# zphj1987'Blog

http://www.zphj1987.com/tags/ceph/

# 基于docker部署ceph以及修改docker image-DockerInfo

http://www.dockerinfo.net/4440.html

基于docker部署ceph以及修改docker image | zphj1987'Blog

http://www.zphj1987.com/2017/03/15/base-on-docker-deploy-ceph/

2017-03-21 分类：[Docker实践分享/开发实战](http://www.dockerinfo.net/practice) 阅读(319) 评论(0) 作者：武汉-运维-磨渣

前言

容器和ceph的结合已经在一些生产环境当中做了尝试，容器的好处就是对运行环境的一个封装，传统的方式是集成为ISO，这个需要一定的维护量，而容器的相关操作会简单很多，也就有了一些尝试，个人觉得如果玩的转容器可以考虑，当然得懂ceph，不然两套系统在一起，问题都不知道是哪个的，就比较麻烦了

本篇是基于之前我的填坑群里面的牛鹏举的一个问题，他的环境出现了创建osd的时候权限问题，我这边没遇到，现在实践了一遍，感觉应该是之前目录提前创建了的问题

实践步骤

安装docker

|  |
| --- |
| yum install docker |

下载ceph镜像

这个镜像是sebastien维护的，他是redhat的ceph工程师，ceph-ansible的负责人,很多一线的资料都是来自他的分享，这个是一个集成好的镜像

|  |
| --- |
| docker pull ceph/daemon |

准备好一些目录

|  |
| --- |
| mkdir -p /etc/ceph  mkdir -p /var/lib/ceph/ |

注意只需要做这个两个目录，不要创建子目录，docker内部有相关的操作

创建一个mon

|  |
| --- |
| sudo docker run -d --net=host --name=mon \  -v /etc/ceph:/etc/ceph \  -v /var/lib/ceph/:/var/lib/ceph \  -e MON\_IP=192.168.8.106 \  -e CEPH\_PUBLIC\_NETWORK=192.168.0.0/16 \  ceph/daemon mon |

MON\_IP就是宿主机的IP地址

执行完了后

|  |
| --- |
| [root@lab8106 ceph]# docker ps -l  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  86ed05173432 ceph/daemon "/entrypoint.sh mon" About a minute ago Up 59 seconds mon |

可以看到退出了，我们来docker logs -f mon看下日志的输出

|  |
| --- |
| [root@lab8106 ceph]# docker logs -f mon  /sbin/ip  creating /etc/ceph/ceph.client.admin.keyring  creating /etc/ceph/ceph.mon.keyring  creating /var/lib/ceph/bootstrap-osd/ceph.keyring  creating /var/lib/ceph/bootstrap-mds/ceph.keyring  creating /var/lib/ceph/bootstrap-rgw/ceph.keyring  monmaptool: monmap file /etc/ceph/monmap-ceph  monmaptool: set fsid to cb5df106-25b3-4f93-9f54-baca2976a47b  monmaptool: writing epoch 0 to /etc/ceph/monmap-ceph (1 monitors)  creating /tmp/ceph.mon.keyring  importing contents of /etc/ceph/ceph.client.admin.keyring into /tmp/ceph.mon.keyring  importing contents of /var/lib/ceph/bootstrap-osd/ceph.keyring into /tmp/ceph.mon.keyring  importing contents of /var/lib/ceph/bootstrap-mds/ceph.keyring into /tmp/ceph.mon.keyring  importing contents of /var/lib/ceph/bootstrap-rgw/ceph.keyring into /tmp/ceph.mon.keyring  importing contents of /etc/ceph/ceph.mon.keyring into /tmp/ceph.mon.keyring  ceph-mon: set fsid to cb5df106-25b3-4f93-9f54-baca2976a47b  ceph-mon: created monfs at /var/lib/ceph/mon/ceph-lab8106 for mon.lab81 |

提示成功了

我们看下生成的文件

|  |
| --- |
| [root@lab8106 ceph]# ll /etc/ceph  total 16  -rw------- 1 root root 137 Mar 14 17:53 ceph.client.admin.keyring  -rw-r--r-- 1 root root 285 Mar 14 17:53 ceph.conf  -rw------- 1 64045 64045 77 Mar 14 17:53 ceph.mon.keyring  -rw-r--r-- 1 64045 64045 187 Mar 14 17:53 monmap-ceph |

从这里可以看到内部的cpeh的用户的id是64045，所以在docker宿主机不要随便去给ceph权限，可能id不匹配，容器内部还是无法操作

创建一个osd

|  |
| --- |
| sudo docker run -d --net=host --name=myosd1 \  --privileged=true \  -v /etc/ceph:/etc/ceph \  -v /var/lib/ceph/:/var/lib/ceph \  -v /dev/:/dev/ \  -e OSD\_DEVICE=/dev/sdb \  ceph/daemon osd\_ceph\_disk |

如果查询日志

|  |
| --- |
| docker logs -f myosd1 |

如果执行命令

|  |
| --- |
| docker exec -it mon ceph -s |

如果想进入容器内部

|  |
| --- |
| docker exec -it mon /bin/bash |

修改集群的副本数

|  |
| --- |
| docker exec -it mon ceph osd pool set rbd size 1 |

查看集群状态

|  |
| --- |
| [root@lab8106 ceph]# docker exec -it mon ceph -s  cluster cb5df106-25b3-4f93-9f54-baca2976a47b  health HEALTH\_WARN  mon.lab8106 low disk space  monmap e2: 1 mons at {lab8106=192.168.8.106:6789/0}  election epoch 4, quorum 0 lab8106  mgr no daemons active  osdmap e7: 1 osds: 1 up, 1 in  flags sortbitwise,require\_jewel\_osds,require\_kraken\_osds  pgmap v15: 64 pgs, 1 pools, 0 bytes data, 0 objects  34288 kB used, 279 GB / 279 GB avail  64 active+clean |

上面的操作都很顺利，但是某些情况可能出现异常情况，或者镜像内部本身就有问题需要自己修改，这个怎么处理

碰上问题想修改image

我们看下我们运行的docker

|  |
| --- |
| [root@lab8106 ceph]# docker ps  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  874d78ccae55 ceph/daemon "/entrypoint.sh osd\_c" 14 hours ago Up 14 hours myosd1  86ed05173432 ceph/daemon "/entrypoint.sh mon" 15 hours ago Up 15 hours mon |

