



Creating a Spark Cluster on your laptop

Patrizio Dazzi

patrizio.dazzi@isti.cnr.it





Virtualization

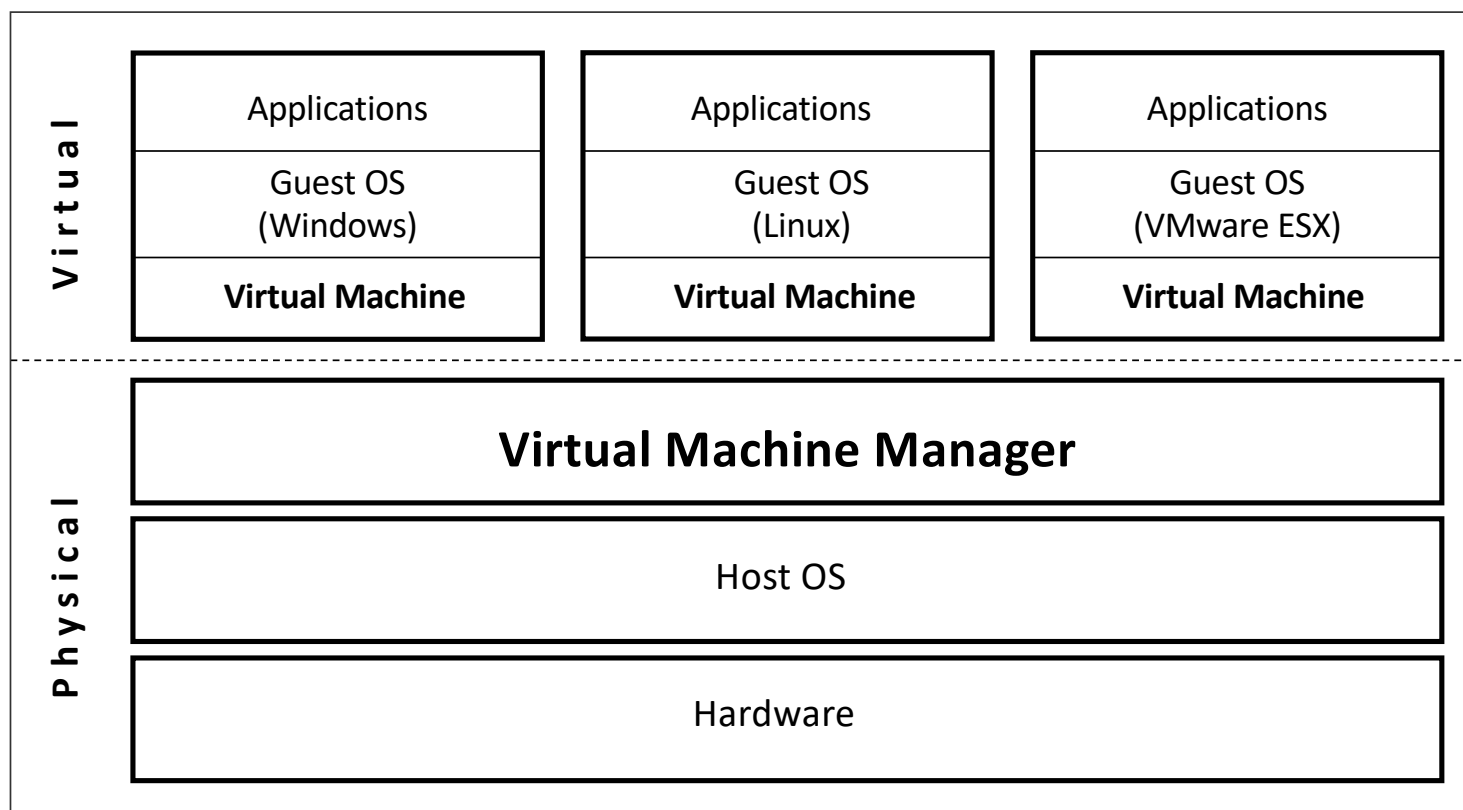
- Desktop Virtualization
- Server Virtualization
- Network Virtualization
- Storage Virtualization
- Application Virtualization



