Dockerfile

Dockerfile是一种被Docker程序解释的脚本，Dockerfile由一条条指令组成，每条指令对应Linux下面的一条命令。Docker程序将这些Dockerfile指令翻译成Linux命令，类似于Makefile，Docker程序解决这些命令间的依赖及运行。Dockerfile的作用是用来创建一个自定义的image，包含了用户指定的软件依赖，当使用docker build命令时，docker会读取当前目录下的命名为Dockerfile的纯文本文件并执行里面的指令构建一个docker image，（另外一种构建docker image的方法是pull基础镜像，然后在启动的容器中安装和配置所需要的环境，然后commit成一个镜像）。下面进行具体的介绍：

# 1.Example

下面是构建jdk+tomcat环境的例子，Dockerfile定义如下：

1. # Pull base image
2. FROM ubuntu:13.10
3. MAINTAINER fys "fengyongshe@cmss.chinamobile.com"
4. # Update source
5. RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
6. RUN apt-get update
7. #Install curl
8. RUN apt-get -y install curl
9. #Install JDK 7
10. RUN cd /tmp && curl -L 'http://download.oracle.com/otn-pub/java/jdk/7u65-b17/jdk-7u65-linux-x64.tar.gz' -H 'Cookie: oraclelicense=accept-securebackup-cookie; gpw\_e24=Dockerfile' | tar -xz
11. RUN mkdir -p /usr/lib/jvm
12. RUN mv /tmp/jdk1.7.0\_65 /usr/lib/jvm/java-7-oracle/
13. #Set Oracke JDK 7 as default java
14. RUN update-alternatives --install /usr/bin/java java /usr/lib/jvm/java-7-oracle/bin/java 300
15. RUN update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java-7-oracle/bin/javac 300
16. ENV JAVA\_HOME /usr/lib/jvm/java-7-oracle
17. #Install tomcat7
18. RUN cd /tmp && curl -L 'http://archive.apache.org/dist/tomcat/tomcat-7/v7.0.8/bin/apache-tomcat-7.0.8.tar.gz' | tar -xz
19. RUN mv /tmp/apache-tomcat-7.0.8 /opt/tomcat7
20. ENV CATALINA\_HOME /opt/tomcat7
21. ENV PATH $PATH:$CATALINA\_HOME/bin
22. ADD tomcat7.sh /etc/init.d/tomcat7
23. RUN chmod 755 /etc/init.d/tomcat7
24. #Expose ports
25. EXPOSE 8090
26. #Define default command
27. ENTRYPOINT service tomcat7 start && tail -f /opt/tomcat7/logs/catalina.out

tomcat7.sh脚本内容如下：

1. export JAVA\_HOME=/usr/lib/jvm/java-7-oracle/
2. export TOMCAT\_HOME=/opt/tomcat7
3. case $1 in
4. start)
5. sh $TOMCAT\_HOME/bin/startup.sh
6. ;;
7. stop)
8. sh $TOMCAT\_HOME/bin/shutdown.sh
9. ;;
10. restart)
11. sh $TOMCAT\_HOME/bin/shutdown.sh
12. sh $TOMCAT\_HOME/bin/startup.sh
13. ;;
14. esac
15. exit 0

使用docker build命令构建docker images，命令格式如下：

Usage: docker build [OPTIONS] PATH | URL | -

[root@fys1 docker-tomcat]# docker build -t fys/jdk-tomcat .

*Sending build context to Docker daemon 4.096 kB*

*Sending build context to Docker daemon*

*Step 0 : FROM ubuntu:13.10*

*---> 195eb90b5349*

*Step 1 : MAINTAINER fys "fengyongshe@cmss.chinamobile.com"*

*---> Using cache*

*---> 8166522902e3*

*Step 2 : RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list*

*---> Running in 701bec054df7*

*---> a266ab4d9a12*

*Removing intermediate container 701bec054df7*

*Step 3 : RUN apt-get update*

[root@fys1 docker-tomcat]# docker images

*REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE*

*fys/jdk-tomcat latest 8166522902e3 About a minute ago 184.6 MB*

使用该docker image，启动Docker Container如下：

[root@fys1 docker-tomcat]# docker run -d -p 8091:8090 fys/jdk-tomcat

*72f825ee806f63b7e15a1aa3fae0e7c5b49bdf575273a688df6d12b38c758b70*

[root@fys1 docker-tomcat]# docker ps

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

*72f825ee806f fys/jdk-tomcat "/bin/sh -c 'service 6 seconds ago Up 6 seconds 0.0.0.0:8091->8090/tcp sharp\_shockley*

