

[Get started](#)[Open in app](#)[Follow](#)

582K Followers



You have **2** free member-only stories left this month. [Sign up for Medium and get an extra one](#)

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 ★



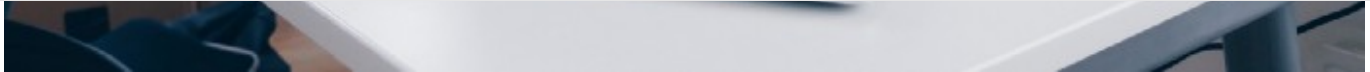
[Get started](#)[Open in app](#)

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).

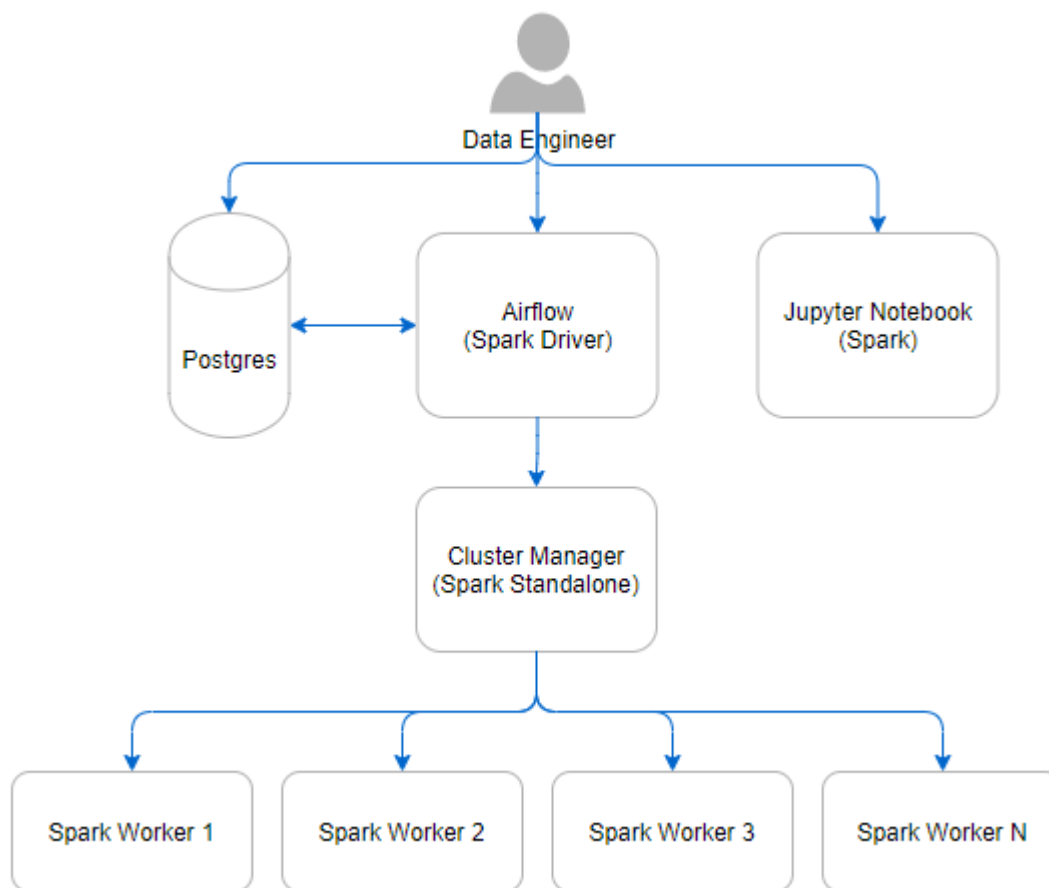
Architecture Components

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

[Get started](#)[Open in app](#)

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](#)

[Get started](#)[Open in app](#)

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 .
```

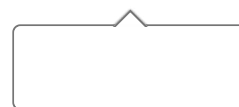
Default versions:

- 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 .
```

[Get started](#)[Open in app](#)

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
```

Get started

Open in app



- Spark Master: <http://localhost:8101>
- 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.

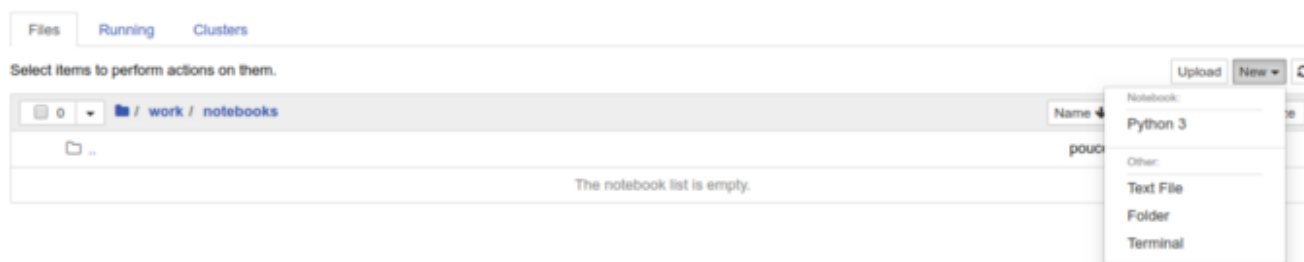
```
$ docker logs -f docker_jupyter-spark_1
```

```
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 started](#)[Open in app](#)

```
# 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:

```
In [6]: # Set file
        logFile = "/home/jovyan/work/data/airflow.cfg"