Benefits from Virtualization

- Save money and energy
- Simplify management

Conceptual representation of VMs



- What is Docker?
 - Docker vs. Virtual Machine
 - History, Status, Run Platforms
 - Hello World
- Images and Containers
- Volume Mounting, Port Publishing, Linking
- Around Docker, Docker Use Cases

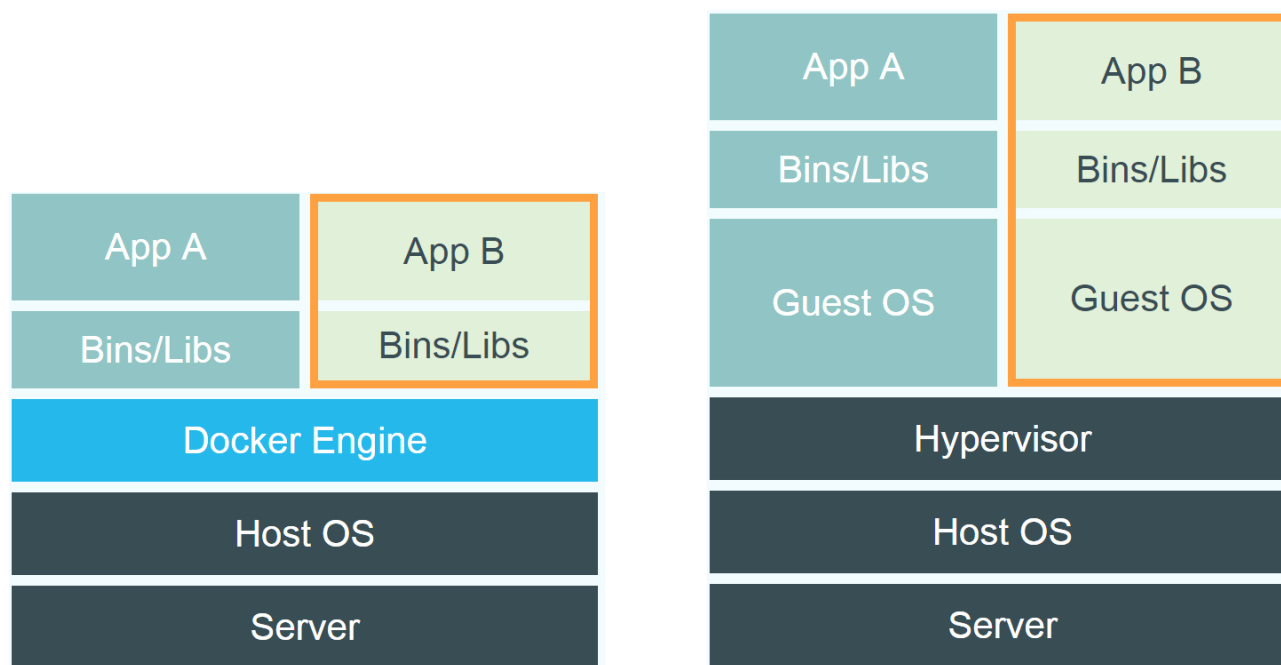


Docker

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system–level virtualization on Linux.