# 2.Dockerfile关键字

Dockerfile的格式如下：# CommentINSTRUCTION arguments

*FROM //指定基础image*

必须指定且在Dockerfile其他指令的签名，后续的指令都依赖于该指令指定的image。FROM指令指定的基础image可以是官方远程仓库的，也可以位于本地仓库

两种格式：

FORM <image>

FROM <image>:<tag>

指定基础image为该image的tag版本

*MAINTANER //用于指定镜像创建者的信息*

构建指令，用于将image的创建者相关信息写入image中，当对image执行docker inspect命令时，输出中有相应的字段记录该信息

格式：MAINTAINER <name>

*RUN //运行任何被基础image支持的命令*

如果基础image使用，ubuntu，那么软件管理部分只能使用ubuntu的命令，有两种格式：

1）RUN <command> (the command is run in a shell - `/bin/sh -c`)

2）RUN ["executable", "param1", "param2" ... ]  (exec form)

*CMD //设置container启动时执行的操作*

设置container启动时指定的操作，该操作可以是执行自定义脚本，也可以是执行系统命令，该指令只能在在文件中存在一次，如果有多个，执行最后一条，格式:

CMD ["executable","param1","param2"] (like an exec, this is the preferred form)

CMD command param1 param2 (as a shell)

当Dockerfile指定ENTRYPOINT，那么使用下面的格式：

CMD ["param1","param2"] (as default parameters to ENTRYPOINT)

ENTRYPOINT指定的是一个可执行的脚本或者程序的路径，该指定的脚本或者程序将会以param1和param2作为参数执行。所以如果cmd执行使用上面的形式，那么Dockerfile中必须要有配套的ENTRYPOINT

*ENTRYPOINT //设置container启动是执行的操作*

指定容器启动时执行的命令，可以多次设置，但是只有最后一个有效，格式：

ENTRYPOINT ["executable", "param1", "param2"] (like an exec, the preferred form)

ENTRYPOINT command param1 param2 (as a shell)

该指令的使用有两种情况，一种是独立使用，另一种和CMD指令配合使用，当独立使用时，如果还使用CMD命令且CMD是一个完整的可执行命令，那么CMD指令和ENTRYPOINT会互相覆盖只有一个CMD或者ENTRYPOINT有效

另一种用法和CMD指令配合使用来指定ENTRYPOINT的默认参数，这时CMD指令不是一个完整的可执行命令，仅仅是参数部分，ENTRYPOINT指令只能使用JSON方式来执行命令，而不能指定参数：

FROM ubuntu

CMD ["-l"]

ENTRYPOINT ["/usr/bin/ls"]

*USER //设置container容器的用户，默认是ROOT用户*

ENTRYPOINT ['memcached']

USER daemon

*EXPOSE //指定容器需要映射到宿主机器的端口*

该指令会将容器中的端口映射成宿主机器中的某个端口，当需要访问容器的时候，可以不是用容器的IP地址而是使用宿主机器的IP地址和映射后的端口，要完成整个操作需要两个步骤：

1）在Dockerfile中使用EXPOSE设置需要映射的容器端口

2）在运行容器的时候指定-p选项加上EXPOSE设置的端口，这样EXPOSE设置的端口号会被映射到宿主机器中的一个端口号。

也可以指定需要映射到宿中机器的那个端口，这是要确保宿主机器的端口没有被使用。EXPOSE指令可以一次设置多个端口号，相应的运行容器的时候，可以配套的多次使用使用-p选项，格式：EXPOSE <port> [<port>...]

