Docker

# Docker安装

## 2.1、检测系统内核

使用 uname -r  命令检测系统内核，内核需在3.10以上。

$ uname -r

3.11.0-15-generic

## 2.2、更新apt源

$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D

打开 /etc/apt/sources.list.d/docker.list 文件查看是否有 docker.list 文件，若没有则创建，并在其中添加如下内容：

deb https://apt.dockerproject.org/repo ubuntu-precise main

更新apt包的index

$ sudo apt-get update

下载docker

$ sudo apt-get install docker-engine

开启docker服务

$ sudo service docker start

验证是否安装成功

$ docker info

# Docker命令

## 3.1、Docker镜像相关的命令

|  |  |  |
| --- | --- | --- |
| 命令 | 使用方法 | 功能 |
| pull/push | docker pull/push <image> | 下载、上传镜像 |
| images | docker images | 显示docker镜像 |
| import | docker import file|URL|- [REPOSITORY[:TAG]] | 导入docker镜像 |
| load | docker load file|URL|- [REPOSITORY[:TAG]] | 导入docker镜像 |
| save | docker save [OPTIONS] IMAGE [IMAGE...] | 导出docker镜像 |
| rmi | docker rmi IMAGE | 移除指定镜像 |
| tag | docker tag [OPTIONS] IMAGE[:TAG] NAME[:TAG] | 重命名镜像 |

Docker获取镜像：

docker pull ubuntu

docker默认是从Docker Hub上下载，Docker Hub是Docker官网的镜像仓库，也是世界上最大的镜像仓库，基本你能想到的镜像都能在其中找到。

Docker镜像的重命名：

可以使用tag命令对镜像重命名，如：

docker tag ubuntu myubuntu

将ubuntu镜像重命名成myubuntu

Docker镜像的导入、导出：

导出：docker save -o /opt/dockerimagesfile/ubuntu-14.04.tar ubuntu:14.04

将ubuntu:14.04镜像文件导出到/opt/dockerimagesfile文件夹中，并命名为ubuntu-14.04.tar

导入：docker load -i /opt/dockerimagesfile/ubuntu-14.04.tar

Docker 镜像的删除：

docker rmi ubuntu:14.04

在删除docker镜像前需将由此镜像的容器先删除。

push命令是推送镜像的意思，可将你做好的镜像推送至docker Hub上也可以推送至个人搭建的私人仓库中去。

## 3.2、Docker容器相关的命令

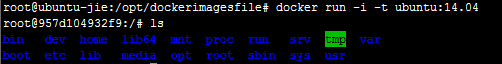
|  |  |  |
| --- | --- | --- |
| 命令 | 使用方法 | 功能 |
| run | docker run [OPTIONS] IMAGE | 从镜像中运行一个容器 |
| start/stop/restart | docker start/stop/restart CONTAINER | 开始、暂停、重启容器 |
| attach | docker attach [OPTIONS] CONTAINER | 进入正在后台运行的容器 |
| build | docker build [OPTIONS] PATH | 从Dockerfile中构建新的image |
| inspect | docker inspect CONTAINER|IMAGE [CONTAINER|IMAGE...] | 查看容器运行时详细信息，了解一个Image或者Container的完整构建信息 |
| kill | docker kill [OPTIONS] CONTAINER [CONTAINER...] | 杀死容器的进程 |
| ps | docker ps [-a] | 查看正在运行的容器，-a查看全部的容器 |
| rm | docker rm [OPTIONS] CONTAINER | 移除容器 |
| top | docker top CONTAINER [ps OPTIONS] | 显示容器内部的进程 |
| commit | docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]] | docker容器转换为docker镜像 |

运行容器：

docker run -i -t ubuntu:14.04

在ubuntu:14.04镜像的基础上开一个具有标准的输入和输出流，一个伪终端的容器，如下图：

其中， 957d104932f9 为此容器的id 。



暂停容器：

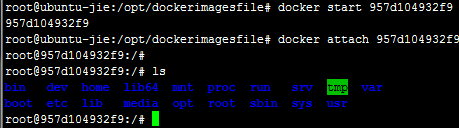
docker stop 957d104932f9

启动容器：

docker start 957d104932f9

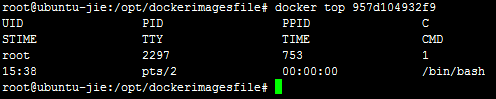
进入容器：

docker attach 957d104932f9



查看docker内部进程：

docker top 957d104932f9



移除容器：

docker rm 957d104932f9

容器转换为镜像：

docker commit 957d104932f9 myubuntu



# Dockerfile创建镜像

## 4.1、Dockerfile的基本语法

例子1：以ubuntu：14.04为基础创建一个带有openjdk7的镜像。

#test

#author jie

#install open-jdk-7

# Format: FROM repository[:version]

FORM ubuntu:14.04

# Format: MAINTAINER Name <email@addr.ess>

MAINTAINER Jie [<jie147@yahoo.com>](mailto:<jie147@yahoo.com>)

