Airflow and MySQL with docker containers

Rajat Biswas · Follow

4 min read · Apr 24, 2020

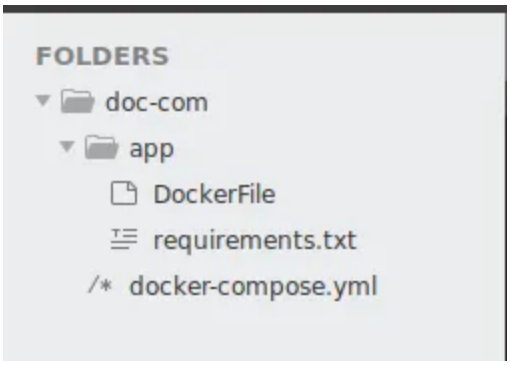
I wanted to create two containers one for airflow and another for MySQL DB. A dag in the airflow container would hit an API to get the response data and save it in the MySQL DB.

For storing the data from airflow container into the MySQL container I needed a way so that the two containers would connect with each other. The containers can’t connect to each other if we just define then as services in the docker-compose file. Here comes the network to the rescue.

Using network we can create a bridge connection between the containers, and it will provide us with an IP that we can use in the connection in airflow.

Another important thing is that at first I used the latest version of the two images but it gave me an error. Later on, going through the internet I found that there is a problem with the MySQL latest version and thus I used version 5.7 for MySQL image as suggested and it worked.

Below is my folder structure:



folder structure

Below is the content of DockerFile:

```
WORKDIR /work_dir

COPY app/requirements.txt /work_dir/
RUN pip install -r requirements.txt
```

Below is the content of requirements.txt:

I have wanted to test how the importing library works in dockers, that's why I did it. You can remove or add other libraries if you need them.

```
pandas==1.0.2
```

Below is the content of the docker-compose.yml file:

```
version: "3"

services:
  webserver:
    build:
      context: ./app/
      dockerfile: DockerFile
    image: puckel/docker-airflow:latest
    container_name: test_container_test
    volumes:
      - /home/ubuntu/dags1/:/usr/local/airflow/dags
    ports:
      - 8080:8080
    networks:
      - backend
    restart: always
    depends_on:
      - mysqldb
  mysqldb:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: helloworld
      MYSQL_DATABASE: testdb
    container_name: mysql_container_test
    ports:
      - 3306:3306
    networks:
      - backend
networks:
  backend:
    driver: "bridge"
```

After runing the ```docker-compose up``` command, it will pull the defined images from hub and will create a bridge network for us.

Here’s a look at the containers

```
ubuntu@ubuntu:~/doc-com$ sudo docker ps
[sudo] password for ubuntu:
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                                        NAMES
aa369e74447d        puckel/docker-airflow:latest   "/entrypoint.sh webs…"   7 minutes ago       Up 7 minutes        5555/tcp, 8793/tcp, 0.0.0.0:8080->8080/tcp   test_container_test
fd5abd402a41        mysql:5.7                      "docker-entrypoint.s…"   7 minutes ago       Up 7 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp            mysql_container_test
ubuntu@ubuntu:~/doc-com$
```