端口映射是docker比较重要的一个功能，每次运行容器的ip地址不能指定而在桥接网卡的地址范围内随机生成的，而宿主机制的IP地址是固定的，可以将容器的端口映射到宿主机器的一个端口，免去每次访问容器中的某个服务都要查看容器的IP的地址。对于一个运行的容器，可以使用docker port加上容器中需要映射的端口和容器的ID来查看该端口号在宿主端口号在宿主机器上的映射端口。

*ENV //用于设置环境变量*

在image中设置一个环境变量，格式为：ENV <key> <value>

设置了以后，后续的RUN命令都可以使用，container启动后，可以通过docker inspect查看这个环境变量，也可以通过docker run --env key=value时设置或者修改环境变量。例如需要设置JAVA\_HOME，那么可以在Dockerfile中这样设置：

ENV JAVA\_HOME /path/to/java/client

*ADD //从src复制文件到container的dest路径*

拷贝到container中的文件和文件夹权限为0755,uid和gid为0。如果是一个目录，将该目录中的所有文件添加到container中，不包括目录。如果文件是可识别的压缩格式，则docker会解压。如果src是文件且<dest>中不使用斜杠结束，则会将<dest>视为文件，<src>的内容为写入<dest>，斜杠则将<src>文件拷贝到<dest>目录下，格式：ADD <src> <dest>

其中<src>是相对构建的源目录的相对路径，可以是文件或目录的路径，也可以是一个远程的文件url

<dest>是container中的绝对路径

*VALUME //指定挂载点*

使容器中的一个目录具有持久化存储数据的功能，该目录可以被容器本身使用，也可以共享给其他容器使用。容器使用的是AUFS，改文件系统不能持久化数据，当容器关闭后，所有的更改都会丢失。当容器中的应用有持久化数据的需求时可以在Dockerfile中使用该指令。

格式1：VALUME ["<mountpoint>"]

格式2：

FROM base

VALUME ['/tmp/data']

对于格式2，运行通过Dockerfile生成的image容器，/tmp/data目录中的数据在容器关闭后，里面的数据都还在，例如另外一个容器也有持久化数据的需求，且使用上面容器共享/tmp/data目录，那么可以运行下面的命令启动一个容器

docker run -t -i -rm -volumes-from [container1] [image2] bash

[container1]为的一个容器的ID，image2为第二个容器image的名字

*WORKDIR //设置执行目录*

多次切换目录，相当于cd命令，对于RUN、CMD、ENTRYPOINT生效，格式：

WORKDIR [work\_dir]

使用举例：WORKDIR /p1 WORKDIR p2 RUN vim a.txt

在/p1/p2下执行vim a.txt

*ONBUILD，//在子镜像中执行*

ONBUILD [Dockerfile关键字]

ONBUILD指定的命令在构建镜像时不并执行，而是在它的子镜像中执行

# 3.Dockerfile进阶

使用Docker启动Ambari Server，Dockerfile如下：

1. #Pull base image
2. FROM jdeathe/centos-ssh-mysql
3. MAINTAINER fys "fengyongshe@cmss.chinamobile.com"
4. #Update source
5. RUN rm -rf /etc/yum.repos.d/\*
6. RUN curl -o /etc/yum.repos.d/bch.repo http://10.128.2.95/bch/1.3.1/centos/6/x86\_64/bch-centos-virtual.repo
7. #Set SSH
8. RUN yum install -y openssh-server openssh-clients
9. RUN ssh-keygen -t dsa -f /etc/ssh/ssh\_host\_dsa\_key \
10. && ssh-keygen -t rsa -f /etc/ssh/ssh\_host\_rsa\_key \
11. && ssh-keygen -q -N "" -t rsa -f /root/.ssh/id\_rsa
12. RUN cp /root/.ssh/id\_rsa.pub /root/.ssh/authorized\_keys \
13. && mkdir /var/run/sshd \
14. && echo 'root:123456' |chpasswd
15. RUN /bin/sed -i 's/.\*session.\*required.\*pam\_loginuid.so.\*/session optional pam\_loginuid.so/g' /etc/pam.d/sshd
16. WORKDIR /root
17. #Install ambari-server && ambari-agent
18. RUN yum install -y passwd openssl libconfuse libconfuse-devel ambari-server ambari-agent
19. RUN echo 'java.home=/opt/jdk1.7.0\_65' >> /etc/ambari-server/conf/ambari.properties
20. RUN ambari-server setup --silent
21. EXPOSE 8080

构建image

$docker build -t fys/ambari .

