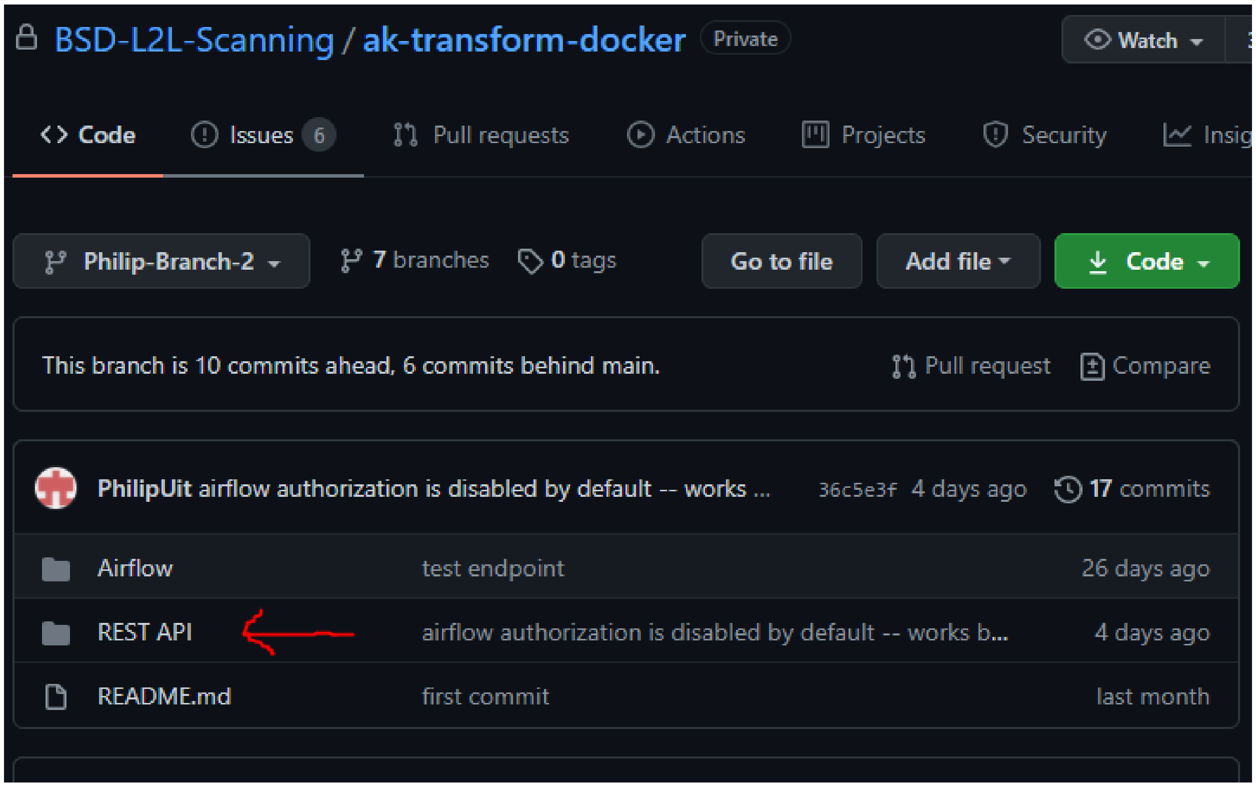
README

Documentation for ak-docker-transform REST API (for Apache Airflow scheduler)

We will be using this REST API folder from Philip-Branch-2 in this example:

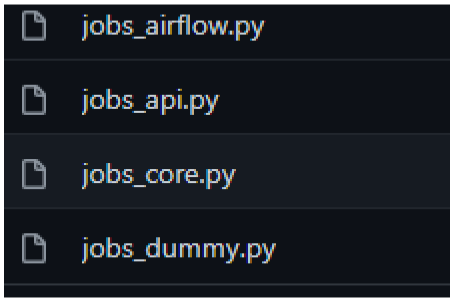


This branch is 10 commits ahead, 6 commits behind main.

PhilipUit airflow authorization is disabled by default -- works ... 36c5e3f 4 days ago 17 commits