COMMAND这里有个/entrypoint.sh

如果存在ENTRYPOINT和CMD，那么CMD就是ENTRYPOINT的参数，如果没有ENTRYPOINT，则CMD就是默认执行指令  
也就是容器启动的时候默认是会去执行/entrypoint.sh 这个了

我们不需要他执行这个，就需要加参数了

|  |
| --- |
| [root@lab8106 ceph]# docker run -i -t --entrypoint /bin/bash ceph/daemon |

比如我上次做的一个操作，把ceph用户绑定到root的id

|  |
| --- |
| root@9b269bf751f9:/# cat /etc/passwd|grep ceph  ceph:x:64045:64045:[Ceph](http://www.dockerinfo.net/docker/ceph) storage service:/var/lib/ceph:/bin/false  root@9b269bf751f9:/# sed -i 's/64045/0/g' /etc/passwd  root@9b269bf751f9:/# cat /etc/passwd|grep ceph  ceph:x:0:0:Ceph storage service:/var/lib/ceph:/bin/false |

退出容器

|  |
| --- |
| root@9b269bf751f9:/# exit |

查询我们最后运行的容器，修改回entrypoint我们再把容器修改提交到基础image

|  |
| --- |
| [root@lab8106 ceph]# docker ps -l  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  9b269bf751f9 ceph/daemon "/bin/bash" 2 minutes ago Exited (0) 15 seconds ago angry\_hawking  [root@lab8106 ceph]# docker commit 9b269bf751f9 ceph/daemon  [root@lab8106 ~]# docker run -i -t --entrypoint /entrypoint.sh ceph/daemon  [root@lab8106 ~]# docker ps -l  CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  c2ea602c18ac ceph/daemon "/entrypoint.sh" 10 seconds ago Exited (1) 7 seconds ago ecstatic\_bartik  [root@lab8106 ceph]# docker commit c2ea602c18ac ceph/daemon |

再次启动容器,并且检查内容，可以看到已经修改好了

|  |
| --- |
| [root@lab8106 ceph]# docker run -i -t --entrypoint /bin/bash ceph/daemon  root@65b538fdc61e:/# cat /etc/passwd|grep ceph  ceph:x:0:0:Ceph storage service:/var/lib/ceph:/bin/false |

如果需要做其他的改动，这样改下就行

总结

本篇主要是根据sebastien的镜像做的部署，并且给出一些常用的命令，以及如何进入固化的容器的内部进行修改，方便自己调试环境

相关资料

[bootstrap-your-ceph-cluster-in-docker/](http://www.sebastien-han.fr/blog/2015/06/23/bootstrap-your-ceph-cluster-in-docker/)

<http://www.zphj1987.com/2017/03/15/base-on-docker-deploy-ceph/>

# 如何使用Docker部署Ceph分布式文件系统-DockerInfo

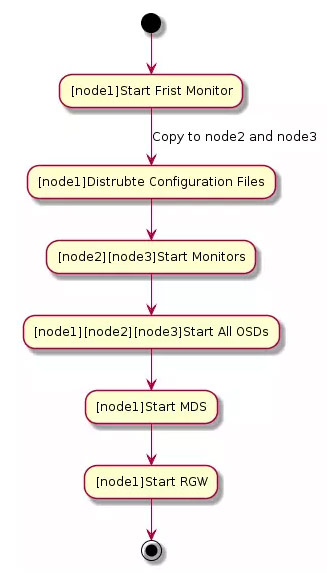
http://www.dockerinfo.net/445.html

[**Docker**](http://www.dockerinfo.net/)作为持久化集成的最佳工具，特别是在部署中有着得天独厚的优势。[Ceph](http://www.dockerinfo.net/docker/ceph)作为开源的分布式存储得到越来越多的使用，但是作为分布式系统，Ceph在部署和运维上仍然有不小的难度,本文重点介绍利用Docker快速的进行Ceph集群的创建，以及各个组件的安装。

部署环境

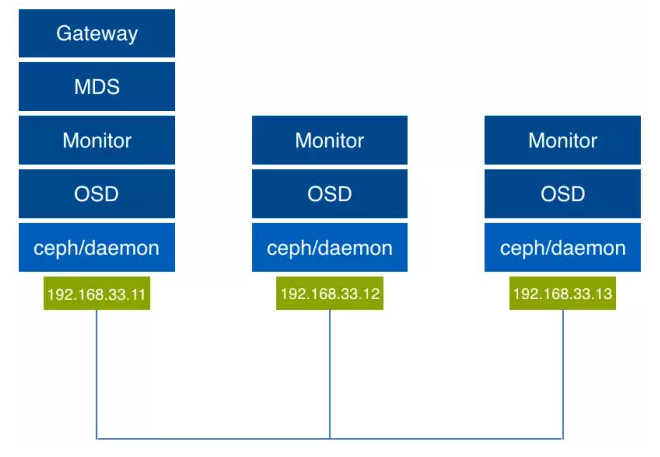
* 至少需要三台虚拟机或者物理机，每台虚拟机或者物理机至少有两块硬盘，这里我是在一台物理机上用vagrant模拟出三台[CentOS](http://www.dockerinfo.net/docker/centos) 6.6虚拟机进行的实验
* 获取ceph/daemon镜像
* 三台虚拟机需要安装docker，本文附带Docker加速方案

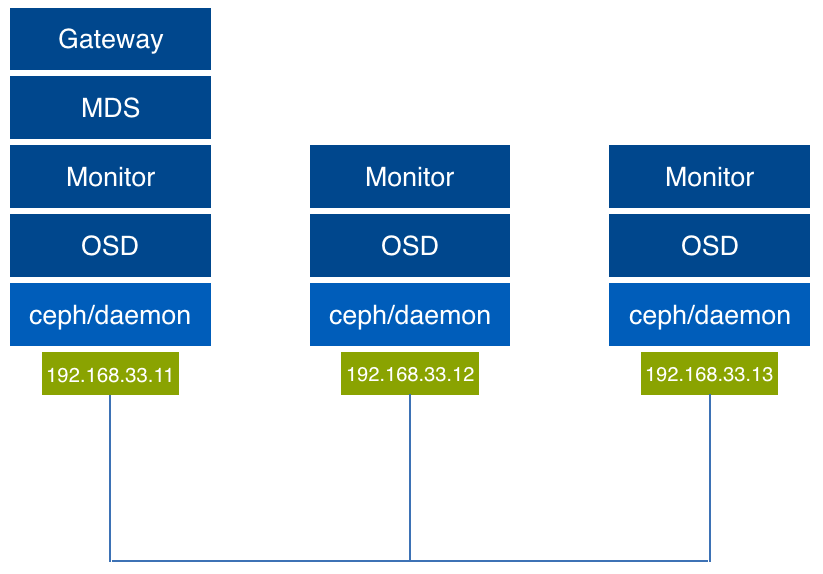
部署流程



部署架构

主机名和集群的对应关系如下：

* node1 -> 192.168.33.11
* node2 -> 192.168.33.12
* node3 -> 192.168.33.13



环境准备

安装Docker，下载镜像

国内安装Dcoker还是速度很慢的，这里推荐使用daocloud的加速方案。不但docker安装速度提高了，pull镜像的速度也大幅度提高。

curl -sSL https://get.daocloud.io/docker | sh

我是在[**CentOS**](http://www.dockerinfo.net/docker/centos)系统上进行的测试，将docker加入自动启动，并启动docker，接下来pull ceph daemon镜像，该镜像包含了所有的ceph服务和entrypoint。