通过命令docker images可以查找到fys/ambari的images

启动ambari on docker

$docker run -d -P fys/ambari

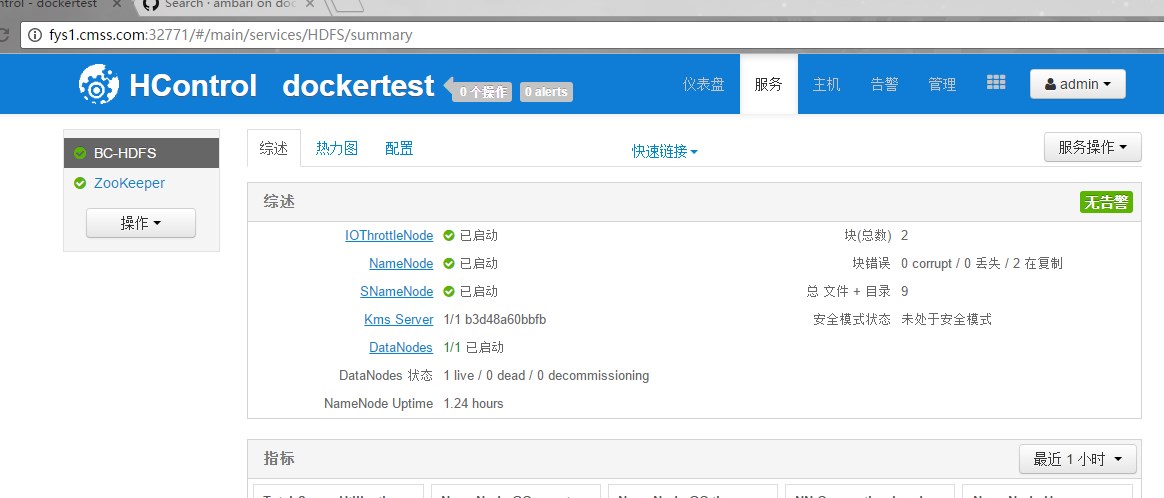
结果如下所示：

*[root@fys1 ~]# docker ps*

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

*b3d48a60bbfb fys/ambari "/usr/bin/supervisor About an hour ago Up About an hour 0.0.0.0:32773->22/tcp, 0.0.0.0:32772->3306/tcp, 0.0.0.0:32771->8080/tcp dreamy\_engelbart*