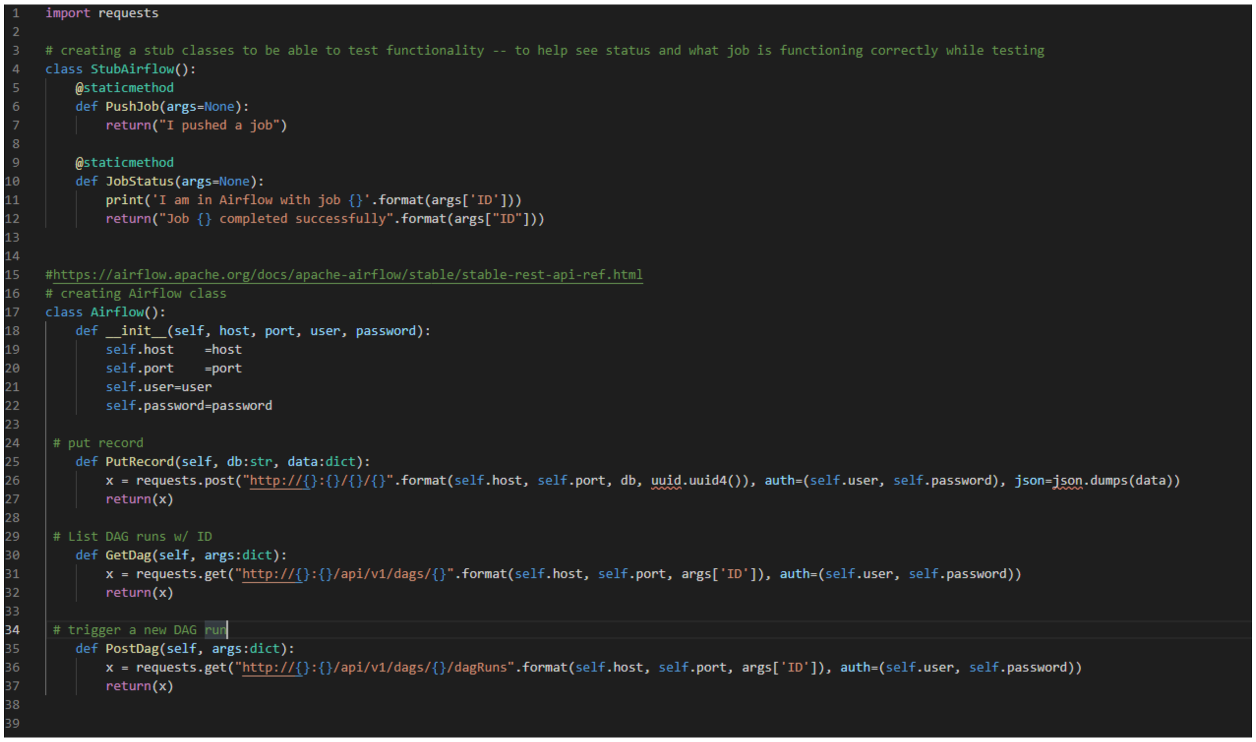
Airflow	test endpoint	26 days ago
REST API	airflow authorization is disabled by default -- works b...	4 days ago
README.md	first commit	last month

We will discuss these 4 files in more detail below (as they are primary code for this section):



