Building a Spark and Airflow development environment with Docker

A brief guide on how to set up a development environment with Spark, Airflow and Jupyter Notebook

Thiago Cordon May 1, 2020 · 6 min read ★

Photo by Christopher Gower on Unsplash

Brief context

As a Data Engineer, it is common to use in our daily routine the Apache Spark and Apache Airflow (if you do not yet use them, you should try) to overcome typical Data Engineering challenges like build pipelines to get data from someplace, do a lot of transformations and deliver it in another place.

In this article, I will share a guide on how to create a Data Engineering development environment containing a Spark Standalone Cluster, an Airflow server and a Jupyter Notebook instance.

In the Data Engineering context, Spark acts as the tool to process data (whatever you can think as data processing), Airflow as the orchestration tool to build pipelines and Jupyter Notebook to interactively develop Spark applications.

Motivation

Think how amazing it would be if you could develop and test Spark applications integrated with Airflow pipelines using your machine, without the necessity to wait someone give you access to a development environment or having to share server resources with others using the same development environment or even to wait this environment be created if it does not exist yet.

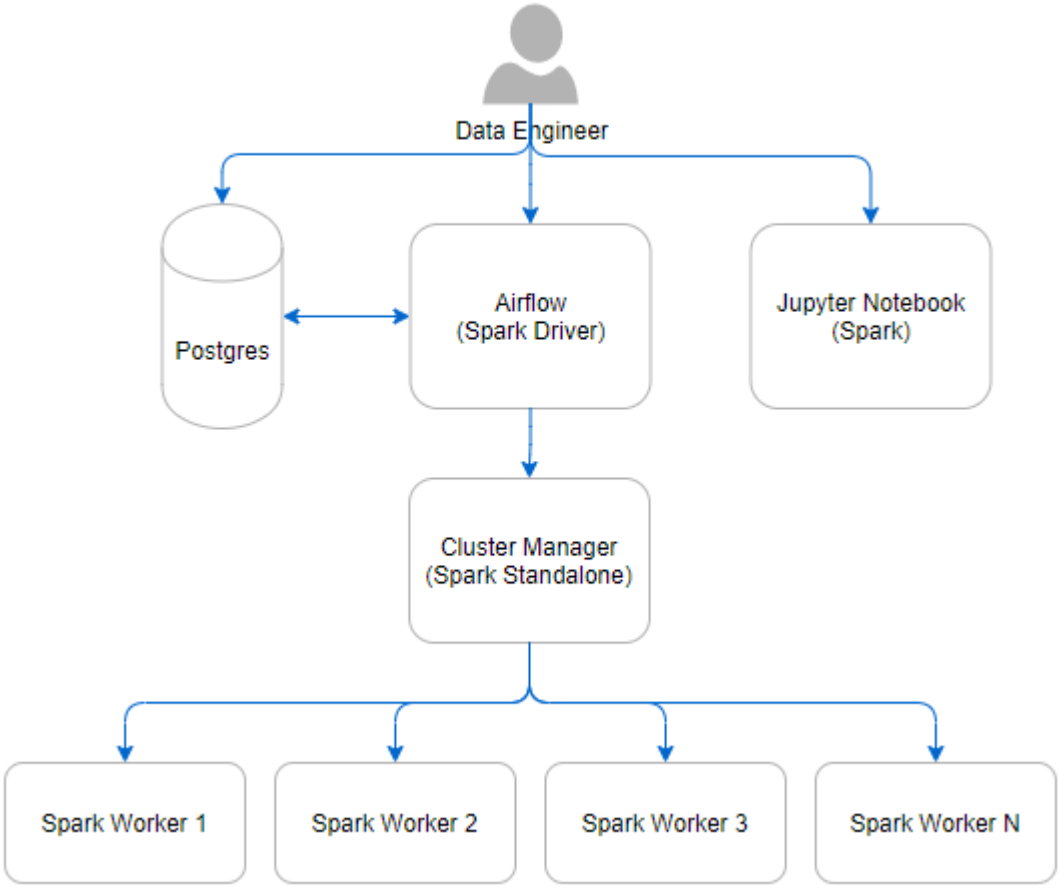
Thinking about this, I started to search how I could create this environment without these dependencies but, unfortunately, I did not find a decent article explaining how to put these things to work together (or maybe I did not have lucky googling).