Ambari的端口映射为32771，根据提示创建hdfs的服务，如下所示：



可以通过命令行对hdfs进行基本操作，停止该Container

*[root@fys1 ~]# docker kill b3d48a60bbfb*

*b3d48a60bbfb*

*[root@fys1 ~]# docker ps*

*CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES*

重启后服务可以正常启动，例如上传的hdfs数据，如下：

*[hdfs@b3d48a60bbfb ~]$ hdfs dfs -ls /tmp/*

*Found 2 items*

*-rw-r--r-- 3 hdfs hdfs 12 2016-09-18 12:16 /tmp/helloworld*

*-rwxr-xr-x 3 hdfs hdfs 980 2016-09-18 12:03 /tmp/id11ac2d00\_date031816*

*[hdfs@b3d48a60bbfb ~]$ hdfs dfs -cat /tmp/helloworld*

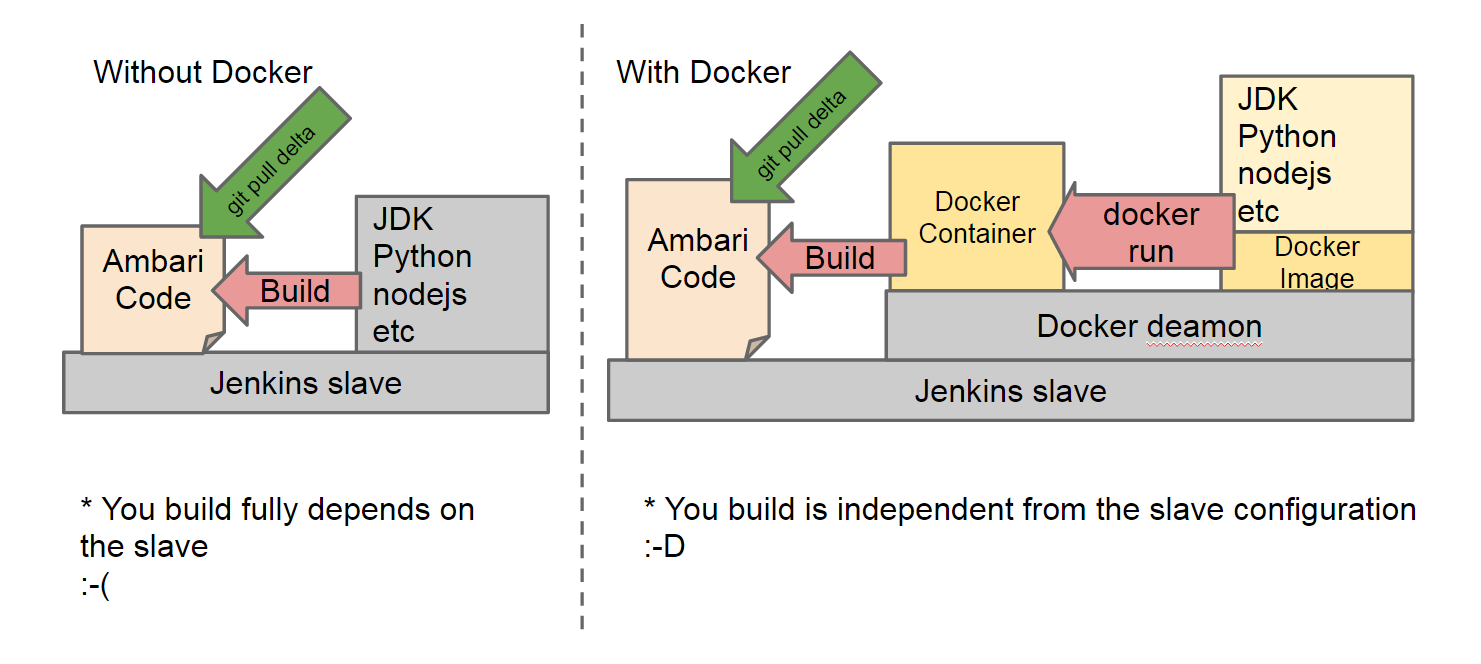
*hello*

*world*

# 5.Ambari Cluster on Docker

## 5.1 Ambari in Docker

Development in Docker，该小节为Ambari的官网上的内容，用于在Docker中开发、构建和测试Ambari，其目的仅是用于测试而非生产环境，系统结构图如下：