#update apt-get and install openjdk 7

RUN apt-get update && apt-get install openjdk-7-jre-headless

在此例中我们相当于有了一个能够运行java代码的容器，在例2中，将会将mongodb数据库安装在docker中，docker容器具有快速创建和启动等有点，是一个相当不错想法。

|  |  |  |
| --- | --- | --- |
| 命令 | 基本用法 | 功能 |
| FROM | FROM <Image> | 以image为基础 |
| MAINTAINER | MAINTAINER <author name> | Dockerfile作者 |
| RUN | RUN <common> | 容器中运行的命令 |
| ADD | ADD <src> <dest> | 将本机的文件添加至容器中 |
| CMD | CMD [“executable”,”param1”,  ”param2”]  or  CMD [“param1”,”param2”]  or  CMD command param1 param2 | CMD有三种格式： |
| EXPOSE | EXPOSE <port> | 暴露出容器端口 |
| ENTRYPOINT | ENTRYPOINT [‘executable’,  ’param1’,’param2’]  or  ENTRYPOINT command param1 param2 | 与CMD命令类似 |
| WORKDIR | WORKDIR /path/to/workdir | 容器的工作路径 |
| ENV | ENV <key> <value> | 环境变量 |
| UID | USER <UID> | 设置UID |
| VOLUME | VOLUME [‘/data’] | 添加数据卷 |

## 4.2、Dockerfile创建mongodb

例2：在ubuntu中创建一个mongodb数据库的Docker镜像

#test

#author jie

#build mongodb

#come from https://docs.docker.com/engine/examples/mongodb/

# Format: FROM repository[:version]

FROM ubuntu:14.04

# Format: MAINTAINER Name <email@addr.ess>

MAINTAINER Jie <jie147@yahoo.com>

# Installation:

# Import MongoDB public GPG key AND create a MongoDB list file

RUN apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10

RUN echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb\_release -sc)"/mongodb-org/3.0 multiverse" | tee /etc/apt/sources.list.d/mongodb-org-3.0.list

# Update apt-get sources AND install MongoDB

RUN apt-get update && apt-get install -y mongodb-org

# Create the MongoDB data directory

RUN mkdir -p /data/db

# Expose port 27017 from the container to the host

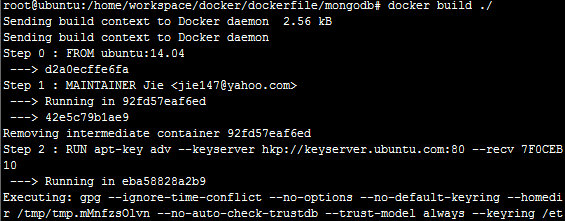
EXPOSE 27017

# Set usr/bin/mongod as the dockerized entry-point application

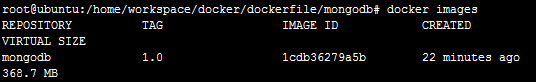
ENTRYPOINT ["/usr/bin/mongod"]

使用docker build 命令创建mongodb镜像

docker build ./

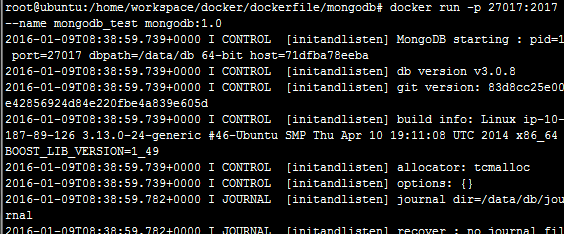


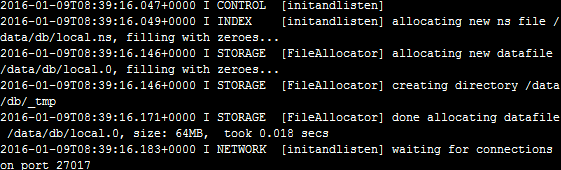
使用docker images查看镜像，存在一个nongodb：1.0



通过mongodb镜像创建容器，使用如下命令：

docker run -p 27017:2017 --name mongodb\_test mongodb:1.0





创建完成。

# Docker实现Spark集群

## 5.1、目标

## 5.1、基础镜像的制作

基础镜像是在ubuntu 14.04的基础上安装了jdk7，scala，hadoop2.6，spark1.4等软件。

使用Dockerfile创建dspark:basic镜像，Dockerfile内容如下：

FROM ubuntu:precise

# build dspark docker image

# Upgrade package index

# install a few other useful packages plus Open Jdk 7

# Remove unneeded /var/lib/apt/lists/\* after install to reduce the

# docker image size (by ~30MB)

RUN apt-get update && apt-get install -y less openjdk-7-jre-headless net-tools vim-tiny sudo openssh-server && rm -rf /var/lib/apt/lists/\*