containers

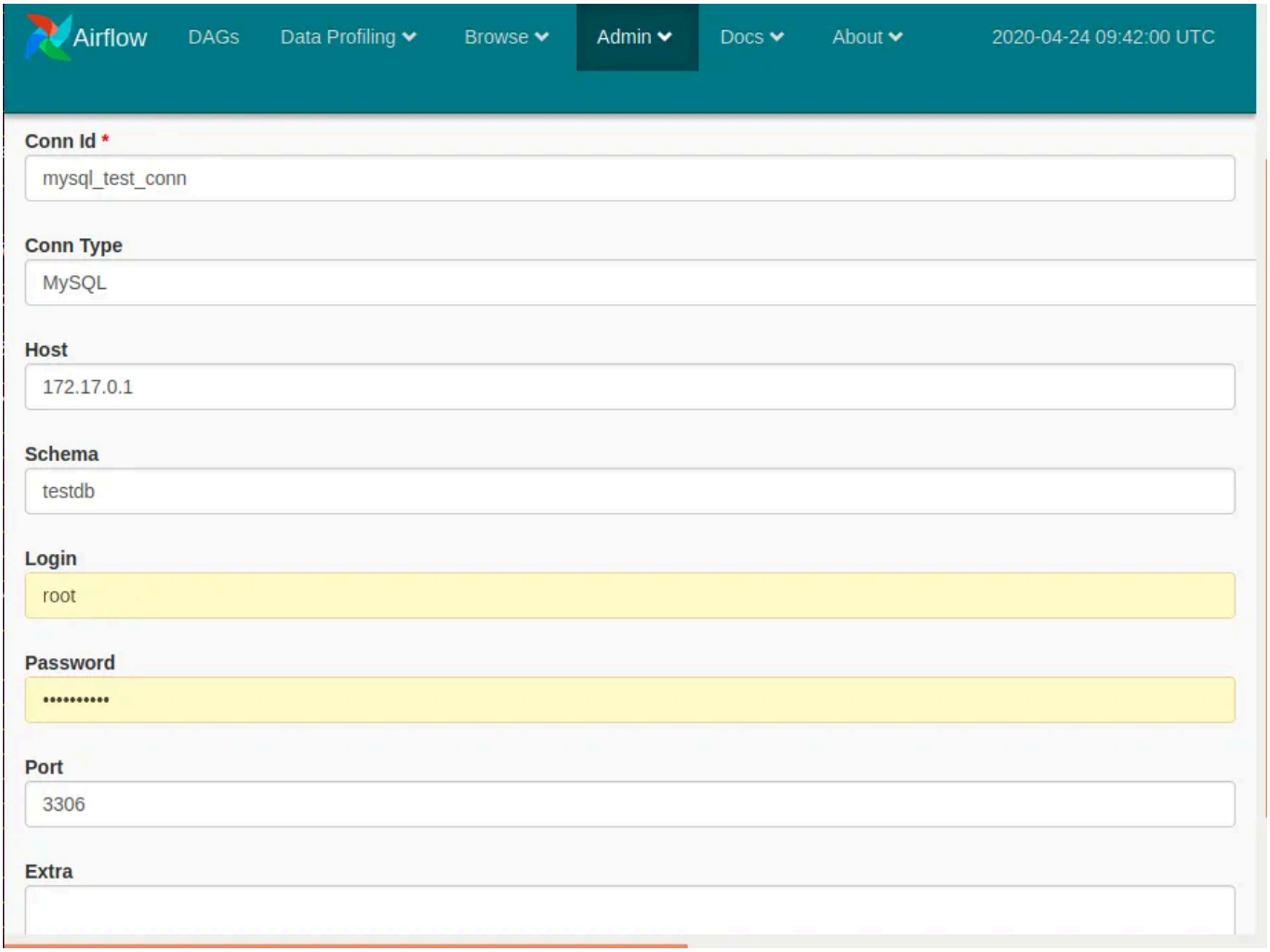
We can see the details of the bridge network with ```docker inspect <network_name>```

The response of the above command on my pc gives me:

```
[
    {
        "Name": "bridge",
        "Id":
"8474d0c9e5bdceae170c8cee345739d94e2a026a144f85e60b31428e58d9cc37",
        "Created": "2020-04-23T22:51:53.133841017-07:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
```

```
                "com.docker.network.bridge.enable_ip_masquerade": "true",
                "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
                "com.docker.network.bridge.name": "docker0",
                "com.docker.network.driver.mtu": "1500"
            },
            "Labels": {}
        }
    ]
```


We will then use the IP to create a connection in Airflow using the database and password that we defined in docker-compose.yml file.



