**VirtualBox 安装**

virtualBox安装扩展工具提示header版本不对时

修改yum源到光盘

挂载光盘(centos安装光盘)

mount /dev/cdrom /media/cdrom

yum remove kernel-headers -y

yum install kernel-headers-$(uname -r) kernel-devel-$( uname -r) -y

yum install gcc make -y

卸载光盘，从新挂载virtualBox安装目录下的VBoxGuestAdditions.iso

umount /dev/cdrom

mount -t /dev/cdrom /mnt/cdrom

./VBoxLinuxAdditions.run

挂载共享文件夹

mount -t vboxsf 共享文件夹名称 /mnt/share

**安装jdk版本**

1.查看版本

java -version

2.检查jdk安装的包

rpm -qa | grep java

显示

java-1.7.0-openjdk-1.7.0.111-2.6.7.8.el7.x86\_64

python-javapackages-3.4.1-11.el7.noarch

tzdata-java-2016g-2.el7.noarch

javapackages-tools-3.4.1-11.el7.noarch

java-1.8.0-openjdk-1.8.0.102-4.b14.el7.x86\_64

java-1.8.0-openjdk-headless-1.8.0.102-4.b14.el7.x86\_64

java-1.7.0-openjdk-headless-1.7.0.111-2.6.7.8.el7.x86\_64

3.卸载openjdk

[root@localhost software]# rpm -e --nodeps tzdata-java-2016g-2.el7.noarch

[root@localhost software]# rpm -e --nodeps java-1.7.0-openjdk-1.7.0.111-2.6.7.8.el7.x86\_64

[root@localhost software]# rpm -e --nodeps java-1.7.0-openjdk-headless-1.7.0.111-2.6.7.8.el7.x86\_64

[root@localhost software]# rpm -e --nodeps java-1.8.0-openjdk-1.8.0.102-4.b14.el7.x86\_64

[root@localhost software]# rpm -e --nodeps java-1.8.0-openjdk-headless-1.8.0.102-4.b14.el7.x86\_64

或使用 yum remove \*openjdk\*

4.再使用 rpm -qa | grep java 查看卸载情况

[root@localhost software]# rpm -qa | grep java

python-javapackages-3.4.1-11.el7.noarch

javapackages-tools-3.4.1-11.el7.noarch

5.解压jdk

tar -zxvf jdk-8u131-linux-x64.tar.gz -C /usr/lib/jvm

6.设置环境变量

vim /etc/profile

在最前面添加：

export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_131

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

7.执行profile文件

[root@localhost software]# source /etc/profile

**基本概念**

[**Docker**](http://lib.csdn.net/base/docker) 包括三个基本概念

* 镜像（Image）
* 容器（Container）
* 仓库（Repository）

先理解了这三个概念，就理解了 Docker 的整个生命周期。

### ****1、docker安装与启动****

yum install -y epel-release

yum install docker-io # 安装docker

# 配置文件 /etc/sysconfig/docker

检查已经安装的docker

Rpm -qa|grep docker

移除旧的版本

Yum remove ‘上条命令查询的结果’

安装最新版本docker

curl -fsSL https://get.docker.com/ | sudo sh

chkconfig docker on # 加入开机启动

service docker start # 启动docker服务

也可使用systemctl start docker.service启动

docker启动报错

使用 systemctl status docker.service -l 查看日志如下

Error starting daemon: SELinux is not supported with the overlay2 graph driver on this kernel. Either boot into a newer kernel or disable selinux in docker (--selinux-enabled=false)

解决办法:

(1)禁用SELinux ，在文件/etc/sysconfig/selinux 中设置 SELINUX=disabled， 然后重启电脑(setenforce 0 不行)

(2)修改docker配置文件,在/ect/sysconfig/docker 中设置 OPTIONS='--selinux-enabled=false'

# 基本信息查看

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 pull centos 下载

docker images [ centos ] 查看

docker run -i -t centos /bin/bash

### ****2、镜像的获取与容器的使用****

# 搜索镜像

docker search <image> # 在docker index中搜索image

# 下载镜像

docker pull <image> # 从docker registry server 中下拉image

# 查看镜像

docker images： # 列出images

docker images -a # 列出所有的images（包含历史）

docker rmi <image ID>： # 删除一个或多个image

# 使用镜像创建容器

docker run -i -t sauloal/ubuntu14.04

docker run -i -t sauloal/ubuntu14.04 /bin/bash # 创建一个容器，让其中运行 bash 应用，退出后容器关闭

docker run -itd --name centos\_aways --restart=always centos #创建一个名称centos\_aways的容器，自动重启

