Docker+Spark+SpringBoot

一、Centos集群规划

3台主机：1个master、2个slaver

master：

主机名： master、ip地址： 172.18.1.2

slaver1：

主机名： slaver1、ip地址： 172.18.1.3

主机名： slaver2、ip地址： 172.18.1.4

二、Centos安装Docker

1、 Docker 要求 CentOS 系统的内核版本高于 3.10 ， 查看本页面的前提条件来验证你的CentOS 版本是否支持 Docker 。

```
uname -r
```

2、使用 `root` 权限登录 Centos。确保 yum 包更新到最新。

```
sudo yum update
```

3、卸载旧版本(如果安装过旧版本的话)

```
sudo yum remove docker  docker-common docker-selinux docker-engine
```

4、安装需要的软件包， yum-util 提供yum-config-manager功能，另外两个是devicemapper驱动依赖的

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

5、 设置yum源

```
sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
```

6、可以查看所有仓库中所有docker版本，并选择特定版本安装

```
yum list docker-ce --showduplicates | sort -r
```

7、安装docker

```
 #sudo yum install docker-ce
```

由于repo中默认只开启stable仓库，故这里安装的是最新稳定版17.12.0

```
sudo yum install docker-ce-17.12.0.ce
```

8、启动并加入开机启动

```
sudo systemctl start docker
```

```
sudo systemctl enable docker
```

9、验证安装是否成功(有client和service两部分表示docker安装启动都成功了)

```
docker version
```

三、Docker安装Centos7

1、获取centos7镜像

```
docker search centos
```

```
docker pull centos
```

2、查看镜像列表的命令：

```
docker images
```

3、构建SSH的Centos镜像

3.1以centos7镜像为基础，构建一个带有SSH功能的centos

```
 vi Dockerfile
```

内容：

```
FROM centos

MAINTAINER xxx@126.com



RUN yum install -y openssh-server sudo

RUN sed -i 's/UsePAM yes/UsePAM no/g' /etc/ssh/sshd_config

\#RUN echo "UsePAM no" >> /etc/ssh/sshd_config
```

```
RUN yum install -y openssh-clients



RUN echo "root:123456" | chpasswd

RUN echo "root  ALL=(ALL)    ALL" >> /etc/sudoers

RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key -N ""

RUN ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ""

RUN ssh-keygen -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ""

RUN ssh-keygen -t dsa -f /etc/ssh/ssh_host_ed25519_key -N ""

RUN mkdir /var/run/sshd

EXPOSE 22

CMD ["/usr/sbin/sshd", "-D"]
```

这段内容的大意是：以 centos 镜像为基础，安装SSH的相关包，设置了root用户的密码为 123456，并启动SSH服务

执行构建镜像的命令，新镜像命名为 centos7-ssh

```
$ docker build -t centos7-ssh .
```

执行完成后，可以在镜像列表中看到

```
$ docker images
```

之后就可以启动容器，然后逐个登录容器并上传相应的jdk和hadoop包进行hadoop安装部署。

四、基于上面构建的ssh镜像构建Hadoop（含spark）镜像

上面是运行了3个centos容器，需要在每个容器中单独安装Hadoop环境，我们可以像构建SSH镜像一样，构建一个Hadoop镜像，然后运行3个Hadoop容器，这样就更简单了

这里是基于 centos7-ssh 这个镜像，把 JAVA 和 Hadoop 的环境都配置好了。

前提：在Dockerfile所在目录下准备好 jdk-8u131-linux-x64.tar.gz 与 hadoop-2.7.3.tar.gz、scala-2.11.8.tgz、spark-2.2.2-bin-hadoop2.7.tgz