jobs_airflow.py
jobs_api.py
jobs_core.py
jobs_dummy.py

1. **jobs_airflow.py**

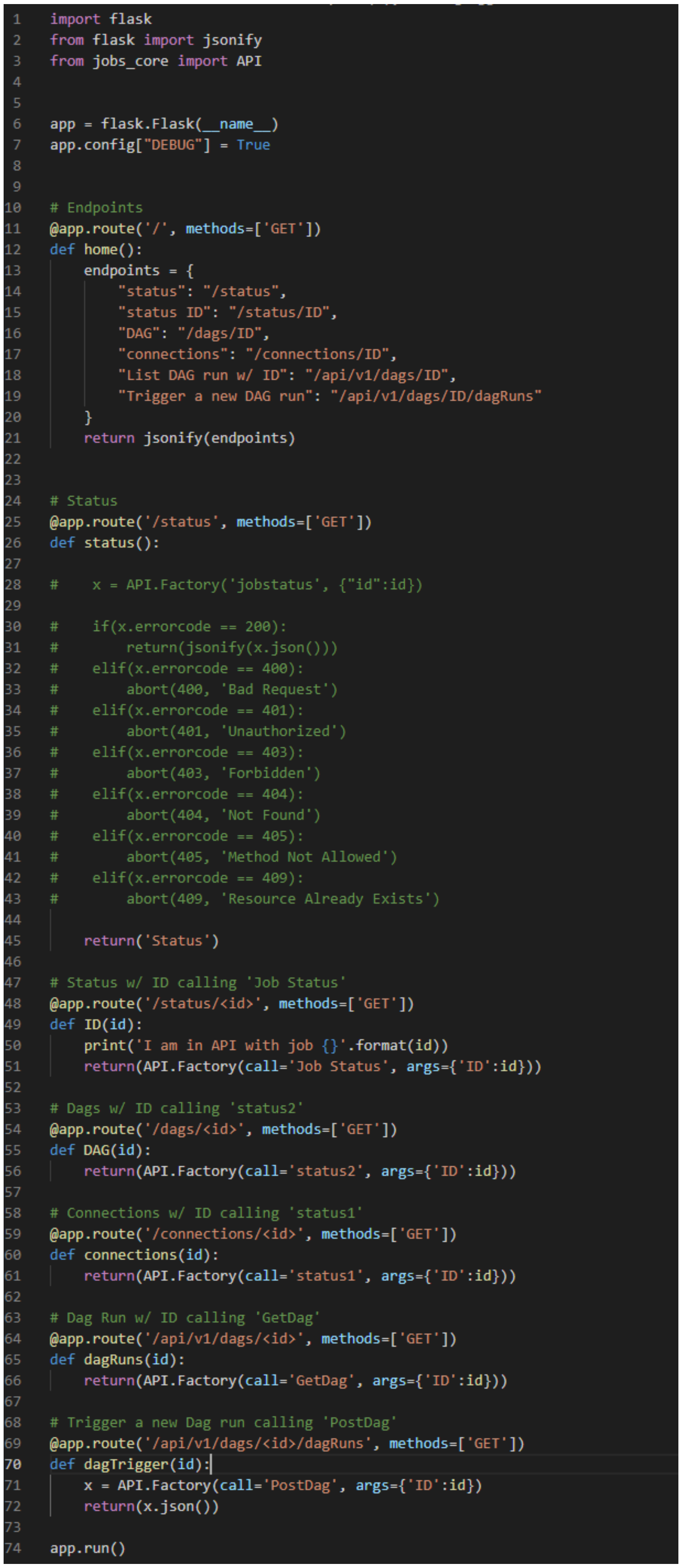


```
1   import requests
2
3   # creating a stub classes to be able to test functionality -- to help see status and what job is functioning correctly while testing
4   class StubAirflow():
5       @staticmethod
6       def PushJob(args=None):
7           return("I pushed a job")
8
9       @staticmethod
10      def JobStatus(args=None):
11          print('I am in Airflow with job {}'.format(args['ID']))
12          return("Job {} completed successfully".format(args["ID"]))
13
14
15  #https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html
16  # creating Airflow class
17  class Airflow():
18      def __init__(self, host, port, user, password):
19          self.host    =host
20          self.port    =port
21          self.user=user
22          self.password=password
23
24   # put record
25      def PutRecord(self, db:str, data:dict):
26          x = requests.post("http://{}:{}/{}/{}".format(self.host, self.port, db, uuid.uuid4()), auth=(self.user, self.password), json=json.dumps(data))
27          return(x)
28
29   # List DAG runs w/ ID
30      def GetDag(self, args:dict):
31          x = requests.get("http://{}:{}/api/v1/dags/{}".format(self.host, self.port, args['ID']), auth=(self.user, self.password))
32          return(x)
33
34   # trigger a new DAG run
35      def PostDag(self, args:dict):
36          x = requests.get("http://{}:{}/api/v1/dags/{}/dagRuns".format(self.host, self.port, args['ID']), auth=(self.user, self.password))
37          return(x)
38
39
```

- Lines 3-12 demonstrate a stubbed class to be able run static method used in PushJob & JobStatus. This will populate in the terminal as we run them, shown below in this documentation.
- Lines 15-22 create a class, Airflow, and establishes how to enter host, port, user, and password required.
- Currently lines 24-27 are not passing through to an active endpoint
- Using the Apache Airflow documentation: https://airflow.apache.org/docs/apache-airflow/stable/stable-rest-api-ref.html -- we see in Lines 29-37 that we are able to connect endpoints using "/api/v1/dags/{}" & "/api/v1/dags/{}/dagRuns". We will demonstrate how to connect to these endpoints using the browser below.

