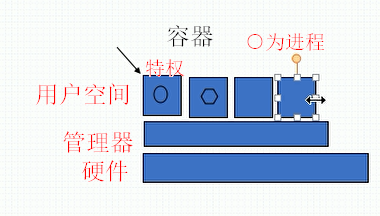
一 docker容器技术基础

1、容器技术



用户空间隔离包括：UTS（主机名用户名，以名称空间为单位隔离），Mount（文件系统），IPC（进程间通信） ，PID（进程号），User（用户组隔离），Net（网络）

内核级实现↑：名称空间namespaces隔离机制 通过系统调用向外输出.

容器级虚拟化技术： namespaces + chroot （Linux内核版本要求3.8以上）

核心技术 namespaces，CGroups，chroot,



CGroups (control groups)

把系统级的资源分组，然后把每组的资源量 分配给特点用户空间的进程。 组可以被再次划分。

1. LXC解决方案——Linux Container

把容器技术用一组简易使用的工具和模板（安装脚本） 简化了使用。基于模板template创建和启动容器。

3、docker ——LXC的简化升级，将其二次封装发行

镜像image: 将用户进程运行需要的一套组件事先装进镜像中，存入仓库，创建容器基于镜像。

一个容器中只运行一个进程。

缺点：

①整体占用空间大（组件有重复），

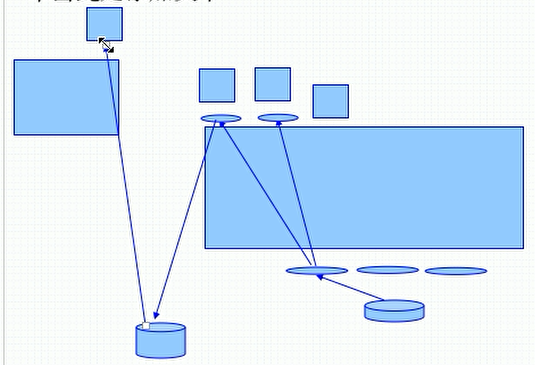
②调试工具也要准备多套 每个镜像一套 自带。调试工具平时不启动，需要时启动一个调试进程运行在当前容器中，不影响主进程。

分层构建，联合挂载：

一个功能在一层上实现，叠在一起合成统一视角 基础镜像+附加镜像叠加（如centOS+Nginx, centOS+Tomcat）。

镜像为只读 不会被由它创建的容器改变，除非在联合挂载的镜像栈顶端创建一个能读能写的新层。

不在容器内部存储数据，在容器外挂载一个持久存储设备在，这样进程终止后文件和数据不会丢，所以进程用完后可以直接把容器删了。



1. 容器编排

能在docker之上 把应用程序之间的依赖从属关系 反映在启动关闭时的次序和管理逻辑中。

容器编排工具：swarm + compose + machine + 统一资源调度：mesos + marathon +集群 kubernetes

二 docker使用

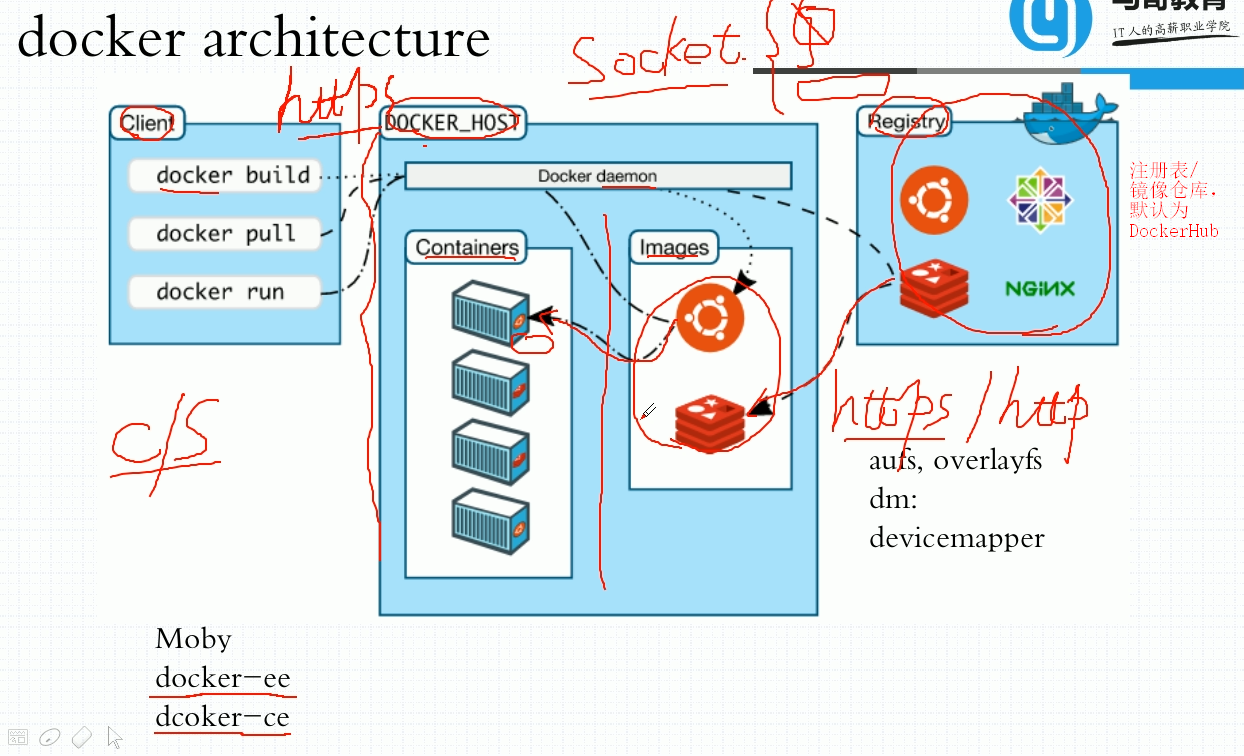
1.