Architecture Components

- Airflow configured with LocalExecutor meaning that all components (scheduler, webserver and executor) will be on the same machine.

database.

- Spark standalone cluster with 3 workers but you can configure more workers as explained further in this article.
- Jupyter notebook with Spark embedded to provide interactive Spark development.



Architecture components. Image by author.

Running your Data Engineer environment

Below, a step by step process to get your environment running. The complete project on Git can be found here.

Prerequisites

- Docker

Download Git project

```
$ git clone https://github.com/cordon-thiago/airflow-spark
```

Build Airflow Image

```
$ cd airflow-spark/docker/docker-airflow
$ docker build --rm --force-rm -t docker-airflow-spark:1.10.7_3.0.1 .
```

Default versions:

- Airflow `1.10.7`
- Spark `3.0.1`

Optionally, you can override the arguments in the build to choose specific Spark, Hadoop and Airflow versions. As an example, here is how to build an image containing the Airflow version `1.10.14`, Spark version `2.4.7` and Hadoop version `2.7`:

```
$ docker build --rm --force-rm \
-t docker-airflow-spark:1.10.14_2.4.7 . \
--build-arg AIRFLOW_VERSION=1.10.14 \
--build-arg SPARK_VERSION=2.4.7 \
--build-arg HADOOP_VERSION=2.7
```

Build Jupyter image

```
$ cd airflow-spark/docker/docker-jupyter
$ docker build --rm --force-rm -t jupyter/pyspark-notebook:3.0.1 .
```

an image containing the Spark version `2.4.7` and Hadoop version `2.7`:

```
$ docker build --rm --force-rm \
-t jupyter/pyspark-notebook:2.4.7 . \
--build-arg spark_version=2.4.7 \
--build-arg hadoop_version=2.7
```

If you change the name or the tag of the docker image when building, remember to update the name/tag in docker-compose file.

Check your images

```
$ docker images
```

```
REPOSITORY                                                          TAG
jupyter/pyspark-notebook                                            3.0.1
docker-airflow-spark                                                1.10.7_3.0.1
```

Start containers

```
$ cd airflow-spark/docker
$ docker-compose up -d
```

Note that when running the docker-compose for the first time, the images `postgres:9.6` and `bitnami/spark:3.0.1` will be built before the containers started.

At this moment you will have an output like below and your stack will be running :).

```
Creating network "docker_default_net" with the default driver
Creating docker_jupyter-spark_1     ... done
Creating docker_spark_1             ... done
Creating docker_spark-worker-3_1 ... done
Creating docker_postgres_1       ... done
Creating docker_spark-worker-2_1    ... done
Creating docker_spark-worker-1_1    ... done
Creating docker_airflow-webserver_1 ... done
```

- Spark Master: http://localhost:8181
- Airflow: http://localhost:8282
- Postgres DB Airflow: Server: localhost, port: 5432, User: airflow, Password: airflow
- Postgres DB Test: Server: localhost, port: 5432, User: test, Password: postgres
- Jupyter notebook: you need to run the code below to get the URL + Token generated and paste in your browser to access the notebook UI.

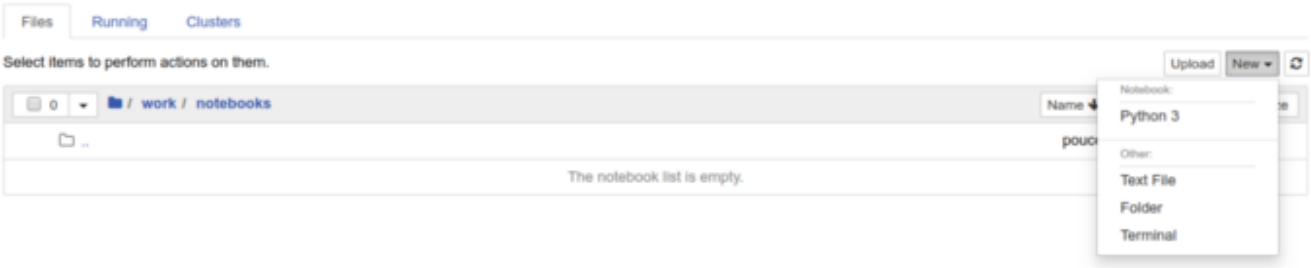
```
$ docker logs -f docker_jupyter-spark_1
```

```
To access the notebook, open this file in a browser:
    file:///home/jovyan/.local/share/jupyter/runtime/nbserver-6-open.html
Or copy and paste one of these URLs:
    http://b46e8587b3bd:8888/?token=9abfc9f63ab14eee96eac00038b62fca31673f50aa2762d0
 or http://127.0.0.1:8888/?token=9abfc9f63ab14eee96eac00038b62fca31673f50aa2762d0
```