# --restart参数：always始终重启；on-failure退出状态非0时重启；默认为，no不重启

# 查看容器

docker ps ：列出当前所有正在运行的container

docker ps -l ：列出最近一次启动的container

docker ps -a ：列出所有的container（包含历史，即运行过的container）

docker ps -q ：列出最近一次运行的container ID

# 再次启动容器

docker start/stop/restart <container> #：开启/停止/重启container

docker start [container\_id] #：再次运行某个container （包括历史container）

使用docker镜像nginx:latest以后台模式启动一个容器,并将容器命名为mynginx。

-d : 以后台方式运行

docker run --name mynginx -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,并将容器的80端口映射到主机随机端口。

docker run -P -d nginx:latest

使用镜像nginx:latest以后台模式启动一个容器,将容器的80端口映射到主机的80端口,主机的目录/data映射到容器的/data。

docker run -p 80:80 -v /data:/data -d nginx:latest

使用自己的镜像包如下错误

oci runtime error: container\_linux.go:247: starting container process caused "exec: \"java\": executable file not found in $PATH".

解决方法1.docker中的jdk环境变量不要设置到/etc/profile中，docker启动时不会加载,需要配置在/root/.bash\_profile文件中

2.启动命令中添加/bin/sh，如下

docker run -itd -p 4091:9091 -v /software:/software myimage/client /bin/sh java -jar eurekaclient1-0.0.1-SNAPSHOT.jar

使用镜像nginx:latest以交互模式启动一个容器,在容器内执行/bin/bash命令。

runoob@runoob:~$ docker run -it nginx:latest /bin/bash

root@b8573233d675:/#

下面开启一个 nginx 服务，将本机 8080 端口映射到容器的 80 端口：

|  |  |
| --- | --- |
|  | $ docker run -d -p 8080:80 nginx  23e725098712d061a1382f33d6fe54da23ae37597a62f8debdd3731b5f9cc4b9 |

$ docker ps 可查看容器端口信息

**查看映射端口**

使用 docker port 命令来查看当前映射的端口配置，也可以查看到绑定的地址。命令格式如下：

[?](http://www.jb51.net/article/127630.htm)

|  |  |
| --- | --- |
| 1 | $ docker port CONTAINER [PRIVATE\_PORT[/PROTO]] |

容器有自己的内部网络和 IP 地址，可以使用 docker inspect + 容器ID 获取容器的具体信息。

**DOCKER 给运行中的容器添加映射端口**

**1、获得容器IP**

将container\_name 换成实际环境中的容器名

[?](http://www.jb51.net/article/127630.htm)

|  |  |
| --- | --- |
| 1 | docker inspect `container\_name` | grep IPAddress |

**2、 iptable转发端口**

将容器的8000端口映射到docker主机的8001端口

复制代码代码如下:

iptables -t nat -A  DOCKER -p tcp --dport 8001 -j DNAT --to-destination 172.17.0.19:8000

**方法2**

1.提交一个运行中的容器为镜像

[?](http://www.jb51.net/article/127630.htm)

|  |  |
| --- | --- |
| 1 | docker commit containerid foo/live |

2.运行镜像并添加端口

[?](http://www.jb51.net/article/127630.htm)

|  |  |
| --- | --- |
| 1 | docker run -d -p 8000:80 foo/live /bin/bash |

#进入正在运行的docker容器

docker exec -it [container\_id] /bin/bash

docker run -i -t -p <host\_port:contain\_port> #：映射 HOST 端口到容器，方便外部访问容器内服务，host\_port 可以省略，省略表示把 container\_port 映射到一个动态端口。