#set environment

ENV SCALA\_HOME /opt/scala

ENV HADOOP\_HOME /opt/hadoop

ENV SPARK\_HOME /opt/spark

ENV PATH $HADOOP\_HOME/bin:$SPARK\_HOME/bin:$SCALA\_HOME/bin:$PATH

#install scala

ADD /home/pkg/scala/scala-2.11.7.tgz /opt

ADD /home/pkg/hadoop/hadoop-2.6.0.tar.gz /opt

ADD /home/pkg/spark/spark-1.4.1-bin-hadoop2.6.tgz /opt

RUN cd /opt && ln -s ./spark-1.4.1-bin-hadoop2.6 spark && ln -s ./hadoop-2.6.0 hadoop && ln -s ./scala-2.11.7 scala

# add some profiles : hadoop spark host

#

COPY ./hadoop/ /opt/hadoop/etc/hadoop/

COPY ./conf/ /opt/spark/conf/

# ssh login without password

RUN ssh-keygen -t dsa -P '' -f ~/.ssh/id\_dsa

RUN mkdir -p /opt/data/hadoop/dfs/name && mkdir -p /opt/data/hadoop/dfs/data

# Hdfs ports

EXPOSE 50010 50020 50070 50075 50090 8020

# Mapred ports

EXPOSE 19888 10020 9001 9010

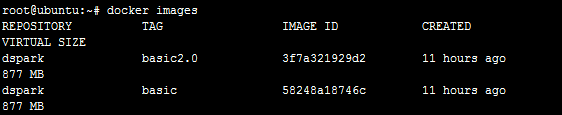
#Yarn ports

EXPOSE 8030 8031 8032 8033 8040 8042 8088

#SSH port

EXPOSE 22 8080

使用命令 docker build -t dspakr:basic ./ 创建镜像，并将镜像命名为dspark:basic .



接下来将使用dspark:basic 这个基础镜像制作出整个集群。

## 5.2、集群的制作

集群中包含一个master节点，两个slave节点。

设置MASTER\_IP=172.17.1.1，修改slaves文件，DockerFile如下：

FROM dspark:basic

# build dsparkmaster1:1.0

###

# Add slave

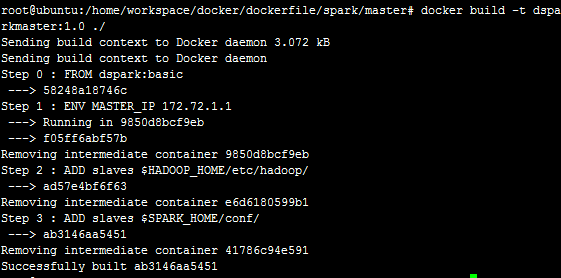
###

ENV MASTER\_IP 172.72.1.1

ADD slaves $HADOOP\_HOME/etc/hadoop/

ADD slaves $SPARK\_HOME/conf/

使用docker build -t dsparkmaster:1.0 ./ 创建镜像，创建过程如下：



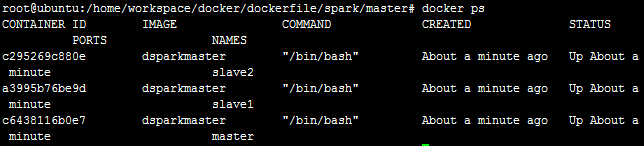
启动master容器：

docker run -dit -p 22 -p 8080 -p 8088 -p 50070 -m 8g --net=none --name master dsparkmaster /bin/bash

