Seguimos el tutorial de instalación. Para ello, primeramente requerimos de un container o virtual machine manager. En nuestro caso instalamos docker:

- Primero actualizamos *apt* e instalamos los paquetes que le permiten usar repositorios https:

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
```

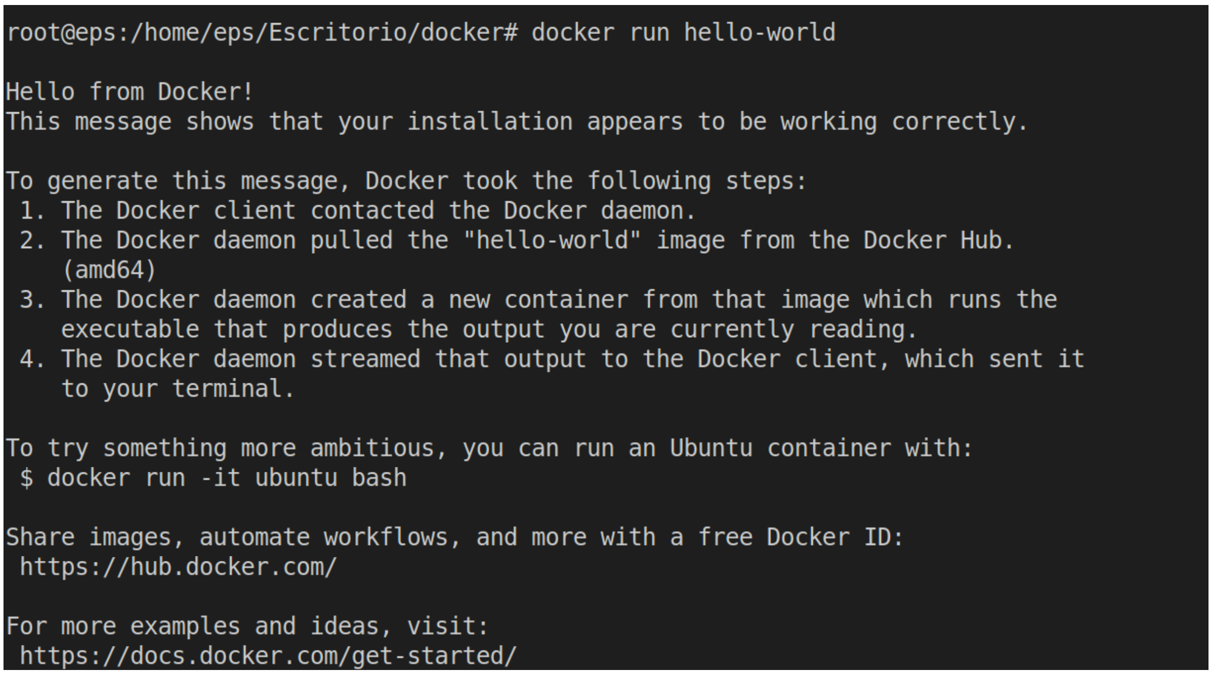
- Añadimos *official GPG key*:

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
        sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

- Finalmente instalamos *docker*:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Para comprobar, podemos ejecutar '*docker run hello-world'*:



```
root@eps:/home/eps/Escritorio/docker# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Una vez tenemos *docker* instalado, volvemos con la instalación de *minikube*:

- Descargamos la última versión estable de minikube:

```
curl -LO \
      https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

- Y a continuación lo instalamos mediante el comando:

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Con todo esto, podemos arrancar *minikube*:

- Para ello, creamos un usuario con permisos:

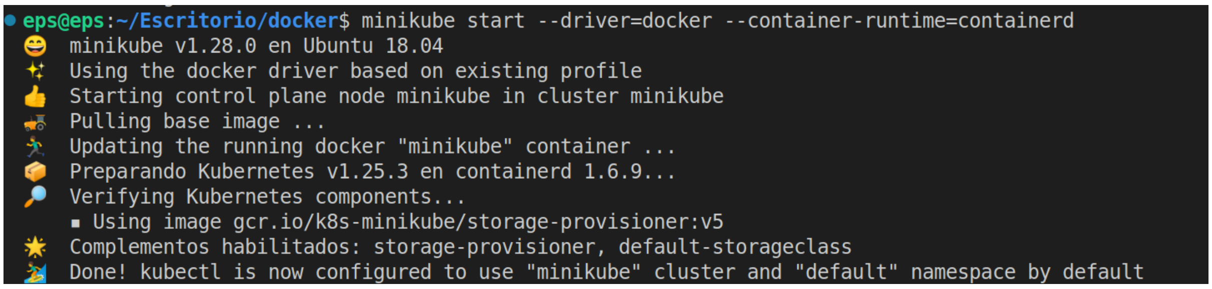
  sudo usermod -a -G docker $USER
  sudo chown -R $USER $HOME/.minikube; chmod -R u+wrx $HOME/.minikube

- Y para poder acceder al config:

  sudo chown -R $USER /home/eps/.kube

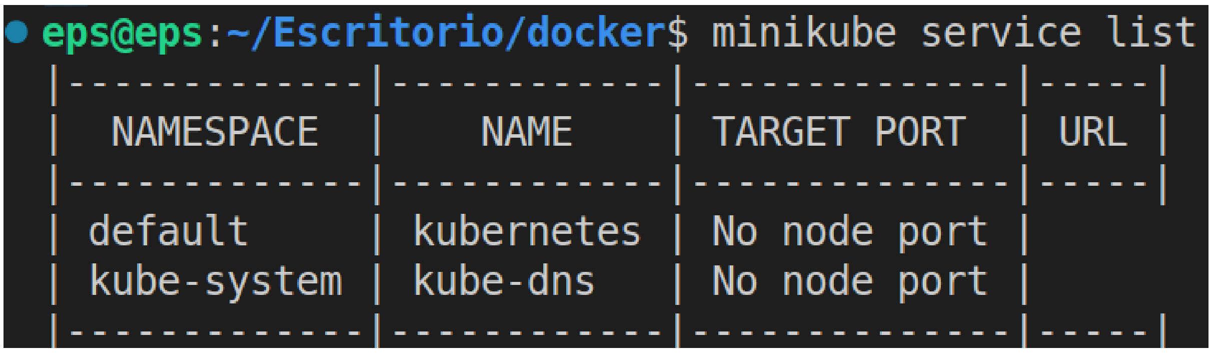
- Finalmente podemos arrancar *minikube*:

  minikube start --driver=docker --container-runtime=containerd



```
eps@eps:~/Escritorio/docker$ minikube start --driver=docker --container-runtime=containerd
😄  minikube v1.28.0 en Ubuntu 18.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🏃  Updating the running docker "minikube" container ...
📦  Preparando Kubernetes v1.25.3 en containerd 1.6.9...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Complementos habilitados: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Para comprobar que se guardó correctamente ejecutamos '*minikube service list*':



```
eps@eps:~/Escritorio/docker$ minikube service list
|-------------|------------|--------------|-----|
|  NAMESPACE  |    NAME    | TARGET PORT  | URL |
|-------------|------------|--------------|-----|
| default     | kubernetes | No node port |
| kube-system | kube-dns   | No node port |
|-------------|------------|--------------|-----|
```

Tras la instalación de *minikube* y *docker*, procedemos con la creación del '*base_dockerfile*':

- Nos indican que debe estar basado en ubuntu 22.04:

    FROM ubuntu:22.04

- Luego pasamos a modificar las variables de entorno:

```
ENV SPARK_VERSION=3.3.1
ENV HADOOP_VERSION=3.3.4
ENV JDK_VERSION=11
```

- Creamos la carpeta */opt* en caso de que no exista. Actualizamos *apt-get* e instalamos *wget* para descargar *hadoop* y *spark*

```
RUN mkdir -p /opt
RUN apt-get update && apt-get install -y wget
```

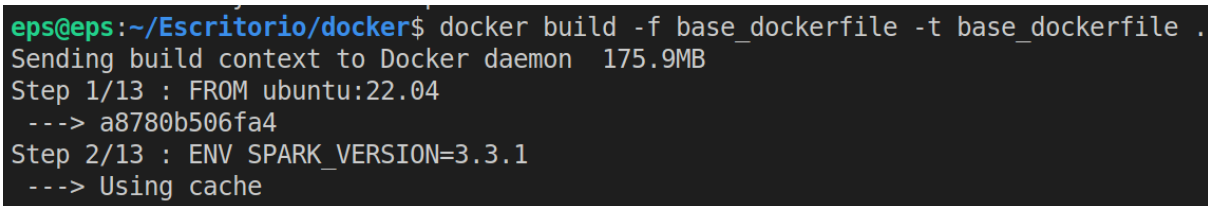
- Finalmente descargamos *openjdk*, *hadoop* y *spark*:

```
RUN apt-get install -y openjdk-${JDK_VERSION}-jdk
RUN wget
https://downloads.apache.org/spark/spark-${SPARK_VERSION}/spark-${SPARK
_VERSION}-bin-hadoop3.tgz
RUN tar -xzf spark-${SPARK_VERSION}-bin-hadoop3.tgz --no-same-owner; mv
spark-${SPARK_VERSION}-bin-hadoop3 /opt
RUN wget
https://dlcdn.apache.org/hadoop/common/hadoop-${HADOOP_VERSION}/hadoop-
${HADOOP_VERSION}-src.tar.gz
RUN tar -xzf hadoop-${HADOOP_VERSION}-src.tar.gz; mv
hadoop-${HADOOP_VERSION}-src /opt
```