chkconfig docker

service docker start

docker pull ceph/daemon

启动第一个Monitor

在node1上启动第一个Monitor，注意，如果你的环境中IP和我不同，请修改MON\_IP。

sudo docker run -d \

    --net=host \

    -v /etc/ceph:/etc/ceph \

    -v /var/lib/ceph/:/var/lib/ceph/ \

    -e MON\_IP=192.168.33.11 \

    -e CEPH\_PUBLIC\_NETWORK=192.168.33.0/24 \

    ceph/daemon mon

验证一下效果：

docker ps

CONTAINER ID        IMAGE               COMMAND                CREATED             STATUS              PORTS               NAMES

7babea544ef1        ceph/daemon         "/entrypoint.sh mon"   3 seconds ago       Up 2 seconds                            backstabbing\_brattain

查看一下集群状态：

docker exec 7babea544ef1 ceph -s

当前集群状态，能看到当前已经有一个mon启动起来了。

   cluster 0de1fc5a-084d-4396-bb0b-59db72a9a439

    health HEALTH\_ERR

           64 pgs stuck inactive

           64 pgs stuck unclean

           no osds

    monmap e1: 1 mons at {node1.docker.com=192.168.33.11:6789/0}

           election epoch 2, quorum 0 node1.docker.com

    osdmap e1: 0 osds: 0 up, 0 in

           flags sortbitwise

     pgmap v2: 64 pgs, 1 pools, 0 bytes data, 0 objects

           0 kB used, 0 kB / 0 kB avail

                 64 creating

复制配置文件

接下来需要将node1的配置文件复制到node2和node3上，复制的路径包含/etc/ceph和/var/lib/ceph/bootstrap-\*下的所有内容。这些配置文件非常重要，如果没有这些配置文件的存在，我们在其他节点启动新的docker ceph daemon的时候会被认为是一个新的集群。  
我们在node1执行以下命令：

ssh root@node2 mkdir -p /var/lib/ceph

scp -r /etc/ceph root@node2:/etc

scp -r /var/lib/ceph/bootstrap\* root@node2:/var/lib/ceph

ssh root@node3 mkdir -p /var/lib/ceph

scp -r /etc/ceph root@node3:/etc

scp -r /var/lib/ceph/bootstrap\* root@node3:/var/lib/ceph

启动第二个和第三个Monitor

在node2上执行：

sudo docker run -d \

    --net=host \

    -v /etc/ceph:/etc/ceph \

    -v /var/lib/ceph/:/var/lib/ceph/ \

    -e MON\_IP=192.168.33.12 \

    -e CEPH\_PUBLIC\_NETWORK=192.168.33.0/24 \

    ceph/daemon mon

在node3上执行：

sudo docker run -d \

    --net=host \

    -v /etc/ceph:/etc/ceph \

    -v /var/lib/ceph/:/var/lib/ceph/ \

    -e MON\_IP=192.168.33.13 \

    -e CEPH\_PUBLIC\_NETWORK=192.168.33.0/24 \

    ceph/daemon mon

在node1上查看集群状态：

   cluster 0de1fc5a-084d-4396-bb0b-59db72a9a439

    health HEALTH\_ERR

           64 pgs stuck inactive

           64 pgs stuck unclean

           no osds

    monmap e3: 3 mons at {node1.docker.com=192.168.33.11:6789/0,node2.docker.com=192.168.33.12:6789/0,node3.docker.com=192.168.33.13:6789/0}

           election epoch 6, quorum 0,1,2 node1.docker.com,node2.docker.com,node3.docker.com

    osdmap e1: 0 osds: 0 up, 0 in

           flags sortbitwise

     pgmap v2: 64 pgs, 1 pools, 0 bytes data, 0 objects

           0 kB used, 0 kB / 0 kB avail

                 64 creating

启动OSD的遇到的问题

按照原视频的介绍的方法，启动OSD可以直接指定某个分区，然后用osd\_ceph\_disk作为启动ceph/daemon的参数，之后docker镜像会自动的进行分区等动作。但是经过实际验证却发现在mkjournal创建错误，OSD无法启动。

经过和社区确认，发现这个Bug在之前版本中得到过修复，但是之后的版本又出现了。根据社区的建议使用jewel版本的ceph daemon进行了再次验证，发现问题依旧，所以这里介绍的方法只能退而求其次，采用手动方式分区、格式化，之后用osd\_directory启动ceph/daemon。

这是用osd\_ceph\_disk方式启动后的错误日志：

command\_check\_call: Running command: /usr/bin/ceph-osd --cluster ceph --mkfs --mkkey -i 4 --monmap /var/lib/ceph/tmp/mnt.BT8FXG/activate.monmap --osd-data /var/lib/ceph/tmp/mnt.BT8FXG --osd-journal /var/lib/ceph/tmp/mnt.BT8FXG/journal --osd-uuid 89e240e1-17e9-4d6c-8d4f-f1a3e0278b91 --keyring /var/lib/ceph/tmp/mnt.BT8FXG/keyring --setuser ceph --setgroup disk

2016-06-12 23:37:26.180610 7f8889654800 -1 filestore(/var/lib/ceph/tmp/mnt.BT8FXG) mkjournal error creating journal on /var/lib/ceph/tmp/mnt.BT8FXG/journal: (2) No such file or directory

2016-06-12 23:37:26.180752 7f8889654800 -1 OSD::mkfs: ObjectStore::mkfs failed with error -2

