大型架构及配置技术

[Docker概述 1](#_Toc651887026)

[部署Docker 2](#_Toc362797311)

[docker基本操作 3](#_Toc2061253459)

[创建启动 容器的命令 4](#_Toc1894907122)

[镜像常用命令 6](#_Toc1625875127)

[容器命令列表 8](#_Toc756899524)

[连接正在运行的docker 9](#_Toc166878607)

[设置IP伪装访问网络 11](#_Toc1109160565)

[定义个私人镜像 11](#_Toc119709560)

[使用Dockerfile创建镜像 12](#_Toc1851759850)

[自定义镜像仓库 14](#_Toc1393817937)

[持久化存储 15](#_Toc558421790)

[Docker网络架构 16](#_Toc1316830124)

[查看Docker默认网络模型 17](#_Toc1787758423)

[实战】五个Docker监控工具的对比 20](#_Toc1000726360)

[Docker Stats命令 20](#_Toc774948015)

[CAdvisor 21](#_Toc602117295)

[Scout 22](#_Toc1088397287)

[Data Dog 28](#_Toc83911440)

[Sensu Monitoring Framework 33](#_Toc337408888)

当启动docker服务时，系统iptables会自动创建防火墙规则

iptables-save可以看到

开机不启动docker服务就没有这么规则

如果手动修改防火墙规则会对docker服务产生影响

所以一般会卸载Firewalld服务

# Docker概述

什么是容器

• 容器技术已经成为应用程序封装和交付的核心技术

• 容器技术的核心有以下几个内核技术组成:

– CGroups(Control Groups)-资源管理

– NameSpace-进程隔离

– SELinux安全

• 由于是在物理机上实施隔离,启动一个容器,可以像启动一个进程一样快速

什么是Docker

• Docker是完整的一套容器管理系统

• Docker提供了一组命令,让用户更加方便直接地使

用容器技术,而不需要过多关心底层内核技术

Docker优点

• 相比于传统的虚拟化技术,容器更加简洁高效

• 传统虚拟机需要给每个VM安装操作系统

• 容器使用的共享公共库和程序

Docker的缺点

• 容器的隔离性没有虚拟化强

• 共用Linux内核,安全性有先天缺陷

• SELinux难以驾驭

• 监控容器和容器排错是挑战

虚拟机:在真机上再启用一个系统. 兼容性更高

容器: 云技术的重要组成部分, 打包程序和运行环境,快速的安装部署及迭代. 受真机限制,不能跨系统

虚拟机里面执行命令,真机使用ps -ef 是看不到的

容器里面执行命令,真机能后看到.也能杀死

容器只打包运行环境,不打包数据.数据可以通过nfs挂载

Docker 只推荐使用rhel7

# 部署Docker

安装前准备

• 需要64位操作系统

• 至少RHEL6.5以上的版本,强烈推荐RHEL7

• 关闭防火墙(不是必须)

安装Docker

• 软件包列表:

– docker-engine

– docker-engine-selinux(如果启用selinux)

准备2台虚拟机，6G内存，两台都安装docker，主机名docker01和docker02

[root@localhost ~]# systemctl start docker

[root@localhost ~]# systemctl enable docker

[root@localhost ~]# ifconfig docker0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0

ether 02:42:8f:62:2b:5c txqueuelen 0 (Ethernet)

RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

使用docker version可以查看docker 的版本

# docker基本操作

什么是镜像

• 在Docker中容器是基于镜像启动的

• 镜像是启动容器的核心

• 镜像采用分层设计

• 使用快照的COW技术,确保底层数据不丢失

加载时使用联合mount技术

docker是建立在前端镜像（node1.img），后端的模板镜像不能改（node.qcow2）

建立docker的过程就是根据模板镜像自动创建一个前端镜像

[root@docker01 ~]# docker images 查看本机的后端存储node.qcow2

docker search 搜索镜像

docker pull 下载镜像

docker push 上传镜像

查看帮助 docker help pull

docker search busybox

docker pull busybox

[root@docker01 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 8c811b4aec35 5 weeks ago 1.146 MB

[root@docker01 ~]# docker save busybox:latest >busybox.tar //将docker01下载的镜像打包，可用于dock02

[root@docker01 ~]# scp busybox.tar 192.168.1.113:./

[root@docker02 ~]# docker load < busybox.tar

**导入、导出镜像**

• 导出镜像(将本地镜像导出为tar文件)

docker save 名称：标签 > 名称.tar 保存镜像

• 导入镜像(通过tar包文件导入镜像)

docker load < 名称.tar 载入镜像

[root@docker01 ~]# lftp 192.168.1.254

lftp 192.168.1.254:/> cd docker/

lftp 192.168.1.254:/docker> get docker\_images.zip 下载要使用的镜像文件

[root@docker01 ~]# unzip docker\_images.zip 解压缩

[root@docker01 ~]# cd docker\_images/

[root@docker01 docker\_images]# ls

centos.tar nginx.tar redis.tar registry.tar ubuntu.tar

[root@docker01 docker\_images]# for i in \*;do docker load < $i;done //将5个镜像导入docker

[root@docker01 ~]# docker images

## 创建启动 容器的命令

docker run -it 镜像的名称：镜像的标签 【启动命令】 （如果没有写标签，启动的默认的latest标签）

这里的启动命令是容器里面的命令，容器里面没有，就启动不了

-i 表示交互的

-t 表示终端

-d 后台进程，非交互式 //服务类的镜像 如nginx和redis docker run -d nginx

docker run **-p** 物理端口：容器端口 //将物理机端口和容器端口进行绑定

docker run -it centos cmd 启动一个交互式的容器，在前台运行

docker run -d centos cmd 启动一个非交互式的容器，在后台运行

docker run -itd centos cmd 启动一个交互式的容器，在后台运行

交互式的进程启动要使用 it ， 非交互式的使用 d ，交互式的放后台使用 itd

**因为:**

对于有交互式的镜像启动docker 如果只使用-d ，这个容器会启动后立即关闭。

因为他相当于运行一个非交互式shell脚本

a.sh

#!/binbash

./a.sh 会瞬间退出，启动完马上结束。

交互式的特点：打开了标准输入，标准输出，标准错误

如果关闭这三个标准，就相当于变成了非交互式，终端会直接关闭

在终端运行exit就是这样

或者

exec >&- 0<&- //关闭了标准输入和标准输出

直接退出终端

docker run -itd centos /bin/bash

docker run -itd redis

docker run -d -p 80:80 nginx

这样就会全都正常启动放入后台

例如 镜像名字和标签 centos6 apache；； centos7 mysql

latest 是使用docker save 命令时，默认的标签

【启动命令】 容器运行的命令 /bin/bash等等，如果不写命令，使用的是默认的命令

查看详细信息

docker inspect 镜像的名称：镜像的标签 //可以看到默认的启动命令和其他信息

[root@docker01 ~]# docker run -it centos /bin/bash

[root@7c8ea5750f5e /]# cd /etc/yum.repos.d/

[root@7c8ea5750f5e yum.repos.d]# rm -rf \*

[root@7c8ea5750f5e yum.repos.d]# cat > dvd.repo <<EE

> [dvd]

> name=dvd

> baseurl=ftp://192.168.1.254/centos7

> enabled=1

> gpgcheck=0

> EE

[root@7c8ea5750f5e yum.repos.d]# ifconfig eth0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.17.0.2 netmask 255.255.0.0 broadcast 0.0.0.0

可以看到已经进入了docker容器

[root@7c8ea5750f5e ~]# ls -al

total 24

dr-xr-x--- 2 root root 114 Apr 2 18:39 .

drwxr-xr-x 16 root root 255 Jun 29 00:29 ..

-rw-r--r-- 1 root root 18 Dec 29 2013 .bash\_logout

-rw-r--r-- 1 root root 176 Dec 29 2013 .bash\_profile

-rw-r--r-- 1 root root 176 Dec 29 2013 .bashrc

-rw-r--r-- 1 root root 100 Dec 29 2013 .cshrc

-rw-r--r-- 1 root root 129 Dec 29 2013 .tcshrc

-rw------- 1 root root 3270 Apr 2 18:39 anaconda-ks.cfg

只要退出，再重新docker run -it centos /bin/bash 进入，又会恢复一个新的docker

1、容器退出后，怎么进入原来的容器

容器启动后，相当于一个系统的进程，退出后就关闭了，

如果想进入原来的容器，需要重新启动这个容器

2、为什么有的容器不加启动命令就起不来（nginx，redis）

因为启动容器的默认命令是非交互式命令

查看redis镜像的信息时，启动命令是redis-server不是交互式的命令

3、启动参数-it是干什么用的

i 表示交互的

t 表示终端

4、redis系统为什么是空的

debian redhat

dpkg rpm

aptget yum

yum search vim

apt-cache search vim

5、怎么查看系统中有几个容器

在另一个终端查看

docker ps 正在运行的

docker ps -a 所有的镜像

6、怎么管理这些容器

用docker相关命令

Docker基本命令

## 镜像常用命令

• 命令列表

– docker images //查看镜像列表

– docker history //查看镜像制作历史，这个镜像打过几次包，执行的历史命令

– docker inspect //查看镜像底层信息

– docker pull //下载镜像

– docker push //上传镜像

– docker rmi //删除本地镜像 需要先删除前端docker容器，再删除后端镜像

– docker save //镜像另存为tar包

– docker load //使用tar包导入镜像

– docker search //搜索镜像

– docker tag //复制镜像 类似于创建一个软连接

[root@docker01 ~]# docker inspect centos

。。。

"Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

], //系统环境变量

"Cmd": [

"/bin/sh",

"-c",

"#(nop) ",

"CMD [\"/bin/bash\"]"

], //默认启动命令

[root@docker01 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 8c811b4aec35 5 weeks ago 1.146 MB

ubuntu latest 452a96d81c30 8 weeks ago 79.62 MB

centos latest e934aafc2206 11 weeks ago 198.6 MB

registry latest d1fd7d86a825 5 months ago 33.26 MB

nginx latest a5311a310510 20 months ago 181.4 MB

redis latest 1aa84b1b434e 21 months ago 182.8 MB

[root@docker01 ~]# docker tag 8c811b4aec35 nsd1802:latest

[root@docker01 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

nsd1802 latest 8c811b4aec35 5 weeks ago 1.146 MB

busybox latest 8c811b4aec35 5 weeks ago 1.146 MB

ubuntu latest 452a96d81c30 8 weeks ago 79.62 MB

centos latest e934aafc2206 11 weeks ago 198.6 MB

registry latest d1fd7d86a825 5 months ago 33.26 MB

nginx latest a5311a310510 20 months ago 181.4 MB

redis latest 1aa84b1b434e 21 months ago 182.8 MB

[root@docker01 ~]# docker tag nsd1802:latest aaa:v1

[root@docker01 ~]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

busybox latest 8c811b4aec35 5 weeks ago 1.146 MB

nsd1802 latest 8c811b4aec35 5 weeks ago 1.146 MB

aaa v1 8c811b4aec35 5 weeks ago 1.146 MB

ubuntu latest 452a96d81c30 8 weeks ago 79.62 MB

centos latest e934aafc2206 11 weeks ago 198.6 MB

registry latest d1fd7d86a825 5 months ago 33.26 MB

nginx latest a5311a310510 20 months ago 181.4 MB

redis latest 1aa84b1b434e 21 months ago 182.8 MB

## 容器命令列表

• 命令列表

– docker run //运行容器

– docker ps //查看容器列表

– docker stop 容器id//关闭容器

– docker start 容器id//启动容器

– docker restart 容器id//重启容器

– docker attach|exec 容器id/名字 //进入容器

– docker inspect 容器id//查看容器底层信息

– docker top 容器id//查看容器进程列表

– docker rm 容器id //删除容器

docker ps

• 列出容器列表

– docker ps 查看正在运行的容器

– docker ps -a 查看所有曾经启动过的容器列表

– docker ps -aq 仅显示容器id

[root@docker01 ~]# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

e0b6f138a241 nginx "nginx -g 'daemon off" 35 minutes ago Up 35 minutes 0.0.0.0:80->80/tcp, 443/tcp backstabbing\_pike

[root@docker01 ~]# docker ps -q

e0b6f138a241

[root@docker01 ~]# docker ps -qa

e0b6f138a241

43d92886e122

7c8ea5750f5e

[root@docker01 ~]# docker rm 43d92886e122

43d92886e122

[root@docker01 ~]# docker ps -qa

e0b6f138a241

7c8ea5750f5e

[root@docker01 ~]# docker rm $(docker ps -qa) 全部删除

启动容器时，比如docker run -it centos /bin/bash

进入之后ps -ef 可以看到pid为1的技能成是 /bin/bash 也就是说/bin/bash就是上帝进程，

运行exit pid为1的进程就会没了，整个系统就没了,直接停止了

容器进程和物理机的系统进程是相同的，在物理机可以看到docker里面的进程，如果杀掉，docker里面的也停止了

虚拟机和物理机是隔离的，虚拟机里面的进程物理机看不到，也没法干预

## 连接正在运行的docker

docker exec -it 容器id /bin/bashls

docker attach 容器id

exec 与 attach **的区别**

exec 单独启动命令运行，与容器启动的终端无关

attach 不启动新的命令，直接连接 console 终端

exec 退出不会影响容器的运行

attach 退出后，容器结束

Exec 后面得加it, 相当于ssh连接,exit退出后溶剂还存活.推荐

Attach 不用加it ,exit退出后容器消失. 使用快捷键 ctrl 不放, 按pq, 退出容器还存活.

当容器报错在终端,用attach连接过去排错

docker run -it 镜像名：标签 命令 （交互）

docker run -d 镜像名：标签 （非交互）

docker run -itd 镜像名：标签 命令 （交互式后台进程）

docker attach 容器id 进入容器console终端

docker exec -it 容器id 命令（/bin/bash）

############################################################################

docker 问题列表，及解答

问题：docker run -it busybox /bin/bash 报错问题

原因：

最后的命令是容器内部命令，容器内，存在才可以执行

可以不指定，不指定启动默认命令

问题：docker run -it centos 退出后，配置全部丢失

原因：

run 启动的是新的容器，老的容器退出后就停止了

启动管理一个老的容器可以使用 docker start|stop|restart

问题：老容器启动以后，怎么在进入？

docker exec -it 容器id /bin/bash

docker attach 容器id

exec 与 attach 的区别

exec 单独启动命令运行，与容器启动的终端无关

attach 不启动新的命令，直接连接 console 终端

exec 退出不会影响容器的运行

attach 退出后，容器结束

问题：如果我使用attach 连接容器后，怎么才能不结束容器？

解决方法：把容器放后台，使用快捷键 ctrl + pq

问题：attach 为什么退出后，容器会结束？

因为 attach 连接进容器的 pid 1 的进程，当 attach 结束时候，pid 为 1 的进程被结束

所有整个容器被销毁

问题：docker run -it nginx 没响应？

因为 nginx 启动的默认 cmd 时 nginx daemon，该进程不是一个交互式的进程

################################################################################

## 设置IP伪装访问网络

[root@room9pc19 docker]# ifconfig

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.40.50.119 netmask 255.255.255.0 broadcast 172.40.50.255

ether 94:de:80:81:e3:53 txqueuelen 1000 (Ethernet)

RX packets 5947707 bytes 593709850 (566.2 MiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 12611909 bytes 18788418441 (17.4 GiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

物理机上查找上网用的网卡

打开路由转发

sysctl -w net.ipv4.ip\_forward=1

设置伪装上网

[root@room9pc19 docker]# iptables -t nat -I POSTROUTING -s 192.168.4.0/24 -o enp2s0 -j MASQUERADE

在虚拟机里面设置默认路由

ip route replace default via 192.168.4.254

模拟 docker 端口绑定转发

iptables -t nat -I PREROUTING -d 192.168.4.10 -p tcp --dport 8080 -i eth0 -j DNAT --to 172.17.0.3:80

docker run -d -p 8080:80 nginx

# 定义个私人镜像

1、启动一个centos容器

[root@docker01 ~]# docker run -itd centos /bin/bash

2、进入容器

[root@docker01 ~]# docker exec -it f3d488dd03ce /bin/bash

（1）配置yum源

[root@f3d488dd03ce /]# cd /etc/yum.repos.d/

[root@f3d488dd03ce yum.repos.d]# rm -rf \*

[root@f3d488dd03ce yum.repos.d]# vi local.repo

[local\_repo]

name=Centos-$releasever-Base

baseurl=ftp://192.168.1.254/centos7/

enabled=1

gpgcheck=0

（2）安装常用软件包

yum provides 想用的命令

yum -y install net-tools iproute lftp bash-completion

3、配置完成后，停止容器，创建镜像

docker commit 容器ID 新镜像名称：标签

[root@f3d488dd03ce ~]# exit

exit

[root@docker01 ~]# docker attach f3 //f3是docker id

[root@f3d488dd03ce /]# exit

[root@docker01 ~]# docker ps -a

[root@docker01 ~]# docker commit f3d4 myos:latest

[root@docker01 ~]# docker rm f3

[root@docker01 ~]# docker images //可以看到新的镜像myos

使用run命令用myos镜像进入一个新镜像，可以看到yum源和软件包都有了

## 使用Dockerfile创建镜像

Dockerfile

• Dockerfile语法格式

– FROM:基础镜像

– MAINTAINER:镜像创建者信息

– EXPOSE:开放的端口

– ENV:设置变量

– ADD:复制文件到镜像

– RUN:制作镜像时执行的命令,可以有多个

– WORKDIR:定义容器默认工作目录

– CMD:容器启动时执行的命令,仅可以有一条CMD

使用dockerfile

[root@docker01 ~]# mkdir ooxx

[root@docker01 ~]# cd ooxx/

[root@docker01 ooxx]# vim Dockerfile //先进入myos查看命令历史，copy进来，然后删除不必要的

FROM centos:latest

RUN rm -rf /etc/yum.repos.d/\*

ADD local.repo /etc/yum.repos.d/ooxx.repo

RUN yum -y install net-tools iproute lftp bash-completion psmisc

CMD ["/usr/bin/python"]

[root@docker01 ooxx]# cp /etc/yum.repos.d/local.repo .

[root@docker01 ooxx]# docker build -t myos:python . 一定要在当前目录，创建了一个包含基本配置的镜像

提供docker的ssh服务

[root@docker01 ]# docker run -it myos

15 yum search sshd

16 yum -y install openssh-server

17 vim /usr/lib/systemd/sshd.service

18 vi /usr/lib/systemd/sshd.service

19 find / -name sshd.service

20 vi /usr/lib/systemd/system/sshd.service

21 /usr/sbin/sshd -D $OPTIONS

22 /usr/sbin/sshd -D 这里提示有一些key缺失，使用下面的命令生成

23 sshd-keygen

25 ls /etc/ssh/

26 /usr/sbin/sshd -D 操作这一步时，挂住不动了，说明服务已经启动了

27 history

(EnvironmentFile=/etc/sysconfig/sshd

ExecStart=/usr/sbin/sshd -D $OPTIONS) 这两行是查看启动服务的脚本的环境变量和启动命令

退出docker 然后创建这个镜像

[root@docker01 ~]# mkdir sshd

[root@docker01 ~]# cd sshd

[root@docker01 sshd]# vim Dockerfile

先在第一行添加

FROM myos:latest

将前面的history粘贴进来，删除不必要的操作

RUN yum -y install openssh-server

ENV EnvironmentFile=/etc/sysconfig/sshd

RUN /usr/sbin/sshd-keygen

EXPOSE 22

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

[root@docker01 sshd]# docker build -t myos:sshd .

[root@docker01 sshd]# docker run -d -p 1022:22 myos:sshd

[root@docker01 sshd]# docker ps 可以看到运行的容器和端口

用客户机去连接

[root@room9pc01 ~]# ssh 192.168.1.11 -p 1022 可以正常连接

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

一个docker启动多个服务

因CMD 只能启动一个命令，所以编写一个脚本用来作为启动命令

[root@docker01 ooxx]# vim Dockerfile

FROM myos:latest

RUN echo 123456 | passwd --stdin root

RUN yum -y install httpd openssh-server

RUN echo "ServerName www.example.com:80" >> /etc/httpd/conf/httpd.conf

WORKDIR /var/www/html

RUN echo test123 > index.html

RUN /usr/sbin/sshd-keygen

EXPOSE 22 80

ADD mixed.sh /root/mixed.sh

RUN /usr/bin/chmod +x /root/mixed.sh

CMD ["/root/mixed.sh"]

[root@docker01 ooxx]# vim mixed.sh

#!/bin/bash

EnvironmentFile=/etc/sysconfig/httpd

/usr/sbin/httpd -DFOREGROUND & 这里需要放入后台，因为这个服务本身会占用终端

EnvironmentFile=/etc/sysconfig/sshd

/usr/sbin/sshd -D

这里不能放入后台，因为放入后台就相当于一个非交互的命令，执行完毕就退出，导致docker启动即停止

[root@docker01 sshd]# docker build -t myos:mixed .

[root@docker01 sshd]# docker run -d -p 1022:22 -p 80:80 myos:mixed

[root@room9pc01 ~]# ssh 192.168.1.11 -p 1022 可以正常连接

[root@room9pc01 ~]# curl http://192.168.1.11

# 自定义镜像仓库

1、创建配置文件/etc/docker/daemon.json

{

"insecure-registries":["192.168.1.11:5000"] //这里填写的是启动5000服务的物理机ip

}

2、重启docker服务

[root@docker01 ~]# systemctl restart docker.service

3、启动私有仓库服务

[root@docker01 ~]# docker run -d -p 5000:5000 registry

[root@docker01 ~]# ss -anptul | grep 5000

tcp LISTEN 0 128 :::5000 :::\* users:(("docker-proxy",pid=1305,fd=4))

4、上传镜像到私有仓库

先给镜像打标签后上传

[root@docker01 ~]# docker tag busybox:latest 192.168.1.11:5000/busybox:latest

[root@docker01 ~]# docker push 192.168.1.11:5000/busybox:latest //看到v1表示失败

5、在另一台docker上测试

配置daemon.json文件，重启docker服务

[root@docker02 ~]# docker run -it 192.168.1.11:5000/busybox:latest

查询私有镜像仓库里面的镜像

使用docker api

curl http://192.168.1.11:5000/v2/\_catalog

查询私有仓库里面的镜像标签

curl http://192.168.1.11:5000/v2/busybox/tags/list

# 持久化存储

容器内不存放数据，使用存储卷来存放数据

存储卷

卷的概念

• docker容器不保持任何数据

• 重要数据请使用外部卷存储(数据持久化)

• 容器可以挂载真实机目录或共享存储为卷

主机卷的映射

• 将真实机目录挂载到容器中提供持久化存储

[root@jacob ~]# docker run -v /data:/data -it centos bash

共享存储

共享存储基本概念

• 一台共享存储服务器可以提供给所有Docker主机使用

• 共享存储服务器(NAS、SAN、DAS等)

• 如:

– 使用NFS创建共享存储服务器

– 客户端挂载NFS共享,并最终映射到容器中

[root@room9pc01 ~]# mkdir /nfs

[root@room9pc01 ~]# vim /etc/exports

/nfs 192.168.1.0/24(rw)

[root@room9pc01 ~]# systemctl restart nfs-server

[root@docker01 ooxx]# mkdir /public

[root@docker01 ooxx]# mount 192.168.1.254:/nfs /public

[root@docker01 ooxx]# docker run -d -v /public:/var/www/html -p 80:80 myos:http

[root@room9pc01 ~]# echo haha > /nfs/index.html

[root@room9pc01 ~]# curl http://192.168.1.11

haha

# Docker网络架构

Linux网桥

创建虚拟网卡

• 真实网卡配置文件

– cat /etc/sysconfig/network-scripts/ifcfg-eth0

• 虚拟网卡配置文件

– cat /etc/sysconfig/network-scripts/ifcfg-eth0:0

[root@jacob ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0:0

TYPE=Ethernet

BOOTPROTO=static

... ...

NAME=eth0:0

DEVICE=eth0:0

ONBOOT=yes

IPADDR=192.168.4.15

创建虚拟网桥

[root@jacob ~]# cat /etc/sysconfig/network-scripts/ifcfg-br0

TYPE=Bridge

BOOTPROTO=static

... ...

NAME=br0

DEVICE=br0

ONBOOT=yes

IPADDR=192.168.4.15

[root@jacob ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0

TYPE=Ethernet

BOOTPROTO=static

... ...

NAME=eth0

DEVICE=eth0

BRIDGE=br0

ONBOOT=yes

IPADDR=192.168.4.15

[root@jacob ~]# ~]# brctl show

Docker网络拓扑

容器之间能够联通的原理

多个容器之间连接的是同一个网桥docker0

[root@docker01 ~]# yum -y install bridge-utils

[root@docker01 ~]# brctl show

bridge name bridge id STP enabled interfaces

docker0 8000.0242b74d5c1a no veth6df0072

veth713ad00

在一个容器中去ping另一个容器

ping 172.17.0.2可以ping通

可以看到两个容器都是连接在docker0的网桥上，使用如下命令，断开一个接口

[root@docker01 ~]# brctl delif docker0 veth6df0072

在一个容器中去ping另一个容器

ping 172.17.0.2无法ping通，其实中间的虚拟交换机就是172.17.0.1的这个网桥

[root@docker01 ~]# brctl addif docker0 veth6df0072

再去ping，就可以ping通

这个原理就是相当于在物理机上创建用一个vbr的网桥，创建虚拟机时使用这个vbr的设备，多个虚拟机之间

就可以互相ping通，一旦断开与vbr这个虚拟交换机的连接，互相就无法连通

## 查看Docker默认网络模型

• 查看默认Docker创建的网络模型

[root@docker01 ~]# docker network list //查看虚拟交换机

NETWORK ID NAME DRIVER SCOPE

08cd328a0d49 bridge bridge local 桥接模型

9f51c2b04ec9 host host local 主机模型

55b968117c19 none null local 无网络

[root@docker01 ~]# ip a s docker0

[root@docker01 ~]# brctl show docker0 //启动容器会绑定该网桥

• 新建Docker网络模型

[root@jacob ~]# docker network create --subnet=172.30.0.0/16 --driver bridge test01

[root@jacob~]# docker network list

[root@jacob~]# ip a s

[root@jacob~]# docker network inspect test01 查看网桥的详细信息

[root@jacob ~]# docker network rm test01 删除网桥

命令网桥的名字，而不是自动创建的名字需要加一个参数，在 docker network inspect bridge 可以看到

"com.docker.network.bridge.name": "docker0",

创建时，加上这个选项就可以命名

[root@docker01 ~]# docker network create --subnet=172.11.0.0/16 --driver bridge -o com.docker.network.bridge.name=test02 test02

[root@docker01 ~]# ifconfig

就可以看到有个名字为test02的网桥

使用新创建的网桥建立容器

[root@docker01 ~]# docker run -it --network=test02 myos

[root@66e60d2d9c9c /]# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 172.11.0.2 netmask 255.255.0.0 broadcast 0.0.0.0

[root@66e60d2d9c9c /]# ping 172.17.0.1 这里也可以ping通docker0的ip，因为默认开启了路由功能

客户端访问容器内的资源

• 默认容器通过SNAT可以访问外网

• 但外部网络的主机不可以访问容器内的资源

• 端口映射

– 使用端口映射可以实现外部网络访问容器内的资源

物理端口和docker端口绑定，实现的原理是使用netfilter框架

就是通过真机的iptables规则

使用iptables-save

可以看到有两条规则

-A POSTROUTING -s 172.17.0.4/32 -d 172.17.0.4/32 -p tcp -m tcp --dport 80 -j MASQUERADE

-A DOCKER ! -i docker0 -p tcp -m tcp --dport 8080 -j DNAT --to-destination 172.17.0.4:80

第二条就是把访问主机8080端口的转发到容器0.4的80端口

类似于firewall的端口转发

使用命令

iptables -t nat -A DOCKER ! -i docker0 -p tcp -m tcp --dport 9080 -j DNAT --to-destination 172.17.0.3:80

就可以访问192.168.1.11的9080端口访问容器0.3的http服务

（查看一个容器的IP地址可以使用 docker inspect -f '{{.NetworkSettings.IPAddress}}' 容器id ）

问题现象

• 推送镜像到registry,提示错误:

[root@jacob ~]# docker push centos

The push refers to a repository [docker.io/library/centos]

Put https://index.docker.io/v1/repositories/library/centos/: dial tcp:

lookup index.docker.io on 172.40.1.10:53: read udp

172.40.50.118:43696->172.40.1.10:53: i/o timeout

故障分析及排除

• 原因分析

– 问题1:提示The push refers to a repository

[docker.io/library/centos]

• 解决办法

– 问题1:先要修改镜像tag,在可以继续push镜像到registry

docker run -it -m 1000m 镜像

启动容器时限制内存使用最高为1000MB

数据和服务要解耦

kubernetes

迁移性非常强

# 五个Docker监控工具的对比

【编者的话】这篇文章作者是Usman，他是服务器和基础架构工程师，有非常丰富的分布式构建经验。该篇文章主要分析评估了五种Docker监控工具，包括免费的和不免费的：Docker Stats、CAdvisor、Scout、Data Dog以及Sensu。不过作者还是推荐使用Data Dog。另外还有两个工具：Prometheus与Sysdig Cloud会在下一篇做介绍分析，敬请期待。  
  
随着Docker被大规模的部署应用，如何通过可视化的方式了解Docker环境的状态以及健康变得越来越重要。这篇文章我们来回顾下监控容器的常用工具。我会基于以下标准评估这些工具：

易于部署

信息呈现的详细度

整个部署过程中日志的聚集程度

数据报警能力

是否可以监控非Docker的资源

成本

这些评估标准可能并不全面，但是我试图强调的是最常用的工具以及优化此六项评估标准的工具。

## Docker Stats命令

本文中所有使用的命令只在亚马逊EC2上的RancherOS实例中测试过。但是我想它们应该可以在任何的Docker容器中运行。  
  
我将讨论的第一个工具是Docker本身。你可能不知道Docker客户端已经提供了基本的命令行工具来检查容器的资源消耗。想要查看容器统计信息只需运行docker stats [CONTAINER\_NAME]。这样就可以查看每个容器的CPU利用率、内存的使用量以及可用内存总量。请注意，如果你没有限制容器内存，那么该命令将显示您的主机的内存总量。但它并不意味着你的每个容器都能访问那么多的内存。另外，还可以看啊都容器通过网络发送和接收的数据总量。

$ docker stats determined\_shockley determined\_wozniak prickly\_hypatia

CONTAINER             CPU %               MEM USAGE/LIMIT       MEM %               NET I/O

determined\_shockley   0.00%               884 KiB/1.961 GiB     0.04%               648 B/648 B

determined\_wozniak    0.00%               1.723 MiB/1.961 GiB   0.09%               1.266 KiB/648 B

prickly\_hypatia       0.00%               740 KiB/1.961 GiB     0.04%               1.898 KiB/648 B

如果想要看到更为详细的容器属性，还可以通过netcat，使用Docker远程API来查看（见下文）。发送一个HTTP GET请求/containers/[CONTAINER\_NAME]，其中CONTAINER\_NAME是你想要统计的容器名称。你可以从[这里](https://gist.github.com/usmanismail/0c4922ffec4a0220d385)看到一个容器stats请求的完整响应信息。在上述的例子中你会得到缓存、交换空间以及内存的详细信息。如果要了解什么是metrics，那么你就需要精读Docker文档的[Run Metrics部分](https://docs.docker.com/articles/runmetrics/)。

评分：

1. 易于部署程度：※※※※※  
2. 信息详细程度：※※※※※  
3. 集成度：无  
4. 生成警报的能力：无  
5. 监测非Docker的资源的能力：无  
6. 成本：免费

## CAdvisor

我们可以使用docker stats命令和远程API来获取容器的状态信息。但是，如果你想要在图形界面中直接查看这些信息，那你就需要诸如[CAdvisor](https://github.com/google/cadvisor)这类的工具。CAdvisor提供了早docker stats命令所显示的数据的可视化界面。运行以下Docker命令，并在浏览器里访问http://&lt;your-hostname>:8080/可以看到CAdvisor的界面。你将看到CPU的使用率、内存使用率、网络吞吐量以及磁盘空间利用率。然后，你可以通过点击在网页顶部的Docker Containers链接，然后选择某个容器来详细了解它的使用情况。

docker run                                      \

--volume=/:/rootfs:ro                         \

--volume=/var/run:/var/run:rw                 \

--volume=/sys:/sys:ro                         \

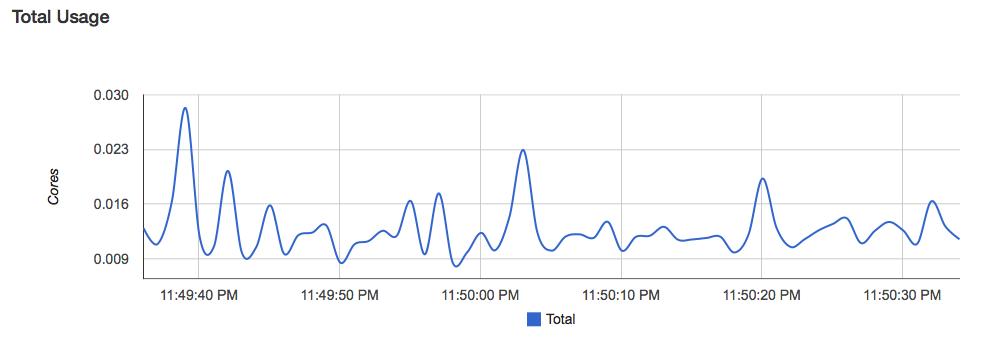
--volume=/var/lib/docker/:/var/lib/docker:ro  \

--publish=8080:8080                           \

--detach=true                                 \

--name=cadvisor                               \

google/cadvisor:latest

[](http://dockone.io/uploads/article/20150525/35c654a1c1f880702fe3bea6d9fd5e90.png)

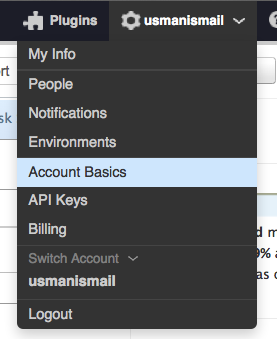
CAdvisor是一个易于设置并且非常有用的工具，我们不用非要SSH到服务器才能查看资源消耗，而且它还给我们生成了图表。此外，当集群需要额外的资源时，压力表（pressure gauges ）提供了快速预览。而且，与本文中的其他的工具不一样的是CAdvisor是免费的，并且还开源。另外，它的资源消耗也比较低。但是，它有它的局限性，它只能监控一个Docker主机。因此，如果你是多节点的话，那就比较麻烦了，你得在所有的主机上都安装一个CAdvisor，者肯定特别不方便。值得注意的是，如果你使用的是Kubernetes，你可以使用[heapster](https://github.com/GoogleCloudPlatform/heapster)来监控多节点集群。另外，在图表中的数据仅仅是时长一分钟的移动窗口，并没有方法来查看长期趋势。如果资源使用率在危险水平，它却没有生成警告的机制。如果在Docker节点的资源消耗方面，你没有任何可视化界面，那么CAdvisor是一个不错的开端来带你步入容器监控，然而如果你打算在你的容器中运行任何关键任务，那你就需要一个更强大的工具或者方法。

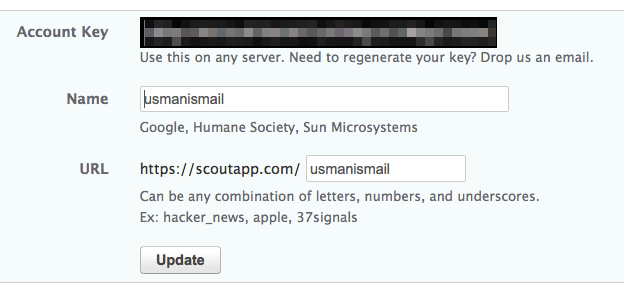
评分：（忽略了heapster，因为它仅支持Kubernetes）

1. 易于部署程度：※※※※※  
2. 信息详细程度：※※  
3. 集成度：※  
4. 生成警报的能力：无  
5. 监测非Docker的资源的能力：无  
6. 成本：免费

## Scout

下一个Docker监控的方法是Scout，它解决了CAdvisor的局限性。 Scout是一个应用监控服务，它能够从很多主机和容器中获得各项监测数据，并将数据呈现在有更长时间尺度的图标中。它也可以基于这些指标生成警报。要获取Scout并运行，第一步，在[scoutapp.com](https://scoutapp.com/)注册一个Scout帐户，免费的试用账号足以用来集成测试。一旦你创建了自己的帐户并登录，点击右上角的帐户名称，然后点击Account Basics来查看你的Account Key，你需要这个Key从Docker服务器来发送指标。

[](http://dockone.io/uploads/article/20150525/7cf7b0501d2fe19e84235bf3059b0d07.png)

[](http://dockone.io/uploads/article/20150525/d1441536f4084971652ac5acf8fcdd00.png)

现在在你的主机上，创建一个名为scouts.yml的文件并将下面的文字复制到该文件中，用上边得到的Key替换到account\_key。您可以对主机指定任何有意义的变量：display\_name、environment与roles等属性。当他们在scout界面上呈现时，这些将用于分离各种指标。我假设有一组网站服务器列表正在运行Docker，它们都将采用如下图所示的变量。

# account\_key is the only required value

account\_key: YOUR\_ACCOUNT\_KEY

hostname: web01-host

display\_name: web01

environment: production

roles: web

现在，你可以使用scout配置文件通过Docker-scout插件来运行scout。

docker run -d  --name scout-agent                              \

-v /proc:/host/proc:ro                                               \

-v /etc/mtab:/host/etc/mtab:ro                                   \

-v /var/run/docker.sock:/host/var/run/docker.sock:ro    \

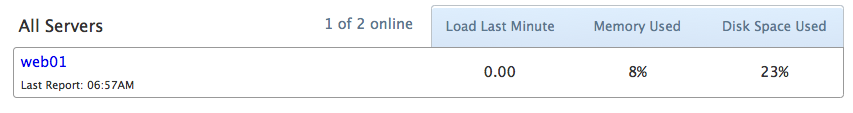
-v `pwd`/scoutd.yml:/etc/scout/scoutd.yml                     \

-v /sys/fs/cgroup/:/host/sys/fs/cgroup/                           \

--net=host --privileged                                                   \

soutapp/docker-scout

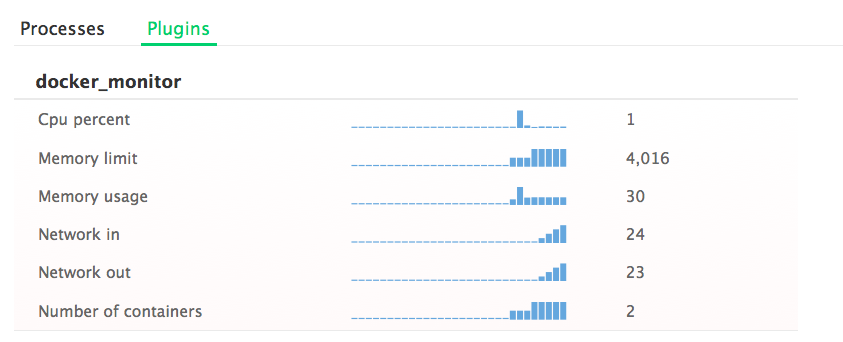
这样你查看Scout网页就能看到一个条目，其中display\_name参数（web01）就是你在scoutd.yml里面指定的。

[](http://dockone.io/uploads/article/20150525/29910859cdcb3d9157953788b9245400.png)

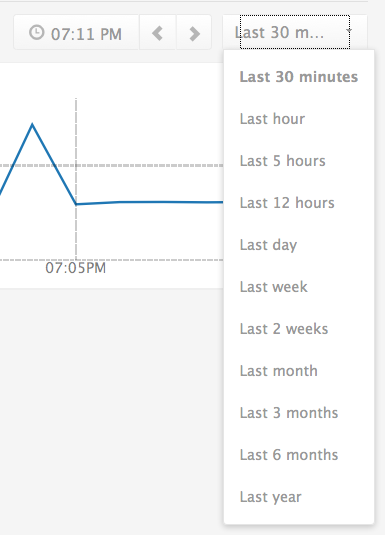
如果你点击它（web01）就会显示主机的详细信息。其中包括任何运行在你主机上的进程计数、cpu使用率以及内存利用率，值得注意的是在docker内部并没有进程的限制。

[](http://dockone.io/uploads/article/20150525/c4bc2c58260871ba2dda04333f957184.png)

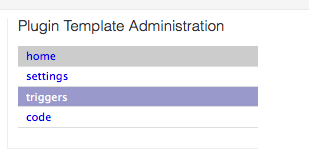
如果要添加Docker监控服务，需要单击Roles选项卡，然后选择所有服务。现在点击+插件模板按钮，接下来的Docker监视器会加载详细信息视图。一旦详细信息呈现出来，选择安装插件来添加到您的主机。接着会给你提供一个已安装插件的名称以及需指定要监视的容器。如果该字段是空的，插件将监控主机上所有的容器。点击完成按钮，一分钟左右你就可以在[Server Name] > Plugins中看到从Docker监控插件中获取的详细信息。该插件为每个主机显示CPU使用率、内存使用率、网络吞吐量以及容器的数量。

[](http://dockone.io/uploads/article/20150525/af2a47b769f00f4bdf9e26379309b5b0.png)

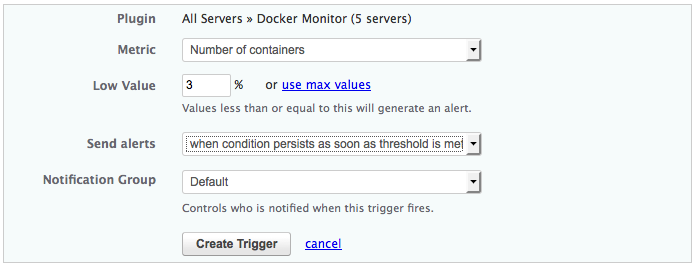
你点击任何一个图表，都可以拉取该指标的详细视图，该视图可以让你看到时间跨度更长的趋势。

[](http://dockone.io/uploads/article/20150525/15c39df8c20b738ba39d75eb29365b15.png)

该视图还支持过滤基于环境和服务器角色的指标。此外，你可以创建“Triggers”或警报，如果指标高于或低于配置的阈值它就给你发送电子邮件。这就允许您设置自动警报来通知您，比如，如果你的一些容器异常关闭以及容器计数低于一定数量。您还可以设置对平均CPU利用率的警报，举例来说，如果你正在运行的容器超过CPU利用率而发热，你会得到一个警告，当然你可以开启更多的主机到你的Docker集群。  
  
要创建触发器，请选择顶部菜单的Roles>All Servers，然后选择plugins部分的Docker monitor。然后在屏幕的右侧的Plugin template Administration菜单里选择triggers。您现在应该看到一个选项“Add a Trigger”，它将应用到整个部署。

[](http://dockone.io/uploads/article/20150525/27ba374316b40e516bb15cd0e2fd865e.png)

下面是一个触发器的例子，如果部署的容器数量低于3就会发出警报。

[](http://dockone.io/uploads/article/20150525/852c58e6989dad25c71d8ea1612f69da.png)

它的创建是为“所有的服务器”，当然你也可以用不同的角色标记你的主机使用服务器上创建的scoutd.yml文件。使用角色。你可以通过使用不同角色来应用触发器到部署的服务器的一个子集上。例如，你可以设置一个当在你的网络的节点的容器数量低于一定数量时的警报。即使是基于角色的触发器我仍然觉得Scout的警报系统可能做的更好。这是因为许多Docker部署具有相同主机上的多种多样的容器。在这种情况下为特定类型的容器设置触发器将是不可能的由于角色被应用到主机上的所有容器。  
  
比起CAdvisor，使用Scout的另一个优点是，它有[大量的插件](https://scoutapp.com/plugin_urls)，除了Docker信息他们可以吸收其他有关你的部署的数据。这使得Scout是你的一站式监控系统，而无需对系统的各种资源来安装各种不同的监控系统。  
  
Scout的一个缺点是，它不显示有关每个主机上像CAdvisor的单独容器的详细信息。这是个问题，如果你在同一台服务器上运行大量的容器。例如，如果你想有一个触发器来提醒您的Web容器的警报，但不是Jenkins容器，这时Scout就无法支持该情况。尽管有这个缺点，Scout还是一个相当有用的工具来监控你的Docker部署。当然这要付出一些代价，每个监控的主机十美元。如果你要运行一个有多台主机的超大部署，这个代价会是个考虑因素。

评分：

1. 易于部署程度：※※※※  
2. 信息详细程度：※※  
3. 集成度：※※※  
4. 生成警报的能力：※※※  
5. 监测非Docker的资源的能力：支持  
6. 成本：每个主机$10

## Data Dog

从Scout移步到另一个监控服务——DataDog，它既解决几个Scout的缺点又解除了CAdvisor的局限性。要使用DataDog，先在[https://www.datadoghq.com/](https://www.datadoghq.com/" \t "/root/文档\\x/_blank)注册一个DataDog账户。一旦你登录到您的帐户，您将看到支持集成的每种类型的指令列表。从列表中选择Docker，你会得到一个Docker run命令（如下），将其复制到你的主机。该命令需要你的预先设置的API密钥，然后你可以运行该命令。大约45秒钟后您的代理将开始向DataDog系统报告。

docker run -d --privileged --name dd-agent             \

-h `hostname`                                      \

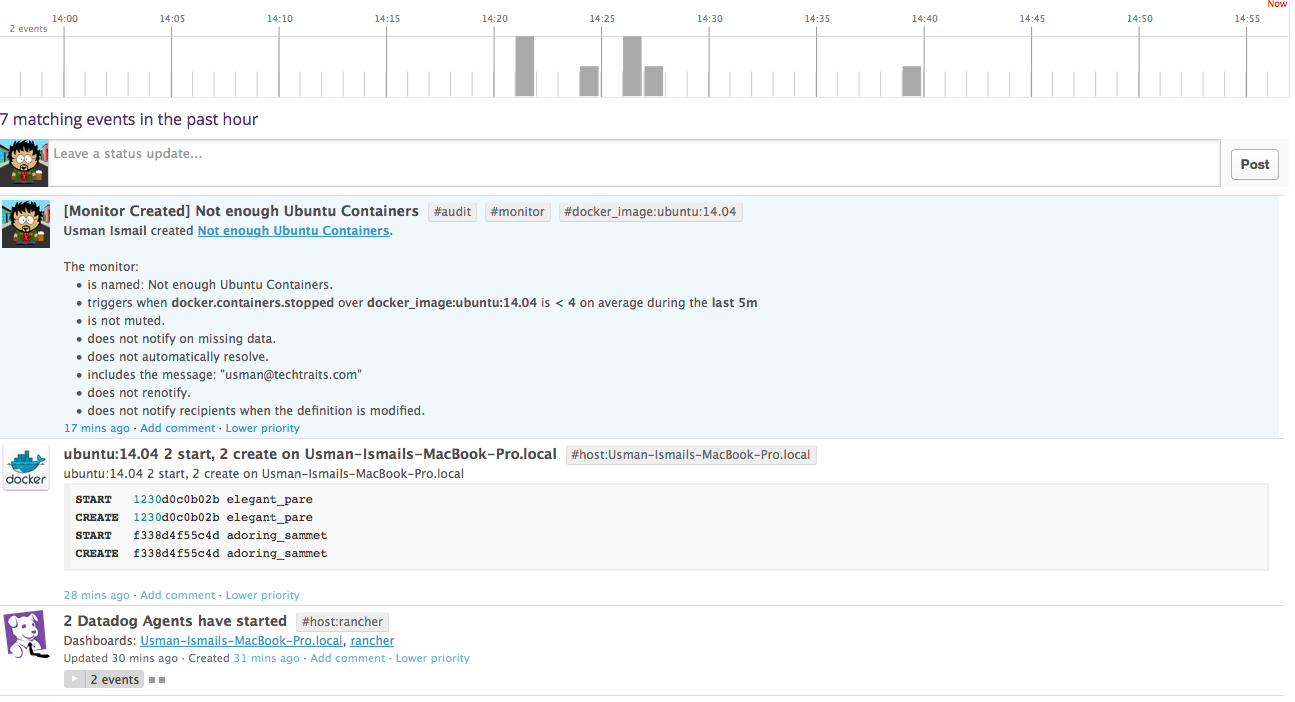
-v /var/run/docker.sock:/var/run/docker.sock       \

-v /proc/mounts:/host/proc/mounts:ro               \

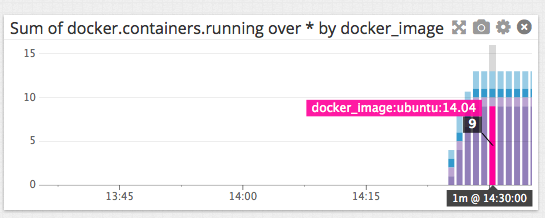
-v /sys/fs/cgroup/:/host/sys/fs/cgroup:ro          \

-e API\_KEY=YOUR\_API\_KEY datadog/docker-dd-agent    \

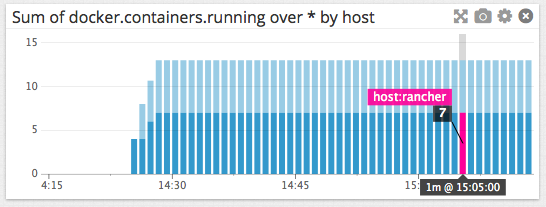
此时，容器提示你可以在DataDog Web的Events tab上处理和查看有关集群的所有动态。所有容器的启动和终止都是事件流的一部分。

[](http://dockone.io/uploads/article/20150525/a28a4b2cde0331788067f2ea474213f1.png)

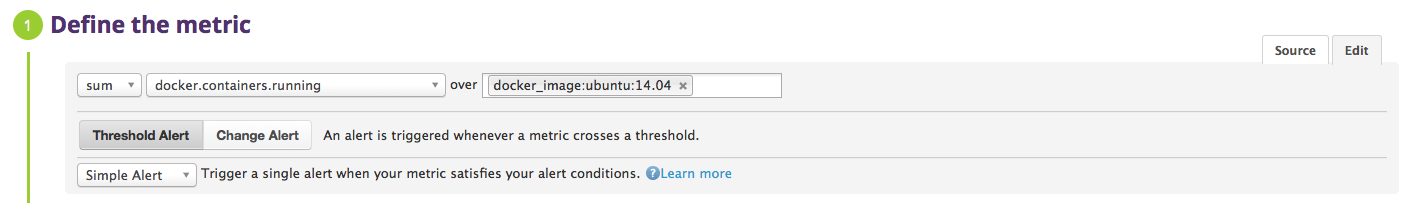
您也可以点击Dashboards标签并点击创建仪表板以合计您整个群集的指标。 Datadog收集在系统中运行的所有容器的CPU使用率、内存以及I/O的指标。此外，也可以获得容器运行和停止次数以及Docker的镜像数量。Dashboard视图可以创建任何数据的图标，或者设置整个部署、主机群、容器镜像指标的图表。例如下图显示了运行容器的数量并加以镜像类型分类，此刻在我的集群运行了9个Ubuntu:14.04的容器。

[](http://dockone.io/uploads/article/20150525/477efd7d9a94fd754608afd2bf8ab7bd.png)

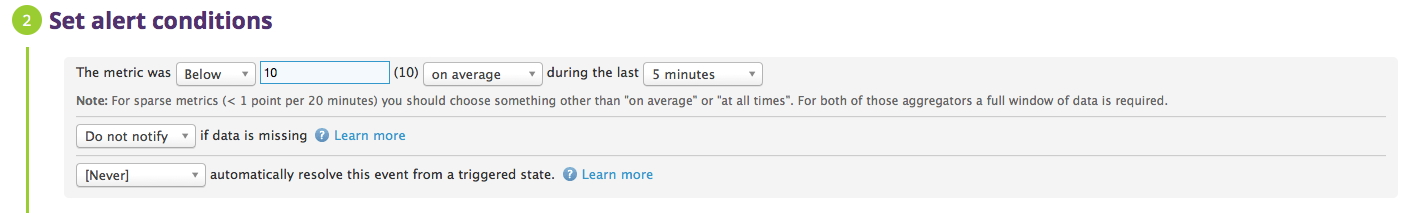
您还可以通过主机分类同样的数据，如下图所示，7个容器在我的Rancher主机上运行，其余的在我的本地的笔记本电脑。

[](http://dockone.io/uploads/article/20150525/6c8d6654c3b280ec3d1dfffc2a175a08.png)

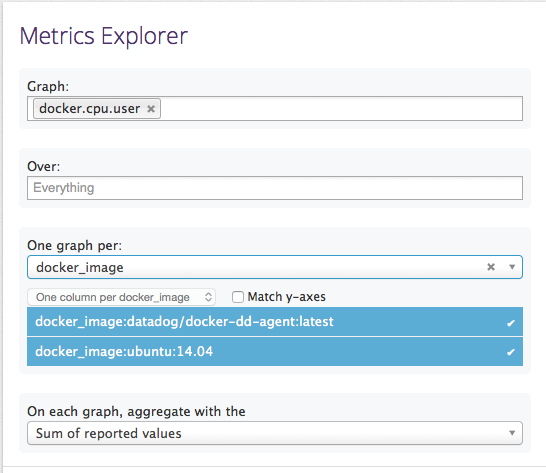
DataDog还支持一种称为Monitors的警报功能。DataDog的一个monitor相当于Scout的一个触发器，并允许您定义各种指标的阈值。 比起Scout，DataDog的警报系统相当灵活与详细。下面的例子说明如何指定您监视的Ubuntu容器的终止，因此你会监视从Ubuntu:14.04的Docker镜象所创建容器的docker.containers.running信息。

[](http://dockone.io/uploads/article/20150525/d44927f0557fc50f15deeb52d1113be3.png)

然后，特定的警报情况是，如果在我们的部署中最后5分钟有（平均）少于十个Ubuntu容器，你就会被警报。尽管这里没有显示，你会被要求填写发送出去时的指定消息在这个警报被触发后，而且还有受到此警报的目标。在当前的例子中，我用一个简单的绝对阈值。您也可以指定一个基于增量的警报，比如是在最后五分钟里停止的容器的平均计数是四的警报。

[](http://dockone.io/uploads/article/20150525/7b9014b99e95a4c6d7002c02c370acaf.png)

最后，使用Metrics Explorer选项卡可以临时聚集你的指标来帮助调试问题或者提取具体的数据信息。该视图允许您基于对容器镜像或主机绘制任何指标的图表。您可以将输出的数据组合成一个单一的图形或者通过镜像或主机的分组来生成一组图形。

[](http://dockone.io/uploads/article/20150525/56fdde3a2037b9a4c0b98bdef025e032.png)

DataDog相比scout在某些功能上做了显著地改善，方便使用以及用户友好的设计。然而这一级别伴随着额外的成本，因为每个DataDog agent价格为$15。

评分：

1. 易于部署程度：※※※※※  
2. 信息详细程度：※※※※※  
3. 集成度：※※※※※  
4. 生成警报的能力：支持  
5. 监测非Docker的资源的能力：※※※※※  
6. 成本：每个主机$15

## Sensu Monitoring Framework

Scout和Datadog提供集中监控和报警系统，然而他们都是被托管的服务，大规模部署的话成本会很突出。如果你需要一个自托管、集中指标的服务，你可以考虑[sensu open source monitoring framework](https://sensuapp.org/)。要运行Sensu服务器可以使用[hiroakis/docker-sensu-server](https://registry.hub.docker.com/u/hiroakis/docker-sensu-server/)容器。这个容器会安装sensu-server、uchiwa Web界面、Redis、rabbitmq-server以及sensu-api。不幸的是sensu不支持Docker。但是，使用插件系统，您可以配置支持容器指标以及状态检查。  
  
在开启sensu服务容器之前，你必须定义一个可以加载到服务器中检查。创建一个名为check-docker.json的文件并添加以下内容到此文件。这个文件告诉Sensu服务器在所有有docker标签的客户端上每十秒运行一个名为load-docker-metrics.sh的脚本。

{

"checks": {

"load\_docker\_metrics": {

  "type": "metric",

  "command": "load-docker-metrics.sh",

  "subscribers": [

    "docker"

  ],

  "interval": 10

}

}

}

现在，您可以使用下面的命令通过我们的检查配置文件来运行Sensu服务器Docker容器。一旦你运行该命令，你就可以在浏览器输入http://YOUR\_SERVER\_IP:3000来访问uchiwa界面。

docker run -d --name sensu-server                                           \

-p 3000:3000                                                            \

-p 4567:4567                                                            \

-p 5671:5671                                                            \

-p 15672:15672                                                          \

-v $PWD/check-docker.json:/etc/sensu/conf.d/check-docker.json           \

hiroakis/docker-sensu-server

这样Sensu服务器就开启了，你就可以对每个运行有我们的Docker容器的主机上开启sensu客户端。你告诉容器将有一个名为load-docker-metrics.sh的脚本，所以让我们创建脚本，并将其放到我们的客户端容器内。创建该文件并添加如下所示的文本，将HOST\_NAME替换为您的主机的逻辑名称。下面的脚本是为运行容器、所有容器以及镜像而使用Docker远程API来拉取元数据。然后它打印出来sensu的键值标示的值。该sensu服务器将读取标准输出并收集这些指标。这个例子只拉取这三个值，但根据需要，你可以使脚本尽可能详细。请注意，你也可以添加多个检查脚本，如thos，只要早前在服务配置文件中你引用过它们。你也可以定义你想要检查运行Docker容器数量降至三个以下的失败。你还可以使检查通过从检查脚本返回一个非零值失败。

#!/bin/bash

set -e

# Count all running containers

running\_containers=$(echo -e "GET /containers/json HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5                                                           \

| python -m json.tool                                                  \

| grep \"Id\"                                                          \

| wc -l)

# Count all containers

total\_containers=$(echo -e "GET /containers/json?all=1 HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5 \

| python -m json.tool \

| grep \"Id\" \

| wc -l)

# Count all images

total\_images=$(echo -e "GET /images/json HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5 \

| python -m json.tool \

| grep \"Id\" \

| wc -l)

echo "docker.HOST\_NAME.running\_containers ${running\_containers}"

echo "docker.HOST\_NAME.total\_containers ${total\_containers}"

echo "docker.HOST\_NAME.total\_images ${total\_images}"

if [ ${running\_containers} -lt 3 ]; then

exit 1;

fi

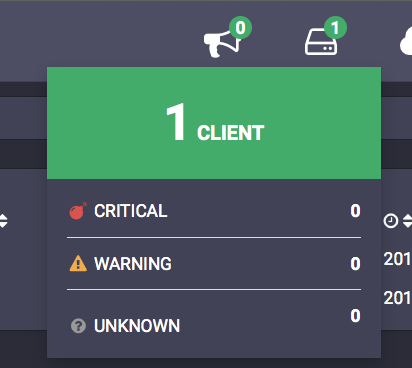
现在你已经定义了Docker载入指标检查，那就需要使用[usman/sensu-client](https://registry.hub.docker.com/u/usman/sensu-client/)容器来启动sensu客户端。您可以使用如下所示的命令启动sensu客户端。需要注意的是，容器必须以privileged来运行以便能够访问Unix sockets，它必须有Docker socket挂载以及你上面定义的load-docker-metrics.sh脚本。确保load-docker-metrics.sh脚本在你的主机的权限标记为可执行。容器也需要将SENSU\_SERVER\_IP、RABIT\_MQ\_USER、RABIT\_MQ\_PASSWORD、CLIENT\_NAME以及CLIENT\_IP作为参数，请指定这些参数到您设置的值。其中RABIT\_MQ\_USER与RABIT\_MQ\_PASSWORD默认值是sensu和password。

docker run -d --name sensu-client --privileged                                \

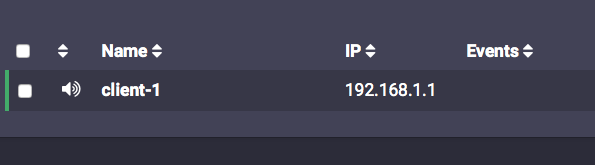
-v $PWD/load-docker-metrics.sh:/etc/sensu/plugins/load-docker-metrics.sh  \

-v /var/run/docker.sock:/var/run/docker.sock                              \

usman/sensu-client SENSU\_SERVER\_IP RABIT\_MQ\_USER RABIT\_MQ\_PASSWORD CLIENT\_NAME CLIENT\_IP

[](http://dockone.io/uploads/article/20150525/9a65fee9fd150d8d8935eaff27b4abe7.png)

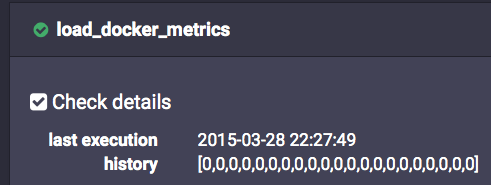
运行完此命令，一会儿你应该看到客户端计数增加1在uchiwa界面。如果您点击客户端图标，你应该看到，包括你刚才添加的客户端的客户端名单。我的客户端1是client-1以及指定的主机IP为192.168.1.1。

[](http://dockone.io/uploads/article/20150525/c2fbd10c19cef81888b7a5b043899fbd.png)

如果你点击客户端名称你应该得到检查的进一步细节。你可以看到load\_docker\_metrics检查在3月28日的10:22运行过。

[](http://dockone.io/uploads/article/20150525/f3483381af3a878a18e2ab1bbf5935f3.png)

如果你点击检查名称就可以看到检查运行的进一步细节。零表明没有错误，如果脚本失败（例如，如果您的Docker守护进程死掉），你会看到一个错误代码（非零）值。虽然在目前的文章中没有涉及这个，你也还可以使用[Handlers](http://sensuapp.org/docs/0.11/adding_a_handler)在sensu设置这些检查失败处理程序来提醒您。此外，uchiwa只显示检查的值，而不是收集的指标。需要注意的是sensu不存储所收集的指标，它们必须被转发到一个时间序列的数据库如InfluxDB或Graphite。这也是通过Handlers做到的。如何配置指标转发到graphite[可以参考这里](http://www.joemiller.me/2012/02/02/sensu-and-graphite/)。

[](http://dockone.io/uploads/article/20150525/5e860c6b296f7911ea9efa8d3a1b0e4d.png)

Sensu支持我们所有的评价标准，你可以对我们Docker容器和主机收集尽可能多的细节。此外，你能够聚合所有主机的值到一个地方，并对这些检查发出警报。这些警报并没有DataDog或Sc​​out的先进，因为你仅能够提​​醒单独的主机上检查失败。然而，Sensu的大缺点是部署的难度。虽然我已经使用Docker容器自动部署许多步骤，Sensu仍然是一个需要我们安装，启动和分开维护Redis、RabitMQ、Sensu API、uchiwa与Sensu Core的复杂系统。此外，你将需要更多的工具，如Graphite来呈现指标值以及生产部署需要定制容器，今天我已经使用了一个容器为了密码的安全以及自定义的SSL证书。除了您重启容器后才能添加更多的检查，你将不得不重新启动Sensu服务，因为这是它开始收集新的标准的唯一途径。由于这些原因，我对Sensu的在易于部署的评价相当的低。

评分：

1. 易于部署程度：※  
2. 信息详细程度：※※※※  
3. 集成度：※※※※  
4. 生成警报的能力：支持但有限  
5. 监测非Docker的资源的能力：※※※※※  
6. 成本：免费

总结

今天的文章涵盖了多种选项用于监控Docker容器，从免费的选择， 如Docker stats、CAdvisor或Sensu，到有偿服务，如Scout和DataDog。我的研究到目前为止DataDog似乎是用于监控Docker部署的最好的系统。只需几秒的安装以及单行命令，所有主机都在同一个地方报告指标，在UI方面，历史趋势是显而易见的，并且Datadog支持更深层次的指标以及报警。然而，$15一个主机系统对于大型部署是昂贵的。对于较大规模，自托管部署，Sensu是能够满足大多数的要求，不过在建立和管理一个Sensu集群的复杂性可能让人望而却步。很显然，有很多其他的自托管的选项，如Nagios或Icinga，他们都类似Sensu。  
  
但愿今天这篇文章会给你一些想法对于监视容器的选择。我会继续调查其他选项，包括使用CollectD、Graphite或InfluxDB与Grafana的更精简的自我管理的容器监控系统。敬请关注更多的细节。  
  
其他信息：发布本文后，我有一些建议去评估Prometheus和Sysdig云，两个非常好的监控Docker的选择。我在这两个服务上花了一些时间，并添加了第二部分到这个文章中。你可以在[这里](http://rancher.com/docker-monitoring-continued-prometheus-and-sysdig/)（译注：过后会翻译这篇）找到它。  
  
想要了解更多关于监控和管理Docker，请参加我们的下一个Rancher在线meetup。  
  
原文：[Comparing Five Monitoring Options for Docker](http://rancher.com/comparing-monitoring-options-for-docker-deployments/) （翻译：[田浩浩](http://dockone.io/people/llitfkitfk) 校对：魏小红）