**一、安装**

**查看端口与进程对应关系 lsof -i:6001**

**内核：centos6.5一般是2.6x的内核**

查看内核：uname -r uname -a cat /etc/issue

**1、导入public key**

[root@MidApp ~]# rpm --import <https://www.elrepo.org/RPM-GPG-KEY-elrepo.org>

curl: (35) SSL connect error

error: https://www.elrepo.org/RPM-GPG-KEY-elrepo.org: import read failed(2).

[root@MidApp pgsql]# yum update nss

**2、安装ELRepo到CentOS**

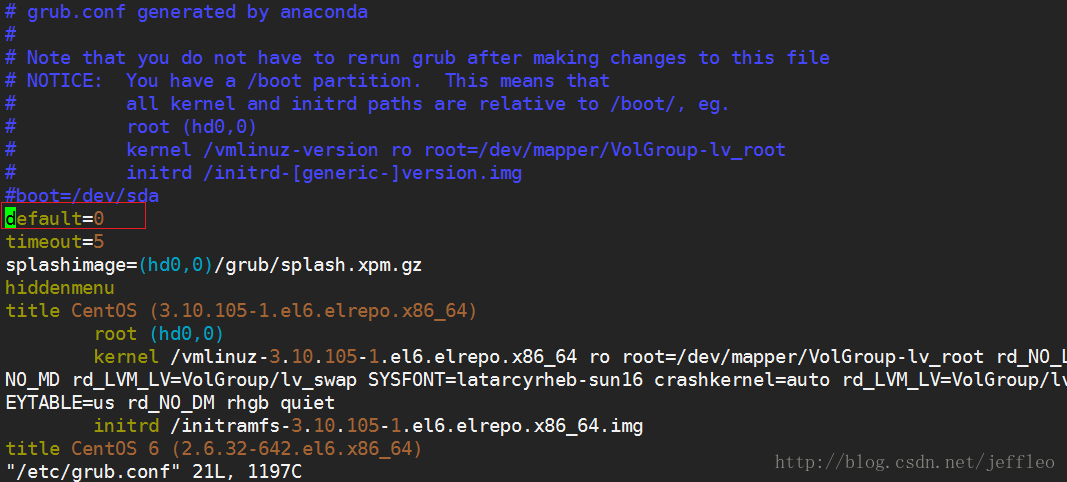
**rpm -Uvh http://www.elrepo.org/elrepo-release-6-8.el6.elrepo.noarch.rpm**

**3、安装 kernel-lt（lt=long-term）**

[root@localhost ~]# yum --enablerepo=elrepo-kernel install kernel-lt -y

**4、编辑grub.conf文件，修改Grub引导顺序**

vim /etc/grub.conf



1. 查看内核版本：uname -r
2. docker安装: yum -y install docker-io

sudo yum install docker-io (如果提示 "No package docker-io available"说明不能连接epel库，则需要执行 $ sudo yum install http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.noarch.rpm后再安装docker-io）

1. [root@runoob ~]# service docker start
2. [root@runoob ~]#docker run hello-world
3. 出现错误信息

[root@mu4 ~]# service docker restart

Stopping docker: [FAILED]

Starting docker: [ OK ]

针对错误信息执行：yum install device-mapper-event-libs

配置加速镜像库：新建/etc/docker/daemon.json

//{ "registry-mirrors": [ "加速地址" ], "insecure-registries": [] }

{"registry-mirrors": ["http://hub-mirror.c.163.com"]}

1. service docker restart 然后运行docker run hello-world

Docker升级：

curl -sSL -O https://get.docker.com/builds/Linux/x86\_64/docker-1.9.1

chmod +x docker-1.9.1

mv docker-1.9.1 /usr/local/bin/docker

cp /usr/local/bin/docker /usr/bin/docker

加速

curl -sSL https://get.daocloud.io/daotools/set\_mirror.sh | sh -s http://ef017c13.m.daocloud.io