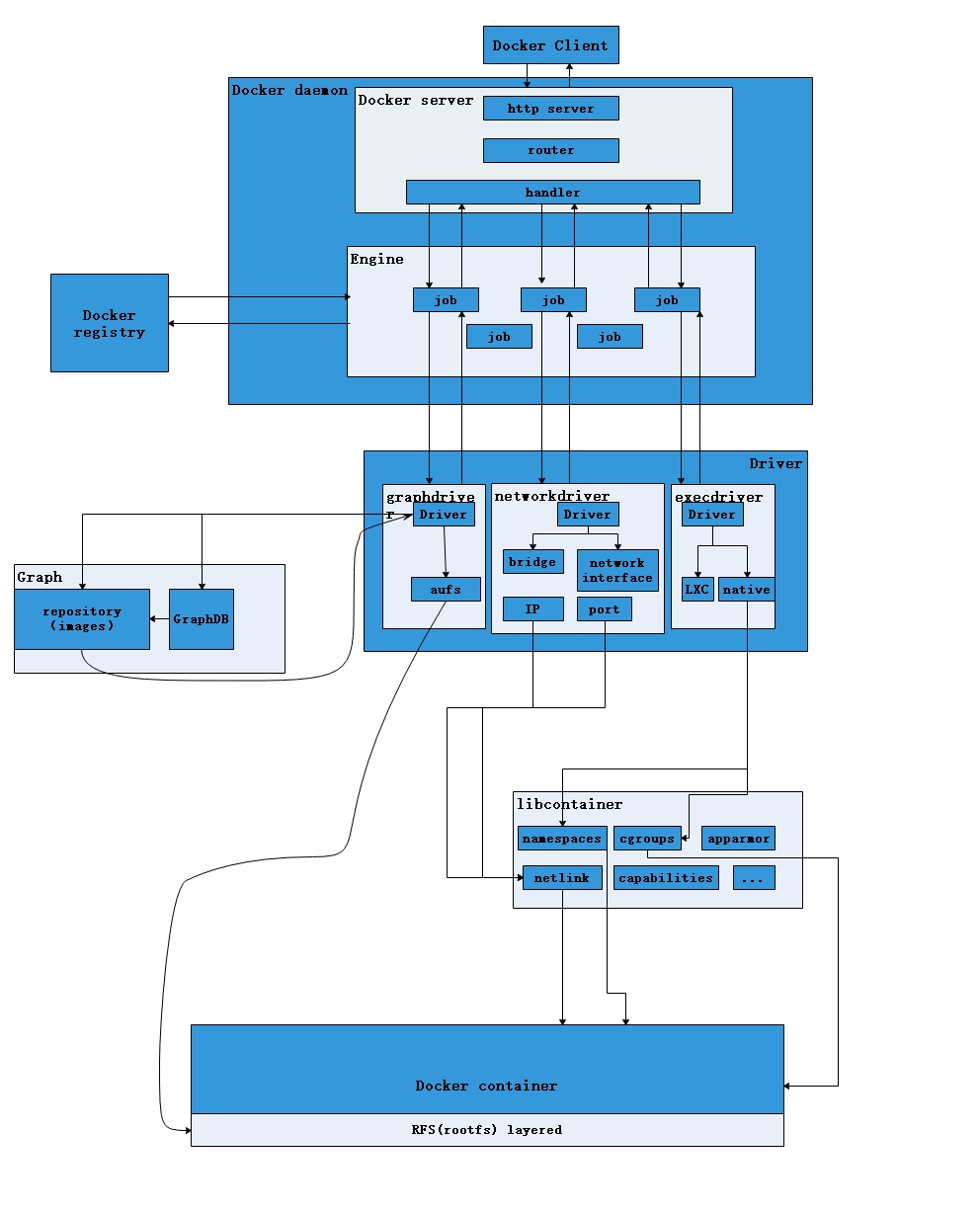
OCI：围绕容器格式和运行时制定一个开放的工业化标准

两部分：运行时标准、镜像格式标准

OCF（开放容器标准，）：runC为其主要实现 和运行引擎

1. Docker架构





用户是使用Docker Client与Docker Daemon建立通信，并发送请求给后者。

Docker Daemon作为Docker架构中的主体部分，首先提供Server的功能使其可以接受Docker Client的请求；而后Engine执行Docker内部的一系列工作，每一项工作都是以一个Job的形式的存在。

Job的运行过程中，当需要容器镜像时，则从Docker Registry中下载镜像，并通过镜像管理驱动graphdriver将下载镜像以Graph的形式存储；当需要为Docker创建网络环境时，通过网络管理驱动networkdriver创建并配置Docker容器网络环境；当需要限制Docker容器运行资源或执行用户指令等操作时，则通过execdriver来完成。

而libcontainer是一项独立的容器管理包，networkdriver以及execdriver都是通过libcontainer来实现具体对容器进行的操作。当执行完运行容器的命令后，一个实际的Docker容器就处于运行状态，该容器拥有独立的文件系统，独立并且安全的运行环境等。

主要模块功能：

1）docker daemon:

docker架构中一个常驻在后台的系统进程.其功能是：接收处理docker client发送的请求。该守护进程在后台启动一个server，server负载接受docker client发送的请求；接受请求后，server通过路由与分发调度，找到相应的handler来执行请求。docker daemon的架构可以分为：docker server、engine、job。

2）docker client:

docker架构中用户用来和docker daemon建立通信的客户端，用户使用的可执行文件为docker，通过docker命令行工具可以发起众多管理container的请求。

docker client发送容器管理请求后，由docker daemon接受并处理请求，当docker client 接收到返回的请求相应并简单处理后，docker client 一次完整的生命周期就结束了，当需要继续发送容器管理请求时，用户必须再次通过docker可以执行文件创建docker client。

1. docker registry

Docker Registry是存储容器镜像的仓库 + 用户获取镜像时的认证。同时提供了当前服务器上所有可用镜像的搜索索引。

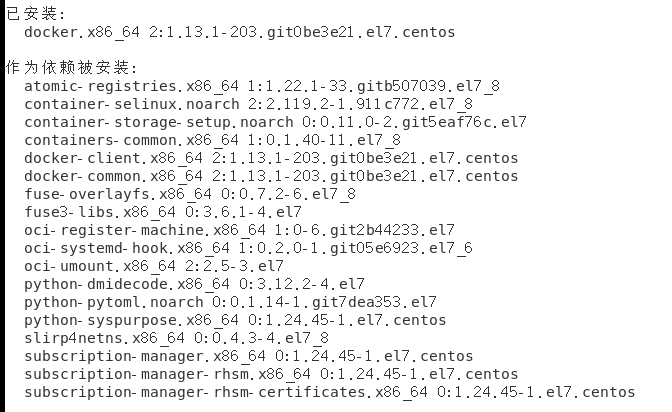
在Registry中可以有多个仓库(repository)，一个仓库放一个应用程序的镜像（不同版本用tag标签标识）；

而容器镜像是在容器被创建时，被加载用来初始化容器的文件架构与目录。在Docker的运行过程中，Docker Daemon会与Docker Registry通信，并实现搜索镜像、下载镜像、上传镜像三个功能，这三个功能对应的job名称分别为”search”，”pull” 与 “push”。

在Docker架构中，Docker可以使用公有的Docker Registry，即大家熟知的Docker Hub，如此一来，Docker获取容器镜像文件时，必须通过互联网访问Docker Hub；同时Docker也允许用户构建本地私有的Docker Registry，这样可以保证容器镜像的获取在内网完成。