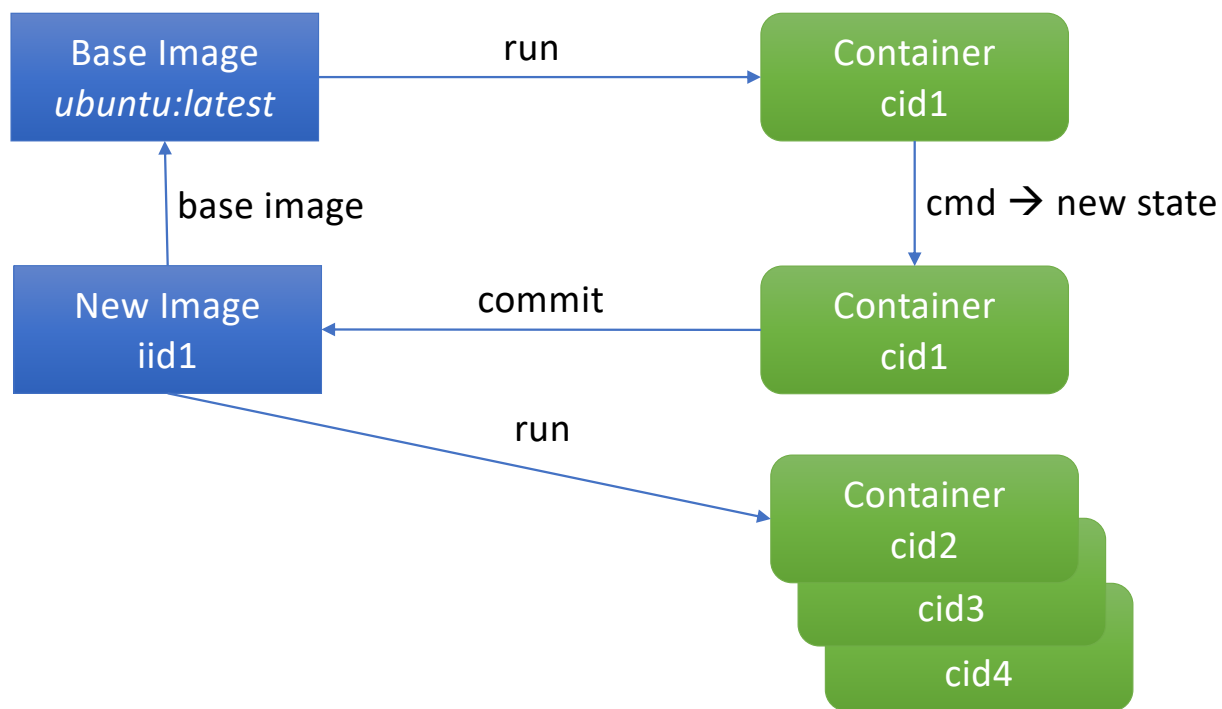
[Source: en.wikipedia.org]

Virtual Machines vs Containers



Source: <https://www.docker.com/whatisdocker/>

Docker: Images vs Containers





Dockerfile

- Create images automatically using a build script: «Dockerfile»
- Can be versioned in a version control system like Git or SVN, along with all dependencies
- Docker Hub can automatically build images based on dockerfiles on Github

Dockerfile example

- Dockerfile:
 - FROM ubuntu
 - ENV DOCK_MESSAGE Hello My World
 - ADD dir /files
 - CMD ["bash", "someScript"]
- docker build [DockerFileDir]
- docker inspect [imageId]



Create container

Simple Command - Ad-Hoc Container

- `docker run ubuntu echo Hello World`

The Docker File System

- In order to understand Docker volumes, it is important to first understand how the Docker file system works.
- A Docker image is a collection of read-only layers. When you launch a container from an image, Docker adds a read-write layer to the top of that stack of read-only layers. Docker calls this the Union File System.
- Any time a file is changed, Docker makes a copy of the file from the read-only layers up into the top read-write layer. This leaves the original (read-only) file unchanged.
- When a container is deleted, that top read-write layer is lost. This means that any changes made after the container was launched are now gone.

How a Volume Can Help

- A volume allows data to persist, even when a container is deleted. Volumes are also a convenient way to share data between the host and the container.
- Mounting a volume is a good solution if you want to:
 - Push data to a container.
 - Pull data from a container.
 - Share data between containers.
- Docker volumes exist outside the Union File System of read-only and read-write layers. The volume is a folder which is shared between the container and the host machine. Volumes can also be shared between containers.

The Basics of Docker Volumes

- A Docker volume "lives" outside the container, on the host machine.
- From the container, the volume acts like a folder which you can use to store and retrieve data. It is simply a mount point to a directory on the host.
- There are several ways to create and manage Docker volumes. Each method has its own advantages and disadvantages.

Using Docker's "volume create" Command

- Creation

- `sudo docker volume create --name [volume name]`
 - E.g., `sudo docker volume create --name data-volume`

- Linkage

- `sudo docker run -it --name my-volume-test -v data-volume:/data centos /bin/bash`

Port publishing

- `docker run -t -p 8080:80 ubuntu nc -l 80`
- Map container port 80 to host port 8080
- Check on host: `nc localhost 8080`
- Link with other docker container
- `docker run -ti --link containerName:alias ubuntu`
- See link info with `set`

Docker: key commands

- images: List all local images
- run: Create a container from an image and execute a command in it
- tag: Tag an image
- pull: Download image from repository
- rmi: Delete a local image
- This will also remove intermediate images if no longer used



Creating a Spark Cluster



Requirements

- The actual versions used for this exercise are:
 - Spark (spark-2.3.2-bin-hadoop2.7)
 - Java (JDK8 update 231)

The easy way...