# 删除容器

docker rm <container...> #：删除一个或多个container

docker rm `docker ps -a -q` #：删除所有的container

docker ps -a -q | xargs docker rm #：同上, 删除所有的container

**docker run 和 docker create 参数基本一样，run是创建容器并后台启动，create是只创建容器。   
docker run 相当于docker create 和 docker start**

run创建容器：docker run -itd

create创建： docker create -it

-t, --tty Allocate a pseudo-TTY

-i, --interactive Keep STDIN open even if not attached

-d, --detach Run container in background and print container ID #run的参数

**容器资源限制参数**

-m 1024m --memory-swap=1024m # 限制内存最大使用（bug：超过后进程被杀死）

--cpuset-cpus="0,1" # 限制容器使用CPU

**docker容器随系统自启参数**

docker run --restart=always redis

* **no** – 默认值，如果容器挂掉不自动重启
* **on-failure** – 当容器以非 0 码退出时重启容器   
  + 同时可接受一个可选的最大重启次数参数 (e.g. **on-failure:5**).
* **always** – 不管退出码是多少都要重启

docker run -itd --name test01 -p IP:sport:dport -m 1024m --memory-swap=1024m --cpuset-cpus="0,1" --restart=always <image ID>

docker exec -it test01 bash # 进入容器也可以用exec命令

**查看容器状态信息**

[root@localhost ~]# docker stats

[root@localhost ~]# docker stats --no-stream

### ****进入容器 - nsenter 命令****

**nsenter安装**   
nsenter 工具在 util-[**Linux**](http://lib.csdn.net/base/linux) 包2.23版本后包含。 如果系统中 util-linux 包没有该命令，可以按照下面的方法从源码安装。

cd /usr/src ; wget https://www.kernel.org/pub/linux/utils/util-linux/v2.28/util-linux-2.28.tar.gz

./configure --without-ncurses

make nsenter && sudo cp nsenter /usr/local/bin

**nsenter使用**   
nsenter 可以访问另一个进程的名字空间。nsenter 要正常工作需要有 root 权限。   
为了连接到容器，你还需要找到容器的第一个进程的 PID，可以通过下面的命令获取。

PID=$(docker inspect --format "{{ .State.Pid }}" <container>)

通过这个 PID，就可以连接到这个容器：

nsenter --target $PID --mount --uts --ipc --net –pid

更简单的，建议下载 .bashrc\_docker，并将内容放到 .bashrc 中。

get -P ~ https://github.com/yeasy/docker\_practice/raw/master/\_local/.bashrc\_docker;

echo "[ -f ~/.bashrc\_docker ] && . ~/.bashrc\_docker" >> ~/.bashrc; source ~/.bashrc

这个文件中定义了很多方便使用 Docker 的命令，例如 docker-pid 可以获取某个容器的 PID；而   
docker-enter 可以进入容器或直接在容器内执行命令。

### ****3、持久化容器与镜像****

#### ****3.1 通过容器生成新的镜像****

运行中的镜像称为容器。你可以修改容器（比如删除一个文件），但这些修改不会影响到镜像。不过，你使用docker commit 命令可以把一个正在运行的容器变成一个新的镜像。

docker commit <container> [repo:tag] # 将一个container固化为一个新的image，后面的repo:tag可选。

#### ****3.2 持久化容器****

export命令用于持久化容器

docker export <CONTAINER ID> > /tmp/export.tar

#### ****3.3 持久化镜像****

Save命令用于持久化镜像

docker save 镜像ID > /tmp/save.tar

**3.4 导入持久化**[**Container**](http://lib.csdn.net/base/docker)

删除container 2161509ff65e

docker rm 2161509ff65e

导入export.tar文件

cat /tmp/export.tar | docker import - export:latest

#### ****3.5 导入持久化image****

删除image daa11948e23d

docker rmi daa11948e23d

导入save.tar文件

docker load < /tmp/save.tar

对image打tag

docker tag daa11948e23d load:tag

#### ****3.6 export-import与save-load的区别****

导出后再导入(export-import)的镜像会丢失所有的历史，而保存后再加载（save-load）的镜像没有丢失历史和层(layer)。这意味着使用导出后再导入的方式，你将无法回滚到之前的层(layer)，同时，使用保存后再加载的方式持久化整个镜像，就可以做到层回滚。（可以执行docker tag 来回滚之前的层）。

#### ****3.7 一些其它命令****

docker logs $CONTAINER\_ID #查看docker实例运行日志，确保正常运行

docker inspect $CONTAINER\_ID #docker inspect <image|container> 查看image或container的底层信息