启动slave容器：

docker run -dit --net=none --name slave1 -m 6g dsparkmaster

docker run -dit --net=none --name slave2 -m 6g dsparkmaster



docker 并无固定IP地址的命令或工具，可以通过虚拟网桥对来实现IP的固定，方法如下写成的shell文件：

#!/bin/bash

#program : using it to set the IP for docker contaiter

#Made by : jie

#data :2015/08/14

#

# 1 master ip is 172.17.1.1

# 2 slave ip is 172.17.2.1 - 172.17.2.2

#

if [ ! -d "/var/run/netns" ]; then

mkdir /var/run/netns

fi

#for the master

master\_name=master

declare -i master\_ip

master\_ip=1

docker start ${master\_name}

pid=$(docker inspect -f '{{.State.Pid}}' ${master\_name})

echo "master pid=$pid"

ln -s /proc/$pid/ns/net /var/run/netns/$pid

ip link add veth\_${master\_name}\_b type veth peer name veth\_${master\_name}\_c

brctl addif docker0 veth\_${master\_name}\_b

ip link set veth\_${master\_name}\_b up

ip link set veth\_${master\_name}\_c netns $pid

ip netns exec $pid ip link set dev veth\_${master\_name}\_c name eth0

ip netns exec $pid ip link set eth0 address 12:34:56:78:9a:${master\_ip}

ip netns exec $pid ip link set eth0 up

ip netns exec $pid ip addr add 172.17.1.${master\_ip}/16 dev eth0

ip netns exec $pid ip route add default via 172.17.42.1

#end master set IP

#for the slave network

declare -i slave\_num slave\_i

slave\_num=2

for (( slave\_i=1; $slave\_i <= $slave\_num; slave\_i=$slave\_i+1 ))

do

slave\_name="slave$slave\_i"

declare -i slave\_ip

slave\_ip=$slave\_i

docker start ${slave\_name}

pid=$(docker inspect -f '{{.State.Pid}}' ${slave\_name})

echo "$slave\_name pid=$pid"

ln -s /proc/$pid/ns/net /var/run/netns/$pid

ip link add veth\_${slave\_name}\_b type veth peer name veth\_${slave\_name}\_c

brctl addif docker0 veth\_${slave\_name}\_b

ip link set veth\_${slave\_name}\_b up

ip link set veth\_${slave\_name}\_c netns $pid

ip netns exec $pid ip link set dev veth\_${slave\_name}\_c name eth0

ip netns exec $pid ip link set eth0 address 12:34:56:78:9a:${slave\_ip}

ip netns exec $pid ip link set eth0 up

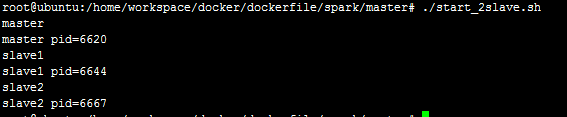
ip netns exec $pid ip addr add 172.17.2.${slave\_ip}/16 dev eth0

ip netns exec $pid ip route add default via 172.17.42.1

done

#end slave set IP

将创建的三个容器一个master、两个slave容器启动后运行shell文件，即可固定ip



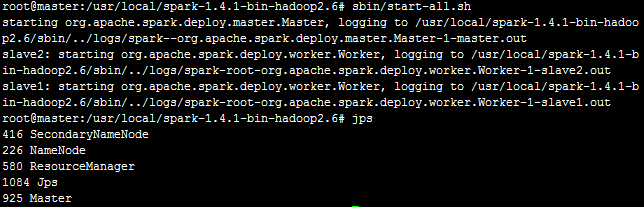
真心希望Docker官网能够提供固定ip的方法，通过这种方法容器重启后ip又要重新固定相当麻烦。

## 5.3、集群启动测试

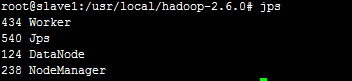
先格式化namenode节点

hdfs namenode -format

现在就启动了，看看效果吧。哈哈.....搞了几天终于弄好啦。



这是namenode节点的



这是slave节点的

成功！！！！！！！！！！！！

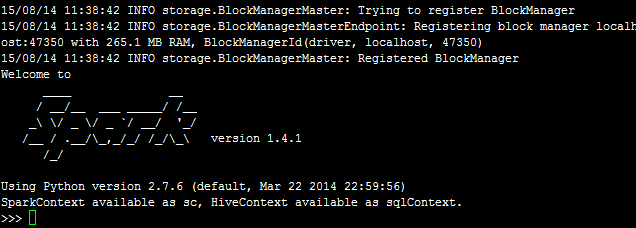
***我们接下来去官网找个例子，测试一下看是否可用吧***

https://spark.apache.org/docs/1.4.1/quick-start.html

我们用python吧，scala实在有点不会。

启动

./bin/pyspark



参考文献：

https://docs.docker.com

http://yaxin-cn.github.io/Docker/docker-container-use-static-IP.html

<http://www.cnblogs.com/rilley/archive/2012/02/13/2349858.html>

<http://www.cnblogs.com/rilley/archive/2012/02/13/2349858.html>

<http://yuedu.baidu.com/ebook/d817967416fc700abb68fca1?fr=aladdin&key=docker>