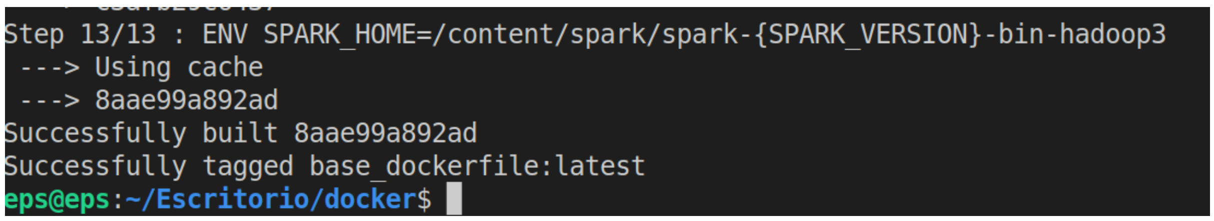
- Por último, creamos nuevas variables de entorno:

```
ENV JAVA_HOME=/usr/lib/jvm/java-${JDK_VERSION}-openjdk-amd64
ENV SPARK_HOME=/opt/spark-${SPARK_VERSION}-bin-hadoop3
```

Una vez acabado, construimos la *base* con un tag que nos permite su uso en las imágenes de *worker* y *master*:

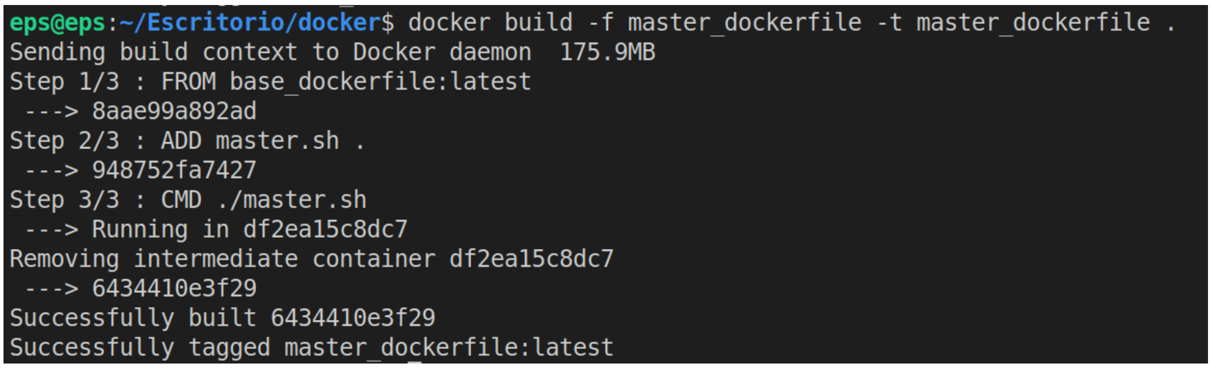


```
eps@eps:~/Escritorio/docker$ docker build -f base_dockerfile -t base_dockerfile .
Sending build context to Docker daemon  175.9MB
Step 1/13 : FROM ubuntu:22.04
 ---> a8780b506fa4
Step 2/13 : ENV SPARK_VERSION=3.3.1
 ---> Using cache
```