[root@node-1 ~]# docker pull alpine

[root@node-1 ~]# docker pull library/rethinkdb

[root@node-1 ~]# docker pull microbox/etcd

[root@node-1 ~]# docker pull shipyard/docker-proxy

[root@node-1 ~]# docker pull swarm

[root@node-1 ~]# docker pull shipyard/shipyard

curl -s https://shipyard-project.com/deploy | bash -s

shipyard访问页面默认登录用户名：admin，密码：shipyard，登录进去后，可以在"ACCOUNTS"选项里管理用户，可以添加用户，并对用户进行角色授权。

2）添加node节点

# curl -sSL http://dockerclub.net/public/script/deploy | ACTION=node DISCOVERY=etcd://<shipyard部署机ip> bash -s                ==> 中文版

# curl -sSL https://shipyard-project.com/deploy | ACTION=node DISCOVERY=etcd://<shipyard部署机ip> bash -s                       ==> 英文版

3）删除shipyard（在节点机上执行，就会将节点从shipyard管理里踢出）

# curl http://dockerclub.net/public/script/deploy | ACTION=remove bash -s                   ==> 中文版

# curl -sSL https://shipyard-project.com/deploy | ACTION=remove bash -s                     ==> 英文版

2）停止运行镜像

[root@node-1 ~]# docker stop shipyard-proxy shipyard-certs shipyard-discovery shipyard-rethinkdb shipyard-swarm-agent shipyard-swarm-manager shipyard-controller

3）启动运行的镜像

[root@node-1 ~]# docker start shipyard-proxy shipyard-certs shipyard-discovery shipyard-rethinkdb shipyard-swarm-agent shipyard-swarm-manager shipyard-controller

比如添加182.48.115.235作为一个节点。那么就需要在182.48.115.235这台机器上执行下面的命令：

[root@k8s-node-1 ~]# curl -sSL https://shipyard-project.com/deploy | ACTION=node DISCOVERY=etcd://182.48.115.233:4001 bash -s

注意：

1）上面命令中的182.48.115.233是shipyard的部署机的ip；

2）182.48.115.233机器的iptables防火墙要打开4001端口。最好是关闭防火墙

同理，添加其他节点操作均是在节点机上执行下面命令（节点机器要能和shipyard部署机通信）

# curl -sSL https://shipyard-project.com/deploy | ACTION=node DISCOVERY=etcd://182.48.115.233:4001 bash -s

**集群部署：**

**shipyard:**集成管理docker容器、镜像、Registries的工具，提供webUI管理swarm平台

**docker三剑客:**

**docker machine:**用于在各种平台上快速创建具有docker服务的虚拟机的技术

**docker-compose:**通过一个.yml配置文件，将所有的容器的部署方法、文件映射、容器连接等等一系列的配置写在一个配置文件里，最后只需要执行docker-compose up命令就会像执行脚本一样的去一个个安装容器并自动部署他们，极大的便利了复杂服务的部署。

**docker swarm:**基于docker平台实现的集群技术

**1、镜像准备：**

# docker pull rethinkdb

# docker pull microbox/etcd

# docker pull shipyard/docker-proxy

# docker pull swarm

# docker pull shipyard/shipyard/

**手动安装启动未成功！**

**手动安装（以本地ip：10.211.55.20为例）**

1、数据存储 rethinkdb

docker run -ti -d --name shipyard-rethinkdb rethinkdb

2、服务发现（etcd）

为了启用Swarm leader选择，我们必须使用来自Swarm容器的外部键值存储。此处，我们使用Etcd作为服务发现工具。可以选用的服务发现工具还有Consul、Zookeeper等。

|  |  |
| --- | --- |
| 1 | docker run -ti -d -p 4001:4001 -p 7001:7001 --name shipyard-discovery microbox/etcd -name discovery |

3. Docker代理服务（shipyard/docker-proxy）

默认情况下，Docker引擎只侦听套接字。 我们可以重新配置引擎以使用TLS，或者您可以使用代理容器。 这是一个非常轻量级的容器，它只是将请求从TCP转发到Docker监听的Unix套接字。

格式：docker run -ti -d -p 2375:2375 --hostname=$HOSTNAME --name shipyard-proxy -v /var/run/docker.sock:/var/run/docker.sock -e PORT=2375 shipyard/docker-proxy

例：docker run -ti -d -p 2375:2375 --hostname=mu4 --name shipyard-proxy -v /var/run/docker.sock:/var/run/docker.sock -e PORT=2375 shipyard/docker-proxy

4.Swarm管理节点（swarm manager）

将IP-OF-HOST替换为对应的ip;

将SWARM-INNER-PORT换成对应的端口，默认2375.