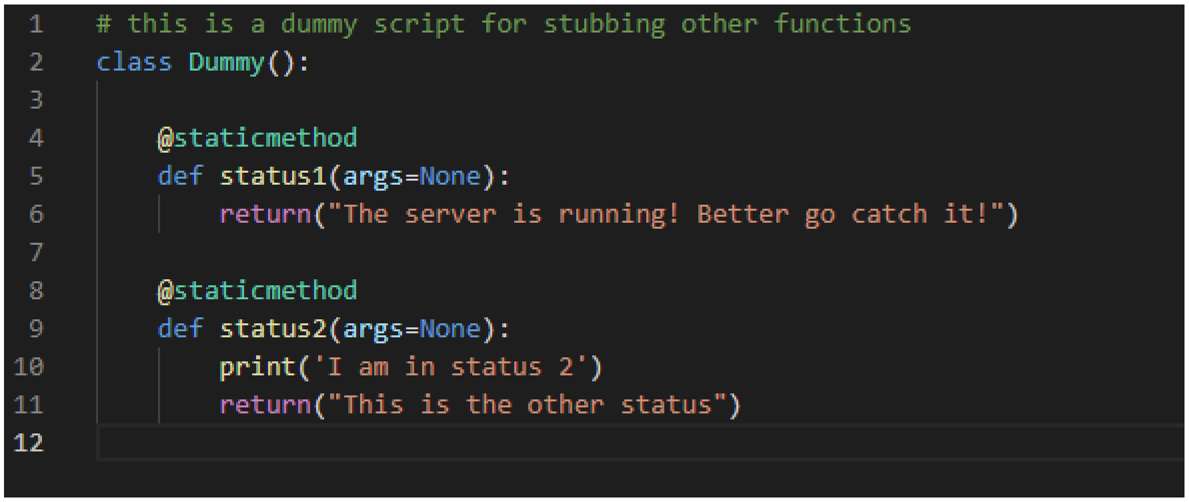
2. jobs_api.py

- We start by importing packages in lines 1-3
- Lines 6-7 are setting up flask
- Lines 10-21 are the active endpoints. Note some are for testing purposes.
- -Lines 24-45 show the error codes that should populate given the status of running the endpoints. They are still commented out and not implemented.
- Lines 47-61 demonstrate the different tests of endpoints using the ID and various status/connections.
- Lines 63-66 demonstrate a Dag Run w/ ID calling ‘GetDag’. This is a successful endpoint connecting to Apache Airflow.
- Lines 68-72 demonstrate Triggering a Dag Run calling ‘PostDag’. This is a successful endpoint connecting to Apache Airflow.
- Line 74 runs the application.



```
1   import flask
2   from flask import jsonify
3   from jobs_core import API
4
5
6   app = flask.Flask(__name__)
7   app.config["DEBUG"] = True
8
9
10  # Endpoints
11  @app.route('/', methods=['GET'])
12  def home():
13      endpoints = {
14          "status": "/status",
15          "status ID": "/status/ID",
16          "DAG": "/dags/ID",
17          "connections": "/connections/ID",
18          "List DAG run w/ ID": "/api/v1/dags/ID",
19          "Trigger a new DAG run": "/api/v1/dags/ID/dagRuns"
20      }
21      return jsonify(endpoints)
22
23
24  # Status
25  @app.route('/status', methods=['GET'])
26  def status():
27
28  #    x = API.Factory('jobstatus', {"id":id})
29
30  #    if(x.errorcode == 200):
31  #        return(jsonify(x.json()))
32  #    elif(x.errorcode == 400):
33  #        abort(400, 'Bad Request')
34  #    elif(x.errorcode == 401):
35  #        abort(401, 'Unauthorized')
36  #    elif(x.errorcode == 403):
37  #        abort(403, 'Forbidden')
38  #    elif(x.errorcode == 404):
39  #        abort(404, 'Not Found')
40  #    elif(x.errorcode == 405):
41  #        abort(405, 'Method Not Allowed')
42  #    elif(x.errorcode == 409):
43  #        abort(409, 'Resource Already Exists')
44
45      return('Status')
46
47  # Status w/ ID calling 'Job Status'
48  @app.route('/status/<id>', methods=['GET'])
49  def ID(id):
50      print('I am in API with job {}'.format(id))
51      return(API.Factory(call='Job Status', args={'ID':id}))
52
53  # Dags w/ ID calling 'status2'
54  @app.route('/dags/<id>', methods=['GET'])
55  def DAG(id):
56      return(API.Factory(call='status2', args={'ID':id}))
57
58  # Connections w/ ID calling 'status1'
59  @app.route('/connections/<id>', methods=['GET'])
60  def connections(id):
61      return(API.Factory(call='status1', args={'ID':id}))
62
63  # Dag Run w/ ID calling 'GetDag'
64  @app.route('/api/v1/dags/<id>', methods=['GET'])
65  def dagRuns(id):
66      return(API.Factory(call='GetDag', args={'ID':id}))
67
68  # Trigger a new Dag run calling 'PostDag'
69  @app.route('/api/v1/dags/<id>/dagRuns', methods=['GET'])
70  def dagTrigger(id):
71      x = API.Factory(call='PostDag', args={'ID':id})
72      return(x.json())
73
74  app.run()
```