把之前的Dockerfile清空，新建：

```
$ vi Dockerfile
```

内容：

```
FROM centos7-ssh
```

```
ADD jdk-8u131-linux-x64.tar.gz /usr/local/

RUN mv /usr/local/jdk1.8.0_131 /usr/local/jdk1.8

ENV JAVA_HOME=/usr/local/jdk1.8

ENV PATH=$JAVA_HOME/bin:$PATH


ADD hadoop-2.7.3.tar.gz /usr/local

RUN mv /usr/local/hadoop-2.7.3 /usr/local/hadoop

ENV HADOOP_HOME=/usr/local/hadoop

ENV PATH=$HADOOP_HOME/bin:$PATH


ADD scala-2.11.8.tgz /usr/local

ENV SCALA_HOME=/usr/local/scala-2.11.8

ENV PATH=$SCALA_HOME/bin:$PATH


ADD spark-2.2.2-bin-hadoop2.7.tgz /usr/local

RUN mv /usr/local/spark-2.2.2-bin-hadoop2.7 /usr/local/spark

ENV SPARK_HOME=/usr/local/spark

ENV PATH=$SPARK_HOME/bin:$PATH


\# 用echo添加多行内容

RUN echo $'#java\n\

JAVA_HOME=/usr/local/jdk1.8\n\

PATH=$JAVA_HOME/bin:$PATH\n\

CLASSPATH=.:$JAVA_HOME/lib/rt.jar\n\

export JAVA_HOME PATH CLASSPATH\n\


\#hadoop\n\

HADOOP_HOME=/usr/local/hadoop\n\

PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH\n\

export HADOOP_HOME PATH\n\

\#export HADOOP_ROOT_LOGGER=DEBUG,console\n\
```

```
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native\n\

export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native\n\

export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"\n\


\#scala\n\

export SCALA_HOME=/usr/local/scala-2.11.8\n\

export PATH=$SCALA_HOME/bin:$PATH\n\


\#spark\n\

export SPARK_HOME=/usr/local/spark\n\

export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin\n'\

\>> /root/.bash_profile


RUN yum install -y which sudo
```

执行构建命令，新镜像命名为 hadoop

```
$ docker build -t hadoop .
```

Docker创建容器时默认采用bridge网络，自行分配ip，不允许自己指定。

在实际部署中，我们需要指定容器ip，不允许其自行分配ip，尤其是搭建集群时，固定ip是必须的。我们可以创建自己的bridge网络： hadoopnet，创建容器的时候指定网络为hadoopnet并指定ip即可。

查看网络模式

```
docker network ls
```

创建一个新的bridge网络

```
docker network create --driver bridge --subnet=172.18.1.0/16 --
gateway=172.18.1.1 hadoopnet
```

查看网络信息

```
docker network inspect hadoopnet
```

我们需要在3台容器中的/etc/hosts文件中添加3台主机的主机名和ip地址对应信息，我们不需要进入容器手动修改，在运行容器时直接使用选项即可。对应的信息为：

172.18.1.2 hadoop2

172.18.1.3 hadoop3

172.18.1.4 hadoop4

在docker中直接修改/etc/hosts文件，在重启容器后会被重置、覆盖。因此需要通过容器启动脚本docker run的--add-host参数将主机和ip地址的对应关系传入，容器在启动后会写入hosts文件中。如：

```
docker run -it -d --name hadoop2test --add-host hadoop2:172.18.1.2 --add-host hadoop3:172.18.1.3 --add-host hadoop4:172.18.1.4 hadoop
```

```
 docker exec -it hadoop2test bash
```

```
$ ip addr
```

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
$ chmod 0600 ~/.ssh/authorized_keys
```

4、1 hadoop部署

1.在slaves文件中定义工作节点

在hadoop根目录下的etc/hadoop目录下修改slaves文件，添加工作节点主机信息。

```
docker exec -it hadoop2test bash
```

按照步骤一中的主机规划，工作节点主机为hadoop3和hadoop4两台主机。如：

```
[root@1d434392f05d /]# cat > /usr/local/hadoop/etc/hadoop/slaves<<EOF

hadoop3

hadoop4

EOF
```

2、修改配置文件信息

a、hadoop-env.sh，添加JAVA_HOME信息

```
[root@hadoop2 hadoop]# cat /usr/local/hadoop/etc/hadoop/hadoop-env.sh |grep
JAVA_HOME

*# The only required environment variable is JAVA_HOME. All others are*

*# set JAVA_HOME in this file, so that it is correctly defined on*

*export JAVA_HOME=${JAVA_HOME}*

sed -i 's/export JAVA_HOME=\${JAVA_HOME}/export
JAVA_HOME=\/usr\/local\/jdk1.8/g' /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

b、core-site.xml

```
# cd /usr/local/hadoop/etc/hadoop

[root@hadoop2 hadoop]# vi core-site.xml

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://hadoop2:9000</value>

</property>

<property>

<name>io.file.buffer.size</name>

<value>131072</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadoop/tmp</value>

<description>Abase for other temporary directories.</description>

</property>

</configuration>
```