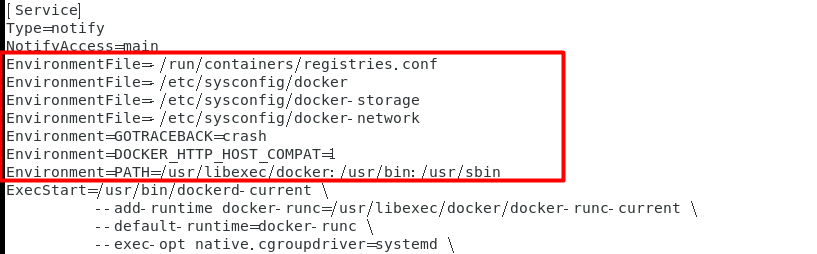
4.docker安装与使用

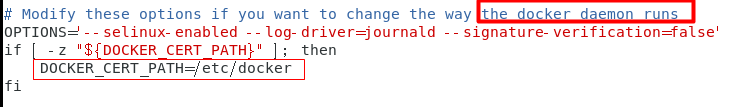
yum install docker



用命令查看docker各个配置文件位置

cat /usr/lib/systemd/system/docker.service





配置docker镜像加速： docker-cn、阿里云加速器、中国科技大学

创建 /etc/docker/daemon.json

在配置文件中加入：

vim /etc/docker/daemon.json

{

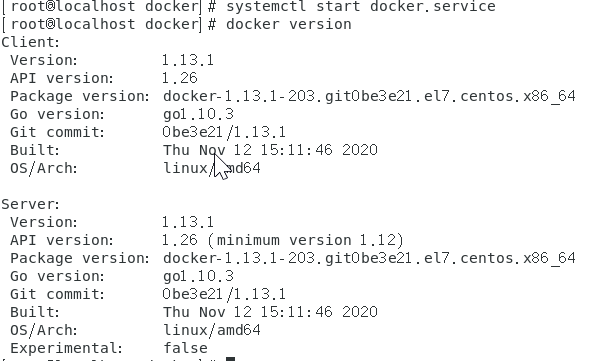
"registry-mirrors": ["https://registry.docker-cn.com"]

}

docker启动：

systemctl start docker.service

docker version





从docker hub 拉取镜像：

docker image pull nginx:1.14-alpine (alpine代表最小化安装版本)

本地docker镜像列表：

docker image ls (docker images)





删除镜像：

docker image rm [ ] = docker rmi [ ]