将SWARM-MAP-PORT换成映射到host机的ip，例如2376

格式：docker run -ti -d --name -p [SWARM-MAP-PORT]:[SWARM-INNER-PORT] shipyard-swarm-manager docker.io/swarm manage --host tcp://0.0.0.0:[SWARM-INNER-PORT] etcd://[IP-OF-HOST]:4001

实例化：docker run  -ti -d --name shipyard-swarm-manager swarm manage --host tcp://0.0.0.0:3375 etcd://10.211.55.20:4001

5.Swarm Agent节点将当前docker节点加入到集群中

将IP-OF-HOST替换为对应的ip

将IP-OF-ETCD-HOST换成etcd所在的docker节点ip。

将LOCAL-DOCKER-PORT换成本机docker所监听的端口，例如2375

格式：docker run -ti -d --name shipyard-swarm-agent docker.io/swarm join --addr [IP-OF-HOST]:[LOCAL-DOCKER-PORT] etcd://[IP-OF-ETCD-HOST]:4001

实例： docker run -ti -d --name shipyard-swarm-agent swarm join --addr 10.211.55.20:2375 etcd://10.211.55.20:4001

6.启动Shipyard管理工具

docker run -ti -d --name shipyard-controller --link shipyard-rethinkdb:rethinkdb --link shipyard-swarm-manager:swarm -p 8083:8080 shipyard/shipyard server -d tcp://swarm:3375

7.浏览器访问：http://192.168.1.108:8083/ 即可进入链接

默认用户名/密码 admin/shipyard

防火墙需打开4001端口，否则防火墙会拦住容器。

Nginx做域名转发

在nginx配置文件http块里添加如下代码

upstream shipyard {

     server 192.168.1.108:8083;

}

server {

    listen          80;

    server\_name     mm.shipyard.com;

    location / {

        proxy\_pass http://shipyard;

        proxy\_http\_version 1.1;

        proxy\_set\_header Upgrade $http\_upgrade;

        proxy\_set\_header Connection "upgrade";

    }

}

以下三行代码为nginx做WebSockets跳转否则，在执行shell时会报400错误

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

**二、使用**

**Docker支持的网络模式有：**

none ：关闭容器内的网络连接

bridge ：通过veth接口来连接容器，默认配置。

host ：允许容器使用host的网络堆栈信息。 注意：这种方式将允许容器访问host中类似D-BUS之类的系统服务，所以认为是不安全的。

container ：使用另外一个容器的网络堆栈信息。

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

$ sudo docker run -d --name redis example/redis --bind 127.0.0.1

$ # use the redis container's network stack to access localhost

$ sudo docker run --rm -ti --net container:redis example/redis-cli -h 127.0.0.1

Docker有两个主要的部件：

Docker: 开源的容器虚拟化平台。

Docker Hub: 用于分享、管理Docker容器的Docker SaaS平台。

Docker使用客户端-服务器(client-server)架构模式。Docker客户端会与Docker守护进程进行通信。

Docker守护 进程会处理复杂繁重的任务，例如建立、运行、发布你的Docker容器。

Docker客户端和守护进程可以运行在同一个系统上，当然你也可以使用 Docker客户端去

连接一个远程的Docker守护进程。Docker客户端和守护进程之间通过socket或者RESTful API进行通信。

要理解Docker的内部构建，必须知道以下三种部件：

Docker镜像 (Docker images)。

Docker仓库 (Docker registeries)。

Docker容器(Docker containers)。

Docker镜像是一个只读的模板

Docker仓库用来保存镜像。可以理解为代码控制中的代码仓库。同样的，Docker仓库也有公有和私有的概念。公有的Docker仓库名字是Docker Hub。Docker Hub提供了庞大的镜像集合供使用。这些镜像可以是你自己创建的，或者你也可以在别人的镜像基础上创建。Docker仓库是Docker的 分发 部分。