- The simplest way of using Spark is the Stand Alone Mode:
 - No Hadoop YARN
 - No Mesos
 - No Kubernetes
- Starting on an empty directory, we create a sub-directory downloads and move the previously downloaded packages to there.

Create a folder with Spark and JDK8

- `$ mkdir downloads`
- `$ mv ~/Downloads/jdk-8u231-linux-x64.tar.gz downloads/`
- `$ mv ~/Downloads/spark-2.3.2-bin-hadoop2.7.tgz downloads/`
- Then we will build a Docker image which can be used both as master and slave. The way we achieve that is through a proper Docker file.



Spark.Dockerfile (1)

```
## build our image on top of Debian latest version
```

```
FROM debian:latest
```

```
LABEL maintainer=youremail@domain.ext
```

```
## initial directory
```

```
WORKDIR /home/
```

```
## install necessary packages
```

```
RUN apt-get update && apt-get install -y curl vim wget software-properties-common ssh  
net-tools wamerican wamerican-insane wbrazilian wdutch wbritish-large
```

```
RUN apt-get install -y python3 python3-pip python3-numpy python3-matplotlib python3-  
scipy python3-pandas python3-simp
```



Spark.Dockerfile (2)

```
## set python3 as default (currently python 3.5)
```

```
RUN update-alternatives --install "/usr/bin/python" "python" "$(which python3)" 1
```

```
## move files from downloads folder at the host to /home in the container
```

```
COPY downloads/* /home/
```

Spark.Dockerfile (3)

```
## configure java jdk 8
RUN mkdir -p /usr/local/oracle-java-8
RUN tar -zxf jdk-8u231-linux-x64.tar.gz -C /usr/local/oracle-java-8/
RUN rm jdk-8u231-linux-x64.tar.gz
RUN update-alternatives --install "/usr/bin/java" "java" "/usr/local/oracle-
java-8/jdk1.8.0_231/bin/java" 1
RUN update-alternatives --install "/usr/bin/javac" "javac" "/usr/local/oracle-
java-8/jdk1.8.0_231/bin/javac" 1
RUN update-alternatives --install "/usr/bin/javaws" "javaws"
"/usr/local/oracle-java-8/jdk1.8.0_231/bin/javaws" 1
ENV JAVA_HOME="/usr/local/oracle-java-8/jdk1.8.0_231"
```


Spark.Dockerfile (4)

```
## configure spark
RUN mkdir -p /usr/local/spark-2.3.2
RUN tar -zxf spark-2.3.2-bin-hadoop2.7.tgz -C /usr/local/spark-2.3.2/
RUN rm spark-2.3.2-bin-hadoop2.7.tgz
RUN update-alternatives --install "/usr/sbin/start-master" "start-master" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/start-master.sh" 1
RUN update-alternatives --install "/usr/sbin/start-slave" "start-slave" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/start-slave.sh" 1
RUN update-alternatives --install "/usr/sbin/start-slaves" "start-slaves" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/start-slaves.sh" 1
RUN update-alternatives --install "/usr/sbin/start-all" "start-all" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/start-all.sh" 1
RUN update-alternatives --install "/usr/sbin/stop-all" "stop-all" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/stop-all.sh" 1
RUN update-alternatives --install "/usr/sbin/stop-master" "stop-master" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/stop-master.sh" 1
RUN update-alternatives --install "/usr/sbin/stop-slaves" "stop-slaves" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/stop-slaves.sh" 1
RUN update-alternatives --install "/usr/sbin/stop-slave" "stop-slave" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/stop-slave.sh" 1
```