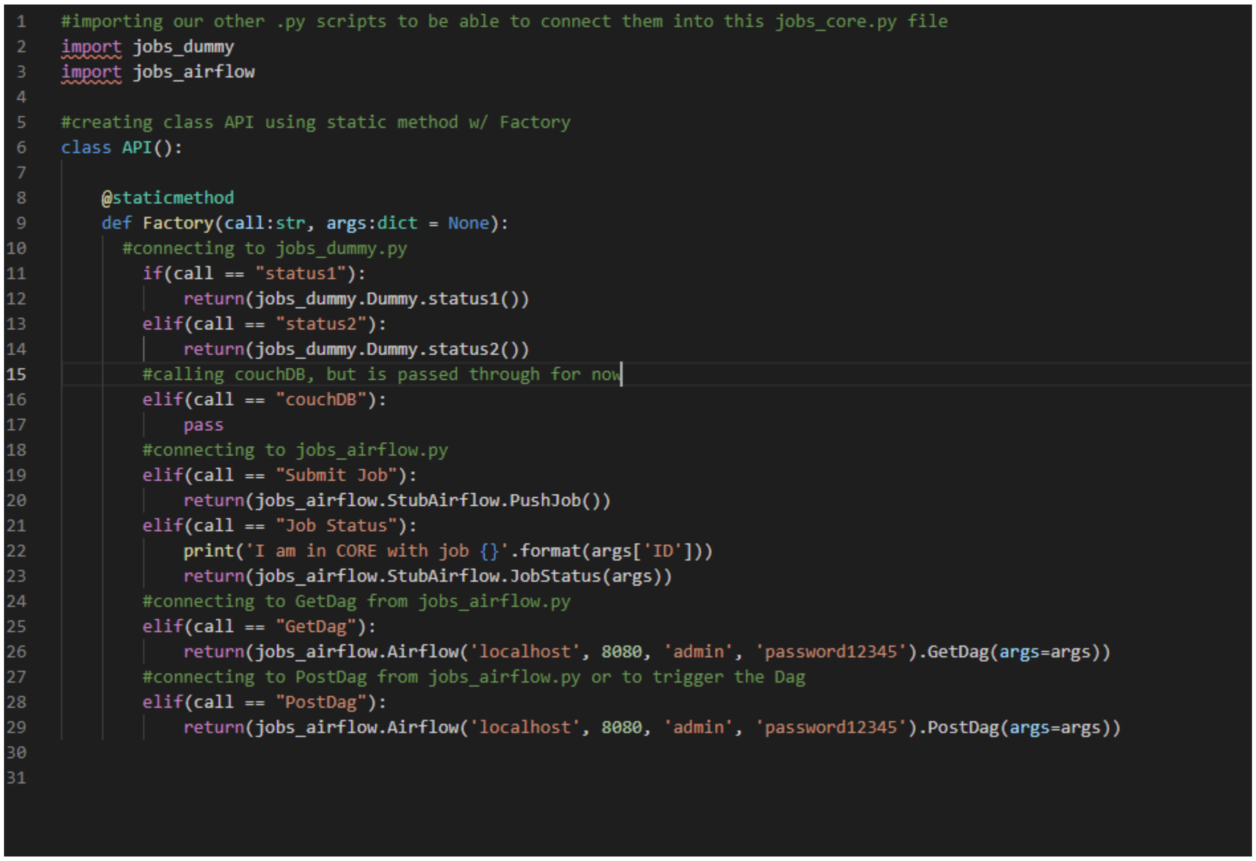
3. **jobs_dummy.py**



```python
# this is a dummy script for stubbing other functions
class Dummy():

    @staticmethod
    def status1(args=None):
        return("The server is running! Better go catch it!")

    @staticmethod
    def status2(args=None):
        print('I am in status 2')
        return("This is the other status")

```

-this script is used to pass static methods for ‘status1’ and ‘status2’ through to the other scripts. This is used to demonstrate our test endpoints and to verify things work during testing.