一个Docker容器包含了所有的某个应用运行所需要的环境。每一个Docker容器都是从Docker镜像创建 的。Docker容器可以运行、开始、停止、移动和删除。每一个Docker容器都是独立和安全的应用平台。Docker容器是Docker的 运行 部分。

搭建私有仓库  
首先在136机器上下载registry镜像  
$ sudo docker pull registry  
下载完之后我们通过该镜像启动一个容器  
$ sudo docker run -d -p 5000:5000 registry  
默认情况下，会将仓库存放于容器内的/tmp/registry目录下，这样如果容器被删除，则存放于容器中的镜像也会丢失，所以我们一般情况下会指定本地一个目录挂载到容器内的/tmp/registry下，如下：  
$ sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry

docker run

-a=[] 　　　 : Attach to `STDIN`, `STDOUT` and/or `STDERR`

-t=false 　　 : Allocate a pseudo-tty

--sig-proxy=true　 : Proxify all received signal to the process (non-TTY mode only) false就不能用ctrl D C退出了

-i=false 　　 : Keep STDIN open even if not attached

-d : 后台运行

--rm=false : Automatically remove the container when it exits (incompatible with -d)

-m="" : Memory limit (format: <number><optional unit>, where unit = b, k, m or g)

-c=0 : CPU shares (relative weight) -c=0完全利用，=512则只能利用50%

package docker:build

docker pull 10.167.202.170/hx-crawler/common-microservicehx-baidurss:0.0.1-SNAPSHOT

docker run -itdP --net over 10.167.202.170/hx-crawler/common-microservicehx-baidurss:0.0.1-SNAPSHOT

停止容器 docker stop dockerid  
搜索镜像 docker search httpd

删除镜像 docker rmi docker rm

删除所有容器 docker rm $(docker ps -a -q)

日志 docker logs dockerid

docker logs -f container name 噼里啪啦 一大堆....太对，清理掉

进入docker docker exec -it 4f6889c062ba /bin/sh  
 docker attach 44fc0f0582d9

docker login 10.167.211.250 -u admin -p Xinhe2017

查看容器运行 docker ps  
docker ps -as

查看容器的root用户密码 docker logs <容器名orID> 2>&1 | grep '^User: ' | tail -n1  
查看容器日志 docker logs -f <容器名orID>  
删除所有镜像 docker rmi $(docker images | grep none | awk '{print $3}' | sort -r)  
  
运行一个新容器，同时为它命名、端口映射、文件夹映射。以redmine镜像为例  
docker run --name redmine -p 9003:80 -p 9023:22 -d -v /var/redmine/files:/redmine/files -v /var/redmine/mysql:/var/lib/mysql sameersbn/redmine  
  
一个容器连接到另一个容器  
docker run -i -t --name sonar -d -link mmysql:db tpires/sonar-server  
sonar  
容器连接到mmysql容器，并将mmysql容器重命名为db。这样，sonar容器就可以使用db的相关的环境变量了。  
  
当需要把一台机器上的镜像迁移到另一台机器的时候，需要保存镜像与加载镜像。  
机器a docker save busybox-1 > /home/save.tar  
机器b docker load < /home/save.tar  
  
查看docker资源 docker attach $ID  
  
从container中拷贝文件出来  
sudo docker cp 7bb0e258aefe:/etc/debian\_version .  
拷贝7bb0e258aefe中的/etc/debian\_version到当前目录下。  
注意：只要7bb0e258aefe没有被删除，文件命名空间就还在，可以放心的把exit状态的container的文件拷贝出来