Setting up a Spark standalone cluster on Docker in layman terms

Marin Aglić · Follow
10 min read · Jan 1, 2023

A while back I decided I wanted to learn PySpark. It seems to be one of the must know technologies for Data engineers. I finally found some time to begin learning and decided I don't want to use Jupyter or Google CodeLab and similar tools. In this post, I'll share how I configured a standalone Spark cluster using Docker. You can find the code here. EDIT: I added another repo that contains only the skeleton for running a Spark standalone cluster on Docker here.

Introduction

When I started investigating on how to set up a Spark cluster locally, I was largely confused on what installations I need, I didn't know what different types of nodes I need, how I'm going to execute an app, nor did I understand Spark configurations.

Spark is an engine that executes data processing operations. Basically, it is a tool that allows effective reading, transforming and loading data (think ETL). For processing your data, you usually set up a Spark cluster. If you're using some cloud provider, they probably have a service that sets up this cluster in the background for you. For example, Google Cloud Platform (GCP) has Dataproc. Of course, you're going to try to learn the tools you require as cheaply as possible, so a standalone cluster on your machine using Docker seems like nice to have.

What are Spark applications? Truth be told, I don't have experience in Scala+Spark, although I did learn Scala for a time. But I did start learning PySpark. Basically, to have a spark application you need to install the Spark (or PySpark) libraries (yes, there are libraries). These libraries are required so that you can reference Spark transformations and actions from your code. To execute said application, you

submit it to a Spark cluster. As we will see by the end of this story, when we will submit a Python script to be executed on the cluster.

The cluster usually has a master node and some worker nodes. Optionally, you can have a history server that shows the data about completed applications. Completed applications are basically jobs that you submitted to Spark. And in terms of PySpark, this can be a Python script.

Let's get started with Dockerfile.

The Dockerfile

I based my Dockerfile on the one I found for Apache Iceberg. You can find the repo under the first reference at the end of this story.

The base of the Docker image is `python:3.10-bullseye`. Usually, you might want to use Debian or Ubuntu. For starters, we install the tools that the OS will need, such as sudo, opejdk, etc.

```
RUN apt-get update && \
    apt-get install -y --no-install-recommends \
      sudo \
      curl \
      vim \
      unzip \
      rsync \
      openjdk-11-jdk \
      build-essential \
      software-properties-common \
      ssh && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*do
```

Next, we setup some directories and environment variables:

```
# Optional env variables
ENV SPARK_HOME=${SPARK_HOME:-"/opt/spark"}
ENV HADOOP_HOME=${HADOOP_HOME:-"/opt/hadoop"}
```

```
RUN mkdir -p ${HADOOP_HOME} && mkdir -p ${SPARK_HOME}
WORKDIR ${SPARK_HOME}
```

We setup base directories for our Spark and Hadoop installations. Although, we won't install Hadoop in this story.

The next lines download, install spark and perform some cleanup:

```
RUN curl https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz
 && tar xvzf spark-3.3.1-bin-hadoop3.tgz --directory /opt/spark --strip-compone
 && rm -rf spark-3.3.1-bin-hadoop3.tgz
```

We could improve the previous lines by specifying the version of Spark as an environment variable so that it's easier to change.

For the next part, let's copy the requirements. In my `requirements.in` I have the following packages:

```
ipython
pandas
pyarrow
numpy
pyspark
```

and I use `pip-compile-multi` to create a `requirements.txt`. I have `ipython` because I'm following the book Data Analysis with Python and PySpark. And copy the requirements in the Dockerfile:

```
# Install python deps
COPY requirements/requirements.txt .
RUN pip3 install -r requirements.txt
```

Next, set some environment variables for Spark, such as the address of the master, the host, port and PySpark python interpreter.

```
ENV PATH="/opt/spark/sbin:/opt/spark/bin:${PATH}"
ENV SPARK_HOME="/opt/spark"
ENV SPARK_MASTER="spark://spark-master:7077"
ENV SPARK_MASTER_HOST spark-master
ENV SPARK_MASTER_PORT 7077
ENV PYSPARK_PYTHON python3
```

We set the Spark binaries and scripts on the Path so it is easier to use the commands and shell scripts that we need. The SPARK_HOME directory is set to `/opt/spark`. The host is set to the same name as the service in docker-compose (we will get to this) that will run the master node.

Next, copy the spark defaults configuration:

```
COPY conf/spark-defaults.conf "$SPARK_HOME/conf"
```

Ok, so what do we have in the default configurations? Here is the `spark-defaults.conf`:

```
spark.master                        spark://spark-master:7077
spark.eventLog.enabled              true
spark.eventLog.dir                  /opt/spark/spark-events
spark.history.fs.logDirectory       /opt/spark/spark-events
```

Here, we're setting that the spark master will be a standalone cluster with the master on port 7077. We're also enabling the eventLog, eventLog directory and history filesystem logDirectory. These three settings are required to use the Spark history server. The setting `spark.eventLog.dir` is the base directory where the events are logged. The `spark.history.fs.logDirectory` is the directory from which the

filesystem history provider will load the logs. These two can be different. For more information see reference number 3.

Make the binaries and scripts executable and set the `PYTHONPATH` environment variable to use the python version that comes with spark:

```
RUN chmod u+x /opt/spark/sbin/* && \
    chmod u+x /opt/spark/bin/*

ENV PYTHONPATH=$SPARK_HOME/python/:$PYTHONPATH
```

Finally, copy the entrypoint script and set the script as the entrypoint.

```
COPY entrypoint.sh .

ENTRYPOINT ["./entrypoint.sh"]
```

