# Developing Multi Nodes Hadoop, Spark Cluster, and Airflow in Docker Compose (Part 1)

Well, you are in the right place here if you are having a headache about how to kicking-off simple Hadoop standalone cluster locally along with Spark standalone cluster and Airflow

Ayyoub Maulana · Follow

8 min read · Mar 19, 2023

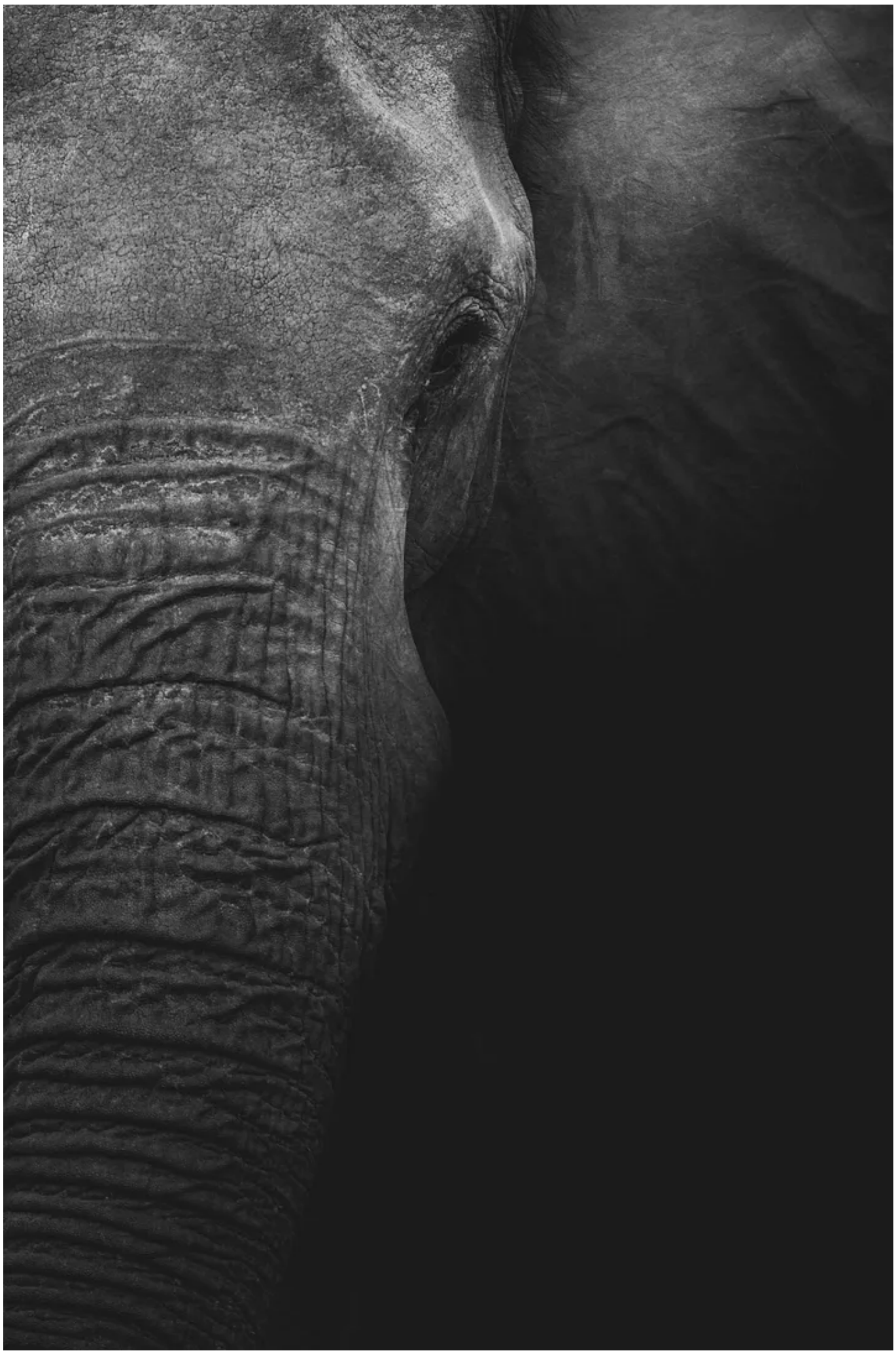

Photo by Geran de Klerk on Unsplash

## Talking Points

So, in this post, I will break down some parts of Hadoop, Airflow, and Spark because those three tools have their own complexities. If you are seeking out what is Hadoop, why we use it, and what the pros and cons of this article are not for you, I will create it separately later on. Same with the rest of the tools. Ok, I will break into three points here :

1. How to start Hadoop Cluster using Docker (Part 1)
2. How to start Spark Cluster using Docker (Part 2)
3. How to setup Airflow using Docker (Part 3)

And, as we can see we use docker to start all of them hence you should have docker installed on your device.