Spark.Dockerfile (5)

```
RUN update-alternatives --install "/usr/sbin/spark-daemon.sh" "spark-daemon.sh" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/spark-daemon.sh" 1
```

```
RUN update-alternatives --install "/usr/sbin/spark-config.sh" "spark-config.sh" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/sbin/spark-config.sh" 1
```

```
RUN update-alternatives --install "/usr/bin/spark-shell" "spark-shell" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/spark-shell" 1
```

```
RUN update-alternatives --install "/usr/bin/spark-class" "spark-class" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/spark-class" 1
```

```
RUN update-alternatives --install "/usr/bin/spark-sql" "spark-sql" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/spark-sql" 1
```

```
RUN update-alternatives --install "/usr/bin/spark-submit" "spark-submit" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/spark-submit" 1
```

```
RUN update-alternatives --install "/usr/bin/pyspark" "pyspark" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/pyspark" 1
```

```
RUN update-alternatives --install "/usr/bin/load-spark-env.sh" "load-spark-env.sh" "/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7/bin/load-spark-env.sh" 1
```

```
ENV SPARK_HOME="/usr/local/spark-2.3.2/spark-2.3.2-bin-hadoop2.7"
```



Spark.Dockerfile (6)

```
## expose for ssh
```

```
EXPOSE 22
```

```
## expose for spark use
```

```
EXPOSE 7000-8000
```

```
## expose for master webui
```

```
EXPOSE 8080
```

```
## expose for slave webui
```

```
EXPOSE 8081
```



Image Creation

- We save this file as spark.Dockerfile and build an image using the following command:

```
$ docker build -f spark.Dockerfile -t username/imagename .
```



Otherwise...

- you can just skip this whole section and just pull our image:
\$ docker pull thiagolcmelo/spark-debian



Create a Network

```
$ docker network create --driver bridge spark-network
```

to ease the communication between worker(s) and master

Creation of the Master

- launch the node as a daemon:

```
$ docker run -d -t \  
    --name master \  
    --network spark-network \  
    thiagolcmelo/spark-debian  
$ docker exec master start-master
```

- launch it in interactive mode, which:

```
$ docker run -it \  
    --name master \  
    --network spark-network \  
    thiagolcmelo/spark-debian \  
    /bin/bash
```

```
root@1e315e2d5e20:/home# start-  
master
```

Comm Ports

- There are two ports important for the master node:
 - 7077 where the slaves connect (default)
 - 8080 the web ui (default)
- We can check if they are working by doing:
 - `root@1e315e2d5e20:/home# netstat -plutn`

Custom Ports

```
$ docker run -d -t \  
    --name master \  
    --network spark-network \  
    thiagolcmelo/spark-debian  
$ docker exec master start-master  
--port 7070 --webui-port 7071
```

```
$ docker run -it \  
    --name master \  
    --network spark-network \  
    thiagolcmelo/spark-debian \  
    /bin/bash  
root@1e315e2d5e20:/home# start-  
master --port 7070 --webui-port  
7071
```