1）创建Docker image

*git clone https://github.com/apache/ambari.git*

*cd ambari*

*docker build -t ambari/build ./dev-support/docker/docker*

2）解析Dockerfile

1. FROM centos:centos6 //基于centos6
2. RUN echo root:changeme | chpasswd
3. ## Install some basic utilities that aren't in the default image
4. RUN yum clean all -y && yum update -y //更新yum源
5. RUN yum -y install vim wget rpm-build sudo which telnet tar openssh-server openssh-clients ntp git python-setuptools httpd //安装所需要的文件
6. # phantomjs dependency
7. RUN yum -y install fontconfig freetype libfreetype.so.6 libfontconfig.so.1 libstdc++.so.6
8. RUN rpm -e --nodeps --justdb glibc-common
9. RUN yum -y install glibc-common
10. ENV HOME /root
11. #Install JAVA //安装JAVA
12. # RUN wget --no-check-certificate --no-cookies --header "Cookie:oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/7u55-b13/jdk-7u55-linux-x64.rpm -O jdk-7u55-linux-x64.rpm
13. # RUN yum -y install jdk-7u55-linux-x64.rpm
14. RUN wget --no-check-certificate --no-cookies --header "Cookie:oraclelicense=accept-securebackup-cookie" http://download.oracle.com/otn-pub/java/jdk/6u45-b06/jdk-6u45-linux-x64-rpm.bin -O jdk-6u45-linux-x64-rpm.bin
15. RUN chmod +x jdk-6u45-linux-x64-rpm.bin
16. RUN ./jdk-6u45-linux-x64-rpm.bin
17. ENV JAVA\_HOME /usr/java/default/
18. #Install Maven //安装maven
19. RUN mkdir -p /opt/maven
20. WORKDIR /opt/maven
21. RUN wget http://apache.cs.utah.edu/maven/maven-3/3.0.5/binaries/apache-maven-3.0.5-bin.tar.gz
22. RUN tar -xvzf /opt/maven/apache-maven-3.0.5-bin.tar.gz
23. RUN rm -rf /opt/maven/apache-maven-3.0.5-bin.tar.gz
24. ENV M2\_HOME /opt/maven/apache-maven-3.0.5
25. ENV MAVEN\_OPTS -Xmx2048m -XX:MaxPermSize=256m
26. ENV PATH $PATH:$JAVA\_HOME/bin:$M2\_HOME/bin
27. # SSH key //设置ssh
28. RUN ssh-keygen -f /root/.ssh/id\_rsa -t rsa -N ''
29. RUN cat /root/.ssh/id\_rsa.pub > /root/.ssh/authorized\_keys
30. RUN chmod 600 /root/.ssh/authorized\_keys
31. RUN sed -ri 's/UsePAM yes/UsePAM no/g' /etc/ssh/sshd\_config
32. # Install python, nodejs and npm
33. RUN yum -y install http://dl.fedoraproject.org/pub/epel/6/x86\_64/epel-release-6-8.noarch.rpm
34. RUN yum -y install nodejs npm --enablerepo=epel
35. RUN npm install -g brunch@1.7.13
36. # Once run some mvn commands to cache .m2/repository
37. WORKDIR /tmp
38. RUN git clone https://github.com/apache/ambari.git
39. WORKDIR /tmp/ambari
40. RUN mvn versions:set -DnewVersion=1.6.1.0
41. RUN mvn -B clean install package rpm:rpm -DskipTests -DnewVersion=1.6.1.0 -Dpython.ver="python >= 2.6" -Preplaceurl
42. # clean git code because I want to use the one on local filesystem.
43. WORKDIR /tmp
44. RUN rm -rf /tmp/ambari
45. RUN mkdir -p /tmp/ambari-build-docker/blueprints //将ambari build的blueprint放入image
46. ADD ./blueprints /tmp/ambari-build-docker/blueprints
47. RUN mkdir -p /tmp/ambari-build-docker/bin
48. ADD ./bin /tmp/ambari-build-docker/bin //将ambari build的脚本写入image
49. WORKDIR /tmp

创建ambari的最新的Image（见maven），然后使用该image进行ambari功能的测试

3）执行单元测试

*cd {ambari\_root}*

*docker run --privileged -h node1.mydomain.com -v $(pwd):/tmp/ambari ambari/build /tmp/ambari/dev-support/docker/docker/bin/ambaribuild.py test -b*

4）部署Hadoop //使用blueprint，安装hadoop

*cd {ambari\_root}*

*docker run --privileged -t -p 80:80 -p 5005:5005 -p 8080:8080 -h node1.mydomain.com --name ambari1 -v $(pwd):/tmp/ambari ambari/build /tmp/ambari-build-docker/bin/ambaribuild.py deploy -b*