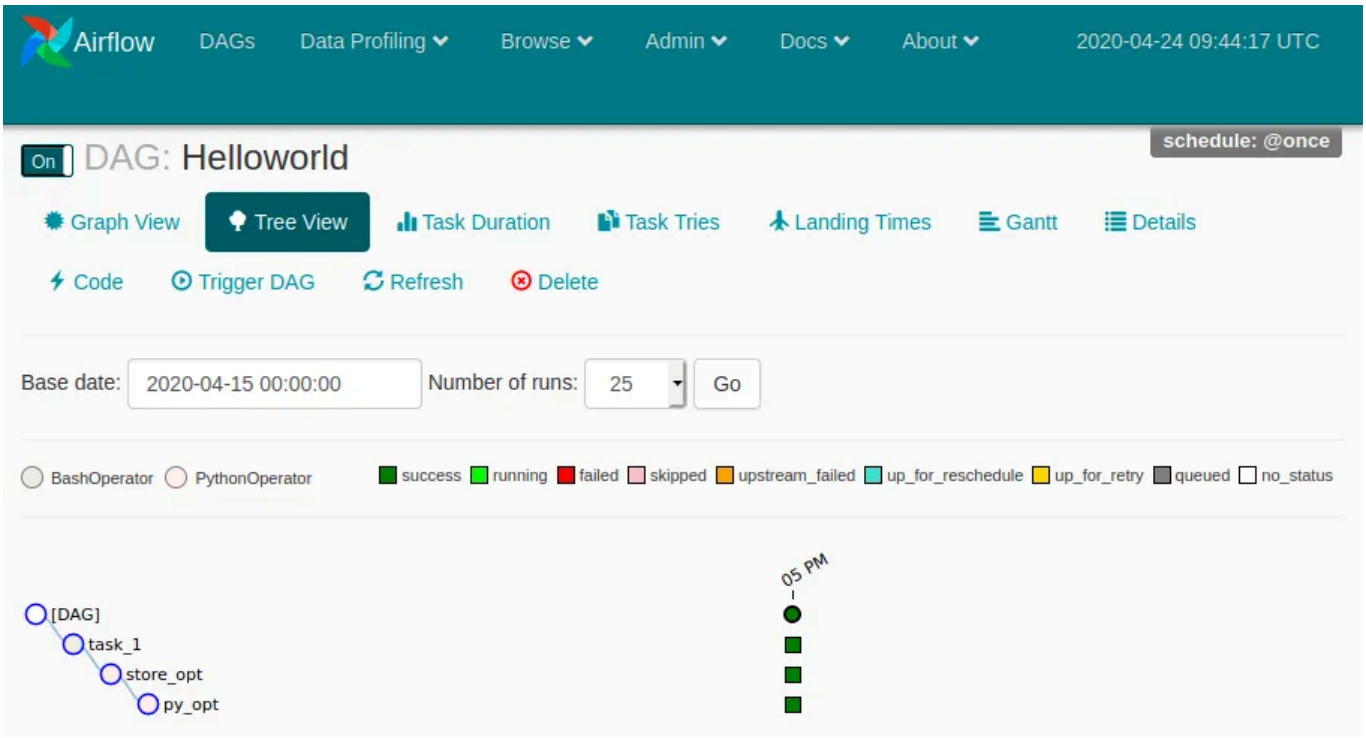
airflow connection

My dag looks like this:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.operators.python_operator import PythonOperator
from datetime import datetime, timedelta
from airflow.hooks.mysql_hook import MySqlHook
from airflow.operators.mysql_operator import MySqlOperator
import pandas as pd
import requests
import json

default_args = {
    'owner': 'user',
    'depends_on_past': False,
    'start_date': datetime(2020, 4, 15),
    'email': ['user@mail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}

dag = DAG('Helloworld', schedule_interval="@once",
default_args=default_args)

def get_data(**kwargs):
    url="https://jsonplaceholder.typicode.com/albums"
    resp = requests.get(url)
    if resp.status_code ==200:
        res = resp.json()
        return res
    return -1

def save_db(**kwargs):
    query = 'select * from test_table'
    mysql_hook = MySqlHook(mysql_conn_id='mysql_test_conn',
schema='testdb')
    connection = mysql_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    for result in results:
        print("*********", result)