4. jobs_core.py

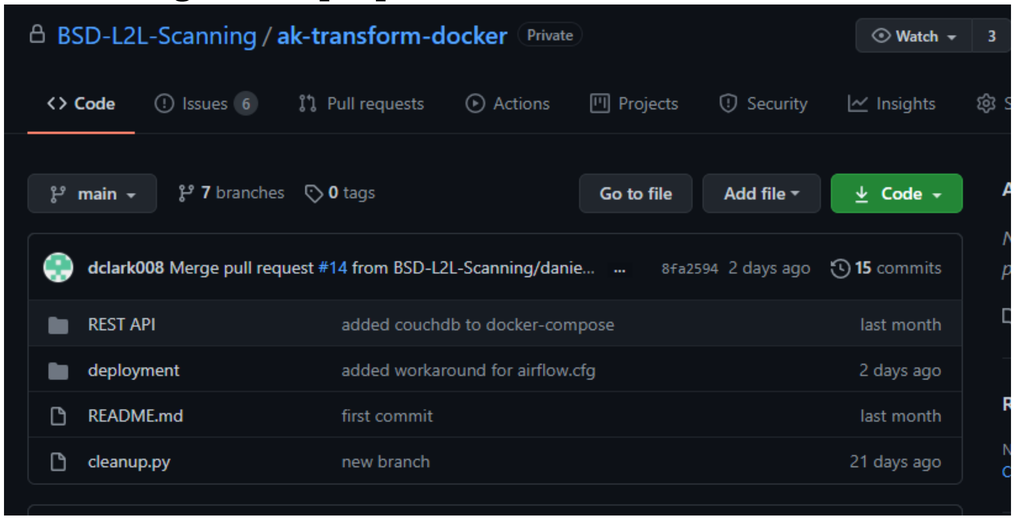


```
1   #importing our other .py scripts to be able to connect them into this jobs_core.py file
2   import jobs_dummy
3   import jobs_airflow
4
5   #creating class API using static method w/ Factory
6   class API():
7
8       @staticmethod
9       def Factory(call:str, args:dict = None):
10        #connecting to jobs_dummy.py
11          if(call == "status1"):
12              return(jobs_dummy.Dummy.status1())
13          elif(call == "status2"):
14              return(jobs_dummy.Dummy.status2())
15          #calling couchDB, but is passed through for now
16          elif(call == "couchDB"):
17              pass
18          #connecting to jobs_airflow.py
19          elif(call == "Submit Job"):
20              return(jobs_airflow.StubAirflow.PushJob())
21          elif(call == "Job Status"):
22              print('I am in CORE with job {}'.format(args['ID']))
23              return(jobs_airflow.StubAirflow.JobStatus(args))
24          #connecting to GetDag from jobs_airflow.py
25          elif(call == "GetDag"):
26              return(jobs_airflow.Airflow('localhost', 8080, 'admin', 'password12345').GetDag(args=args))
27          #connecting to PostDag from jobs_airflow.py or to trigger the Dag
28          elif(call == "PostDag"):
29              return(jobs_airflow.Airflow('localhost', 8080, 'admin', 'password12345').PostDag(args=args))
30
31
```

- we start by importing the jobs_dummy.py and jobs_airflow.py scripts (demonstrated above) in lines 1 & 2
- using the static Factory method we generate to make calls to connect to in lines 6-29
- Lines 11-14 connect to jobs_dummy.py and link to Status1 & Status2
- Line 15-17 passes the CouchDB, but is not implemented yet
- Lines 19-29 connect to the jobs_airlfow.py file passing through calls for 'Submit Job' 'Job Status' 'GetDag' and 'PostDag' --- please note that the 'GetDag' and 'PostDag' are able to connect to Apache Airflow endpoints.

How to stand-up Deployment Docker

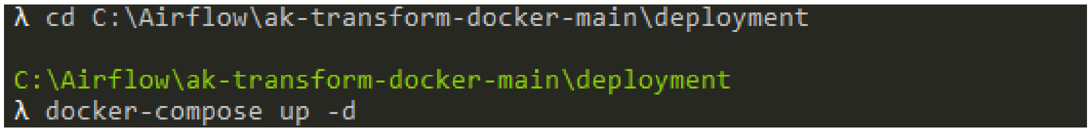
Be sure to go to the proper ak-transform-docker-main:



dclark008 Merge pull request #14 from BSD-L2L-Scanning/danie... … 8fa2594 2 days ago 15 commits

REST API	added couchdb to docker-compose	last month
deployment	added workaround for airflow.cfg	2 days ago
README.md	first commit	last month
cleanup.py	new branch	21 days ago

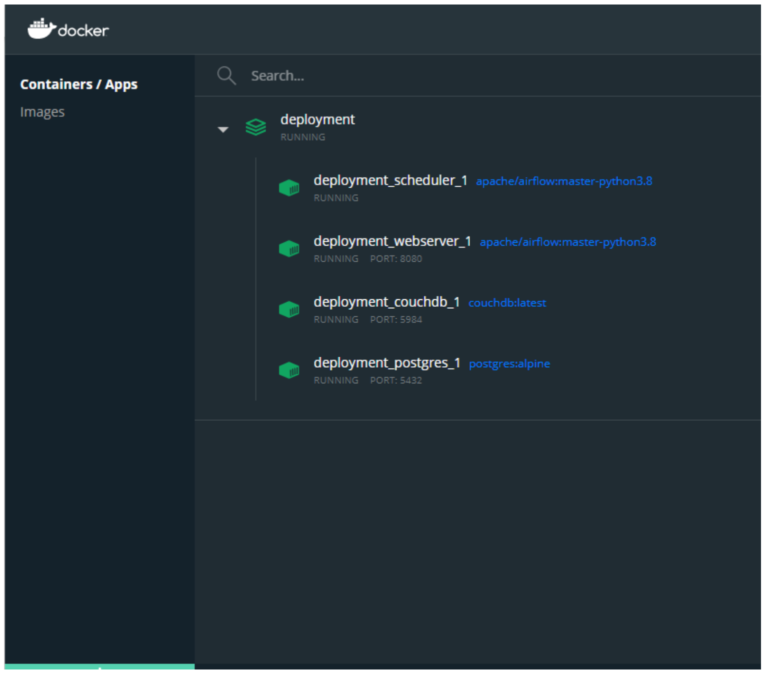
Next, cd into the correct folder:



```
λ cd C:\Airflow\ak-transform-docker-main\deployment

C:\Airflow\ak-transform-docker-main\deployment
λ docker-compose up -d
```