Running a Spark Job inside Jupyter notebook

Not it's time to test if everything is working correctly. Let's first run a Spark application interactively in Jupyter notebook.

Inside Jupyter, go to "work/notebooks" folder and start a new Python 3 notebook.

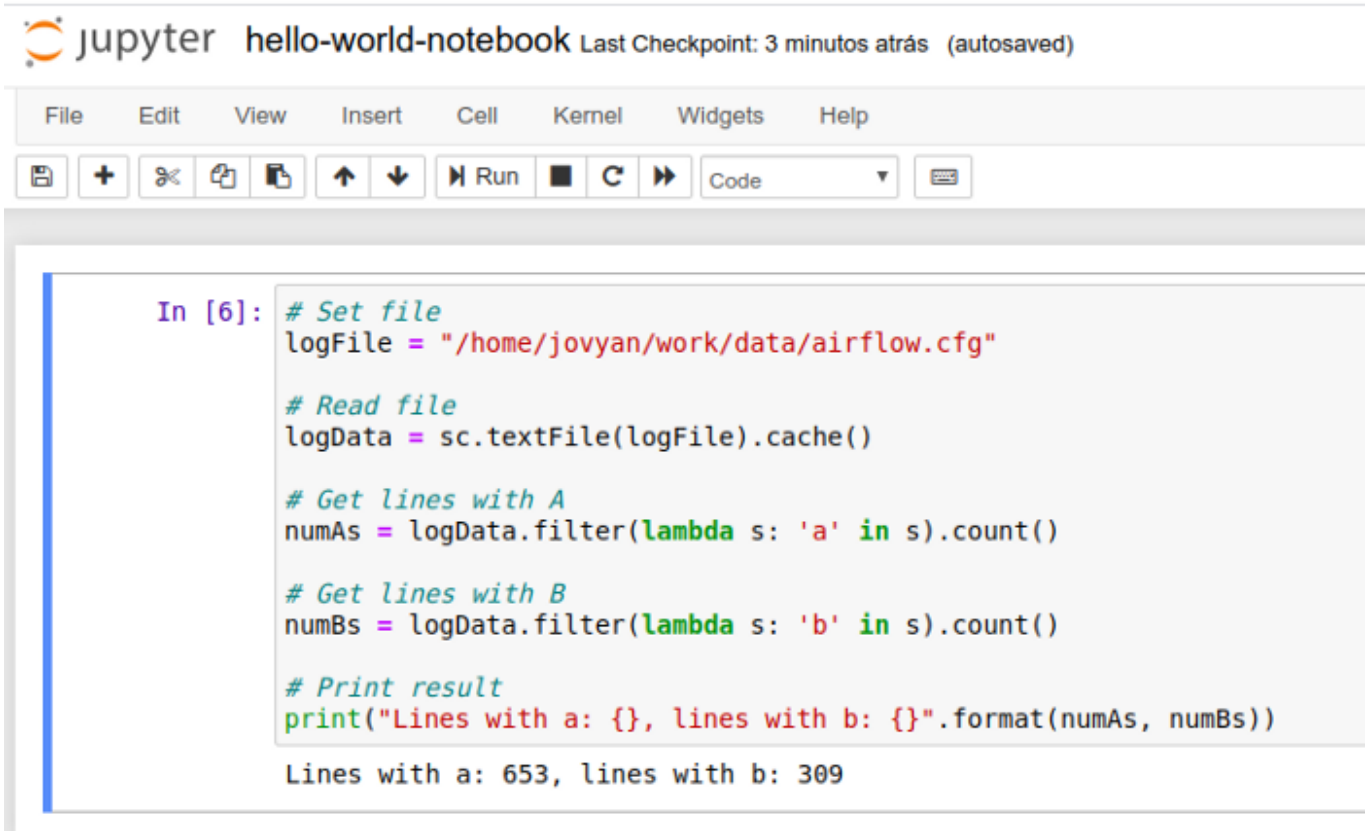


Paste the code below in the notebook and rename it to hello-world-notebook. This Spark code will count the lines with A and lines with B inside the airflow.cfg file.

```
# Set file
logFile = "/home/jovyan/work/data/airflow.cfg"
```