删除容器： docker rm

docker container create = docker create 创建容器：

docker container start 启动处于停止状态的容器

docker container stop 停止容器

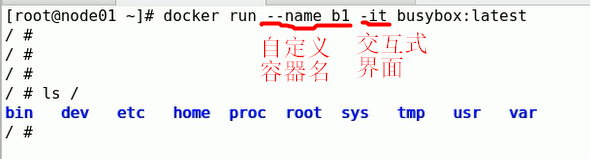
docker container run 创建并直接启动

docker container rm 删除

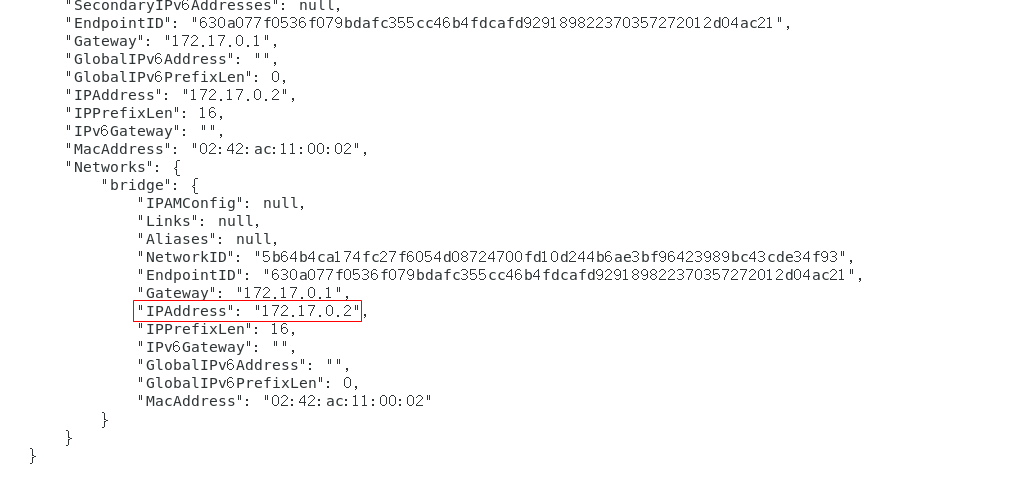
docker container pause/unpause 暂停/重启

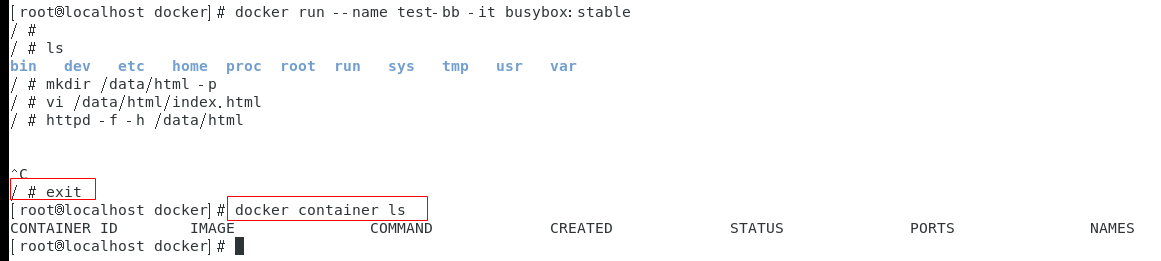
docker container top 查看容器消耗资源的排行

docker container ls 列出所有容器的信息 = docker ps



docker inspect [ 容器name] 查看 容器的具体信息



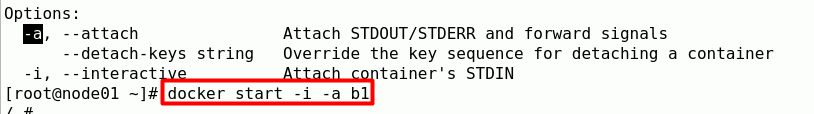


在busybox中输入exit表示退出终端，也就意味着进程终止，也意味着容器停止，

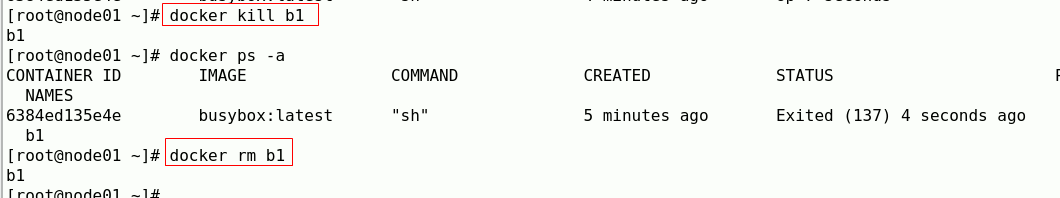
docker container ls = docker ps 默认不显示停止状态的容器