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

例如：

cp /mygithub/cas\_project/eurekaclient1/target/eurekaclient1-0.0.1-SNAPSHOT.jar app/eurekaclient1/

docker build -f app/eurekaclient1/Dockerfile -t myimage/eurekaclient1:0.0.1 .

docker run -d -p 50091:9091 -e host='192.168.1.14' -e port=50091 0de012dae1e0

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

### ****一些使用技巧****

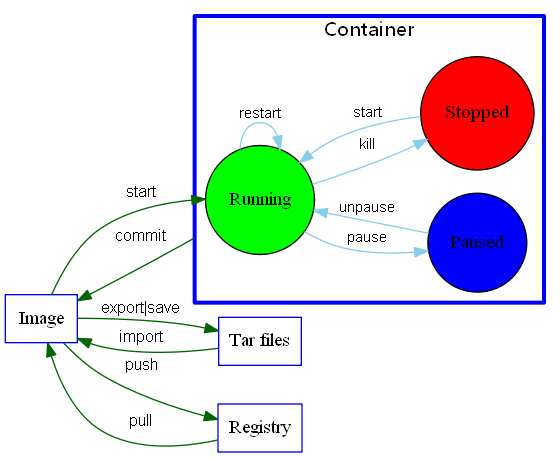
#### ****docker文件存放目录****

Docker实际上把所有东西都放到/var/lib/docker路径下了。

[root@localhost docker]# ls -F

containers/ devicemapper/ execdriver/ graph/ init/ linkgraph.db repositories-devicemapper volumes/

containers目录当然就是存放容器（container）了，graph目录存放镜像，文件层（file system layer）存放在graph/imageid/layer路径下，这样我们就可以看看文件层里到底有哪些东西，利用这种层级结构可以清楚的看到文件层是如何一层一层叠加起来的。



**使用Dockerfile构建镜像**

**构建镜像**

docker build -t keven/centos:tomcat-centos --rm=true .

-t选择指定生成镜像的用户名，仓库名和tag

--rm=true指定在生成镜像过程中删除中间产生的临时容器。

**运行镜像**

docker run -d -p 8090:8080 5207

-p指定主机80端口与容器8080端口进行绑定

-d 指定容器运行后与当前tty分离，后台运行

5207是镜像的ID前4位。

自动映射主机端口的启动方法

[root@wls12c web]$ docker run -d -p 8080 --name myweb 520

de39869a8c560e5e0cf48fc6022c05ed9f9a145bdafb897767fa468dc24ebfff

[root@wls12c web]$ docker port de3

8080/tcp -> 0.0.0.0:32768

[root@wls12c web]$

**查看挂载点**

docker inspect test1

**springcloud 多网卡处理方法**

#忽略eth0,支持正则表达式

spring.cloud.inetutils.ignored-interfaces[0]=eth0

#注册时使用ip而不是主机名

eureka.instance.prefer-ip-address=true

分析可配置的参数   
  
spring.cloud.inetutils   
ignored-interfaces:过滤掉的网卡名，正则匹配   
preferred-networks:偏向接受的网络ip地址，正则匹配   
use-only-site-local-interfaces: true，是否使用本地

在bootstrap.yml中配置后不生效，两种方式处理：   
- UtilAutoConfiguration中，可以看出，若配置了spring.cloud.util.enabled，则会使用配置文件中的配置。   
- 在启动中使用-D加系统参数

### 忽略指定网卡

1. 1 忽略网卡:spring.cloud.inetutils.ignored-interfaces
2. [?](http://www.jb51.net/article/137676.htm)

|  |  |
| --- | --- |
| 1  2 | #忽略eth0, 支持正则表达式  spring.cloud.inetutils.ignored-interfaces[0]=eth0 |

1. 2 指定IP:spring.cloud.inetutils.preferred-networks
2. [?](http://www.jb51.net/article/137676.htm)

|  |  |
| --- | --- |
| 1  2 | #preferred-networks是指倾向于使用的IP地址，接收一个正则表达式数组，用于选择Spring Cloud应用使用的本机的IP地址。  spring.cloud.inetutils.preferredNetworks[0]=^192\.168 |