Run >> **docker-compose up -d**

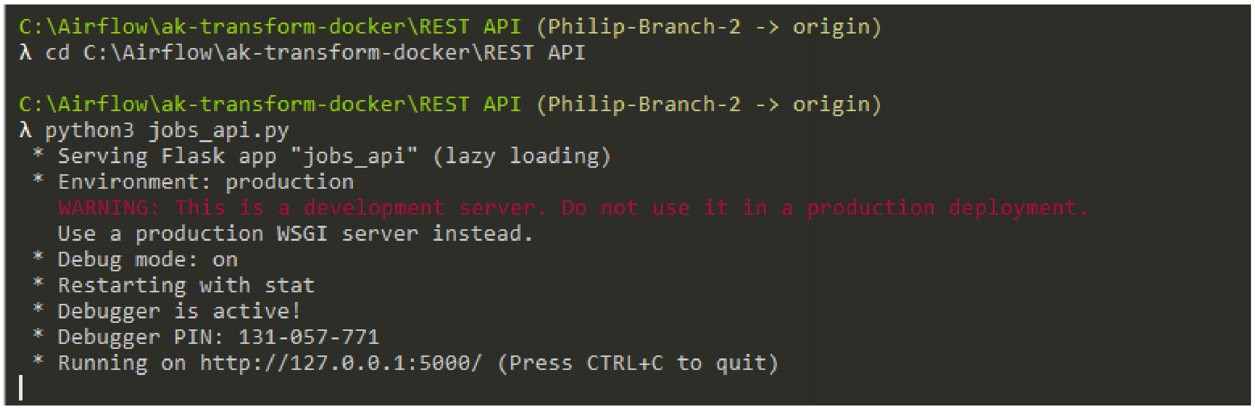
Here we see the Docker container is properly stood-up:



deployment RUNNING
deployment_scheduler_1 apache/airflow:master-python3.8 RUNNING
deployment_webserver_1 apache/airflow:master-python3.8 RUNNING PORT: 8080
deployment_couchdb_1 couchdb:latest RUNNING PORT: 5984
deployment_postgres_1 postgres:alpine RUNNING PORT: 5432

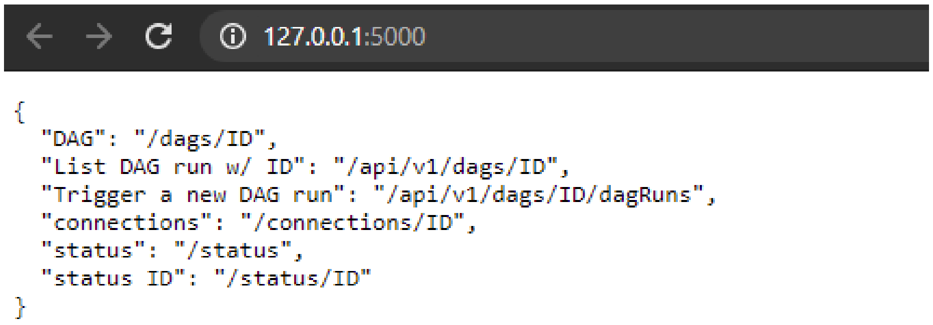
How to test endpoints

Make sure you are in the REST API folder of Philip-Branch-2 using cd:
Then run >> python3 jobs_api.py



```
C:\Airflow\ak-transform-docker\REST API (Philip-Branch-2 -> origin)
λ cd C:\Airflow\ak-transform-docker\REST API

C:\Airflow\ak-transform-docker\REST API (Philip-Branch-2 -> origin)
λ python3 jobs_api.py
 * Serving Flask app "jobs_api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 131-057-771
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

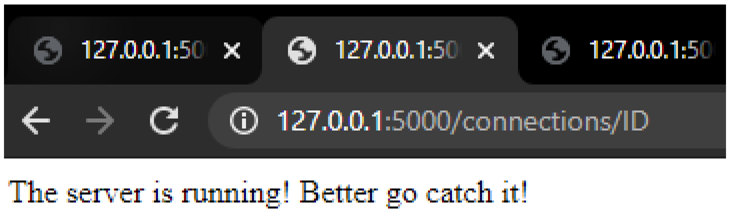
Clicking on the url: http://127.0.0.1:5000/ brings us to our endpoints:



127.0.0.1:5000