2016-06-12 23:37:26.180918 7f8889654800 -1 \*\* ERROR: error creating empty object store in /var/lib/ceph/tmp/mnt.BT8FXG: (2) No such file or directory

mount\_activate: Failed to activate

unmount: Unmounting /var/lib/ceph/tmp/mnt.BT8FXG

command\_check\_call: Running command: /bin/umount -- /var/lib/ceph/tmp/mnt.BT8FXG

启动OSD

第一步先进行分区和格式化，这里只给出node1的操作方式，其他两个节点的方式类似。

先来安装必要的工具：

yum install -y parted xfsprogs

[root@node1 vagrant]# parted /dev/sdb

GNU Parted 2.1

Using /dev/sdb

(parted) mklabel

New disk label type? gpt

(parted) p

Model: ATA VBOX HARDDISK (scsi)

Disk /dev/sdb: 107GB

Sector size (logical/physical): 512B/512B

Partition Table: gpt

Number  Start  End  Size  File system  Name  Flags

(parted) mkpart

Partition name?  []? "Linux filesystem"

File system type?  [ext2]? xfs

Start? 0G

End? 107GB

(parted) p

Model: ATA VBOX HARDDISK (scsi)

Disk /dev/sdb: 107GB

Sector size (logical/physical): 512B/512B

Partition Table: gpt

Number  Start   End    Size   File system  Name              Flags

1      1049kB  107GB  107GB               Linux filesystem

(parted) quit

格式化：

[root@node1 vagrant]# mkfs.xfs /dev/sdb1

meta-data=/dev/sdb1              isize=256    agcount=4, agsize=6553472 blks

        =                       sectsz=512   attr=2, projid32bit=1

        =                       crc=0        finobt=0

data     =                       bsize=4096   blocks=26213888, imaxpct=25

        =                       sunit=0      swidth=0 blks

naming   =version 2              bsize=4096   ascii-ci=0 ftype=0

log      =internal log           bsize=4096   blocks=12799, version=2

        =                       sectsz=512   sunit=0 blks, lazy-count=1

realtime =none                   extsz=4096   blocks=0, rtextents=0

我们把目录在node1上进行挂载。

mkdir -p /ceph/sdb

mount /dev/sdb1 /ceph/sdb

最后启动OSD，这里最重要的就是把我们刚刚挂载好的OSD的实际路径透传给Docker内部的/var/lib/ceph/osd，如果每个节点有多个OSD的情况下，只需要在Host上映射到不同的目录，启动Docker的时候变更和/var/lib/ceph/osd的映射关系即可。

sudo docker run -d \

   --net=host \

   -v /etc/ceph:/etc/ceph \

   -v /var/lib/ceph/:/var/lib/ceph/ \

   -v /dev/:/dev/ \

   -v /ceph/sdb:/var/lib/ceph/osd \

   --privileged=true \

   ceph/daemon osd\_directory

按照同样的方法，将node2和node3的OSD也加入到集群，最终的效果如下：

   cluster 0de1fc5a-084d-4396-bb0b-59db72a9a439

    health HEALTH\_WARN

           clock skew detected on mon.node2.docker.com

           64 pgs degraded

           64 pgs stuck unclean

           64 pgs undersized

           Monitor clock skew detected

    monmap e3: 3 mons at {node1.docker.com=192.168.33.11:6789/0,node2.docker.com=192.168.33.12:6789/0,node3.docker.com=192.168.33.13:6789/0}

           election epoch 6, quorum 0,1,2 node1.docker.com,node2.docker.com,node3.docker.com

    osdmap e13: 3 osds: 3 up, 3 in

           flags sortbitwise

     pgmap v18: 64 pgs, 1 pools, 0 bytes data, 0 objects

           4551 MB used, 11306 MB / 16720 MB avail

                 64 active+undersized+degraded

创建MDS

创建好基本的环境，其他的就容易了很多，下面来启动MDS。

sudo docker run -d \

   --net=host \

   -v /etc/ceph:/etc/ceph \

   -v /var/lib/ceph/:/var/lib/ceph/ \

   -e CEPHFS\_CREATE=1 \

   ceph/daemon mds

启动RGW，并且映射80端口

sudo docker run -d \

   -p 80:80 \

   -v /etc/ceph:/etc/ceph \

   -v /var/lib/ceph/:/var/lib/ceph/ \

   ceph/daemon rgw

最终的集群状态

   cluster 0de1fc5a-084d-4396-bb0b-59db72a9a439

    health HEALTH\_WARN

           clock skew detected on mon.node2.docker.com

           48 pgs stuck inactive

           48 pgs stuck unclean

           Monitor clock skew detected

    monmap e3: 3 mons at {node1.docker.com=192.168.33.11:6789/0,node2.docker.com=192.168.33.12:6789/0,node3.docker.com=192.168.33.13:6789/0}

           election epoch 6, quorum 0,1,2 node1.docker.com,node2.docker.com,node3.docker.com

    mdsmap e5: 1/1/1 up {0=mds-node1.docker.com=up:active}

    osdmap e25: 3 osds: 3 up, 3 in

           flags sortbitwise

     pgmap v38: 128 pgs, 9 pools, 588 bytes data, 11 objects

           6791 MB used, 16996 MB / 25081 MB avail

                 80 active+clean

                 45 creating

                  3 creating+activating

总结

在[**Docker**](http://www.dockerinfo.net/)中部署Ceph并没有想象中的那么顺利，社区的版本中仍然有Bug需要解决。

Docker作为一种快捷的部署方式，的确可以大幅度提高Ceph的部署效率，提高扩展的速度。但是从另一个角度我们应该注意到，随着Docker的引入也改变了Ceph的运维方式，比如在OSD增减的时候，需要到容器中对Ceph集群进行维护。再比如配置文件变更后的重启问题等。

Ceph中国社区Ceph中国社区是由中国开源云联盟WG8工作组领导的开源社区组织，2016年4月份得到Ceph社区官方的正式授权，社区秉承公正、公开、公平的理念，为广大Ceph爱好者提供交流平台。官方网站：www.ceph.org.cn

# How I barely got my first Ceph monitor running in Docker | Sébastien Han

http://www.sebastien-han.fr/blog/2013/09/19/how-I-barely-got-my-first-ceph-mon-running-in-docker/

Docker is definitely the new trend. Thus I quickly wanted to try to put a Ceph monitor inside a Docker container. Story of a tough journey…

First let’s start with the DockerFile, this makes the setup easy and repeatable by anybody:

FROM ubuntu:latest

MAINTAINER Sebastien Han <han.sebastien@gmail.com>

# Hack for initctl not being available in Ubuntu

RUN dpkg-divert --local --rename --add /sbin/initctl

RUN ln -s /bin/true /sbin/initctl

# Repo and packages

RUN echo deb http://archive.ubuntu.com/ubuntu precise main | tee /etc/apt/sources.list

RUN echo deb http://archive.ubuntu.com/ubuntu precise-updates main | tee -a /etc/apt/sources.list

RUN echo deb http://archive.ubuntu.com/ubuntu precise universe | tee -a /etc/apt/sources.list

RUN echo deb http://archive.ubuntu.com/ubuntu precise-updates universe | tee -a /etc/apt/sources.list

RUN apt-get update

RUN apt-get install -y --force-yes wget lsb-release sudo

# Fake a fuse install otherwise ceph won't get installed

RUN apt-get install libfuse2

RUN cd /tmp ; apt-get download fuse

RUN cd /tmp ; dpkg-deb -x fuse\_\* .