docker container ls -a才会显示所有的 包括停止了的

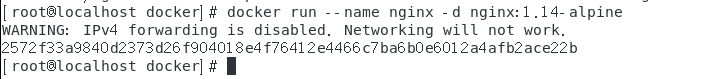
重新激活容器：



（docker container start -ai test-bb）

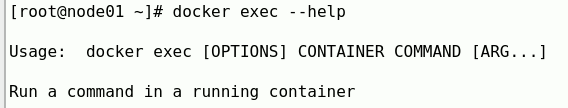


启动nginx容器 （-d表示在后台启动）

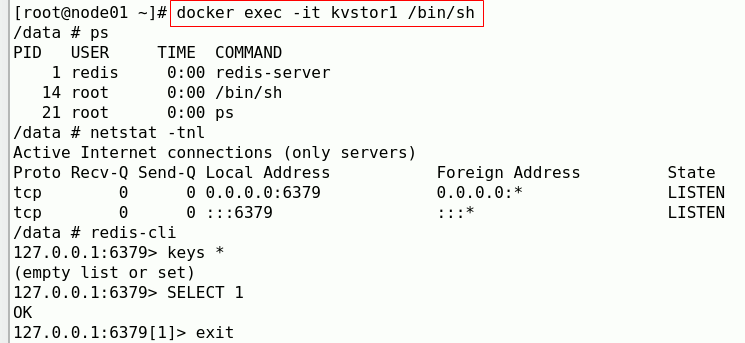