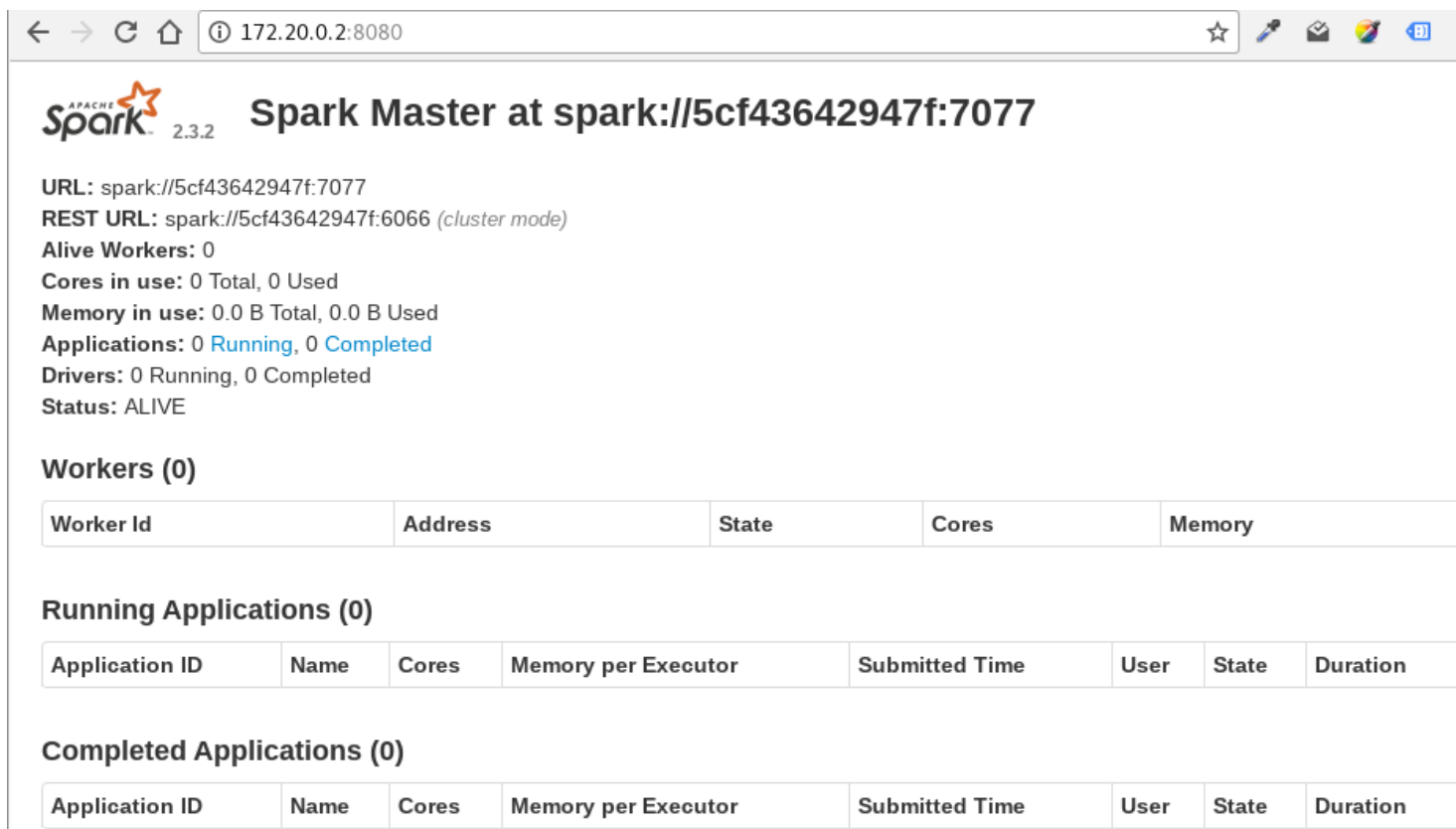
It is possible to check the master's web ui by getting the container's IP:

```
$ root@5cf43642947f:/home# hostname -i  
$ 172.20.0.2
```

If we are running as daemon:

```
$ docker exec master hostname -i  
$ 172.20.0.2
```

Master Web frontend



← → ↻ 🏠 ⓘ 172.20.0.2:8080 ☆ 🔍 📧 🎨 📄 ⋮

APACHE Spark 2.3.2 Spark Master at spark://5cf43642947f:7077

URL: spark://5cf43642947f:7077
 REST URL: spark://5cf43642947f:6066 (cluster mode)
 Alive Workers: 0
 Cores in use: 0 Total, 0 Used
 Memory in use: 0.0 B Total, 0.0 B Used
 Applications: 0 [Running](#), 0 [Completed](#)
 Drivers: 0 Running, 0 Completed
 Status: ALIVE

Workers (0)

Worker Id	Address	State	Cores	Memory
No workers are currently active.				

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
No running applications are currently active.							

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
No completed applications are currently active.							



Mac users be careful

if you are running Docker on Mac, you might need to map the container port 8080 to a local port:

```
$ docker run -it \  
    --name master \  
    --network spark-network \  
    -p 9999:8080 \  
    thiagolcmelo/spark-debian \  
    /bin/bash
```

```
root@1e315e2d5e20:/home# start-master
```

And now the URL must be `http://localhost:9999`

Creation of workers


```
$ docker run -d -t \  
    --name worker-1 \  
    --network spark-network \  
    thiagolcmelo/spark-debian  
$ docker exec worker-1 start-slave spark://master:7077
```

Note: Since we named the master node container as “master”, we can refer to it using its name, at least for containers inside the spark-network. It is also possible to use the master’s IP.