So, here is how my whole Dockerfile looks like:

```
FROM python:3.10-bullseye as spark-base

RUN apt-get update && \
    apt-get install -y --no-install-recommends \
      sudo \
      curl \
      vim \
      unzip \
      rsync \
      openjdk-11-jdk \
      build-essential \
      software-properties-common \
      ssh && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*



## Download spark and hadoop dependencies and install

# Optional env variables
```

```
ENV SPARK_HOME=${SPARK_HOME:-"/opt/spark"}
ENV HADOOP_HOME=${HADOOP_HOME:-"/opt/hadoop"}

RUN mkdir -p ${HADOOP_HOME} && mkdir -p ${SPARK_HOME}
WORKDIR ${SPARK_HOME}


RUN curl https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz
 && tar xvzf spark-3.3.1-bin-hadoop3.tgz --directory /opt/spark --strip-compone
 && rm -rf spark-3.3.1-bin-hadoop3.tgz


FROM spark-base as pyspark

# Install python deps
COPY requirements/requirements.txt .
RUN pip3 install -r requirements.txt

ENV PATH="/opt/spark/sbin:/opt/spark/bin:${PATH}"
ENV SPARK_HOME="/opt/spark"
ENV SPARK_MASTER="spark://spark-master:7077"
ENV SPARK_MASTER_HOST spark-master
ENV SPARK_MASTER_PORT 7077
ENV PYSPARK_PYTHON python3

COPY conf/spark-defaults.conf "$SPARK_HOME/conf"

RUN chmod u+x /opt/spark/sbin/* && \
    chmod u+x /opt/spark/bin/*

ENV PYTHONPATH=$SPARK_HOME/python/:$PYTHONPATH

COPY entrypoint.sh .

ENTRYPOINT ["./entrypoint.sh"]
```

Entrypoint

The shell script `entrypoint.sh` contains which shell script the container should run once it starts, depending on the argument provided through the docker-compose file. The whole script is pretty simple, get the workload that we want from the argument and depending on the value, execute the appropriate Spark script:

```
#!/bin/bash

SPARK_WORKLOAD=$1
```

```
echo "SPARK_WORKLOAD: $SPARK_WORKLOAD"

if [ "$SPARK_WORKLOAD" == "master" ];
then
  start-master.sh -p 7077
elif [ "$SPARK_WORKLOAD" == "worker" ];
then
  start-worker.sh spark://spark-master:7077
elif [ "$SPARK_WORKLOAD" == "history" ]
then
  start-history-server.sh
fi
```

Docker compose

Let's start with the whole docker compose file:

```
version: '3.8'

services:
  spark-master:
    container_name: da-spark-master
    build: .
    image: da-spark-image
    entrypoint: ['./entrypoint.sh', 'master']
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:8080" ]
      interval: 5s
      timeout: 3s
      retries: 3
    volumes:
      - ./book_data:/opt/spark/data
      - ./spark_apps:/opt/spark/apps
      - spark-logs:/opt/spark/spark-events
    env_file:
      - .env.spark
    ports:
      - '9090:8080'
      - '7077:7077'


  spark-history-server:
    container_name: da-spark-history
    image: da-spark-image
    entrypoint: ['./entrypoint.sh', 'history']
    depends_on:
      - spark-master
    env_file:
      - .env.spark
```

```
    volumes:
      - spark-logs:/opt/spark/spark-events
    ports:
      - '18080:18080'

  spark-worker:
#    container_name: da-spark-worker
    image: da-spark-image
    entrypoint: ['./entrypoint.sh', 'worker']
    depends_on:
      - spark-master
    env_file:
      - .env.spark
    volumes:
      - ./book_data:/opt/spark/data
      - ./spark_apps:/opt/spark/apps
      - spark-logs:/opt/spark/spark-events

volumes:
  spark-logs:
```

There are a few things to notice. First, we define three services: master, worker and history server. We can have an arbitrary number of workers by starting the containers with:

```
docker-compose up --scale spark-worker=3
```

Second, the docker image is shared between the service. Therefore, the image is built only once.

The master and worker services have access to the book data and spark apps directories that we map into the containers using the `volumes` setting. The history server uses only the `spark-logs` volume that is defined at the bottom of the file. The directory for the spark logs is the same that is defined in the spark default configuration file that we wrote.

You should also notice that there is an environment file that is being loaded with the containers. The environment file has a single setting:

```
SPARK_NO_DAEMONIZE=true
```

This allows us to control whether the spark processes will run as daemons or not. We're turning that off, since otherwise, the containers will simply shutdown after the entrypoint script is executed.

Makefile

In the Makefile, I defined the instructions to ease starting, tearing down and building containers, and submitting spark jobs.

Here are some of the instructions from the Makefile:

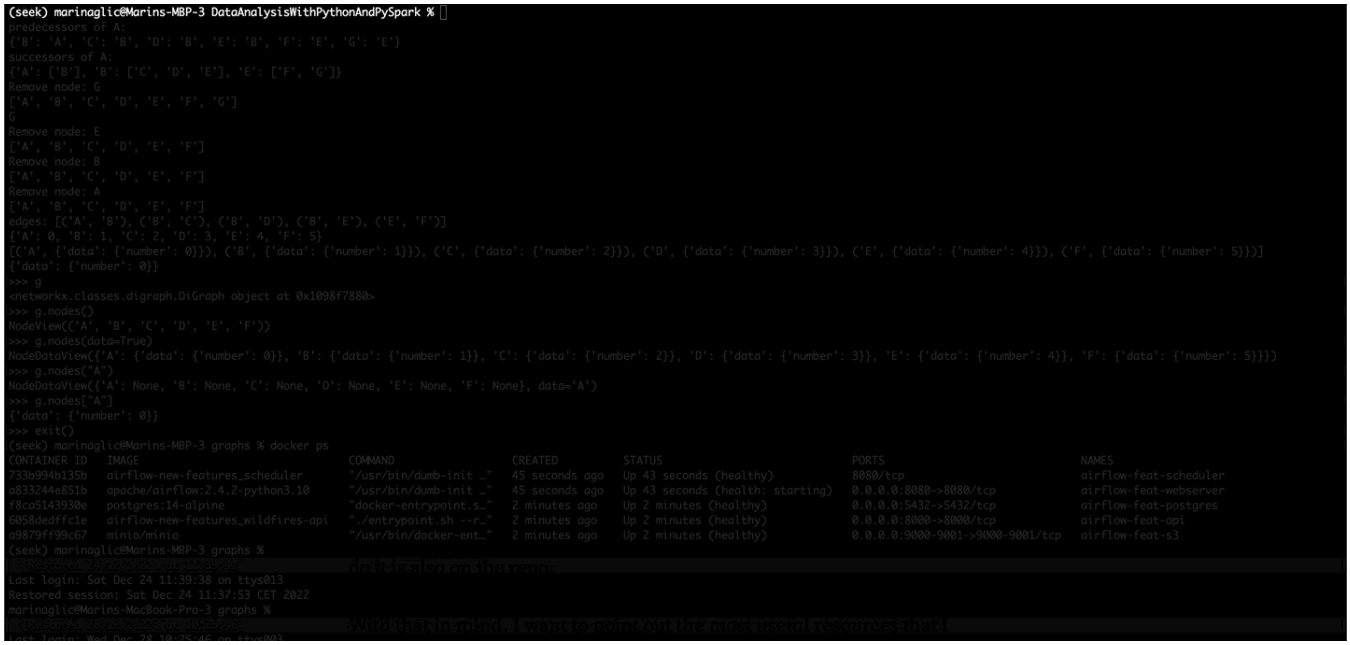
```
build:
	docker-compose build

build-nc:
	docker-compose build --no-cache

build-progress:
	docker-compose build --no-cache --progress=plain

down:
	docker-compose down --volumes

run:
	make down && docker-compose up

run-scaled:
	make down && docker-compose up --scale spark-worker=3

run-d:
	make down && docker-compose up -d

stop:
	docker-compose stop

submit:
	docker exec da-spark-master spark-submit --master spark://spark-master:7077
```