*# once your are done*

*docker kill ambari1 && docker rm ambari1*

5) 参考文献

https://cwiki.apache.org/confluence/display/AMBARI/Development+in+Docker

<https://github.com/apache/ambari/tree/trunk/dev-support/docker>

*该处仅是build最新的ambari代码，完成构建、单元测试及功能测试，与Ambari cluster on docker区别比较大，不再深入分析及测试*

## 5.2 docker-ambari

*[root@fys3 ~]# docker search ambari*

*NAME DESCRIPTION STARS OFFICIAL AUTOMATED*

*sequenceiq/ambari This repository is deprecated! 50 [OK]*

*sequenceiq/ambari-shell 12 [OK]*

*hortonworks/ambari-agent Ambari Agent: https://ambari.apache.org 7 [OK]*

*hortonworks/ambari-server Ambari Server: https://ambari.apache.org 6 [OK]*

*hortonworks/ambari-base Base image containing systemd, java, sshd 1 [OK]*

1) 下载image

*docker pull hortonworks/ambari-base*

*docker pull hortonworks/ambari-server*

*docker pull sequenceiq/ambari-shell*

*docker pull hortonworks/ambari-agent*

docker pull sequenceiq/consul:v0.5.0-v6

2）从github上下载ambari-functions，该脚本用于启动ambari containers

*git clone https://github.com/sequenceiq/docker-ambari.git*

*cd docker-ambari*

*source ambari-functions*

执行以上命令后，可以在系统中执行以下相关命令：

ambari-python-wrap

amb-create-hdfs-dir

amb-scp-to-first

amb-start-cluster

ambari-metrics-collector

amb-clean

amb-deploy-cluster

amb-settings

amb-start-first

ambari-metrics-monitor

amb-copy-to-hdfs

amb-members

amb-shell

amb-start-node

3) 启动ambari docker 集群

执行amb前缀的命令，如下：

*[root@fys3 docker-ambari]# amb-settings*

*NODE\_PREFIX=amb*

*CLUSTER\_SIZE=3*

*AMBARI\_SERVER\_NAME=amb-server*

*AMBARI\_SERVER\_IMAGE=hortonworks/ambari-server:latest*

*AMBARI\_AGENT\_IMAGE=hortonworks/ambari-agent:latest*

*DOCKER\_OPTS=*

*AMBARI\_SERVER\_IP=*

*CONSUL=amb-consul*

*CONSUL\_IMAGE=sequenceiq/consul:v0.5.0-v6*

*EXPOSE\_DNS=false*

*DRY\_RUN=false*

启动集群的命令如下：

*[root@fys3 docker-ambari]# amb-start-cluster 3*

*starting an ambari cluster with: 3 nodes*

*[DEBUG] docker run -d --name amb-consul -h amb-consul.service.consul sequenceiq/consul:v0.5.0-v6 -server -bootstrap*

*[DEBUG] docker run -d --privileged --name amb-server -h amb-server.service.consul hortonworks/ambari-server:latest systemd.setenv=NAMESERVER\_ADDR=172.17.0.1*

*ed4acf35d016571a27e3b8345515e89d1f4fecf2fe11b0af71b18328c826b0b5*

*truetrue[DEBUG] docker run -d --privileged --name amb1 -h amb1.service.consul hortonworks/ambari-agent:latest systemd.setenv=NAMESERVER\_ADDR=172.17.0.1*

*95b3cfd2041d7f60cd0506b7225cb13123f76f304e7b70f8c007c643ed039093*

*true[DEBUG] docker run -d --privileged --name amb2 -h amb2.service.consul hortonworks/ambari-agent:latest systemd.setenv=NAMESERVER\_ADDR=172.17.0.1*

*e8c2af1ccf4ce500b716e2ce1a75166b8b82f6e66975c56f6a224cab6f057d57*

命令执行结果如下：

*[root@fys3 docker-ambari]# docker ps*

*CONTAINER ID IMAGE COMMAND CREATE STATUS PORTS NAMES*

*e8c2af1ccf4c hortonworks/ambari-agent:latest "/usr/sbin/init syst 3 minutes ago Up 3 minutes amb2*