容器中的进程必须是前台运行的状态

在容器中执行命令：

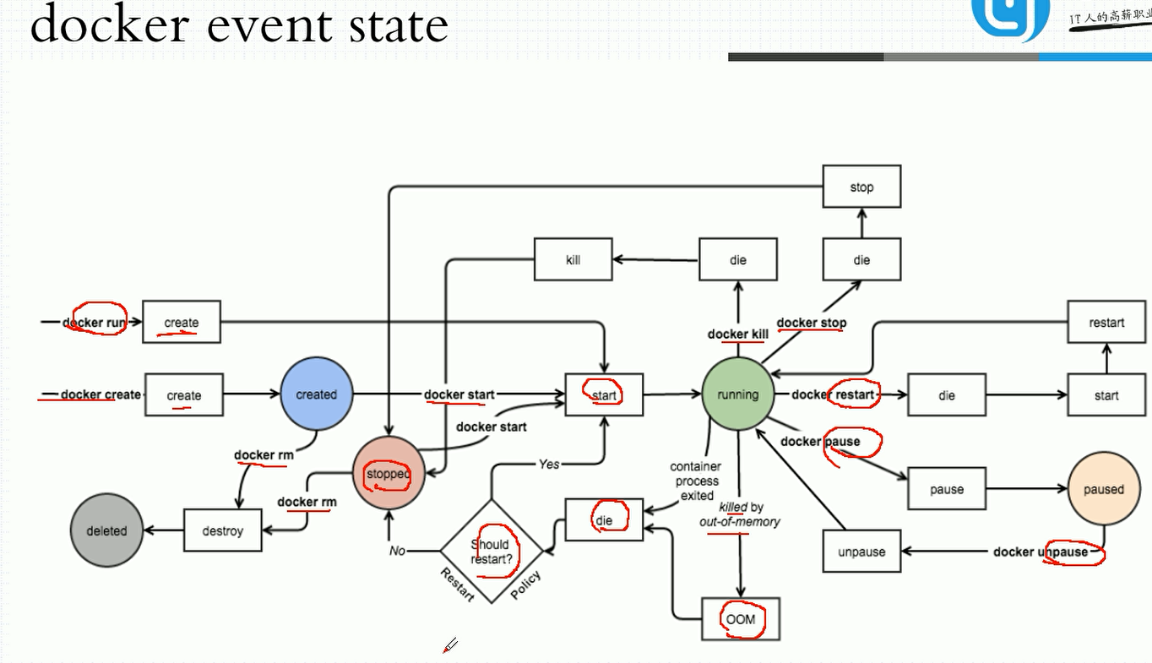


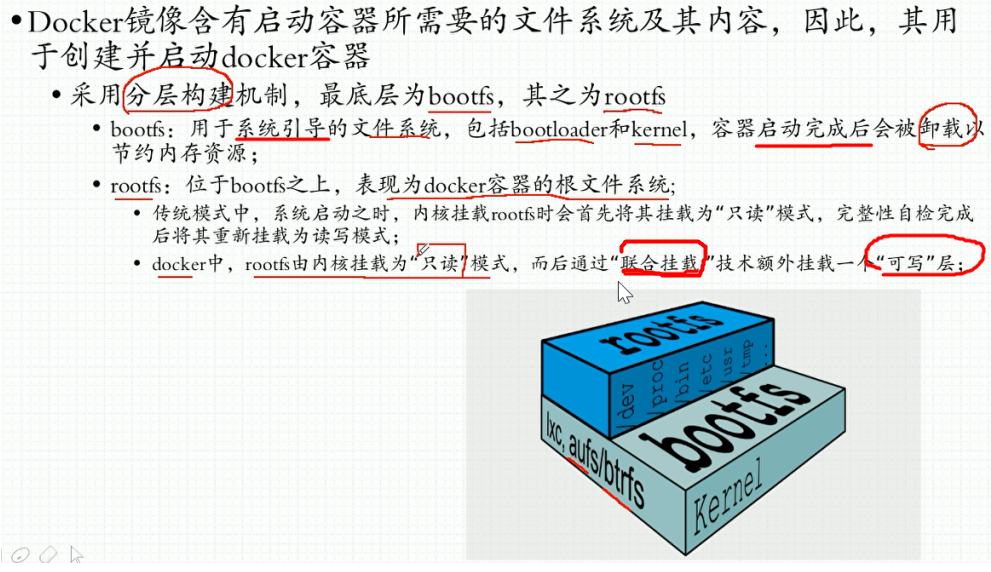
docker exec -it kvstor1 /bin/sh

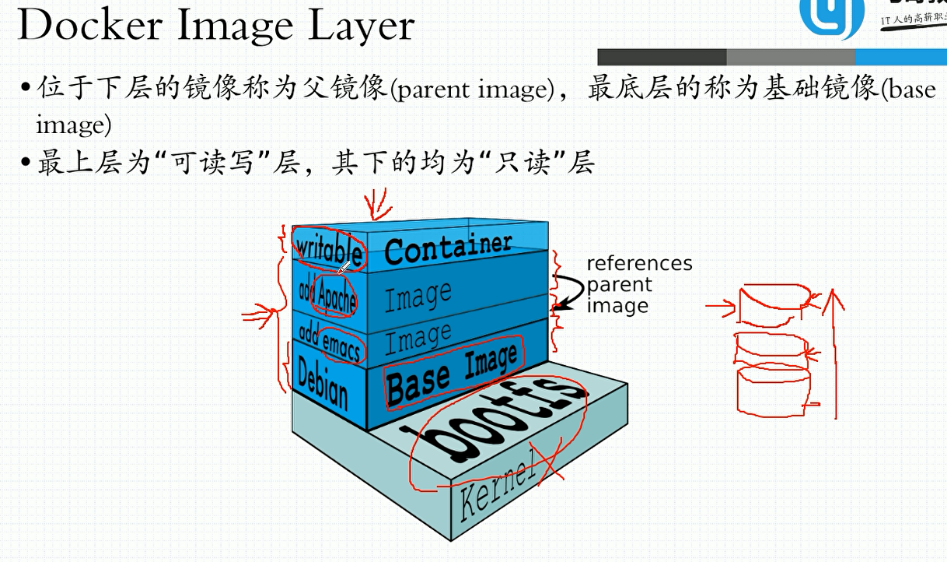


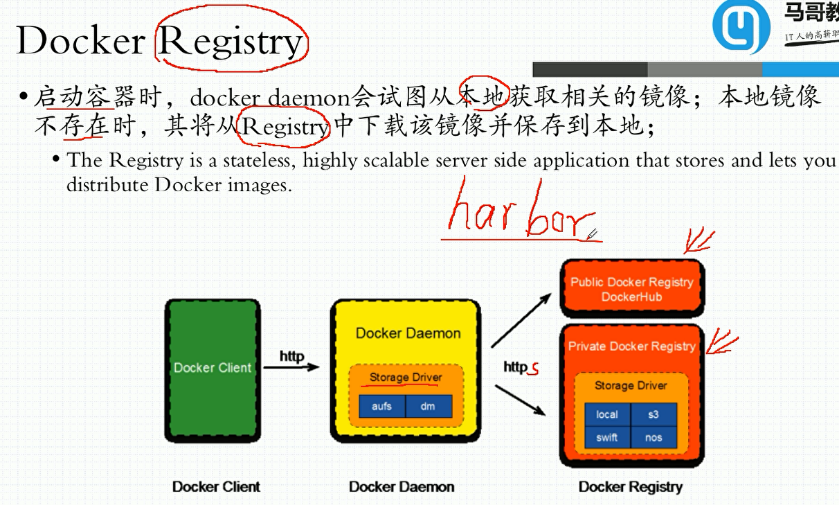
查看容器的日志：