Running the cluster and submitting jobs

Let's run the cluster in scaled mode. To do this, we can simply run the command:

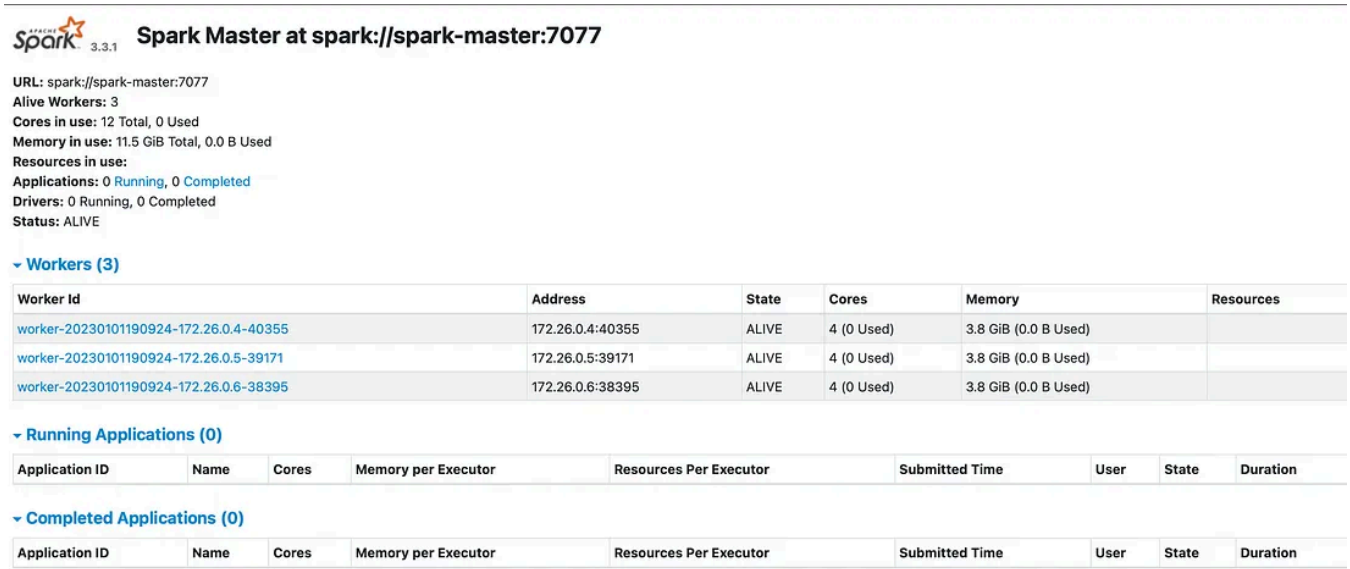
```
make run-scaled
```

This will bring up 5 containers — 1 master, 3 workers and 1 history server.



Starting standalone spark cluster

Once the cluster is started, we can go to `localhost:9090` and see if all of the workers are running.



Web UI on localhost:9090

Ok, now we can submit our first job. Since I prepared the command in the Makefile, we can submit the job with the following command:

```
make submit app=data_analysis_book/chapter03/word_non_null.py
```

The command in the background that will be executed:

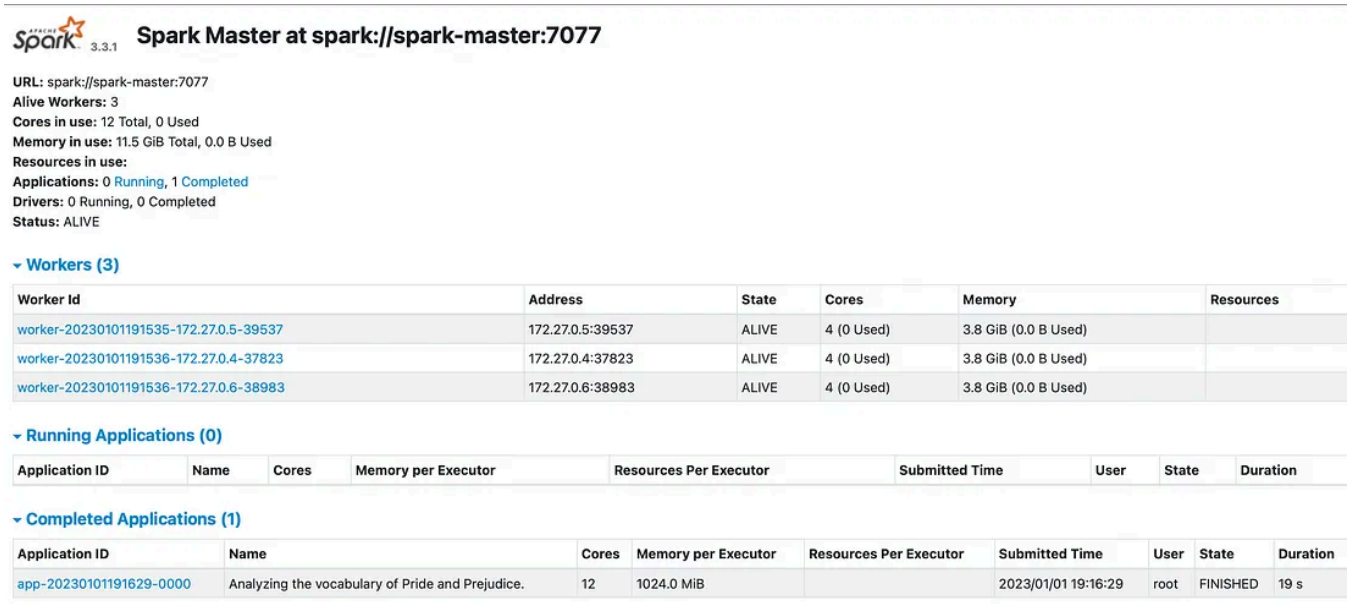
```
docker exec da-spark-master spark-submit --master spark://spark-master:7077 --c
```