c、hdfs-site.xml

```
# cd /usr/local/hadoop/etc/hadoop
```

```
[root@hadoop2 hadoop]# vi hdfs-site.xml

<configuration>

****

**dfs.namenode.http-address**

**hadoop2:50070**

****

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>hadoop2:9001</value>

<description># web HDFS</description>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/home/hadoop/dfs/name</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/home/hadoop/dfs/data</value>

</property>

<property>

<name>dfs.replication</name>

<value>2</value>

<description># Block 2</description>

</property>

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

</configuration>
```

d、yarn-site.xml

```
# cd /usr/local/hadoop/etc/hadoop

[root@hadoop2 hadoop]# vi yarn-site.xml

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>hadoop2:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>hadoop2:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>hadoop2:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>hadoop2:8033</value>
```

```
</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>hadoop2:8088</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>2048</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>1</value>

</property>

</configuration>
```

e、mapred-site.xml

```
 #cd /usr/local/hadoop/etc/hadoop

[root@hadoop2 hadoop]# vi mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>hadoop2:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>
```

```
<value>hadoop2:19888</value>

</property>

</configuration>
```

f、修改启动、停止文件

```
cd /usr/local/hadoop/sbin
```

修改启动和停止文件，在末尾添加：

start-dfs.sh 和 stop-dfs.sh

```
cat >> start-dfs.sh<<EOF

HDFS_DATANODE_USER=root

\#HADOOP_SECURE_DN_USER=hdfs

HDFS_NAMENODE_USER=root

HDFS_SECONDARYNAMENODE_USER=root

HDFS_DATANODE_SECURE_USER=hdfs

EOF
```

```
cat >> stop-dfs.sh<<EOF

HDFS_DATANODE_USER=root

\#HADOOP_SECURE_DN_USER=hdfs

HDFS_NAMENODE_USER=root

HDFS_SECONDARYNAMENODE_USER=root

HDFS_DATANODE_SECURE_USER=hdfs

EOF
```

```
start-yarn.sh  和stop-yarn.sh

cat >> start-yarn.sh<<EOF

YARN_RESOURCEMANAGER_USER=root

HADOOP_SECURE_DN_USER=yarn

YARN_NODEMANAGER_USER=root

EOF
```

```
cat >> stop-yarn.sh<<EOF

YARN_RESOURCEMANAGER_USER=root

HADOOP_SECURE_DN_USER=yarn

YARN_NODEMANAGER_USER=root

EOF
```

退出。

注意：

以上步骤完成以后停止当前容器，并使用docker命令保持到一个新的镜像。使用新的镜像重新启动集群，这样集群每台机器都有相同的账户、配置和软件，无需再重新配置。如：

a、停止容器

```
docker stop hadoop2test
```

b、保存镜像

```
docker commit hadoop2test hadoop_me:v2.0
```

```
root@f-virtual-machine:~# docker images
```

```
REPOSITORY      TAG         IMAGE ID      CREATED       SIZE

hadoop_me       v2.0        9dd2a5ea1ee6    4 seconds ago    1.71GB

hadoop        latest       46f3bcaf5a93    6 minutes ago    1.71GB

centos7-ssh      latest       6f675e46be2e    7 minutes ago    271MB

centos        latest       300e315adb2f    3 months ago    209MB

portainer/portainer  latest       62771b0b9b09    7 months ago    79.1MB
```

4、2运行容器并验证

1、了解端口映射

集群启动后，需要通过web界面观察集群的运行情况，因此需要将容器的端口映射到宿主主机的端口上，可以通过docker run命令的-p选项完成。比如：

将yarn任务调度端口映射到宿主主机8088端口上：

例如docker run -it -p 8088:8088 hadoop_me:v2.0其他端口类似。

2、从新镜像启动3个容器

```
docker run --name hadoop2 -h hadoop2 --network=hadoopnet --ip 172.18.1.2 -d -p 50070:50070 -p 9001:9001 -p 5002:22 -p 9870:9870 -p 8088:8088 -p 19888:19888 hadoop_me:v2.0
```

```
docker run --name hadoop3 -h hadoop3 --network=hadoopnet --ip 172.18.1.3 -d -p 5003:22 -p 8042:8042 hadoop_me:v2.0
```

```
docker run --name hadoop4 -h hadoop4 --network=hadoopnet --ip 172.18.1.4 -d -p 5004:22 -p 8043:8042 hadoop_me:v2.0
```