        # 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))

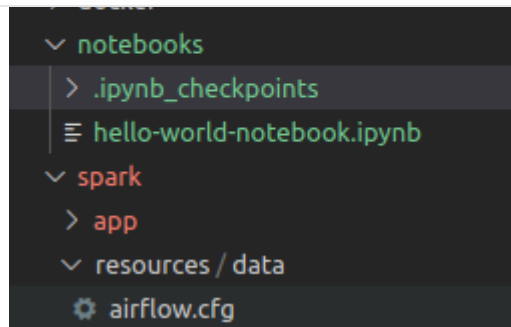
        Lines with a: 653, lines with b: 309
```

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.

Get started

Open in app



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

```


Get started

Open in app



```

22     "start_date": datetime(now.year, now.month, now.day),
23     "email": ["airflow@airflow.com"],
24     "email_on_failure": False,
25     "email_on_retry": False,
26     "retries": 1,
27     "retry_delay": timedelta(minutes=1)
28 }
29
30 dag = DAG(
31     "spark-test",
32     default_args=default_args,
33     schedule_interval=timedelta(1)
34 )
35
36 start = DummyOperator(task_id="start", dag=dag)
37
38 spark_job = SparkSubmitOperator(
39     task_id="spark_job",
40     application="/usr/local/spark/app/hello-world.py", # Spark application path created in airflow
41     name=spark_app_name,
42     conn_id="spark_default",
43     verbose=1,
44     conf={"spark.master":spark_master},
45     application_args=[file_path],
46     dag=dag)
47
48 end = DummyOperator(task_id="end", dag=dag)
49
50 start >> spark_job >> end

```

```

1  import sys
2  from pyspark import SparkConf, SparkContext
3  from pyspark.sql import SparkSession
4
5  #####
6  # You can configure master here if you do not pass the spark.master parameter in conf
7  #####
8  #master = "spark://spark:7077"

```

[Get started](#)[Open in app](#)

```
12
13 # Create spark context
14 sc = SparkContext()
15
16 # Get the second argument passed to spark-submit (the first is the python app)
17 logFile = sys.argv[1]
18
19 # Read file
20 logData = sc.textFile(logFile).cache()
21
22 # Get lines with A
23 numAs = logData.filter(lambda s: 'a' in s).count()
24
25 # Get lines with B
26 numBs = logData.filter(lambda s: 'b' in s).count()
27
28 # Print result
29 print("Lines with a: {}, lines with b: {}".format(numAs, numBs))
```

airflow spark spark app hello world my 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` :

Conn Id *	spark_default
Conn Type	
Host	spark://spark
Schema	
Login	
Password	
Port	7077

Get started

Open in app



spark_default connection inside Airflow.

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

	DAG	Schedule	Owner	Recent Tasks	Last Run	DAG Runs	Links
	spark-test	1 day, 0:00:00	airflow				

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

```
{[spark_submit_hook.py:436]} INFO - 20/05/01 11:07:36 INFO DAGScheduler: ResultStage 1 (count at /usr/local/spark/bin/hadoop) finished: count at /usr/local/spark/bin/hadoop
{[spark_submit_hook.py:436]} INFO - 20/05/01 11:07:36 INFO DAGScheduler: Job 1 finished: count at /usr/local/spark/bin/hadoop
{[spark_submit_hook.py:436]} INFO - Lines with a: 653, lines with b: 309
{[spark_submit_hook.py:436]} INFO - 20/05/01 11:07:36 INFO SparkContext: Invoking stop() from shutdown hook
{[spark_submit_hook.py:436]} INFO - 20/05/01 11:07:36 INFO SparkUI: Stopped Spark web UI at http://da6b7e31ef9:4040
{[spark_submit_hook.py:436]} INFO - 20/05/01 11:07:36 INFO StandaloneSchedulerBackend: Shutting down all executors
```

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



Spark Master at spark://spark:7077

URL: spark://spark:7077

Alive Workers: 3

Cores in use: 3 Total, 0 Used

Memory in use: 3.0 GiB Total, 0.0 B Used

Resources in use:

Applications: 0 Running, 1 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (3)

Worker Id	Address
worker-20210103171910-172.19.0.2-33087	172.19.0.2:33087
worker-20210103171910-172.19.0.3-34305	172.19.0.3:34305
worker-20210103171910-172.19.0.5-43793	172.19.0.5:43793

Running Applications (0)

Application ID	Name	Cores	Memory per Executor
----------------	------	-------	---------------------

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor
----------------	------	-------	---------------------

[Get started](#)[Open in app](#)

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:

```
1  spark-worker-n:
2      image: bitnami/spark:3.0.1
3      networks:
4          - default_net
5      environment:
6          - SPARK_MODE=worker
7          - SPARK_MASTER_URL=spark://spark:7077
8          - SPARK_WORKER_MEMORY=1G
9          - SPARK_WORKER_CORES=1
10         - SPARK_RPC_AUTHENTICATION_ENABLED=no
11         - SPARK_RPC_ENCRYPTION_ENABLED=no
12         - SPARK_LOCAL_STORAGE_ENCRYPTION_ENABLED=no
13         - SPARK_SSL_ENABLED=no
14     volumes:
15         - ../spark/app:/usr/local/spark/app # Spark scripts folder (Must be the same path in
16         - ../spark/resources/data:/usr/local/spark/resources/data #Data folder (Must be the
```

[airflow-spark-docker-add-spark-workers.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.

[Get started](#)[Open in app](#)

Enjoy!

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[Spark](#) [Airflow](#) [Docker](#) [Data Engineering](#)

[About](#) [Help](#) [Legal](#)

Get the Medium app