1. 3 修改Host文件
2. 当网查遍历逻辑都没有找到合适ip时会走JDK的InetAddress.getLocalHost()。该方法会返回当前主机的hostname, 然后会根据hostname解析出对应的ip。因此第二种方案就是配置本机的hostname和/etc/hosts文件，直接将本机的主机名映射到有效IP地址。
3. 4 指定实例IP
4. [?](http://www.jb51.net/article/137676.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | #指定此实例的ip  eureka.instance.ip-address=  #注册时使用ip而不是主机名  eureka.instance.prefer-ip-address=true |

1. 5 通过jar命令参数
2. [?](http://www.jb51.net/article/137676.htm)

|  |  |
| --- | --- |
| 1  2  3 | java -jar xxx.jar --spring.cloud.inetutils.preferred-networks= #需要设置的IP地址  或者  java -jar xxx.jar --spring.cloud.inetutils.ignored-interfaces= #需要过滤掉的网卡 |

1. 以上新手折腾了一下午，大家有好方法一起交流一下，希望对大家的学习有所帮助，也希望大家多多支持脚本之家。

通过上面源码分析可以得知，spring cloud肯定能配置一个网卡忽略列表。通过查文档资料得知确实存在该属性：

spring.cloud.inetutils.ignored-interfaces[0]=eth0 # 忽略eth0, 支持正则表达式

（2）将spring.cloud.inetutils.preferred-networks的配置从application.yml文件移到外部，jar命令-Dspring.cloud.inetutils.preferred-networks: 192.168.239.覆盖或者设置在系统的环境变量里。用dockerfile或者docker compose就更方便了。

## Docker Compose 安装说明

安装docker compose

curl -L https://github.com/docker/compose/releases/download/1.8.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

为Docker Compose脚本添加执行权限

chmod +x /usr/local/bin/docker-compose

安装完成，测试：

docker-compose --version

结果显示：

docker-compose version 1.8.0, build f3628c7

说明Compose已经成功安装完成了。

安装Compose命令补全工具

curl -L https://raw.githubusercontent.com/docker/compose/$(docker-compose version --short)/contrib/completion/bash/docker-compose > /etc/bash\_completion.d/docker-compose

## Docker Machine安装说明

1. 安装官方文档安装

$ base=https://github.com/docker/machine/releases/download/v0.14.0 &&

curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&

sudo install /tmp/docker-machine /usr/local/bin/docker-machine

安装docker machine自动补全脚本

base=https://raw.githubusercontent.com/docker/machine/v0.14.0

for i in docker-machine-prompt.bash docker-machine-wrapper.bash docker-machine.bash

do

sudo wget "$base/contrib/completion/bash/${i}" -P /etc/bash\_completion.d

done

执行

PS1='[\u@\h \W$(\_\_docker\_machine\_ps1)]\$ '

创建docker host

1. 生成秘钥 ssh-keygen -t rsa

id\_rsa :生成的私钥文件

id\_rsa.pub：生成的公钥文件

如果希望公钥生效，需满足以下两个条件：

1) .ssh目录的权限必须是700

2) .ssh/authorized\_keys文件权限必须是600

1. 远程拷贝私钥文件，实现ssh无秘钥登录

$: ssh-copy-id ip地址

1. 执行docker-machine create创建第一个docker1

docker-macine create --driver generic --generic-ip-address=192.168.7.235 docker1

命令分析：

create #创建docker主机  
--driver generic #驱动类型 generic 支持linux通用服务器，还支持很多种云主机  
--generic-ip-address=192.168.1.80 #指定主机  
--generic-ssh-key ~/.ssh/id\_rsa #指定私钥  
--generic-ssh-user=test #指定用户  
docker1 #主机名称

1. 远程执行docker命令，创建docker虚拟机
2. docker-machine ssh docker1 docker ps -a
3. docker-machine ssh docker1 docker run hello-world
4. 也可以通过切换环境变量来实现

eval $(docker-machine env docker1)

1. 查看docker具体配置

查看具体配置

在/etc/systemd/system/docker.service.d下文件名是：10-machine.conf

1. docker-machine upgrate docker1 docker2批量更新docker host版本
2. docker-machine config docker1 查看host的docker daemon配置
3. docker-machine start/status/restart/kill  docker1是对docker host所在的系统的操作，不是对服务的操作
4. docker-machine scp是在docker host之间复制

dockermachine scp docker1:/etc/passwd docker2:/passwd

1. docker-machine rm docker1删除一个docker host