## Prerequisites

1. Docker
2. WSL2 if you are using Windows

I assume that you already know what docker is so we will focus on Hadoop installation instead. If you want to see **all setup** just open it on **Github here.**

Ok, without further ado let's play!

## Hadoop

In short, Hadoop is **a framework that allows for the distributed processing of large data sets across clusters of computers.** We will create 3 datanodes here one namenode, 1 history server, 1 node manager, and 1 resource manager.

For this project, this is my project structure looks like this:

```
hadoop-docker
|_ airflow
|_ conf
|_ datanode
|_ historyserver
|_ namenode
|_ nodemanager
|_ resourcemanager
|_ spark
|_ Dockerfile
|_ docker-compose.yml
|_ start-cluster.sh
```

I will create all dockerfile related to Airflow in the airflow directory, the conf directory consists of all Hadoop configurations, all datanode related will go to datanode directory, all namenode related will build under namenode directory, and so on. You can modify and simplify this project structure feel free to refactor it if it is not so simple structure.

### 1. Namenode

Let's start with namenode, namenode has a crucial function in Hadoop. Namenode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes) including all the HDFS metadata in Hadoop. So, if namenode fails it can be a serious problem in our cluster.

To create a namenode, you can create file named Dockerfile inside namenode directory, and just copy and paste this code to your new Dockerfile :

```
FROM hadoop-base:3.3.1

HEALTHCHECK CMD curl -f http://localhost:9870/ || exit 1

ENV HDFS_CONF_dfs_namenode_name_dir=file:///hadoop/dfs/name
RUN mkdir -p /hadoop/dfs/name
VOLUME /hadoop/dfs/name

ADD run.sh /run.sh
RUN chmod a+x /run.sh

EXPOSE 9870

CMD ["/run.sh"]
```

The namenode Dockerfile is an extended image from hadoop-base:3.3.1 that we will create later on. Once the image was created then it will create a command to check the healthiness of the namenode, then create hadoop dfs file for namenode, and run the run.sh shell script to start the namenode. Oke, to satisfy the needs you can create run.sh file inside namenode directory as well.

run.sh shell script will contains this lines of code :

```
#!/bin/bash

# Check if namenode dir exists
namedir=`echo $HDFS_CONF_dfs_namenode_name_dir | perl -pe 's#file://##'`
if [ ! -d $namedir ]; then
  echo "Namenode name directory not found: $namedir"
  exit 2
fi

# check whether clustername specified or not
# we setup clustername in docker-compose
if [ -z "$CLUSTER_NAME" ]; then
  echo "Cluster name not specified"
  exit 2
fi

# remove some unused files from namenode directory
echo "remove lost+found from $namedir"
rm -r $namedir/lost+found

# format namenode dir if there is no files or directory exists in it
if [ "`ls -A $namedir`" == "" ]; then
  echo "Formatting namenode name directory: $namedir"
  $HADOOP_HOME/bin/hdfs --config $HADOOP_CONF_DIR namenode -format $CLUSTER_NAME
fi

# kick-off namenode server
$HADOOP_HOME/bin/hdfs --config $HADOOP_CONF_DIR namenode
```

We need this file to automate the process of creating, formatting, and checking namenode. Now, you have a namenode here. What's next? let's move on.

### 2. Datanode

Datanode has a different function compared with namenode, it will store your file object, make it distributed with a partition, and has some blocks to actually save our file in a distributed fashion. So, for datanode we will go into datanode directory and create a Dockerfile file in it. This is the contents :

```
FROM hadoop-base:3.3.1

HEALTHCHECK CMD curl -f http://localhost:9864/ || exit 1

ENV HDFS_CONF_dfs_datanode_data_dir=file:///hadoop/dfs/data
RUN mkdir -p /hadoop/dfs/data
VOLUME /hadoop/dfs/data

ADD run.sh /run.sh
RUN chmod a+x /run.sh

EXPOSE 9864

CMD ["/run.sh"]
```

Same with namenode, datanode's Dockerfile is built from an existing image called hadoop-base:3.3.1. Will create the datanode dfs directory after it to



```
#!/bin/bash

# Check if datanode data dir exists
datadir=`echo $HDFS_CONF_dfs_datanode_data_dir | perl -pe 's#file://##'`
if [ ! -d $datadir ]; then
  echo "Datanode data directory not found: $datadir"
  exit 2
fi

# if yes run this code to kick-off datanode
$HADOOP_HOME/bin/hdfs --config $HADOOP_CONF_DIR datanode
```

### 3. History Server

The third thing is the History server, this is where we have historical logs about our jobs in Hadoop. Since the Resource manager and the Job tracker run in memory, then to avoid being out of memory it will delete the finished jobs and you can see it in the History server instead.