3.格式化

进入hadoop2：

```
docker exec -it hadoop2 bash
```

或者

```
ssh hadoop2
```

进入到/usr/local/hadoop/bin目录下

```
cd /usr/local/hadoop/bin
```

执行格式化命令

```
hdfs namenode -format
```

4.在master主机上执行start-all.sh脚本启动集群

在master上 /usr/local/hadoop/sbin 目录下执行：

```
cd /usr/local/hadoop/sbin
```

```
./start-all.sh
```

#如果没设置StrictHostKeyChecking no第一次启动需要输入几次yes

```
[root@hadoop2 sbin]# ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [hadoop2]
hadoop2: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-hadoop2.out
hadoop3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-hadoop3.out
hadoop4: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-hadoop4.out
Starting secondary namenodes [hadoop2]
hadoop2: starting secondarynamenode, logging to /usr/local/hadoop/logs/hadoop-root-secondarynamenode-hadoop2.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn--resourcemanager-hadoop2.out
hadoop4: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop4.out
hadoop3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-hadoop3.out
```

```
[root@hadoop2 sbin]# jps
1392 NameNode
2019 Jps
1592 SecondaryNameNode
1757 ResourceManager
[root@hadoop2 sbin]# ssh hadoop3
Last login: Fri Mar 12 06:35:29 2021 from 172.18.1.2
[root@hadoop3 ~]# jps
387 NodeManager
532 Jps
285 DataNode
[root@hadoop3 ~]# ssh hadoop4
Last login: Fri Mar 12 06:35:40 2021 from 172.18.1.3
[root@hadoop4 ~]# jps
487 NodeManager
632 Jps
383 DataNode
```

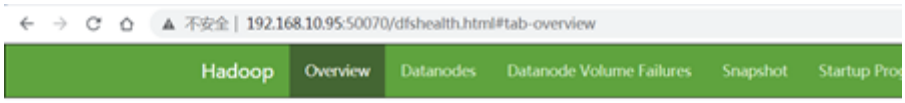
5.执行应用程序测试下：

```
[root@hadoop2 sbin]# hadoop fs -ls /
```

```
[root@hadoop2 sbin]# cd /usr/local/hadoop/share/hadoop/mapreduce/
```

```
[root@hadoop4 ~]# hadoop jar hadoop-mapreduce-examples-2.7.3.jar pi 2 1000
```

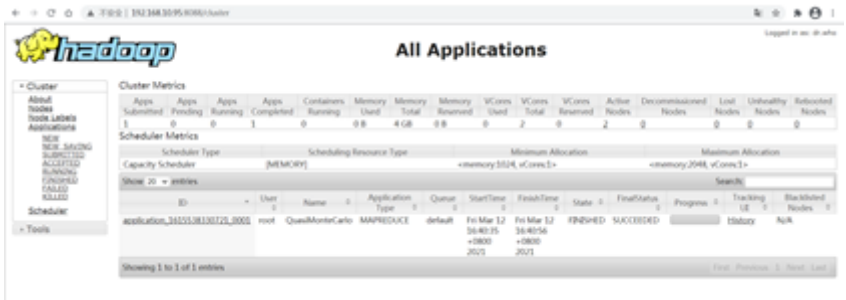
7.通过web页面访问



Overview 'hadoop2:9000' (active)

Started:	Fri Mar 12 08:38:23 UTC 2021
Version:	2.7.3, rbaa91f7c6bc9cb92be5982de4719c1c8af91ccff
Compiled:	2016-08-18T01:41Z by root from branch-2.7.3
Cluster ID:	CID-dd2fa6f6-a568-42f1-b83f-ec85a7474eaa
Block Pool ID:	BP-78281373-172.18.1.2-1615538285998

Summary



五、Spark2.2.2单机配置（root用户）（只Master节点操作）

以下步骤操作都在Master节点进行（hadoop2上面）。

1）前面Dockerfile已重构实现上传和解压spark并修改目录名称。

2）修改相应的配置文件。

```
root@f-virtual-machine:~# docker exec -it hadoop2 bash
```

复制spark-env.sh.template成spark-env.sh

```
  cd $SPARK_HOME/conf/
```

```
  cp spark-env.sh.template spark-env.sh
```

修改$SPARK_HOME/conf/spark-env.sh，末尾添加如下内容：

```
cat >> /usr/local/spark/conf/spark-env.sh<<EOF

export JAVA_HOME=/usr/local/jdk1.8

export SCALA_HOME=/usr/local/scala-2.11.8

export HADOOP_HOME=/usr/local/hadoop

export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop

export SPARK_MASTER_IP=172.18.1.2

export SPARK_MASTER_HOST=172.18.1.2

export SPARK_LOCAL_IP=172.18.1.2

export SPARK_WORKER_MEMORY=1g

export SPARK_WORKER_CORES=2

export SPARK_HOME=/usr/local/spark

export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)

EOF
```