You can see that we're actually using the spark-submit command, specifying the master, deploy mode, and script that we want to submit.

The script we submit is this one:

```
import pyspark.sql.functions as F
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName(
    "Analyzing the vocabulary of Pride and Prejudice."
).getOrCreate()

book = spark.read.text("/opt/spark/data/pride-and-prejudice.txt")

lines = book.select(F.split(F.col("value"), " ").alias("line"))

words = lines.select(F.explode(F.col("line")).alias("word"))

words_lower = words.select(F.lower(F.col("word")).alias("word_lower"))
words_clean = words_lower.select(
    F.regexp_extract(F.col("word_lower"), "[a-z]*", 0).alias("word")
)
words_nonull = words_clean.where(F.col("word") != "")

results = words_nonull.groupby(F.col("word")).count()

results.orderBy(F.col("count").desc()).show(10)

results.coalesce(1).write.csv("/opt/spark/data/results/chapter03/simple_count.c
```

Once the application finishes, you can see it on the web ui:



Spark Master at spark://spark-master:7077

URL: spark://spark-master:7077
Alive Workers: 3
Cores in use: 12 Total, 0 Used
Memory in use: 11.5 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	State	Cores	Memory	Resources
worker-20230101191535-172.27.0.5-39537	172.27.0.5:39537	ALIVE	4 (0 Used)	3.8 GiB (0.0 B Used)	
worker-20230101191536-172.27.0.4-37823	172.27.0.4:37823	ALIVE	4 (0 Used)	3.8 GiB (0.0 B Used)	
worker-20230101191536-172.27.0.6-38983	172.27.0.6:38983	ALIVE	4 (0 Used)	3.8 GiB (0.0 B Used)	

Running Applications (0)

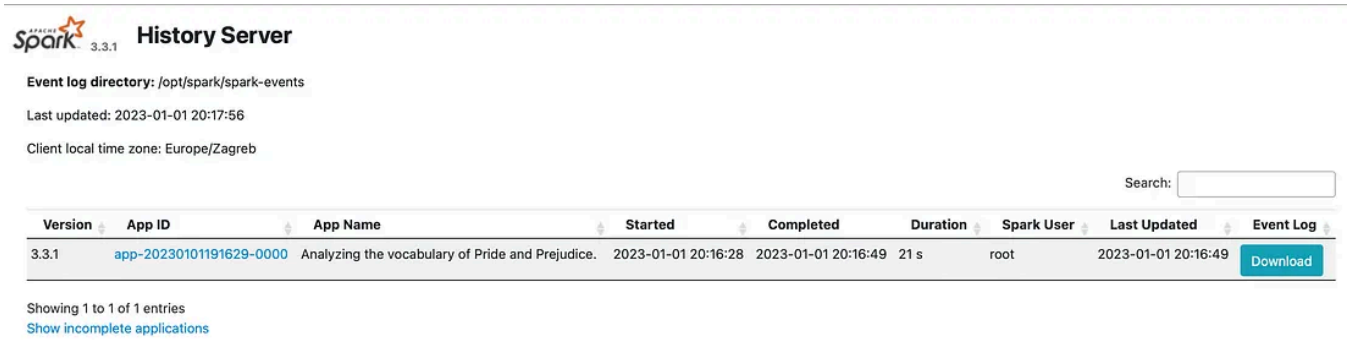
Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20230101191629-0000	Analyzing the vocabulary of Pride and Prejudice.	12	1024.0 MiB		2023/01/01 19:16:29	root	FINISHED	19 s

The finished application is visible on localhost:9090

You can also access the history server on `localhost:18080` and view the application there:



History Server

Event log directory: /opt/spark/spark-events
Last updated: 2023-01-01 20:17:56
Client local time zone: Europe/Zagreb

Search:

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.3.1	app-20230101191629-0000	Analyzing the vocabulary of Pride and Prejudice.	2023-01-01 20:16:28	2023-01-01 20:16:49	21 s	root	2023-01-01 20:16:49	Download

Showing 1 to 1 of 1 entries
Show incomplete applications

History server on localhost:18080

So, what is the client mode? This setting was most confusing for me.

A Spark job consists of Spark Executors that actually execute the task and Spark Driver that schedules the Executors. The book I mentioned defines the executors and drivers: *The executors sit atop a worker and execute the work submitted by the driver. The driver is responsible for completing a given job and requests resources from the master as needed. Workers are a set of computing resources, and the master is the one allocating resources as needed to complete a job (Chapter 1.2.2 from the book Data Analysis with Python and PySpark).*

When running a job in client mode, the driver is running on the client, e.g. a laptop. If the client fails, the job is shutdown. The executors still run on the Spark cluster. For the cluster mode, everything is run on the cluster. If you've started a job using

your laptop, you can feel free to close it (see reference 5). So, **in both modes, the actual work is performed on the cluster.**

Additions to the repo — not covered in this post

After I figured out that you can't submit Python scripts in cluster mode for a standalone spark cluster, I started looking into how to integrate Spark into a Yarn cluster. I managed to do this before writing this post, so the code to do it is also on the repo. The spark documentation also states:

> *Currently, the standalone mode does not support cluster mode for Python applications. (reference 7)*

With that in mind, I want to point out the most useful resources that I found for bringing Spark up on a Yarn cluster. The links are under references 4 and 5.