That's a brief definition of a History server, to set it up, you can jump into the historyserver directory then make Dockerfile again then fill it with this :

```
FROM hadoop-base:3.3.1

HEALTHCHECK CMD curl -f http://localhost:8188/ || exit 1

ENV YARN_CONF_yarn_timeline___service_leveldb___timeline___store_path=/hadoop/ya
RUN mkdir -p /hadoop/yarn/timeline
VOLUME /hadoop/yarn/timeline

ADD run.sh /run.sh
```

```
RUN chmod a+x /run.sh

EXPOSE 8188

CMD ["/run.sh"]
```

Again, same with the two points above it will health check the service, make history server directory logs called yarn timeline, and run.sh to automate creation. Next, create run.sh again inside historyserver directory and paste this script :

```
#!/bin/bash

# Kick-off yarn server
$HADOOP_HOME/bin/yarn --config $HADOOP_CONF_DIR historyserver
```

### 4. Node Manager

Hadoop uses YARN to manage their resource, Node Manager is the YARN agent that is attached to the node, to track which node is available, and which node is saving the requested file by the client. To run the Node Manager you can go into nodemanager directory and make Dockerfile in it then paste this code :

```
FROM hadoop-base:3.3.1

HEALTHCHECK CMD curl -f http://localhost:8042/ || exit 1

ADD run.sh /run.sh
RUN chmod a+x /run.sh

EXPOSE 8042

CMD ["/run.sh"]
```

To automate the creation of Node Manager you can create a new file named run.sh and paste this script in it :

```
#!/bin/bash

# Kick-off nodemanager to manage namenode, datanode inside cluster
# This is second level of resource manager
# Nodemanager will manage resource hand in hand with yarn
$HADOOP_HOME/bin/yarn --config $HADOOP_CONF_DIR nodemanager
```

Once, you run it it will run the service on port 8042.

### 5. Resource Manager

If the Node Manager is the second level that works with YARN, then Resource Manager is the first level service that works with YARN. It will manage the Node Manager to manage resources and communicate to the YARN. To set it up, you can create Dockerfile inside resourcemanager directory and fill it in with this code :

```
FROM hadoop-base:3.3.1

HEALTHCHECK CMD curl -f http://localhost:8088/ || exit 1

ADD run.sh /run.sh
RUN chmod a+x /run.sh

EXPOSE 8088

CMD ["/run.sh"]
```

And, to automate creation as well you can create the run.sh file and paste this script to it :

```
#!/bin/bash

$HADOOP_HOME/bin/yarn --config $HADOOP_CONF_DIR resourcemanager
```

Once you run it, it will turn the resource manager service on.

### 6. Base Hadoop Image

Oke, you have it all, then the last thing is creating the base image from the Dockerfile inside hadoop-docker project. To do so, you can create as we did before and paste this code into the Dockerfile :

```
FROM python:3.7

USER root

# ----------------------------------------------------------
# JAVA
# ----------------------------------------------------------
RUN apt-get update && apt-get install -y --no-install-recommends \
    openjdk-11-jdk
# For AMD based architecture use
# ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk/
ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-arm64/

# ----------------------------------------------------------
# HADOOP
# ----------------------------------------------------------
ENV HADOOP_VERSION=3.3.1
ENV HADOOP_URL=https://downloads.apache.org/hadoop/common/hadoop-$HADOOP_VERSION
ENV HADOOP_PREFIX=/opt/hadoop-$HADOOP_VERSION
ENV HADOOP_CONF_DIR=/etc/hadoop
ENV MULTIHOMED_NETWORK=1
ENV USER=root
ENV HADOOP_HOME=/opt/hadoop-$HADOOP_VERSION
ENV PATH $HADOOP_PREFIX/bin/:$PATH
ENV PATH $HADOOP_HOME/bin/:$PATH
```

```
RUN set -x \
    && curl -fSL "$HADOOP_URL" -o /tmp/hadoop.tar.gz \
    && tar -xvf /tmp/hadoop.tar.gz -C /opt/ \
    && rm /tmp/hadoop.tar.gz*

RUN ln -s /opt/hadoop-$HADOOP_VERSION/etc/hadoop /etc/hadoop
RUN mkdir /opt/hadoop-$HADOOP_VERSION/logs
RUN mkdir /hadoop-data

USER root

COPY conf/core-site.xml $HADOOP_CONF_DIR/core-site.xml
COPY conf/hdfs-site.xml $HADOOP_CONF_DIR/hdfs-site.xml
COPY conf/mapred-site.xml $HADOOP_CONF_DIR/mapred-site.xml
COPY conf/yarn-site.xml $HADOOP_CONF_DIR/yarn-site.xml
```