```
{
  "DAG": "/dags/ID",
  "List DAG run w/ ID": "/api/v1/dags/ID",
  "Trigger a new DAG run": "/api/v1/dags/ID/dagRuns",
  "connections": "/connections/ID",
  "status": "/status",
  "status ID": "/status/ID"
}
```

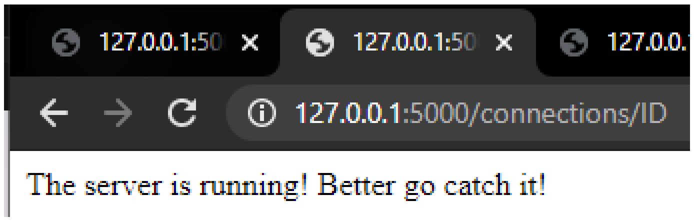
Here we demonstrate running these endpoints in the browser.



127.0.0.1:5000/connections/ID

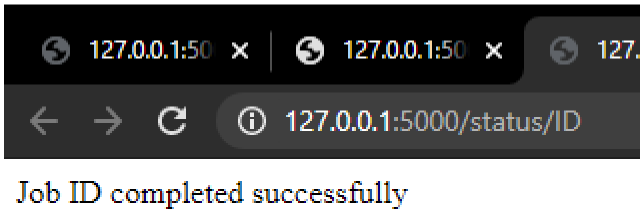
The server is running! Better go catch it!

http://127.0.0.1:5000/connections/ID



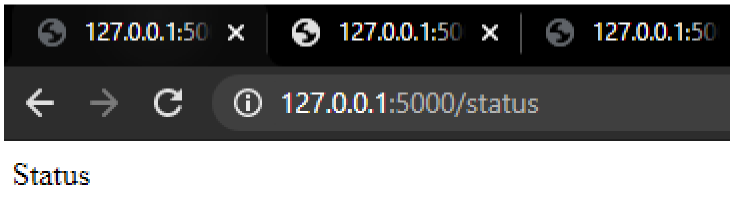
The server is running! Better go catch it!

http://127.0.0.1:5000/status/ID



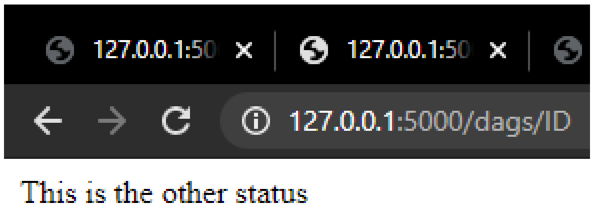
Job ID completed successfully

http://127.0.0.1:5000/status



Status

http://127.0.0.1:5000/dags/ID

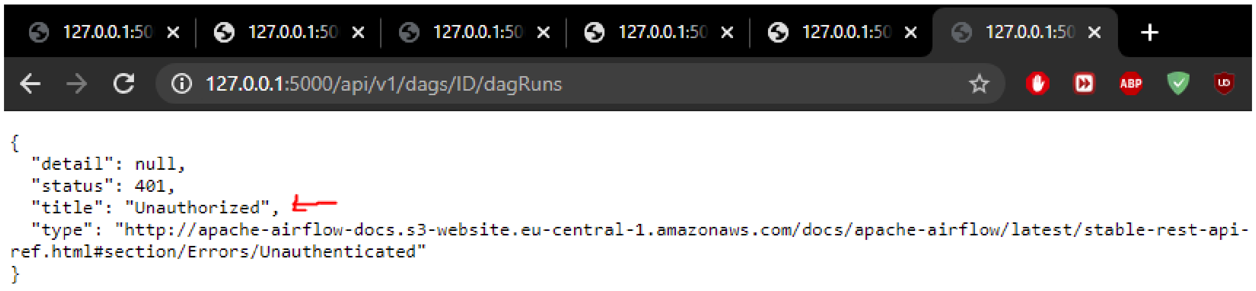


This is the other status

Connecting Endpoint to Airflow

This endpoint allows us to connect to Airflow. Due to Airflow being complicated, it shows as ‘unauthorized’, but we are connected to it.

http://127.0.0.1:5000/api/v1/dags/ID/dagRuns



```
{
  "detail": null,
  "status": 401,
  "title": "Unauthorized",
  "type": "http://apache-airflow-docs.s3-website.eu-central-1.amazonaws.com/docs/apache-airflow/latest/stable-rest-api-ref.html#section/Errors/Unauthenticated"
}
```

Problem:

- This is not unauthorized due to Airflow being difficult
- it works but is not connected

Solution:

- Airflow can be tricky and challenging here
- Download updated deployment folder over current one if not updated
- Create new entrypoint.sh file with the line >> `cp scripts/airflow.cfg airflow.cfg` and the new airflow.cfg file in the scripts folder