I might write another story describing how to integrate Yarn and Spark, but really, the two references I found are great.

Summary

In this post we saw how to:

- setup a Spark standalone cluster on Docker
- submit jobs to the Spark cluster

The code is available on GitHub here.

I should also mention that reference 8 is also an article on medium that describes building a standalone Spark cluster with a Jupyter interface.

EDIT: I created another repo that contains only the skeleton for running a Spark standalone cluster. You can find it here.

References

1. https://github.com/tabular-io/docker-spark-iceberg/tree/main/spark
2. https://www.manning.com/books/data-analysis-with-python-and-pyspark
3. https://stackoverflow.com/questions/32001248/whats-the-difference-between-spark-eventlog-dir-and-spark-history-fs-logdirecto
4. https://www.linode.com/docs/guides/how-to-install-and-set-up-hadoop-cluster/

5. https://www.linode.com/docs/guides/install-configure-run-spark-on-top-of-hadoop-yarn-cluster/
6. https://dev.to/mvillarrealb/creating-a-spark-standalone-cluster-with-docker-and-docker-compose-2021-update-6l4
7. https://spark.apache.org/docs/latest/submitting-applications.html
8. https://towardsdatascience.com/apache-spark-cluster-on-docker-ft-a-juyterlab-interface-418383c95445
9. https://lemaizi.com/blog/creating-your-own-micro-cluster-lab-using-docker-to-experiment-with-spark-dask-on-yarn/

Python Spark Cluster Docker Pyspark

Follow

Written by Marin Aglić

274 Followers

Working as a Software Engineer. Interested in Data Engineering. Mostly working with airflow, python, celery, bigquery. Ex PhD student.

More from Marin Aglić

Marin Aglić

Setting up Hadoop Yarn to run Spark applications

In this post I'll talk about setting up a Hadoop Yarn cluster with Spark. After setting up a Spark standalone cluster, I noticed that I...

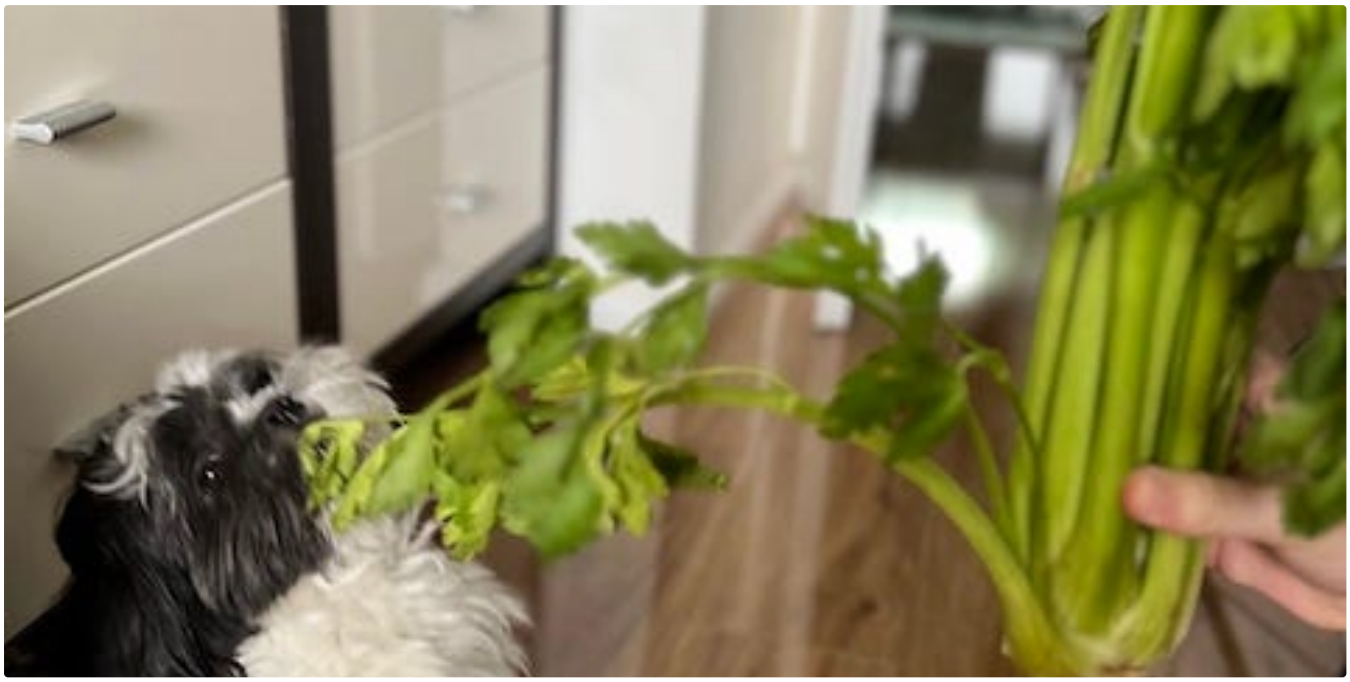
Jan 10, 2023 46 5

Marin Aglić

Learning Apache Iceberg—an introspection

Bridging the gap between what I knew and what I wanted to learn. This is the first story I decided to write about my process of learning...