I create this for ARM64-based architecture, so for you that use AMD64 you can adjust JAVA_HOME with :

```
ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk/
```

Once you run it, it will download the Hadoop binary file, so just sit and relax until the process is done. We will talk about how to run it in a batch later on. So, next to simplify the process, create docker-compose.yml inside hadoop-docker directory, and paste this code to it :

```
version: "2.1"

services:
  namenode:
    build: ./namenode
    container_name: namenode
    volumes:
      - hadoop_namenode:/hadoop/dfs/name
      - ./data/:/hadoop-data/input
      - ./map_reduce/:/hadoop-data/map_reduce
    environment:
      - CLUSTER_NAME=test
    ports:
      - "9870:9870"
      - "8020:8020"
    networks:
      - hadoop_network

  resourcemanager:
    build: ./resourcemanager
    container_name: resourcemanager
    restart: on-failure
    depends_on:
      - namenode
      - datanode1
      - datanode2
      - datanode3
    ports:
      - "8089:8088"
    networks:
      - hadoop_network

  historyserver:
    build: ./historyserver
    container_name: historyserver
    depends_on:
      - namenode
```

```
      - datanode1
      - datanode2
    volumes:
      - hadoop_historyserver:/hadoop/yarn/timeline
    ports:
      - "8188:8188"
    networks:
      - hadoop_network

  nodemanager1:
    build: ./nodemanager
    container_name: nodemanager1
    depends_on:
      - namenode
      - datanode1
      - datanode2
    ports:
      - "8042:8042"
    networks:
      - hadoop_network

  datanode1:
    build: ./datanode
    container_name: datanode1
    depends_on:
      - namenode
    volumes:
      - hadoop_datanode1:/hadoop/dfs/data
    networks:
      - hadoop_network

  datanode2:
    build: ./datanode
    container_name: datanode2
    depends_on:
      - namenode
    volumes:
      - hadoop_datanode2:/hadoop/dfs/data
    networks:
      - hadoop_network

  datanode3:
    build: ./datanode
    container_name: datanode3
    depends_on:
      - namenode
    volumes:
      - hadoop_datanode3:/hadoop/dfs/data
    networks:
      - hadoop_network

volumes:
  hadoop_namenode:
  hadoop_datanode1:
  hadoop_datanode2:
  hadoop_datanode3:
  hadoop_historyserver:

networks:
  hadoop_network:
    name: hadoop_network
```

Once, it is up, it will create 3 datanodes, nodemanager, historyserver, resourcemanager, and namenode services. You can increase or decrease the number of datanodes with your own. Place the data that want to save on the container in the input directory, if you have some Map Reduce script then place it in the map_reduce directory.

Finally to automate the creation, create one shell script file called start-cluster.sh and paste this script inside it :

```
#!/bin/bash

docker network create hadoop_network

docker build -t hadoop-base:3.3.1 .

docker-compose up -d
```

Once you run it, it will create the docker network to make it easy to manage and communicate with the services, then built the Dockerfile that contains a base image and name it with tag 3.3.1, beware to change this tag, if you want to change it then you should change all the tags in all Dockerfile. The final step is creating all Hadoop services with docker-compose up command.

Don’t forget to add execute permission to it using this command :

```
sudo chmod +x start-cluster.sh
```

To run it all, type this command to your terminal :

```
./start-cluster.sh
```

then voila! you have a cluster of Hadoop locally.

Hadoop  Docker  Data Engineering


## Written by Ayyoub Maulana

Follow

42 Followers

Data Enthusiast | Passionate Data Engineer https://www.linkedin.com/in/ayyoub-maulana/ | Wanna mentored by me ? https://adplist.org/mentors/ayyoub-maulana


### More from Ayyoub Maulana