验证：

```
spark-submit --master yarn --deploy-mode cluster --class
org.apache.spark.examples.SparkPi /usr/local/spark/examples/jars/spark-
examples_2.11-2.2.2.jar 1000
```

六、关于spark 以client模式运行虚拟内容超限的问题

解决方法：

修改hadoop的yarn-site.xml配置。（避免虚拟内容超限问题出现）

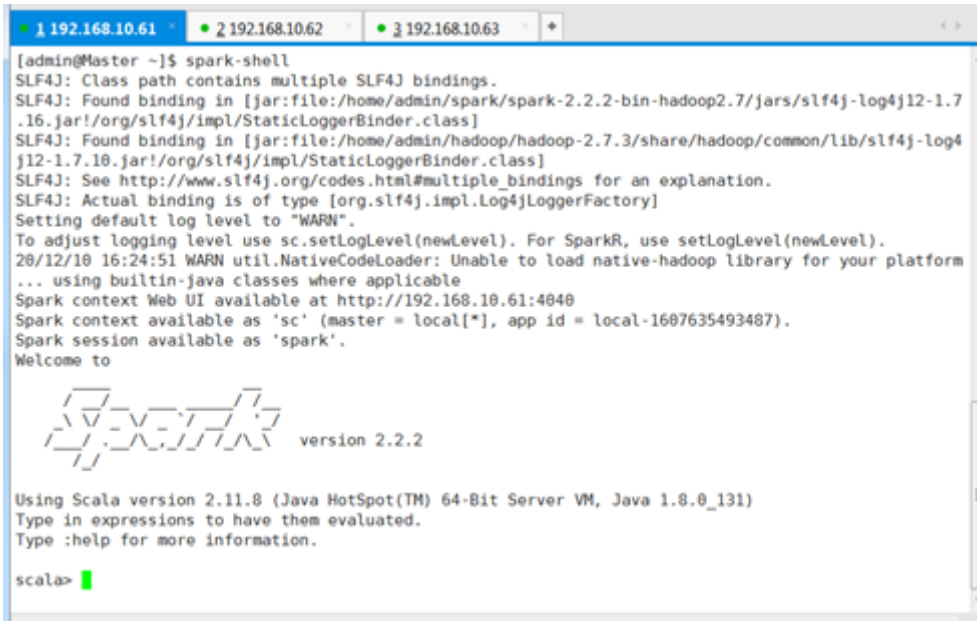
vi $HADOOP_HOME/etc/hadoop/yarn-site.xml

在 内末尾添加如下内容：

```
  <property>

   <name>yarn.nodemanager.vmem-check-enabled</name>

   <value>false</value>

•    <description>Whether virtual memory limits will be enforced for containers</description>

  </property>
```

```
  <property>

   <name>yarn.nodemanager.vmem-pmem-ratio</name>

   <value>4</value>

   <description>Ratio between virtual memory to physical memory when setting memory limits for containers</description>

  </property>
```

运行spark-shell



七、Docker运行Springboot程序

7、1 上传项目修改Dockerfile文件

```
find / -name emp.jar
```

```
[root@localhost Dockerfile]# find / -name emp.jar
/tmp/emp.jar
```

```
cp -r /tmp/emp.jar  ./Dockerfile
```

```
cd  Dockerfile

vi  Dockerfile

FROM java:8
COPY *.jar /app.jar
CMD ["--server.port=8082"
EXPOSE 8082
ENTRYPOINT ["java","-jar","/app.jar"]
~
```

7、2 运行镜像，生成容器

```
docker build  -t  app666 .
```

```
Successfully tagged app666:latest
[root@localhost Dockerfile]# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
app666        latest    523267534a84   10 seconds ago   671MB
hadoop4       latest    196e41d0fc69   4 days ago       2.41GB
hadoop3       latest    188c877855c6   4 days ago       2.41GB
hadoop2       latest    dbb4ecb46786   4 days ago       2.2GB
hello-world   latest    d1165f221234   2 weeks ago      13.3kB
java          8         d23bdf5b1b1b   4 years ago      643MB
```

```
docker run -P 8082:8082 --name app-springboot-boot app666
```