Mar 10, 2023 158

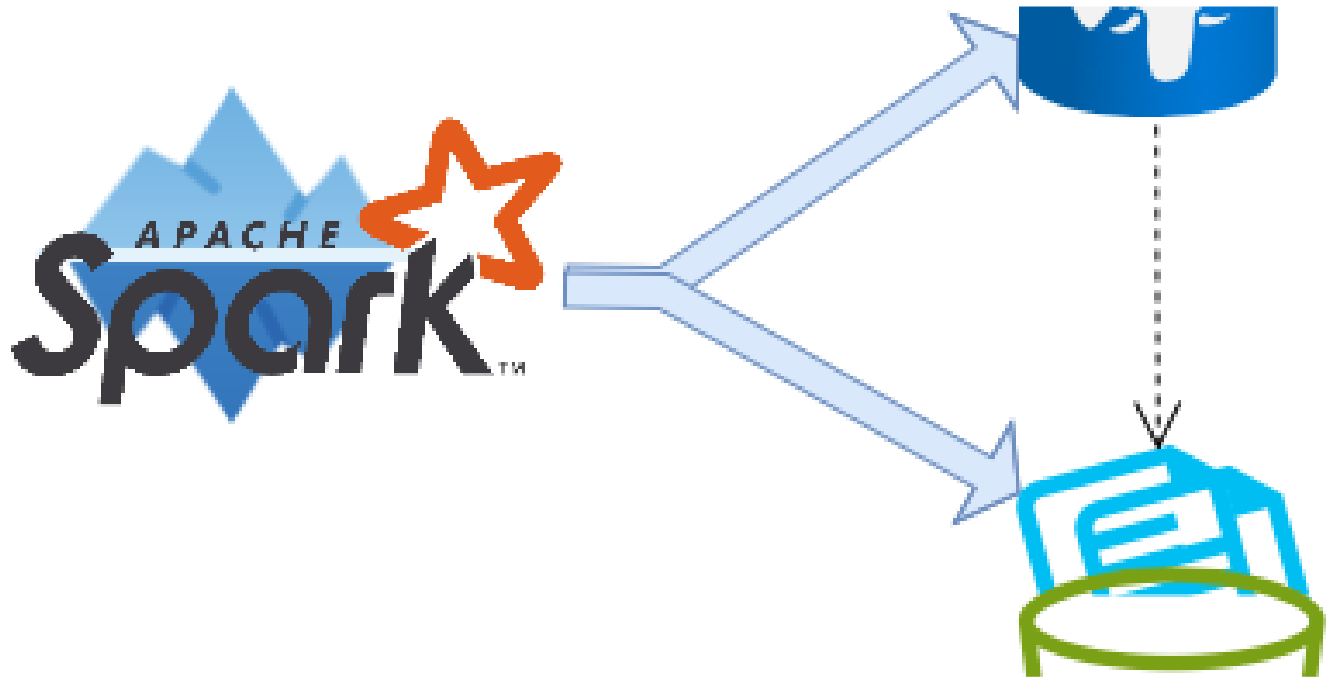


Marin Aglić in Data Engineer Things

Designing Dynamic Workflows with Celery and Python

The why and how to insert chains into chains

Aug 7, 2023 50



Marin Aglić

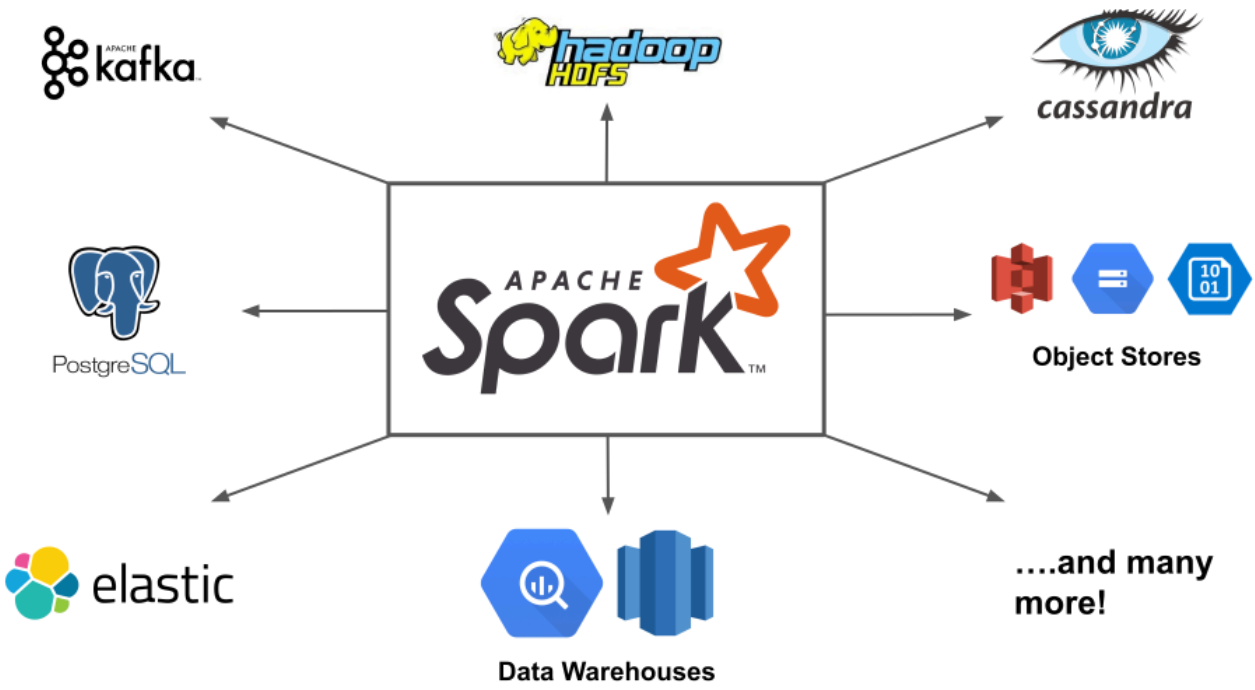
Learning Apache Iceberg—storing the data to Minio S3

Bridging the gap between what I knew and what I wanted to learn. This is the third in a series of articles where I continue my progress in...

Jan 30 92

See all from Marin Aglić

Recommended from Medium



Mahmud Oyinloye

Setting Up Apache Spark (macOS): A Comprehensive Guide

This tutorial walks you through setting up Apache Spark on macOS, (version 3.4.3). It covers installing dependencies like Miniconda...