def check_table_exists(**kwargs):
    query = 'select count(*) from information_schema.tables where
table_name="test_table"'
    mysql_hook = MySqlHook(mysql_conn_id='mysql_test_conn',
schema='testdb')
```

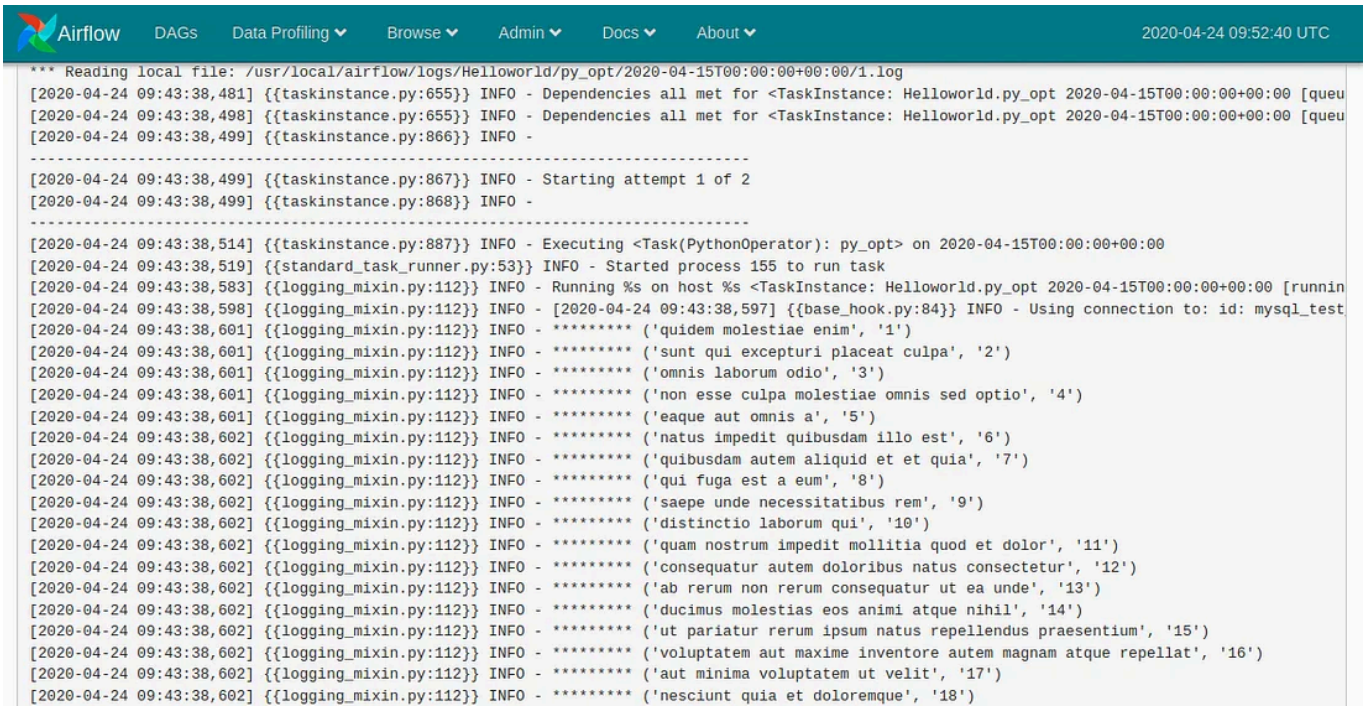
```
    connection = mysql_hook.get_conn()
    cursor = connection.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    return results

def store_data(**kwargs):
    res = get_data()
    table_status = check_table_exists()
    mysql_hook = MySqlHook(mysql_conn_id='mysql_test_conn',
schema='testdb')
    if table_status[0][0] == 0:
        print("----- table does not exists, creating it")
        create_sql = 'create table test_table(col1 varchar(100), col2
varchar(100))'
        mysql_hook.run(create_sql)
    else:
        print("----- table already exists")
    if res != -1:
        for data in res:
            sql = 'insert into test_table values (%s, %s)'
            mysql_hook.run(sql, parameters=(data.get("title"),
data.get("id")))

py = PythonOperator(
    task_id='py_opt',
    dag=dag,
    python_callable=save_db,
)

py1 = PythonOperator(
    task_id='store_opt',
    dag=dag,
    python_callable=store_data,
)

t1 = BashOperator(
    task_id='task_1',
    bash_command='echo "Hello World from Task 1"',
    dag=dag)

t1 >> py1 >> py
```

After triggering the dag the status can be seen below



dag status

The response from the final task of my dag:



dag py_opt output

We can see that indeed the data has been saved in MySQL container