Workers in the frontend as members of the cluster

← → ↻ 🏠 ⓘ 172.20.0.2:8080 ☆ 🔍 📧 🌐 📄 ⋮

 **Spark Master at spark://c7eedd25648c:7077**

URL: spark://c7eedd25648c:7077
REST URL: spark://c7eedd25648c:6066 (cluster mode)
Alive Workers: 1
Cores in use: 4 Total, 0 Used
Memory in use: 2.8 GB Total, 0.0 B Used
Applications: 0 [Running](#), 0 [Completed](#)
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores	Memory
worker-20181026191546-172.20.0.3-44193	172.20.0.3:44193	ALIVE	4 (0 Used)	2.8 GB (0.0 B Used)

Running Applications (0)

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

Completed Applications (0)

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

Workers comm port

- By default, the web ui port for all workers is the 8081. We can discover the worker's IP by doing:

```
$ docker exec worker-1 hostname -i
```

```
$ 172.20.0.3
```



The screenshot shows a web browser window with the address bar containing '172.20.0.3:8081'. The page title is 'Spark Worker at 172.20.0.3:44193'. The page content includes the Apache Spark logo (version 2.3.2) and the following information:

- ID: worker-20181026191546-172.20.0.3-44193
- Master URL: spark://c7eedd25648c:7077
- Cores: 4 (0 Used)
- Memory: 2.8 GB (0.0 B Used)


There is a blue link labeled 'Back to Master'. Below this, the section 'Running Executors (0)' is shown, which contains an empty table with the following columns: ExecutorID, Cores, State, Memory, Job Details, and Logs.

Add additional workers

```
$ docker run -d -t \  
    --name worker-2 \  
    --network spark-network \  
    thiagolcmelo/spark-debian  
$ docker exec worker-2 start-slave spark://master:7077  
  
$ docker run -d -t \  
    --name worker-3 \  
    --network spark-network \  
    thiagolcmelo/spark-debian  
$ docker exec worker-3 start-slave spark://master:7077
```


Workers... workers everywhere!!!

← → ↻ 🏠 ⓘ 172.20.0.2:8080 ☆ 🖨️ 📧 🌐 📄 ⋮

 **Spark Master at spark://c7eedd25648c:7077**

URL: spark://c7eedd25648c:7077
REST URL: spark://c7eedd25648c:6066 (cluster mode)
Alive Workers: 3
Cores in use: 12 Total, 0 Used
Memory in use: 8.3 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (3)

Worker Id	Address	State	Cores	Memory
worker-20181026191546-172.20.0.3-44193	172.20.0.3:44193	ALIVE	4 (0 Used)	2.8 GB (0.0 B Used)
worker-20181026192359-172.20.0.4-41767	172.20.0.4:41767	ALIVE	4 (0 Used)	2.8 GB (0.0 B Used)
worker-20181026192415-172.20.0.5-36185	172.20.0.5:36185	ALIVE	4 (0 Used)	2.8 GB (0.0 B Used)

Running Applications (0)

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

Completed Applications (0)

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

Check containers using docker ps

- We can also make a sanity check using Docker cli:

\$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b1cf2ea57a1c	thiagolcmelo/spark-debian	"bash"	2 minutes ago	Up 2 minutes	22/tcp, 7000-8000/tcp, 8080-8081/tcp	worker-3
d2c209c4722f	thiagolcmelo/spark-debian	"bash"	2 minutes ago	Up 2 minutes	22/tcp, 7000-8000/tcp, 8080-8081/tcp	worker-2
500689c690b7	thiagolcmelo/spark-debian	"bash"	10 minutes ago	Up 10 minutes	22/tcp, 7000-8000/tcp, 8080-8081/tcp	worker-1
c7eedd25648c	thiagolcmelo/spark-debian	"bash"	11 minutes ago	Up 11 minutes	22/tcp, 7000-8000/tcp, 8080-8081/tcp	master



Submission of a simple application

Issue the python –version command

```
$ docker exec master python --version
```

```
Python 3.5.3
```

Unpack spark

- unpacking the Spark files to a folder named as spark:

```
$ tar -xzvf downloads/spark-2.3.2-bin-hadoop2.7.tgz .
```

```
$ mv spark-2.3.2-bin-hadoop2.7 spark
```

The first submission

The first trial can be one of the Spark examples:

```
$ ./spark/bin/spark-submit \  
  --master spark://172.20.0.2:7077 \  
  spark/examples/src/main/python/pi.py 1000
```

The number 1000 in the end means that the job will be split in 1000 pieces which can be processed in parallel.

If we return to the master's web ui during the processing, we can see that there is an application running: