docker 监控命令:

docker ps -a -q | wc -l   监控容器数量

docker ps  -q | wc -l     正在运行的容器的数量

docker ps -a  | grep -v 'Up '  | grep -v 'CONTAINER' | wc -l  非运行状态的容器的数量

Container 退出

ctrl-c退出container,那么container也会stop

ctrl-q可以退出到宿主机，而保持container仍然在运行

Docker 常用命令:

* docker version 查看docker的版本号，包括客户端、服务端、依赖的Go等
* docker info 查看系统(docker)层面信息，包括管理的images, containers数等
* docker search <image> 在docker index中搜索image
* docker pull <image> 从docker registry server 中下拉image
* docker push <image|repository> 推送一个image或repository到registry
* docker push <image|repository>:TAG 同上，指定tag
* docker inspect <image|container> 查看image或container的底层信息
* docker images **TODO** filter out the intermediate image layers (intermediate image layers 是什么)
* docker images -a 列出所有的images
* docker ps 默认显示正在运行中的container
* docker ps -l 显示最后一次创建的container，包括未运行的
* docker ps -a 显示所有的container，包括未运行的
* docker logs <container> 查看container的日志，也就是执行命令的一些输出
* docker rm <container...> 删除一个或多个container
* docker rm `docker ps -a -q` 删除所有的container
* docker rm `docker ps -a | grep Exited | awk '{print $1}'`

docker rmi -f `docker images | grep '<none>' | awk '{print $3}'`

* docker ps -a -q | xargs docker rm 同上, 删除所有的container
* docker rmi <image...> 删除一个或多个image
* docker start/stop/restart <container> 开启/停止/重启container
* docker start -i <container> 启动一个container并进入交互模式
* docker attach <container> attach一个运行中的container
* docker run <image> <command> 使用image创建container并执行相应命令，然后停止
* docker run -i -t <image> /bin/bash 使用image创建container并进入交互模式, login shell是/bin/bash
* docker run -i -t -p <host\_port:contain\_port> 将container的端口映射到宿主机的端口
* docker commit <container> [repo:tag] 将一个container固化为一个新的image，后面的repo:tag可选
* docker build <path> 寻找path路径下名为的Dockerfile的配置文件，使用此配置生成新的image
* docker build -t repo[:tag] 同上，可以指定repo和可选的tag
* docker build - < <dockerfile> 使用指定的dockerfile配置文件，docker以stdin方式获取内容，使用此配置生成新的image
* docker port <container> <container port> 查看本地哪个端口映射到container的指定端口，其实用docker ps 也可以看到

启动参数:

docker  --api-enable-cors=false 是否开启远程API接口

一般应用于DOCKER\_OPTS

DOCKER\_OPTS='--api-enable-cors=false'

docker -d 是绑定网桥, 默认是docker0,

docker -b=br0 是含义, 一般应用于DOCKER\_NETWORK\_OPTIONS

DOCKER\_NETWORK\_OPTIONS='--bip=192.168.227.1/24'

ifconfig docker0 192.168.227.1

使用 none 可以停用容器里的网络。

docker -D, --debug=false 是否debug类型

docker  -d 是守护进程运行

docker  --dns=[]  强制容器使用dns服务器

docker -d --dns 8.8.8.8

docker  --mac-address 强制修改mac地址

docker  --net="bridge" 网络环境(默认选项)

none 关闭container内的网络连接

Docker 默认是将container设置为bridge模式。此时在host上面讲存在一个docker0的网络接口，同时会针对container创建一对veth接口。其中一个veth接口是在host充当网卡桥接作用，另外一个veth接口存在于container的命名空间中，并且指向container的loopback。Docker会自动给这个container分配一个IP，并且将container内的数据通过桥接转发到外部。

Host 是当网络模式设置为host时，这个container将完全共享host的网络堆栈。host所有的网络接口将完全对container开放。container的主机名也会存在于host的hostname中。这时，container所有对外暴露的port和对其它container的link，将完全失效。

Container是当网络模式设置为Container时，这个container将完全复用另外一个container的网络堆栈。同时使用时这个container的名称必须要符合下面的格式：--net container:<name|id>.

比如当前有一个绑定了本地地址localhost的redis container。如果另外一个container需要复用这个网络堆栈，则需要如下操作：

docker run -d --name redis example/redis --bind 127.0.0.1  
docker run --rm -ti --net container:redis example/redis-cli -h 127.0.0.1

docker  -g, --graph="/var/lib/docker"   根目录

docker -G, --group="docker" 在后台运行模式下，赋予指定的Group到相应的unix socket上。注意，当此参数 --group 赋予空字符串时，将去除组信息。

docker  --iptables=true  启动Docker容器自定义的iptable规则

docker  -p, --pidfile="/var/run/docker.pid"  后台进程PID文件路径

计算机生成了可选文字:
container 
container 
read write 
read only 
Lookup Image 
get image 
'rom Ison rile 
ano layer size 
tile grapn 
directory 
contalner_Mount 
/oufsmnt/.id 
be mounted to 
"Outs/ diff/ , i d 
louts/ diff / ;id-init 
mergeAnOVer13'C 
onlg 
compare 
image cc•nng 
and container 
contig. men 
merge mem 
read write 
read only 
setupMc•untsForC 
ontajner 
mount 
mes to 
container 
rcots, e.g. 'etc/ 
Misn 
/aufs/mnt/;id-init 
unlink some files/direc 
and create new one. e.g. / 
proc , /etc/resolv.cont. These 
files will be reset during the 
/oufs/mnt/;id-init 
be mounted to 
/ auf5/diff/id-init 
target •mage 
Oase image 
execonver start 
use Yau's/ 
mnV•JO as 
init time of container. 
finish 
http : // blog. csdn- net/mengzechao 

docker 目录:

containers :  是存放容器的描述文件, 后续重启容器, 依据这些配置文件

日志文件,只显示主进程

vfs : 虚拟挂载

volumes: 是配置文件

tmp : bluid 是产生的垃圾文件

graph : 镜像配置信息

init : docker初始化的二进制文件

容器参数:

docker login 登陆用户,密码

docker info 详细信息

docker 查询

docker search ubunt

docker pull 下载镜像

vim /etc/sysconfig/docker-storage

添加:

DOCKER\_STORAGE\_OPTIONS=--insecure-registry=192.168.144.135:5000

docker pull 192.168.144.135:5000/consol/tomcat-7.0

pull到docker hub

docker commit:

docker commit -m "Added hadoop hive client" -a "wangzhehan" 42062e85e250 test-client/centos:t

docker  run

docker run -u root --env-file=tt -it  test-client/centos:t hadoop fs -ls hdfs://192.168.147.111:8021/

启动一个容器，在其中运行指定命令。

-a stdin 指定标准输入输出内容类型，可选 STDIN/STDOUT / STDERR 三项；

-d 后台运行容器，并返回容器ID；-d命令是设置detach为true，根据官方的文档，意思是让这个命令在后台运行，但并不是一直运行

-i 以交互模式运行容器，通常与 -t 同时使用；

-t 为容器重新分配一个伪输入终端，通常与 -i 同时使用；

--name="nginx-lb" 为容器指定一个名称；

--dns 8.8.8.8 指定容器使用的DNS服务器，默认和宿主一致；

--dns-search example.com 指定容器DNS搜索域名，默认和宿主一致；

-h "mars" 指定容器的hostname；

-e username="ritchie" 设置环境变量；

--env-file=[] 从指定文件读入环境变量；

--cpuset="0-2" or --cpuset="0,1,2" 绑定容器到指定CPU运行；

配置 lxc, --lxc-conf="lxc.cgroup.cpuset.cpus = 0,1"

mpstat -P ALL 5 10

--cpu-shares=0  是相对权重， 设置为一个正整数，代表所分配的相对CPU资源比。

配置 lxc, --lxc-conf="lxc.cgroup.cpu.shares = 1234"

-m 内存限制

--privileged=true 是特权用户

-P 自动选择对外暴漏端口 49000 -49900

--volume-from 绑定盘符的来源, 与来源一模一样

--rm 临时回话,并删除

docker save

docker save jdclient/centos  > ~/hiveclient.tar

docker load

docker load  < hiveclient.tar

docker rm删除容器

docker ps -a -q |xargs docker rm

docker -v启动容器,挂载硬盘

docker run -t -i -v /data2:/data2 -v /data0/:/data0/ -v /data3:/data3 centos:7

docker export

docker export 7691a814370e > ubuntu.tar

docker import

cat shortcalculate4.tar | docker import - shortcalculate/centos6.5:5

容器logs

docker logs -f  -t --tail="100"  42062e85e250

docker --link动态获取另一个容器的环境变量(link)

默认是同网段的ip, 使用link仅仅注册hosts 等, 即便不是用link, 也可以直接使用

1. 使用link后台是初始化环境(env),拼接为tcp://172.17.0.32:5432, 同时利用三个变量:ADDR, PORT, PROTO
2. 添加host文件, 将目标地址插入host文件中

docker run -d --name db training/postgres

docker run -d -P --name web --link db:db training/webapp python app.py

// db 是连接的容器名称, 第二个db是别名

查看iptables :

iptables-save

计算机生成了可选文字:
docker run --rm=true --link=mysqlserver:myserver -it java /bin/bash 
iptables-save 
—A DOCKER —s *72.17.0.4/32 —d 172.17.0.1/32 —i dockerO —o dockerO —p tcp —m tcp 
—A DOCKER —s 172.17.0.1/32 —d 172.17.0.4/32 —i dockerO —o dockerO —p tcp —m tcp 
--dport 3306 -i ACCEPT 
--sport 3306 -j ACCEPT 

这里我们可以看

定制显示docker列:

docker inspect --format='{{.State.Pid}} {{.Name}}' `docker ps -a -q`

获取docker执行进程

docker top `docker ps -q`

docker inspect 查看镜像参数

docker inspect  test-client/centos:t

详细信息:

[{

    "Architecture": "amd64",

    "Author": "wangzhehan",

    "Comment": "Added hadoop hive client",

    "Config": {

        "AttachStderr": false,

        "AttachStdin": false,

        "AttachStdout": false,

        "Cmd": [

            "/bin/bash"

        ],

        "CpuShares": 0,

        "Cpuset": "",

        "Domainname": "",

        "Entrypoint": null,

        "Env": null,

        "ExposedPorts": {

            "22/tcp": {}

        },

        "Hostname": "",

        "Image": "",

        "MacAddress": "",

        "Memory": 0,

        "MemorySwap": 0,

        "NetworkDisabled": false,

        "OnBuild": null,

        "OpenStdin": false,

        "PortSpecs": null,

        "StdinOnce": false,

        "Tty": false,

        "User": "root",

        "Volumes": null,

        "WorkingDir": ""

    },

    "Container": "42062e85e250e46fdc1867da1e5a53bfea16a770515e94d81fd6cfc7ae9ba675",

    "ContainerConfig": {

        "AttachStderr": true,

        "AttachStdin": true,

        "AttachStdout": true,

        "Cmd": [

            "/bin/bash"

        ],

        "CpuShares": 0,

        "Cpuset": "",

        "Domainname": "",

        "Entrypoint": null,

        "Env": null,

        "ExposedPorts": {

            "22/tcp": {}

        },

        "Hostname": "42062e85e250",

        "Image": "jdclient/centos:7",

        "MacAddress": "",

        "Memory": 0,

        "MemorySwap": 0,

        "NetworkDisabled": false,

        "OnBuild": null,

        "OpenStdin": true,

        "PortSpecs": null,

        "StdinOnce": true,

        "Tty": true,

        "User": "root",

        "Volumes": null,

        "WorkingDir": ""

    },

    "Created": "2015-06-18T02:50:53.120470388Z",

    "DockerVersion": "1.5.0",

    "Id": "f0f749cfd553c13842b1c0bf4aadec7ce4b769142c58548cce484e24a6287c95",

    "Os": "linux",

    "Parent": "9b6f4bdec10036c3966c1f2e81bf3d3b3570a277a40d860d669557476d1211ec",

    "Size": 16191,

    "VirtualSize": 1109989744

}

]

重启镜像 start

Docker start 容器名

连接容器 attach

docker attach 容器名/ID

创建守护进程

docker run --name 容器名 -d test-client/centos:t /bin/sh -c "while true;do echo test;sleep 1; done"

Docker log -ft  test-client/centos:t

查看守护进程命令

docker top 42062e85e250

后台进程: docker exec -d 42062e85e250 cat /etc/profile

交互运行: docker exec -t -i 42062e85e250 /bin/bash

Kiil 进程: docker kill -s

停止守护进程

docker stop 42062e85e250

自动重启 restart: always / onfailure(可以设置次数)

docker run -restart=always -name wangzhehan -d  test-client/centos:t /bin/sh -c "while true;do echo test;sleep 1; done"

删除容器

docker rm 42062e85e250

images

docker images --no-trunc

镜像与镜像之间的联系(bridge):

计算机生成了可选文字:
Container 'app' 
Tomcat 
ort 8080 
Container 'db' 
MySQL 
ort 3306 
Bridge docker0 
Host 

默认仅有主机可以访问, 同时会创建私有网络(172.XXX)的IP, 需要绑定映射端口~~

使用--net host 则可以使用主机,不需要映射

计算机生成了可选文字:
—S 'ocker -dt --ret bst pythm -n 

进入容器

Sshd

Nsenter  进入进程

计算机生成了可选文字:
• docker inspect --format 
.State.Pid •ontainer-ib 
• nsenter --target $PID --mount --uts --ipc --net --pid 

exec 是已经运行容器中执行

docker exec 容器ID 命令

私有仓库装载镜像:

1. 启动自建仓库

 docker run -d -p 5000:5000 -v /data0/registry:/tmp/registry docker.io/registry

docker run -p 8090:8080 -e REG1=http:/172.22.96.66:5000/v1/ 172.16.190.49:5000/atcol/docker-registry-ui:latest

仓库:

<http://172.16.190.49:8080/registry/index>

计算机生成了可选文字:
Registries 
172.16.190.49 
Images 
Search images. 
API 
VI 
5,000 

1. Pull  官网下载公用镜像

docker pull consol/tomcat-7.0

1. Tag

docker tag imageID 192.168.144.135:5000/consol/tomcat-7.0

docker tag 18875faf9372 172.16.190.49:5000/hasedon/centos6.5:latest

docker tag cab7598b00f2 dd/test:1

1. Push

docker push  192.168.144.135:5000/consol/tomcat-7.0

docker push 172.16.190.49:5000/hasedon/centos6.5:latest

1. 查看远程库 images

curl 192.168.144.135:5000/v1/search

1. Search

docker search 192.168.144.135:5000/test-client/centos