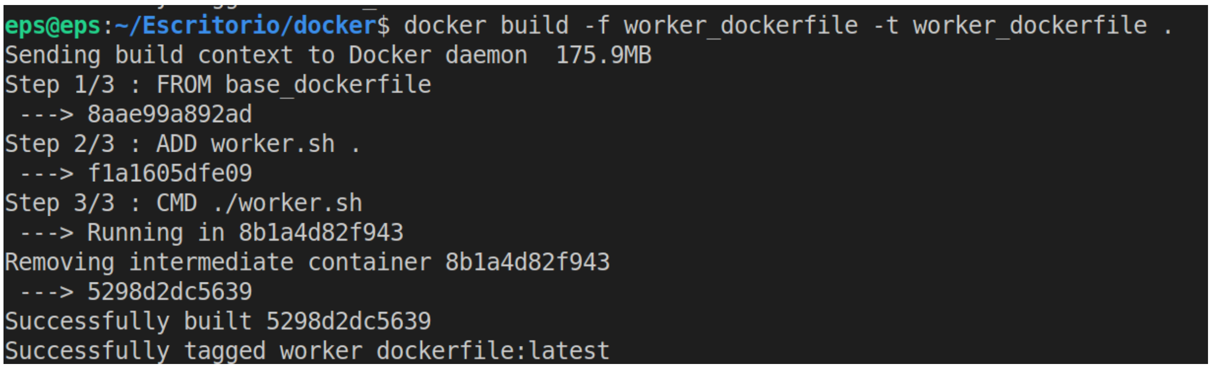


```
Step 13/13 : ENV SPARK_HOME=/content/spark/spark-{SPARK_VERSION}-bin-hadoop3
 ---> Using cache
 ---> 8aae99a892ad
Successfully built 8aae99a892ad
Successfully tagged base_dockerfile:latest
eps@eps:~/Escritorio/docker$
```

Pasamos a la creación del *master* y el *worker*, en los que únicamente heredamos del base, añadimos el script correspondiente y lo ejecutamos (mucha información es obtenida de caché al estar recreando pasos ya realizados con anterioridad):

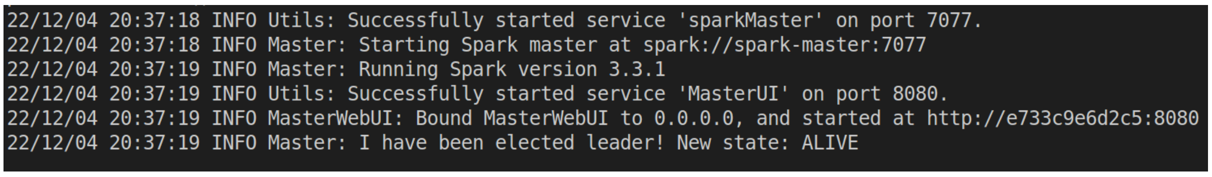


```
eps@eps:~/Escritorio/docker$ docker build -f master_dockerfile -t master_dockerfile .
Sending build context to Docker daemon  175.9MB
Step 1/3 : FROM base_dockerfile:latest
 ---> 8aae99a892ad
Step 2/3 : ADD master.sh .
 ---> 948752fa7427
Step 3/3 : CMD ./master.sh
 ---> Running in df2ea15c8dc7
Removing intermediate container df2ea15c8dc7
 ---> 6434410e3f29
Successfully built 6434410e3f29
Successfully tagged master_dockerfile:latest
```



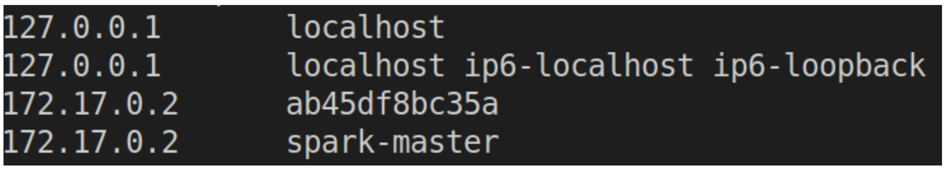
```
eps@eps:~/Escritorio/docker$ docker build -f worker_dockerfile -t worker_dockerfile .
Sending build context to Docker daemon  175.9MB
Step 1/3 : FROM base_dockerfile
 ---> 8aae99a892ad
Step 2/3 : ADD worker.sh .
 ---> f1a1605dfe09
Step 3/3 : CMD ./worker.sh
 ---> Running in 8b1a4d82f943
Removing intermediate container 8b1a4d82f943
 ---> 5298d2dc5639
Successfully built 5298d2dc5639
Successfully tagged worker_dockerfile:latest
```

Y finalmente, al correr el *master* obtenemos lo siguiente:

