail -n +2 | grep -oE '^[^ ]+' "
    "| tr '\n' ' '",
    ...
)
bash_task = BashOperator(
    task_id="gcp_service_list_instances_sql",
    bash_command=
    "gcloud sql instances list | tail -n +2 | grep -oE '^[^ ]+\' "
              "| tr \"\n\" \"",
    xcom_push=True,
    ...
)
List of instances

```python
bash_task = BashOperator(
    task_id="gcp_service_list_instances_sql",
    bash_command=
    "gcloud sql instances list | tail -n +2 | grep -oE '^[^ ]+' "
    " | tr '\n' ' '",
    xcom_push=True,
    dag=dag
)
```
All services

GCP_SERVICES = [
    ('sql', 'Cloud SQL'),
    ('spanner', 'Spanner'),
    ('bigtable', 'BigTable'),
    ('compute', 'Compute Engine'),
]
List of instances - all services

```python
bash_task = BashOperator(
    task_id="gcp_service_list_instances_sql",
    bash_command=
        "gcloud sql instances list | tail -n +2 | grep -oE '^[^ ]+' "
        "| tr '\n' ' '",
    xcom_push=True,
    dag=dag
)
```
List of instances - all services

for gcp_service in GCP_SERVICES:
    bash_task = BashOperator(
        task_id="gcp_service_list_instances_{}".format(gcp_service[0]),
        bash_command=
        "gcloud {} instances list | tail -n +2 | grep -oE \'^[^ ]+\' "
        "| tr '\n' ' '".format(gcp_service[0]),
        xcom_push=True,
        dag=dag
    )
Send Slack message

```python
send_slack_msg_task = PythonOperator(
    python_callable=send_slack_msg,
    provide_context=True,
    task_id='send_slack_msg_task',
    dag=dag
)
```
Send Slack message

```python
send_slack_msg_task = PythonOperator(
    python_callable=send_slack_msg,
    provide_context=True,
    task_id='send_slack_msg_task',
    dag=dag
)
```
def send_slack_msg(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].\
            xcom_pull(task_ids='gcp_service_list_instances_{0}'.\
                format(gcp_service[0]))
    ...

GDG DevFest Warsaw 2018  
@higrys, @sprzedwojski
def send_slack_msg(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].\
        xcom_pull(task_ids='gcp_service_list_instances_{}'\
           .format(gcp_service[0]))
    ...

def send_slack_msg(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].
            xcom_pull(task_ids='gcp_service_list_instances_{}'
                        .format(gcp_service[0]))
        data = ...
    ...
    ...
```python
def send_slack_msg(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].
            xcom_pull(task_ids='gcp_service_list_instances_{}'
                .format(gcp_service[0]))
        data = ...
        requests.post(
            url=SLACK_WEBHOOK,
            data=json.dumps(data),
            headers={"Content-type": "application/json"}
        )
```
Prepare email

```python
prepare_email_task = PythonOperator(
    python_callable=prepare_email,
    provide_context=True,
    task_id='prepare_email_task',
    dag=dag
)
```
Prepare email

```
prepare_email_task = PythonOperator(
    python_callable=prepare_email,
    provide_context=True,
    task_id='prepare_email_task',
    dag=dag
)
```
def prepare_email(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].
                  xcom_pull(task_ids='gcp_service_list_instances_{}'
                                .format(gcp_service[0]))

    ...

    html_content = ...

    context['task_instance'].xcom_push(key='email', value=html_content)
def prepare_email(**context):
    for gcp_service in GCP_SERVICES:
        result = context['task_instance'].\
            xcom_pull(task_ids='gcp_service_list_instances_{}'\
                      .format(gcp_service[0]))
        ...
        html_content = ...
        context['task_instance'].xcom_push(key='email', value=html_content)
Send email

```python
send_email_task = EmailOperator(
    task_id='send_email',
    to='szyman.przedwojski@polidea.com',
    subject=INSTANCES_IN_PROJECT_TITLE,
    html_content=...,
    dag=dag
)
```
Send email

send_email_task = EmailOperator(
    task_id='send_email',
    to='szymon.przedwojski@polidea.com',
    subject=INSTANCES_IN_PROJECT_TITLE,
    html_content=
    "{{ task_instance.xcom_pull(task_ids='prepare_email_task', key='email') }}",
    dag=dag
)
Dependencies

```python
for gcp_service in GCP_SERVICES:
    bash_task = BashOperator(
        ...
    )
    bash_task >> send_slack_msg_task
    bash_task >> prepare_email_task
```
Dependencies

```python
for gcp_service in GCP_SERVICES:
    bash_task = BashOperator(
        ...
    )
    bash_task >> send_slack_msg_task
    bash_task >> prepare_email_task
    prepare_email_task >> send_email_task
```
for gcp_service in GCP_SERVICES:
    bash_task = BashOperator(
        ...
    )
    bash_task >> send_slack_msg_task
    bash_task >> prepare_email_task
prepare_email_task >> send_email_task
Demo

https://github.com/PolideaInternal/airflow-gcp-spy
Complex DAGs

![Real Pipelines of Clover](https://speakerdeck.com/pybay/2016-matt-davis-a-practical-introduction-to-airflow?slide=13)

Complex, Manageable, DAGs
Airflow's General Architecture

Workers

Web UI

Scheduler/Executor

Webserver

Metadata Database

Task Execution Logs

Source: https://medium.com/@dustinstansbury/understanding-apache-airflows-key-concepts-a96efed52b1a
Celery Executor

Controller
- Web server
- Scheduler
- RDBMS
- DAGs

Celery Broker
- RabbitMQ/Redis/AmazonSQS

Node 1
- Worker
- DAGs
- Sync files (Chef/Puppet/Ansible/NFS)

Node 2
- Worker
- DAGs
- Sync files (Chef/Puppet/Ansible/NFS)
(Beta): Kubernetes Executor

Controller

Scheduler

Web server

RDBMS

Scheduler

Node 1

Pod

Pod

Node 2

Pod

Pod

DAGs

Sync files

- Git Init
- Persistent Volume
- Baked-in (future)

Kubernetes Cluster

Kubernetes Master

Controller

Scheduler

Web server

RDBMS

DAGs

Sync files

- Git Init
- Persistent Volume
- Baked-in (future)
GCP - Composer

https://github.com/GoogleCloudPlatform/airflow-operator
Thank You!

Follow us @ polidea.com/blog
We’re hiring!
Questions? :)

Follow us @ polidea.com/blog