May 8

Bayu Adi Wibowo

Deploying a Big Data Ecosystem: Dockerized Hadoop, Spark, Hive, and Zeppelin

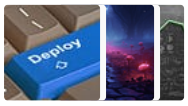
The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of...

May 5 72 2

Lists

Coding & Development

11 stories · 863 saves



Predictive Modeling w/ Python

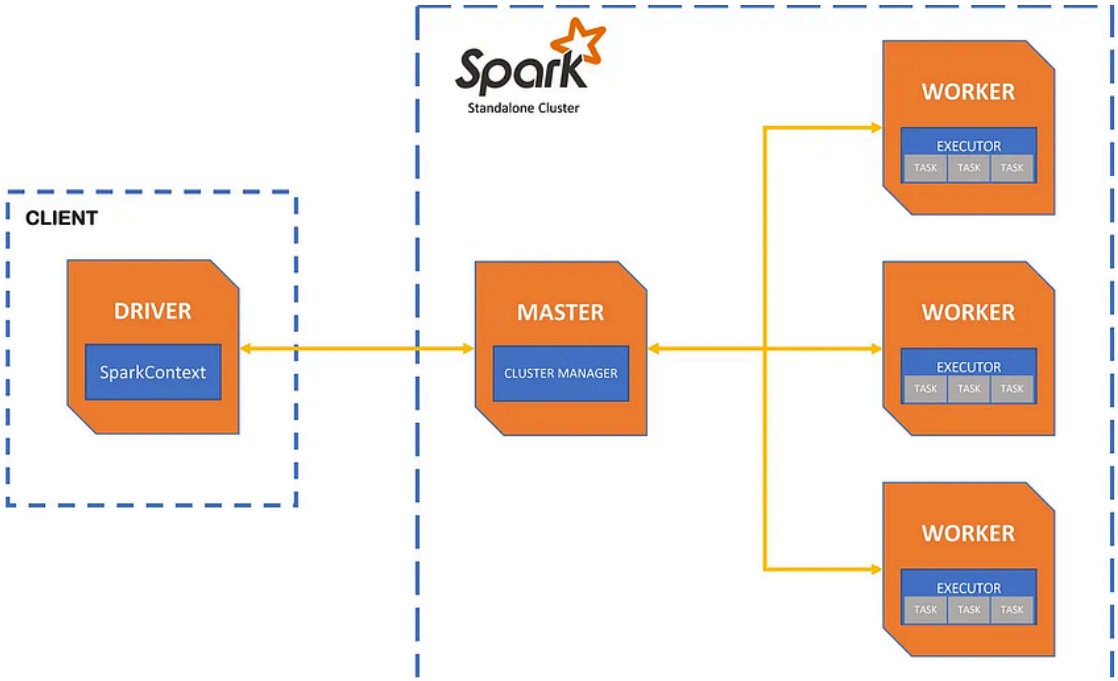
20 stories · 1614 saves

Practical Guides to Machine Learning

10 stories · 1967 saves

ChatGPT

21 stories · 848 saves

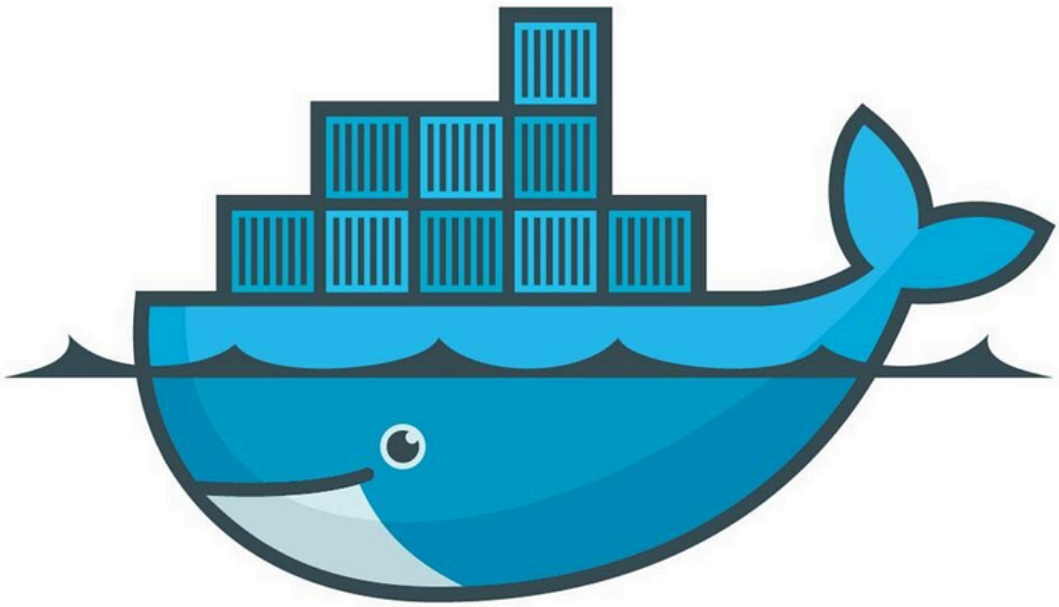


Sanjeet Shukla

Setting Up Apache Spark from Scratch in a Docker Container: A Step-by-Step Guide

Apache Spark is a technology of choice for Data Engineering. However, setting up a full-fledged Spark cluster can be a daunting task. So In...