```
ubuntu@ubuntu:~/doc-com$ sudo docker exec -ti fd5abd402a41 bash
root@fd5abd402a41:/# mysql -uroot -phelloworld
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 105
Server version: 5.7.29 MySQL Community Server (GPL)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use testdb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from test_table;
+-------------------------------------------------------------------------------+------+
| col1                                                                          | col2 |
+-------------------------------------------------------------------------------+------+
| quidem molestiae enim                                                         |    1 |
| sunt qui excepturi placeat culpa                                              |    2 |
| omnis laborum odio                                                            |    3 |
| non esse culpa molestiae omnis sed optio                                      |    4 |
| eaque aut omnis a                                                             |    5 |
| natus impedit quibusdam illo est                                              |    6 |
| quibusdam autem aliquid et et quia                                            |    7 |
| qui fuga est a eum                                                            |    8 |
| saepe unde necessitatibus rem                                                 |    9 |
| distinctio laborum qui                                                        |   10 |
| quam nostrum impedit mollitia quod et dolor                                   |   11 |
| consequatur autem doloribus natus consectetur                                 |   12 |
| ab rerum non rerum consequatur ut ea unde                                     |   13 |
| ducimus molestias eos animi atque nihil                                       |   14 |
| ut pariatur rerum ipsum natus repellendus praesentium                         |   15 |
| voluptatem aut maxime inventore autem magnam atque repellat                   |   16 |
| aut minima voluptatem ut velit                                                |   17 |
| nesciunt quia et doloremque                                                   |   18 |
| velit pariatur quaerat similique libero omnis quia                            |   19 |
| voluptas rerum iure ut enim                                                   |   20 |
| repudiandae voluptatem optio est consequatur rem in temporibus et             |   21 |
| et rem non provident vel ut                                                   |   22 |
| incidunt quisquam hic adipisci sequi                                          |   23 |
| dolores ut et facere placeat                                                  |   24 |
| vero maxime id possimus sunt neque et consequatur                             |   25 |
```

MySQL container output

I hope it helps anyone who is looking into a way to connect two containers.

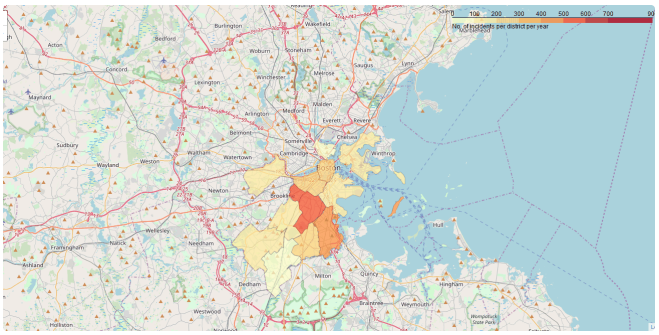
Airflow Docker MySQL

Written by Rajat Biswas

2 Followers

Follow

More from Rajat Biswas



Rajat Biswas

Creating GIFs of Choropleth Maps

This is my first article here on medium. I used the Boston Crime dataset from kaggle.

7 min read · Jan 26, 2020

11 1

Rajat Biswas

A gentle intro to plotly express

This is the second story with the same Boston Crime data from Kaggle that I used in my firs...

5 min read · Feb 13, 2020

See all from Rajat Biswas

Recommended from Medium

Kerry Wang

Building a Simple ETL with Airflow, PostgreSQL, and Docker

A Practical Guide to Building an ETL Pipeline with Airflow, PostgreSQL, and Docker

13 min read · Nov 21, 2023

53

Data Science - Social Media - Mechanical

Airflow Series: 2. Setup Airflow database

The series is the part of the last post. In the post before, I have failed in setup airflow wit...

2 min read · Sep 4, 2023

5

Lists

Coding & Development

11 stories · 480 saves

Natural Language Processing

1253 stories · 733 saves

Prithvijit Guha

Hello World, Airflow + Docker

An introductory tutorial on setting up a local Airflow instance with Docker. Also optionally...

8 min read · Oct 20, 2023

33

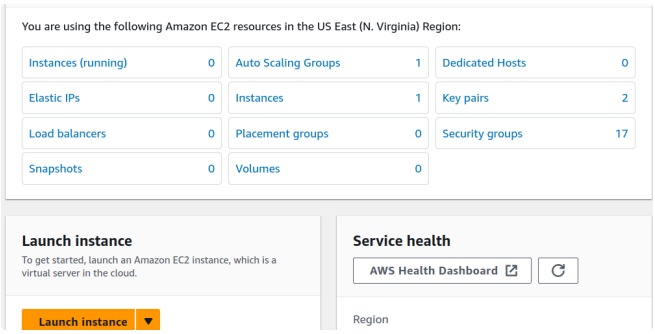
Davor Borcic

Creating Snowflake Data Pipelines in Airflow

Basic concepts behind Airflow and data pipelines

15 min read · Nov 2, 2023

65



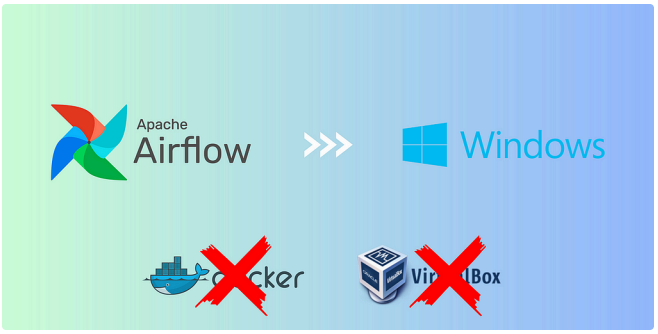
Praveen NG in Dev Genius

How to Install Apache Airflow on AWS EC2 Instance?

This post shows how Airflow can be installed on an EC2 instance. It involves four steps.

6 min read · Oct 16, 2023

64



VivekR

How to easily install Apache Airflow on Windows?

Install Apache Airflow on Windows easily without Docker or VirtualBox

3 min read · Sep 13, 2023

65 1