*95b3cfd2041d hortonworks/ambari-agent:latest "/usr/sbin/init syst 3 minutes ago Up 3 minutes amb1*

*ed4acf35d016 hortonworks/ambari-server:latest "/usr/sbin/init syst 3 minutes ago Up 3 minutes 8080/tcp amb-server*

*3a4c3a82e421 sequenceiq/consul:v0.5.0-v6 "/bin/start -server 3 minutes ago Up 3 minutes 53/tcp, 53/udp, 8300-8302/tcp, 8400/tcp, 8301-8302/udp, 8500/tcp amb-consul*

启动组件包括：amb-server、amb1、amb2和amb-consul，结果如下：

*[root@fys3 docker-ambari]# amb-settings*

*NODE\_PREFIX=amb*

*CLUSTER\_SIZE=3*

*AMBARI\_SERVER\_NAME=amb-server*

*AMBARI\_SERVER\_IMAGE=hortonworks/ambari-server:latest*

*AMBARI\_AGENT\_IMAGE=hortonworks/ambari-agent:latest*

*DOCKER\_OPTS=*

*AMBARI\_SERVER\_IP=172.17.0.2*

*CONSUL=amb-consul*

*CONSUL\_IMAGE=sequenceiq/consul:v0.5.0-v6*

*EXPOSE\_DNS=false*

*DRY\_RUN=false*

通过blueprint部署集群，通过amb-shell来执行

*[root@fys3 ~]# amb-shell*

*[DEBUG] docker run -it --rm -e EXPECTED\_HOST\_COUNT=2 -e BLUEPRINT= --link amb-server:ambariserver --entrypoint /bin/sh hortonworks/ambari-server:latest -c /tmp/ambari-shell.sh*

*AMBARI\_HOST=172.17.0.2*

*[DEBUG] waits for ambari server: 172.17.0.2 RUNNING ...*

*[DEBUG] waits until 2 hosts connected to server ...*

*[DEBUG] connected hosts: 2*

*\_ \_ \_ \_\_\_\_ \_ \_ \_*

*/ \ \_ \_\_ \_\_\_ | |\_\_ \_\_ \_ \_ \_\_ (\_)/ \_\_\_| | |\_\_ \_\_\_ | || |*

*/ \_ \ | '\_ ` \_ \ | '\_ \ / \_` || '\_\_|| |\\_\_\_ \ | '\_ \ / \_ \| || |*

*/ \_\_\_ \ | | | | | || |\_) || (\_| || | | | \_\_\_) || | | || \_\_/| || |*

*/\_/ \\_\|\_| |\_| |\_||\_.\_\_/ \\_\_,\_||\_| |\_||\_\_\_\_/ |\_| |\_| \\_\_\_||\_||\_|*

*Welcome to Ambari Shell. For command and param completion press TAB, for assistance type 'hint'.*

*ambari-shell>help*

*\* blueprint add - Add a new blueprint with either --url or --file*

*\* blueprint defaults - Adds the default blueprints to Ambari*

*\* blueprint list - Lists all known blueprints*

*\* blueprint show - Shows the blueprint by its id*

*\* cluster assign - Assign host to host group*

参数解析：

-d: Detached mode，container在后台运行

-p 8080，publish ambari web和Rest api端口

-h amb0.mycorp.com，使用的hostname

--name，container的名称

sequenceiq/ambari:1.6.0，image的名称

--tag ambari-server=true，这个命令传给entrypoint

4) 参考文献

https://github.com/sequenceiq/docker-ambari

http://blog.sequenceiq.com/blog/2014/06/17/ambari-cluster-on-docker/

http://blog.sequenceiq.com/blog/2014/12/04/multinode-ambari-1-7-0/

https://cwiki.apache.org/confluence/display/AMBARI/Ambari+Shell

https://hub.docker.com/r/masato/ambari-swarm/

https://github.com/masato/docker-ambari

https://github.com/sequenceiq/docker-ambari