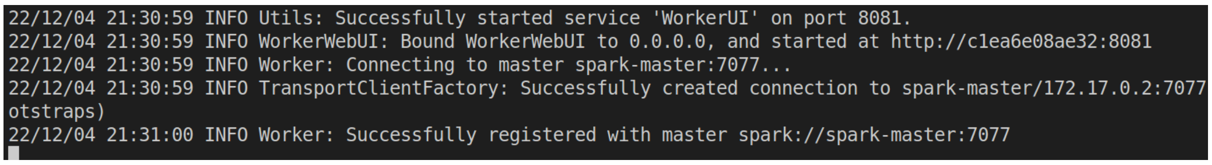


```
22/12/04 20:37:18 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
22/12/04 20:37:18 INFO Master: Starting Spark master at spark://spark-master:7077
22/12/04 20:37:19 INFO Master: Running Spark version 3.3.1
22/12/04 20:37:19 INFO Utils: Successfully started service 'MasterUI' on port 8080.
22/12/04 20:37:19 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://e733c9e6d2c5:8080
22/12/04 20:37:19 INFO Master: I have been elected leader! New state: ALIVE
```

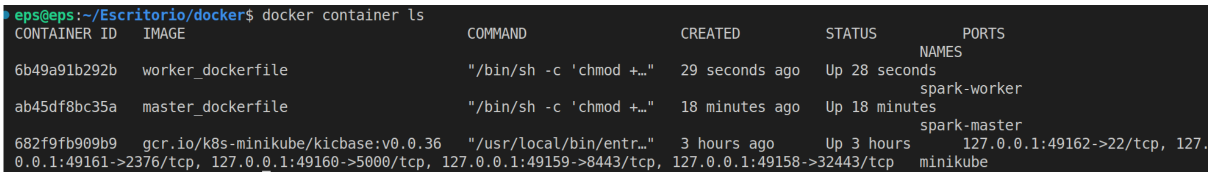
Como podemos ver en los *hosts*, se ve el *spark-master*:



```
127.0.0.1       localhost
127.0.0.1       localhost ip6-localhost ip6-loopback
172.17.0.2      ab45df8bc35a
172.17.0.2      spark-master
```



```
22/12/04 21:30:59 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
22/12/04 21:30:59 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://c1ea6e08ae32:8081
22/12/04 21:30:59 INFO Worker: Connecting to master spark-master:7077...
22/12/04 21:30:59 INFO TransportClientFactory: Successfully created connection to spark-master/172.17.0.2:7077
otstraps)
22/12/04 21:31:00 INFO Worker: Successfully registered with master spark://spark-master:7077
```



```
eps@eps:~/Escritorio/docker$ docker container ls
 CONTAINER ID   IMAGE                                 COMMAND                  CREATED          STATUS          PORTS
                                                                                                            NAMES
 6b49a91b292b   worker_dockerfile                     "/bin/sh -c 'chmod +…"   29 seconds ago   Up 28 seconds
                                                                                                            spark-worker
 ab45df8bc35a   master_dockerfile                     "/bin/sh -c 'chmod +…"   18 minutes ago   Up 18 minutes
                                                                                                            spark-master
 682f9fb909b9   gcr.io/k8s-minikube/kicbase:v0.0.36   "/usr/local/bin/entr…"   3 hours ago      Up 3 hours      127.0.0.1:49162->22/tcp, 127.
 0.0.1:49161->2376/tcp, 127.0.0.1:49160->5000/tcp, 127.0.0.1:49159->8443/tcp, 127.0.0.1:49158->32443/tcp   minikube
```

Para que ambas imágenes tengan conocimiento de la otra usamos:

docker run --add-host spark-master:172.17.0.2 -it --name spark-worker worker_dockerfile

A continuación creamos el *.yaml* del servicio y de los pods de *worker* y *master*.
Una vez creados, lanzamos los deployments:

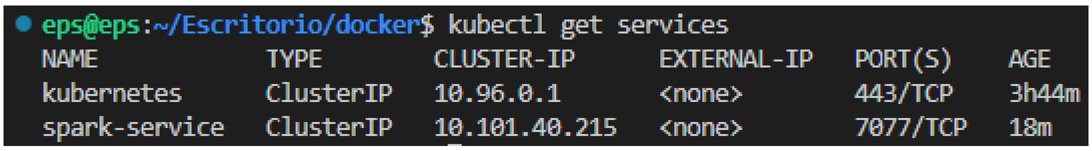
```
kubectl apply -f master.yaml; kubectl apply -f worker.yaml
```

Podemos comprobarlos usando el comando “*kubectl get pods*”:

Y lanzamos el servicio:

```
kubectl apply -y service.yaml
```

Para comprobarlo, podemos conseguir la lista de servicios con “*kubectl get services*”:



```
eps@eps:~/Escritorio/docker$ kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP    3h44m
spark-service   ClusterIP   10.101.40.215   <none>        7077/TCP   18m
```