RUN cd /tmp ; dpkg-deb -e fuse\_\*

RUN cd /tmp ; rm fuse\_\*.deb

RUN cd /tmp ; echo -en '#!/bin/bash\nexit 0\n' > DEBIAN/postinst

RUN cd /tmp ; dpkg-deb -b . /fuse.deb

RUN cd /tmp ; dpkg -i /fuse.deb

# Install Ceph

CMD wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob\_plain;f=keys/release.asc' | apt-key add -

RUN echo deb http://ceph.com/debian-dumpling/ $(lsb\_release -sc) main | tee /etc/apt/sources.list.d/ceph-dumpling.list

RUN apt-get update

RUN apt-get install -y --force-yes ceph ceph-deploy

# Avoid host resolution error from ceph-deploy

RUN echo ::1 ceph-mon | tee /etc/hosts

# Deploy the monitor

RUN ceph-deploy new ceph-mon

EXPOSE 6789

Then build the image:

|  |
| --- |
| $ sudo docker build -t leseb/ceph-mon . ... ... ...  ---> 113b00f4dc3a Step 23 : RUN echo ::1 ceph-mon | tee /etc/hosts  ---> Running **in** 1f67db0c963a ::1 ceph-mon  ---> 556d638a365b Step 24 : RUN ceph-deploy new ceph-mon  ---> Running **in** 547e61297891 /usr/lib/python2.7/dist-packages/pushy/transport/ssh.py:323: UserWarning: No paramiko or native ssh transport  warnings.warn("No paramiko or native ssh transport") [ceph\_deploy.new][DEBUG ] Creating new cluster named ceph [ceph\_deploy.new][DEBUG ] Resolving host ceph-mon [ceph\_deploy.new][DEBUG ] Monitor ceph-mon at ::1 [ceph\_deploy.new][DEBUG ] Monitor initial members are ['ceph-mon'] [ceph\_deploy.new][DEBUG ] Monitor addrs are ['::1'] [ceph\_deploy.new][DEBUG ] Creating a random mon key... [ceph\_deploy.new][DEBUG ] Writing initial config to ceph.conf... [ceph\_deploy.new][DEBUG ] Writing monitor keyring to ceph.mon.keyring...  ---> 2b087f2f3ead Step 25 : EXPOSE 6789  ---> Running **in** 0c174fbe7a5b  ---> 460e2d2c900a Successfully built 460e2d2c900a |

Now we almost have th full image, we just need to instruct Docker to install the monitor. For this, we simply run the image that we just created and we pass the command that creates the monitor:

|  |
| --- |
| $ docker run -d -h="ceph-mon" leseb/ceph-mon ceph-deploy --overwrite-conf mon create ceph-mon e2f48f3cca26 |

Check if it works properly:

|  |
| --- |
| $ docker logs e2f48f3cca26 /usr/lib/python2.7/dist-packages/pushy/transport/ssh.py:323: UserWarning: No paramiko or native ssh transport  warnings.warn("No paramiko or native ssh transport") [ceph\_deploy.mon][DEBUG ] Deploying mon, cluster ceph hosts ceph-mon [ceph\_deploy.mon][DEBUG ] detecting platform **for** host ceph-mon ... [ceph\_deploy.mon][INFO ] distro info: Ubuntu 12.04 precise [ceph-mon][DEBUG ] deploying mon to ceph-mon [ceph-mon][DEBUG ] remote hostname: ceph-mon [ceph-mon][INFO ] write cluster configuration to /etc/ceph/{cluster}.conf [ceph-mon][INFO ] creating path: /var/lib/ceph/mon/ceph-ceph-mon [ceph-mon][DEBUG ] checking **for** **done** path: /var/lib/ceph/mon/ceph-ceph-mon/**done** [ceph-mon][DEBUG ] **done** path does not exist: /var/lib/ceph/mon/ceph-ceph-mon/**done** [ceph-mon][INFO ] creating keyring file: /var/lib/ceph/tmp/ceph-ceph-mon.mon.keyring [ceph-mon][INFO ] create the monitor keyring file [ceph-mon][INFO ] Running command: ceph-mon --cluster ceph --mkfs -i ceph-mon --keyring /var/lib/ceph/tmp/ceph-ceph-mon.mon.keyring [ceph-mon][INFO ] ceph-mon: mon.noname-a [::1]:6789/0 is local, renaming to mon.ceph-mon [ceph-mon][INFO ] ceph-mon: set fsid to b8344267-3857-4ead-bb38-2fb54566341e [ceph-mon][INFO ] ceph-mon: created monfs at /var/lib/ceph/mon/ceph-ceph-mon **for** mon.ceph-mon [ceph-mon][INFO ] unlinking keyring file /var/lib/ceph/tmp/ceph-ceph-mon.mon.keyring [ceph-mon][INFO ] create a **done** file to avoid re-doing the mon deployment [ceph-mon][INFO ] create the init path **if** it does not exist [ceph-mon][INFO ] Running command: initctl emit ceph-mon cluster=ceph id=ceph-mon |

Then commit the last version of your image to save the latest change:

|  |
| --- |
| $ docker commit e2f48f3cca26 leseb/ceph-mon 86f44bce988e |

Finally run the monitor in a new container:

|  |
| --- |
| $ docker run -d -p 6789 -h="ceph-mon" leseb/ceph ceph-mon --conf /ceph.conf --cluster=ceph -i ceph-mon -f 12974394437d root@hp-docker:~*# docker ps* ID IMAGE COMMAND CREATED STATUS PORTS 12974394437d leseb/ceph:latest ceph-mon --conf /cep 2 seconds ago Up 1 seconds 49175->6789 |

Now the tough part, because of the use of ceph-deploy the monitor listens to the IPv6 local address. Which in normal circonstances is not a problem since we can access from either its local IP (lo) or its private address (eth0 or something else). However with Docker, things are a little bit different, the monitor is only accessible from its namespace, so even if you expose a port this won’t work. Basically exposing a port creates an Iptables DNAT rule, that says: everything that goes from anywhere to the host IP address on a specific port is redirected to the IP address within the container namespace. In the end, if you try to access the monintor using the IP address of the host plus the exposed port you will get something like this:

.connect claims to be [::1]:6804/1031425 not [::1]:6804/31537 - wrong node!

Although there is a way to access the monitor! We need to access it from host directly through the namespace.

First grab your container’s ID:

|  |
| --- |
| $ docker ps ID IMAGE COMMAND CREATED STATUS PORTS 9cfa541f6be9 leseb/ceph:latest ceph-mon --conf /cep 25 hours ago Up 25 hours 49156->6789 |

Use this script, **stolen and adapt** from Jérôme Petazzoni [here](https://github.com/jpetazzo/pipework/blob/master/pipework#L54). This script creates the entry point on the host to access the namespace of the container.

|  |
| --- |
| #!/bin/bash set -e  GUESTNAME=$1  *# Second step: find the guest (for now, we only support LXC containers)* CGROUPMNT=$(grep ^cgroup.\*devices /proc/mounts | cut -d" " -f2 | head -n 1) [ "$CGROUPMNT" ] || {  echo "Could not locate cgroup mount point."  exit 1 }  N=$(find "$CGROUPMNT" -name "$GUESTNAME\*" | wc -l) **case** "$N" **in**  0)  echo "Could not find any container matching $GUESTNAME."  exit 1  ;;  1)  true  ;;  \*)  echo "Found more than one container matching $GUESTNAME."  exit 1  ;; **esac**  NSPID=$(head -n 1 $(find "$CGROUPMNT" -name "$GUESTNAME\*" | head -n 1)/tasks) [ "$NSPID" ] || {  echo "Could not find a process inside container $GUESTNAME."  exit 1 } mkdir -p /var/run/netns rm -f /var/run/netns/$NSPID ln -s /proc/$NSPID/ns/net /var/run/netns/$NSPID  echo "" echo "Namespace is ${NSPID}" echo ""  ip netns exec $NSPID ip a s eth0 |

Execute it:

|  |
| --- |
| $ ./pipework.sh 9cfa541f6be9  Namespace is 10660  607: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP qlen 1000  link/ether b6:96:a3:c3:c7:1f brd ff:ff:ff:ff:ff:ff  inet 172.17.0.8/16 brd 172.17.255.255 scope global eth0  inet6 fe80::b496:a3ff:fec3:c71f/64 scope link  valid\_lft forever preferred\_lft forever |

Now, get the monitor’s key:

|  |
| --- |
| $ cp /var/lib/docker/containers/9cfa541f6be97821131355b4005bc24b509baf3028759f0f871bf43840399f96/rootfs/ceph.mon.keyring ceph.mon.docker.keyring [mon.] key = AQANAipSAAAAABAApGcUJIxy+DO56vP4UpIV5g== caps mon = allow \* |

Ouahh YEAH!

|  |
| --- |
| $ sudo ip netns exec 10660 ceph -k ceph.mon.docker.keyring -n mon. -m 172.17.0.8 -s  cluster c957629f-525d-4b60-a6b7-e1ccd9494063  health HEALTH\_ERR 192 pgs stuck inactive; 192 pgs stuck unclean; no osds  monmap e1: 1 mons at {ceph-mon=172.17.0.8:6789/0}, election epoch 2, quorum 0 ceph-mon  osdmap e1: 0 osds: 0 up, 0 **in**  pgmap v2: 192 pgs: 192 creating; 0 bytes data, 0 KB used, 0 KB / 0 KB avail  mdsmap e1: 0/0/1 up |

**III. Issues and caveats**

I’m not really convinced by this first shot. The biggest issue here is that the monitor needs to be known.

Wow that was a hell of a job to get this working. At the end, the effort is quite useless since nothing can reach the monitor except the host itself. Thus, other Ceph components will only work if they share the same network namespace as the monitor. Sharing all the containers namespace into one could quite difficult as well. But what’s the point to have a Ceph cluster stuck within some namespaces, without any clients accessing it?

*I have to admit that this was pretty fun to hack. Although, in practice, that’s not usable at all. Thus you can consider this as an experiment and a way to get into Docker ;-).*

# Bootstrap your Ceph cluster in Docker | Sébastien Han

http://www.sebastien-han.fr/blog/2015/06/23/bootstrap-your-ceph-cluster-in-docker/

Almost two years have passed since my first attempt to [run Ceph inside Docker](http://www.sebastien-han.fr/blog/2013/09/19/how-I-barely-got-my-first-ceph-mon-running-in-docker/). Time has elapsed and I haven’t really got the time to resume this work until recently. For the last couple of months, I have been devoting a third part of my time to contributing on deploying [Ceph in Docker](https://github.com/ceph/ceph-docker). Before we start, I would like to highlight that nothing of this work would have been possible without the help of [Seán C. McCord](http://www.cycoresys.com/). Indeed the current ceph-docker repository is based on Seán’s initial work. Let’s see how you can get this running!

**Rationale**

Running Ceph inside Docker is a bit controversial and many people might believe that there is no point doing this. Where it is not really a problem for Monitors, Metadata Server and Rados Gateway to be containerized things get tricky when it comes to the OSDs. The Ceph OSD is really tighten to the machine it runs on, having such strong relationship with the hardware is not something common for all the softwares. Given that the OSD can not work if the disk that it relies on die is a bit of an issue in this container world.

To be honest at some point, I was thinking this:

*I don’t know why am I doing this. I just know that people out there want it (and yes they probably don’t know why). I can feel it’s important to do it anyway, so let’s do it.*

This does not sound really optimistic I know, but it’s somehow the truth. My vision has slightly changed though, so for what it’s worth let me explain why. We will see if you will change your mind as well. And yes my explanation will be more than: Docker is fancy, so let’s Dockerize everything.

People have started investing a lot of engineering efforts to run containerized softwares on their platforms. Thus they have been using various tools to build and orchestrate their environment. And I won’t be surprised to see Kubernetes being the orchestration tool for this matter. Some people also love to run bleeding edge technologies on production as they might find other things boring (right Seán?). So with the containerize everything approach, they will be happy that something is happening on their favorite open source storage solution :).

Where with yum or apt-get it is not easy to rollback, this is different with containers. Upgrades and rollback are made easier, as you can easily docker stop and docker run a new version of your daemons. You can also potentially run different clusters on an isolated fashion on the same machine. This makes development ideal.

**The project**

As mentioned, everything started from Seán C. McCord work and we iterated around his work together. Currently if you use ceph-docker you will be able to run every single Ceph daemon either on Ubuntu or CentOS. We have a lot of images available on[Docker Hub](https://hub.docker.com/). We have the Ceph namespace, so our images are prefixed as ceph/<daemon>. We use automated builds, as a result everytime we merge a new patch and new build gets triggered and produces a new version of the container image. As we are currently in a refactoring process, you will see that a lot of images are available. Historically we had (and we still do until we merge this [patch](https://github.com/ceph/ceph-docker/pull/94)) one image per daemon. So one container image for monitor, osd, mds and radosgw. This is not really ideal and in practice not needed. This is why we [worked](https://github.com/ceph/ceph-docker/pull/78) on a single container image called daemon. This image contains all the Ceph daemons and you activate the one you want with a parameter while invoking the docker run command. That being said, if you want to start I encourage you to directly use the ceph/daemon image. I will show example in the next section on how to run it.

**Containerize Ceph**

**Monitors**

Given that monitors can not communicate through a NATed network we need to use the --net=host to expose Docker’s host machine network stack:

|  |
| --- |
| $ sudo docker run -d --net=host \ -v /etc/ceph:/etc/ceph \ -v /var/lib/ceph/:/var/lib/ceph \ -e MON\_IP=192.168.0.20 \ -e CEPH\_PUBLIC\_NETWORK=192.168.0.0/24 \ ceph/daemon mon |

List of available options:

* MON\_IP is the IP address of your host running Docker
* MON\_NAME is the name of your monitor (DEFAULT: $(hostname))
* CEPH\_PUBLIC\_NETWORK is the CIDR of the host running Docker, it should be in the same network as the MON\_IP
* CEPH\_CLUSTER\_NETWORK is the CIDR of a secondary interface of the host running Docker. Used for the OSD replication traffic.

**Object Storage Daemon**

The current implementation allows you to run a single OSD process per container Following the microservice mindset we should not run more than one service inside our container. In our case, running multiple OSD processes into a single container breaks this rule and will likely introduce undesirable behaviours. This will also increase the setup and maintenance complexity of the solution.

In this configuration, the usage of --privileged=true is strictly required because we need a full access to /dev/ and other kernel functions. However, we support another configuration based on simply exposing OSD directories, where the operators will do the appropriate preparation of the devices. Then he/she will simply expose the OSD directory and populating (ceph-osd mkfs) the OSD will be done by the entrypoint. The configuration I’m presenting now is easier to start with because you only need to specify a block device and the entrypoint will do the rest.

For those who do not want to use --privileged=true, please fall back on the second example.

|  |
| --- |
| $ sudo docker run -d --net=host \ --privileged=true \ -v /etc/ceph:/etc/ceph \ -v /var/lib/ceph/:/var/lib/ceph \ -v /dev/:/dev/ \ -e OSD\_DEVICE=/dev/vdd \ ceph-daemon osd\_ceph\_disk |

If you don’t want to use --privileged=true you can always prepare the OSD by yourself with the help of your configuration management of your choice.

Example without a privileged mode, in this example we assume that you partitioned, put a filesystem and mounted the OSD partition. To create your OSDs simply run the following command:

|  |
| --- |
| $ sudo docker exec <mon-container-id> ceph osd create. |

Then run your container like so:

|  |
| --- |
| $ sudo docker run -v /osds/1:/var/lib/ceph/osd/ceph-1 -v /osds/2:/var/lib/ceph/osd/ceph-2  $ sudo docker run -d --net=host \ -v /etc/ceph:/etc/ceph \ -v /var/lib/ceph/:/var/lib/ceph \ -v /osds/1:/var/lib/ceph/osd/ceph-1 \ ceph-daemon osd\_disk\_directory |

List of available options:

* OSD\_DEVICE is the OSD device, ie: /dev/sdb
* OSD\_JOURNAL is the device that will be used to store the OSD’s journal, ie: /dev/sdz
* HOSTNAME is the hostname of the hostname of the container where the OSD runs (DEFAULT: $(hostname))
* OSD\_FORCE\_ZAP will force zapping the content of the given device (DEFAULT: 0 and 1 to force it)
* OSD\_JOURNAL\_SIZE is the size of the OSD journal (DEFAULT: 100)

**Metadata Server**

This one is pretty straighforward and easy to bootstrap. The only caviat at the moment is that we require the Ceph admin key to be available in the Docker. This key will be used to create the CephFS pools and the filesystem.

If you run an old version of Ceph (prior to 0.87) you don’t need this, but you will likely do since it is always better to run the last version!

|  |
| --- |
| $ sudo docker run -d --net=host \ -v /var/lib/ceph/:/var/lib/ceph \ -v /etc/ceph:/etc/ceph \ -e CEPHFS\_CREATE=1 \ ceph-daemon mds |

List of available options:

* MDS\_NAME is the name of the Metadata server (DEFAULT: mds-$(hostname))
* CEPHFS\_CREATE will create a filesystem for your Metadata server (DEFAULT: 0 and 1 to enable it)
* CEPHFS\_NAME is the name of the Metadata filesystem (DEFAULT: cephfs)
* CEPHFS\_DATA\_POOL is the name of the data pool for the Metadata Server (DEFAULT: cephfs\_data)
* CEPHFS\_DATA\_POOL\_PG is the number of placement groups for the data pool (DEFAULT: 8)
* CEPHFS\_DATA\_POOL is the name of the metadata pool for the Metadata Server (DEFAULT: cephfs\_metadata)
* CEPHFS\_METADATA\_POOL\_PG is the number of placement groups for the metadata pool (DEFAULT: 8)

**Rados Gateway**

For the Rados Gateway, we deploy it with civetweb enabled by default. However it is possible to use different CGI frontends by simply giving remote address and port.

|  |
| --- |
| $ sudo docker run -d --net=host \ -v /var/lib/ceph/:/var/lib/ceph \ -v /etc/ceph:/etc/ceph \ ceph-daemon rgw |

List of available options:

* RGW\_REMOTE\_CGI defines if you use the embedded webserver of Rados Gateway or not (DEFAULT: 0 and 1 to disable it)
* RGW\_REMOTE\_CGI\_HOST is the remote host running a CGI process
* RGW\_REMOTE\_CGI\_PORT is the remote port of the host running a CGI process
* RGW\_CIVETWEB\_PORT is the listenning port of civetweb (DEFAULT: 80)
* RGW\_NAME is the name of the Rados Gateway instance (DEFAULT: $(hostname))

**Further work**

**Configuration store backends**

By default, the ceph.conf and all the ceph keys are generated during the initial monitor bootstrap. This process assumes that to extand your cluster to multiple nodes you have to distribute these configurations across all the nodes. This is not really flexible and we want to improve this. One thing that I will propose soon is to use Ansible to generate the configuration/keys and to distribute them on all the machines.

Alternatively, we want to be able to store various configuration files on different backends kv store like [etcd](https://github.com/coreos/etcd) and [consul](https://www.consul.io/).

**Orchestrate the deployment**

A very first step is to use [ceph-ansible](https://github.com/ceph/ceph-ansible) where the logic is already implemented. I just need to push some changes, but most of the work is already present.

[Kubernetes](http://kubernetes.io/), a preview on how to bootstrap monitors is already [available](https://github.com/ceph/ceph-docker/tree/master/examples/kubernetes).

**Extending to Rocket and beyond**

There is not much to do here as you can simply port your Docker images into Rocket and launch them (pun intented here).

# ceph/ceph-docker: Docker files and images to run Ceph in containers

https://github.com/ceph/ceph-docker

docker-ceph

Ceph-related Docker files.

Core Components:

* [ceph/base](https://github.com/ceph/ceph-docker/blob/master/ceph-releases/jewel/ubuntu/14.04/base): Ceph base container image. This is nothing but a fresh install of the latest Ceph + Ganesha on Ubuntu LTS (14.04)
* [ceph/daemon](https://github.com/ceph/ceph-docker/blob/master/ceph-releases/jewel/ubuntu/14.04/daemon): All-in-one container for all core daemons.

Utilities and convenience wrappers

* [ceph/config](https://github.com/ceph/ceph-docker/blob/master/config): DEPRECATED (not maintained anymore since the logic is now embedded in the daemon image) Initializes and distributes cluster configuration
* [ceph/docker-registry](https://github.com/ceph/ceph-docker/blob/master/docker-registry): Rados backed docker-registry images repository
* [ceph/rados](https://github.com/ceph/ceph-docker/blob/master/rados): Convenience wrapper to execute the rados CLI tool
* [ceph/rbd](https://github.com/ceph/ceph-docker/blob/master/rbd): Convenience wrapper to execute the rbd CLI tool
* [ceph/rbd-lock](https://github.com/ceph/ceph-docker/blob/master/rbd-lock): Convenience wrapper to block waiting for an rbd lock
* [ceph/rbd-unlock](https://github.com/ceph/ceph-docker/blob/master/rbd-unlock): Convenience wrapper to release an rbd lock
* [ceph/rbd-volume](https://github.com/ceph/ceph-docker/blob/master/rbd-volume): Convenience wrapper to mount an rbd volume

Demo

* [ceph/demo](https://github.com/ceph/ceph-docker/blob/master/ceph-releases/jewel/ubuntu/14.04/demo): Demonstration cluster for testing and learning. This container runs all the major ceph and ganesha components installed, bootstrapped, and executed for you to play with. (not intended for use in building a production cluster)

How to contribute?!

The following assumes that you already forked the repository, added the correct remote and are familiar with git commands.

Prepare your development environment

Simply execute ./generate-dev-env.sh CEPH\_RELEASE DISTRO DISTRO\_VERSION. For example if you run ./generate-dev-env.sh jewel ubuntu 16.04 the script will:

* hardlink the files from ceph-releases/jewel/ubuntu/16.04/{base,daemon,demo} in ./base, ./daemon and ./demo.
* create a file in {base,daemon,demo}/SOURCE\_TREE which will remind you the version you are working on.

From this, you can start modifying your code and building your images locally.

My code is ready, what's next?

Contributions must go in the 'ceph-releases' tree, in the appropriate Ceph version, distribution and distribution version. So once you are done, we can just run:

* cp -av base $(cat base/SOURCE\_TREE)
* cp -av daemon $(cat base/SOURCE\_TREE)
* cp -av demo $(cat base/SOURCE\_TREE)

We identified 2 types of contributions:

Distro specific contributions

The code only changes the base image content of a specific distro, nothing to replicate or change for the other images..

New functionality contributions

If you look at the ceph-releases directory you will notice that for each release the daemon image's content is symlinked to the Ubuntu daemon 14.04. Even if we support multi-distro, Ubuntu 14.04 remains the default. It would nice if you could get familiar with this approach. This basically means that if you are testing on CentOS then you should update the Ubuntu image instead. All the changes in the entrypoints should not diverse from one distro to another, so this should be safe :). We are currently **only**bringing new functionality in the jewel release.

CI

We use Travis to run several tests on each pull request:

* we build both base and daemon images
* we run all the ceph processes in a container based on the images we just built
* we execute a validation script at the end to make sure Ceph is healthy

For each PR, we try to detect which Ceph release is being impacted. Since we can only produce a single CI build with Travis, ideally this change will only be on a single release and distro. If we have multiple ceph release and distro, we can only test one, since we have to build base and daemon. By default, we just pick up the first line that comes from the changes.

You can check the files in travis-builds to learn more about the entire process.

If you don't want to run a build for a particular commit, because all you are changing is the README for example, add [ci skip] to the git commit message. Commits that have [ci skip] anywhere in the commit messages are ignored by Travis CI.

We are also transitioning to have builds in Jenkins, this is still a work in progress and will start taking precedence once it is solid enough. Be sure to check the links and updates provided on pull requests.

Images workflow

Once your contribution is done and merged in master. Either @Ulexus or @leseb will execute ceph-docker-workflow.sh, this will basically compare the content of each tag/branch to master. If any difference is found it will push the appropriate changes in each individual branches. Ultimately new pushed tags will trigger a Docker build on the Docker Hub.

Video demonstration

Manually

A recorded video on how to deploy your Ceph cluster entirely in Docker containers is available here:

[](http://youtu.be/FUSTjTBA8f8)

With Ansible

[](http://youtu.be/DQYZU1VsqXc)