Aug 29 12

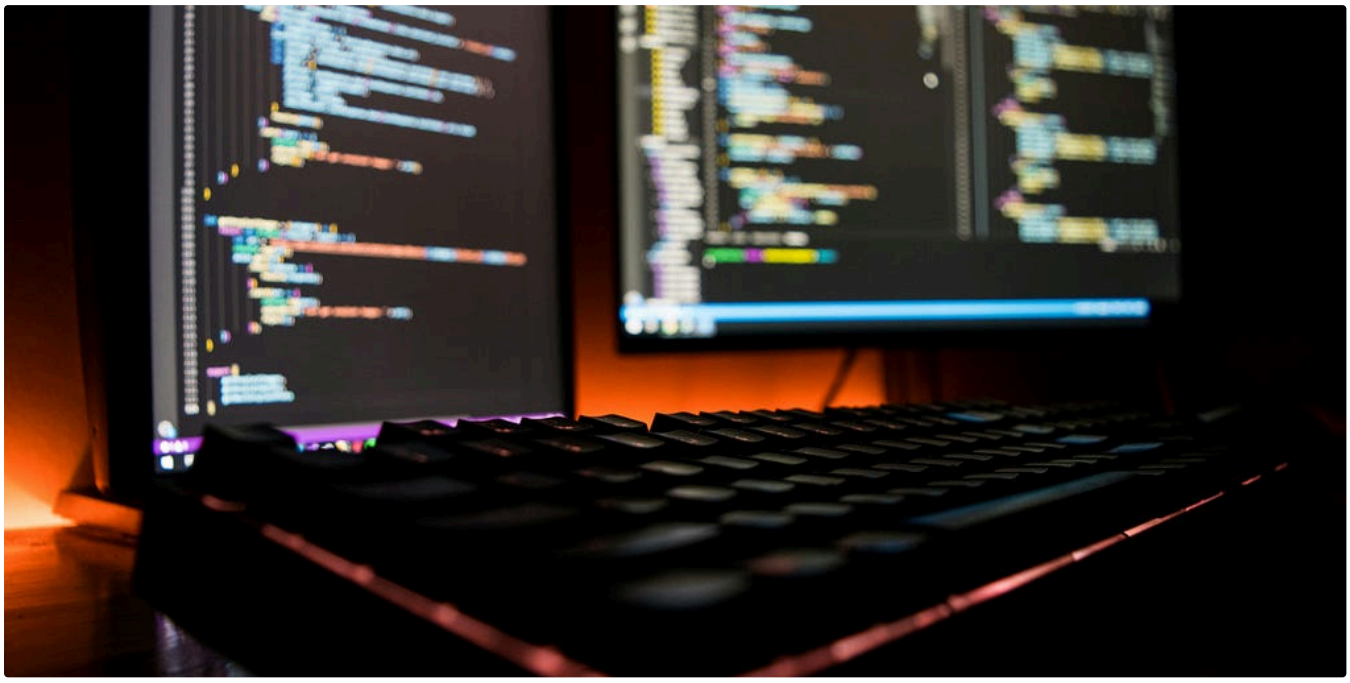


Romain Bruyère

Docker and WSL2 without Docker Desktop

Using Docker on Windows has grown more challenging over the past few years. This guide aims to walk you through the process, from start to...

May 9 194 5

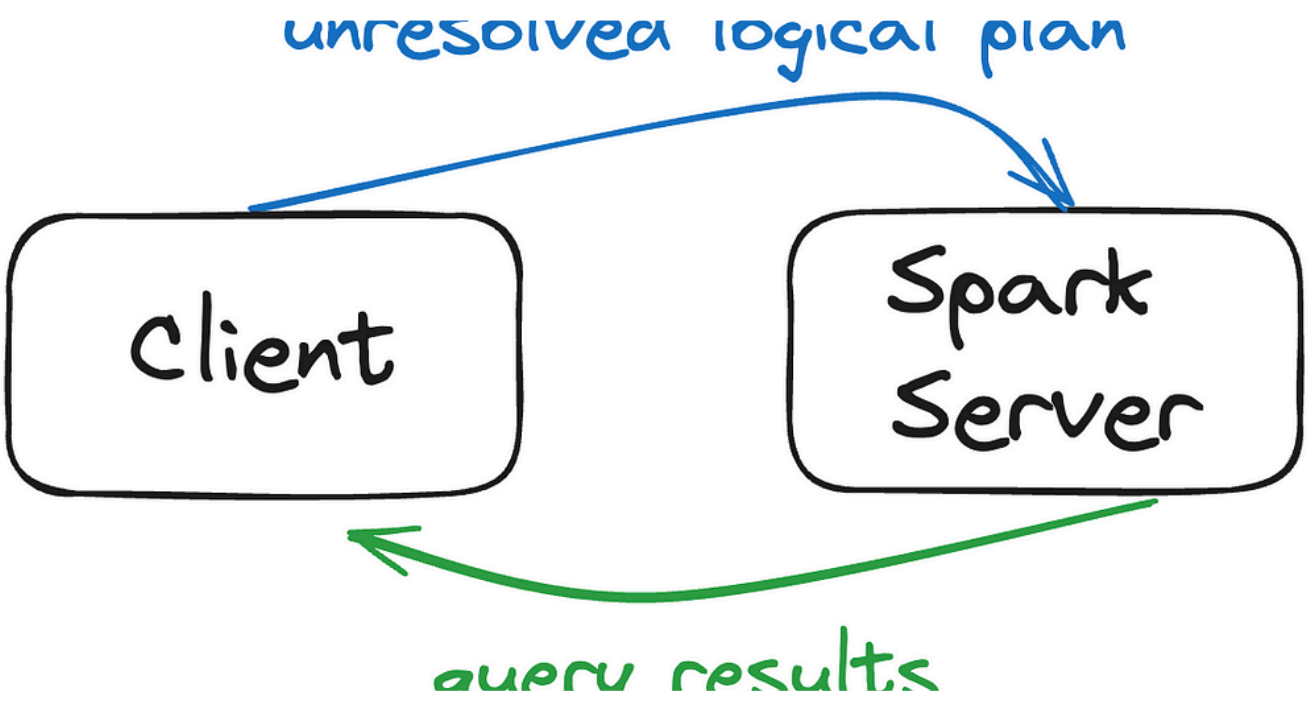


Thomas Lawless

PySpark Development with Poetry & PEX

Managing dependencies for PySpark applications can be challenging, especially when you want to maintain a clean development environment.

Jun 9 2



ymelo

Spark Connect: Launch Spark Applications Anywhere with the Client-Server Architecture + DBT

Apache Spark is a powerful data processing tool that has revolutionized big data analysis. One of its most recent and exciting features is...

May 26 74 1