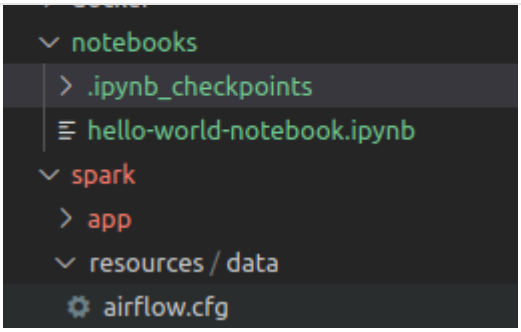
```
# Get lines with A
numAs = logData.filter(lambda s: 'a' in s).count()

# Get lines with B
numBs = logData.filter(lambda s: 'b' in s).count()

# Print result
print("Lines with a: {}, lines with b: {}".format(numAs, numBs))
```

After running the code, you will have:



Note that:

- The path spark/resources/data in your project is mapped to /home/jovyan/work/data/ in Jupyter machine.
- The folder notebooks in your project is mapped to /home/jovyan/work/notebooks/ in Jupyter machine.



Triggering a Spark Job from Airflow

In Airflow UI, you can find a DAG called spark-test which resides in dags folder inside your project.

This is a simple DAG that triggers the same Spark application which we ran in Jupyter notebook with two little differences:

- We need to instantiate the Spark Context (in Jupyter it is already instantiated).
- The file to be processed will be an argument passed by Airflow when calling spark-submit.

```
1
2  from airflow import DAG
3  from airflow.operators.dummy_operator import DummyOperator
4  from airflow.contrib.operators.spark_submit_operator import SparkSubmitOperator
5  from datetime import datetime, timedelta
6
7  ################################################
8  # Parameters
9  ################################################
10 spark_master = "spark://spark:7077"
11 spark_app_name = "Spark Hello World"
12 file_path = "/usr/local/spark/resources/data/airflow.cfg"
13
14 ################################################
15 # DAG Definition
16 ################################################
17 now = datetime.now()
18
```

```
    "start_date": datetime(now.year, now.month, now.day),
    "email": ["airflow@airflow.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=1)
}

dag = DAG(
        "spark-test",
        default_args=default_args,
        schedule_interval=timedelta(1)
    )

start = DummyOperator(task_id="start", dag=dag)

spark_job = SparkSubmitOperator(
    task_id="spark_job",
    application="/usr/local/spark/app/hello-world.py", # Spark application path created in airfl
    name=spark_app_name,
    conn_id="spark_default",
    verbose=1,
    conf={"spark.master":spark_master},
    application_args=[file_path],
    dag=dag)

end = DummyOperator(task_id="end", dag=dag)

start >> spark_job >> end
```