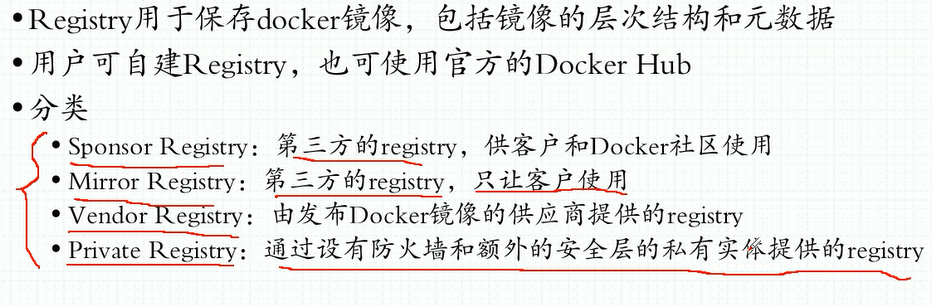


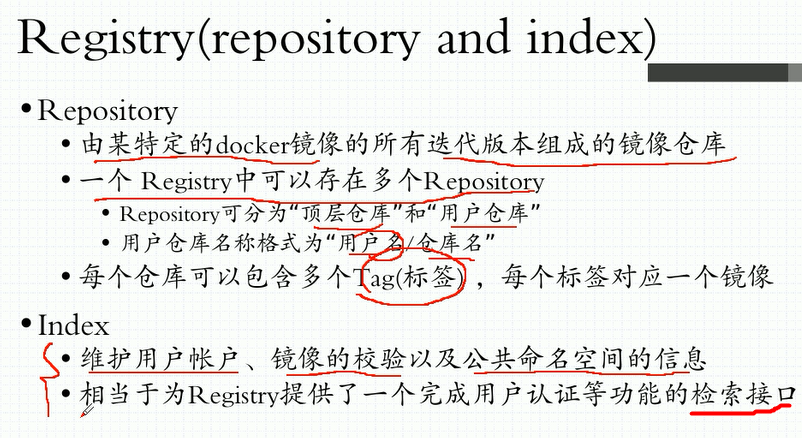


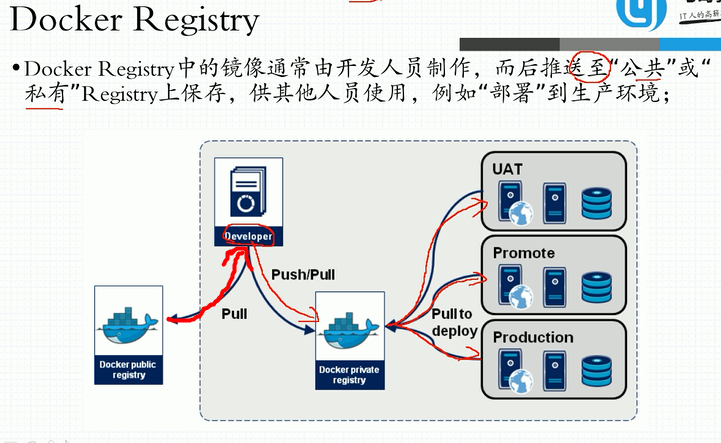




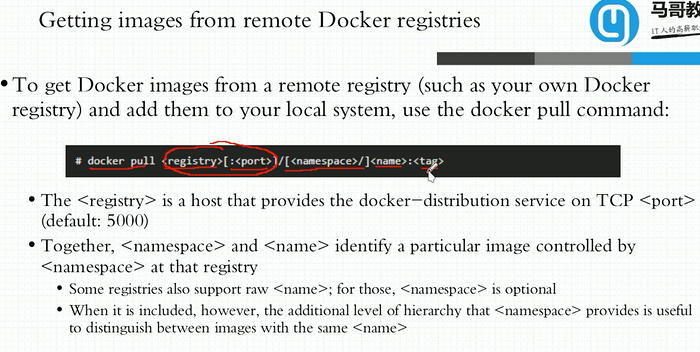






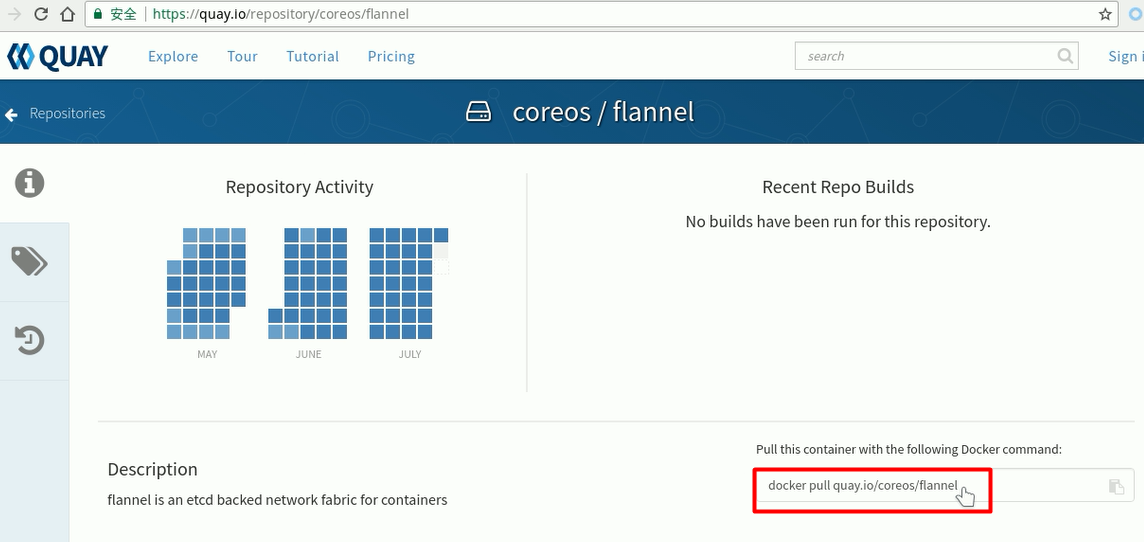