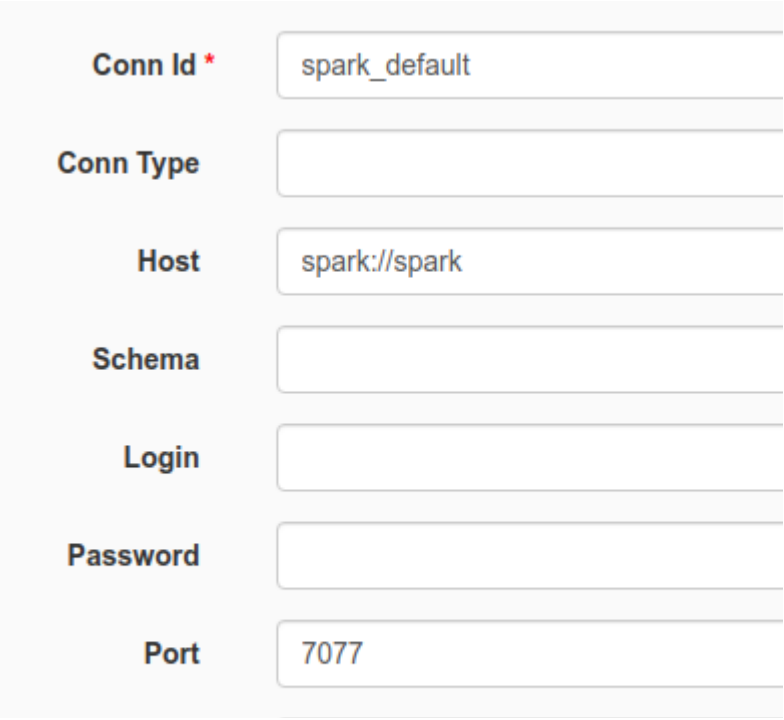
```
import sys
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

############################
# You can configure master here if you do not pass the spark.master paramenter in conf
############################
#master = "spark://spark:7077"
```

```python

# Create spark context
sc = SparkContext()

# Get the second argument passed to spark-submit (the first is the python app)
logFile = sys.argv[1]

# Read file
logData = sc.textFile(logFile).cache()

# Get lines with A
numAs = logData.filter(lambda s: 'a' in s).count()

# Get lines with B
numBs = logData.filter(lambda s: 'b' in s).count()

# Print result
print("Lines with a: {}, lines with b: {}".format(numAs, numBs))
```

airflow-spark-spark-app-hello-world.py hosted with ♡ by GitHub view raw

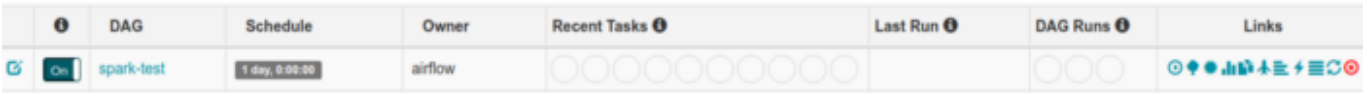
Spark app hello-world.py

Before running the DAG, change the spark_default connection inside Airflow UI to point to `spark://spark` (Spark Master) , `port 7077` :

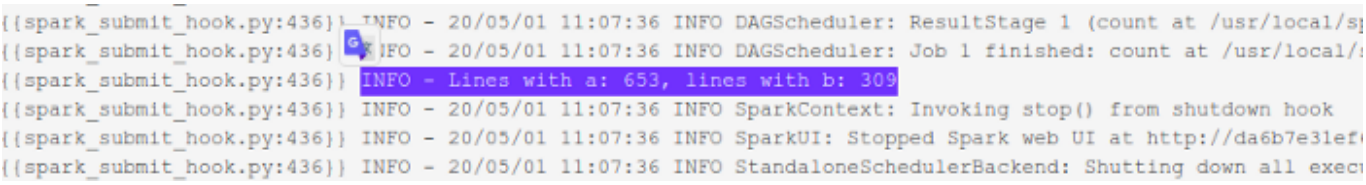


spark_default connection inside Airflow.

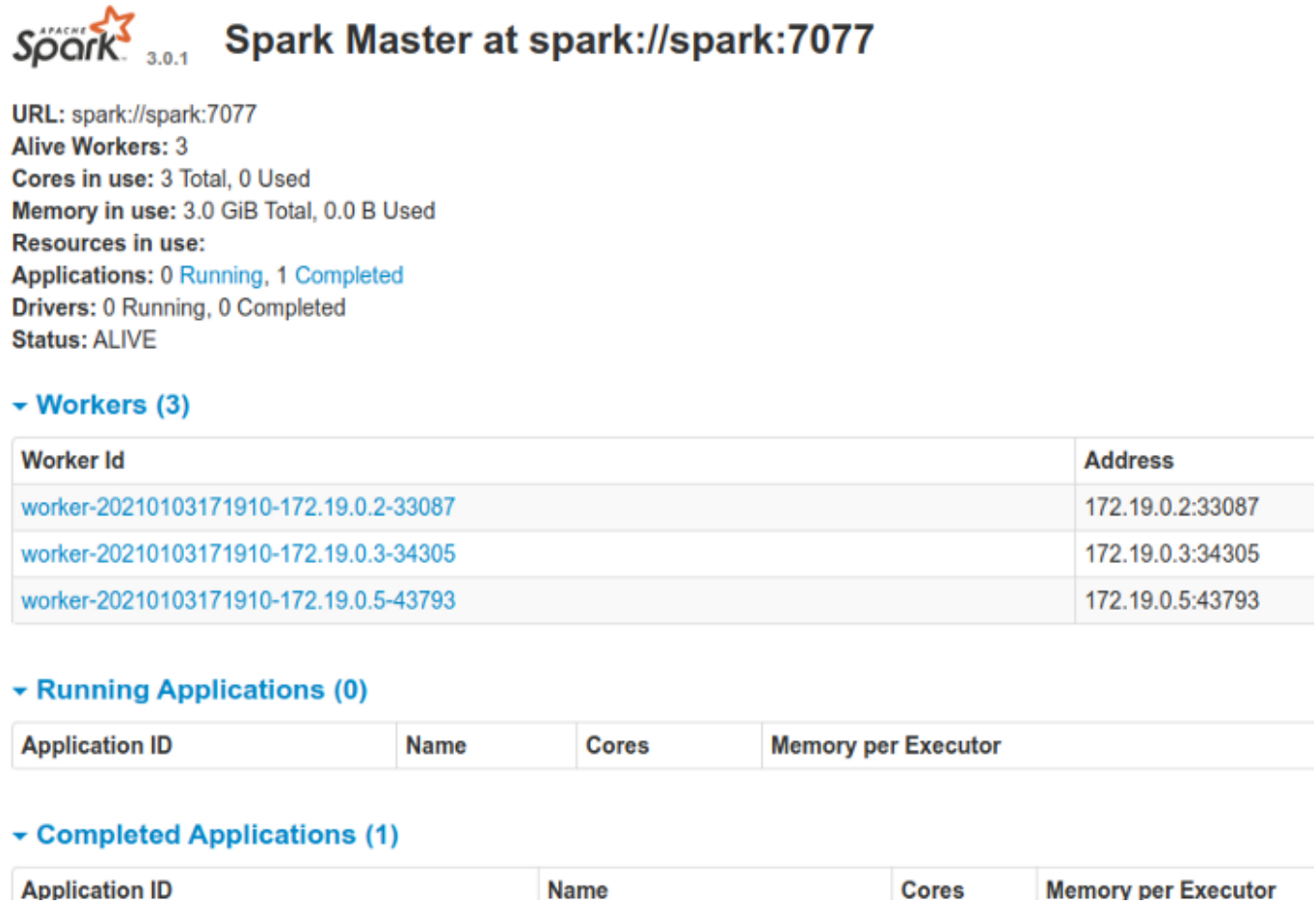
Now, you can turn the DAG on and trigger it from Airflow UI.



After running the DAG, you can see the result printed in the spark_job task log in Airflow UI:



And you can see the application in the Spark Master UI:



Increasing the number of Spark Workers

You can increase the number of Spark workers just adding new services based on `bitnami/spark:latest` image to the `docker-compose.yml` file like following:

```
    spark-worker-n:
        image: bitnami/spark:3.0.1
        networks:
            - default_net
        environment:
            - SPARK_MODE=worker
            - SPARK_MASTER_URL=spark://spark:7077
            - SPARK_WORKER_MEMORY=1G
            - SPARK_WORKER_CORES=1
            - SPARK_RPC_AUTHENTICATION_ENABLED=no
            - SPARK_RPC_ENCRYPTION_ENABLED=no
            - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
            - SPARK_SSL_ENABLED=no
        volumes:
            - ../spark/app:/usr/local/spark/app # Spark scripts folder (Must be the same path in
            - ../spark/resources/data:/usr/local/spark/resources/data #Data folder (Must be the
```

airflow-spark-docker-add-spark-worker.yml hosted with ♡ by GitHub view raw

Environment variables meaning can be found here.

Stopping your environment

When you no longer want to play with this stack, you can stop it to save local resources:

```
$ cd airflow-spark/docker
$ docker-compose down
```

Conclusion

Airflow and Spark are two core Data Engineering tools and you can have this environment on your computer whenever you want as explained in this guide.

Enjoy!

Spark Airflow Docker Data Engineering