云源生： 通过大量的环境变量传递参数 容器启动时，从环境变量中加载 然后注入到配置文件中，以此来改变进程的配置文件

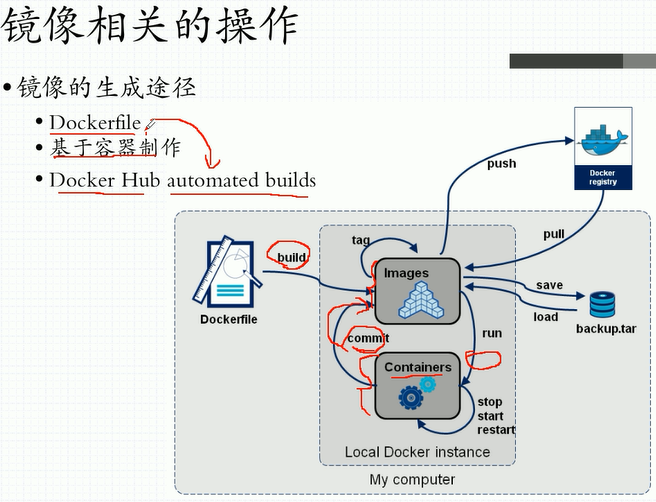


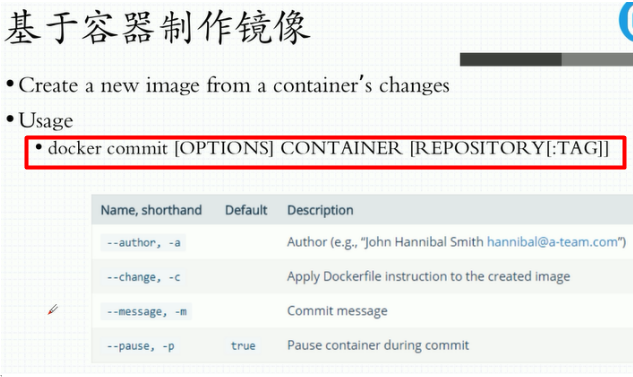
镜像库除了dockerhub 还有https:// quay.io 等

例：获取flannel镜像



docker pull quay.io/coreos/flannel (没加标签默认为latest)

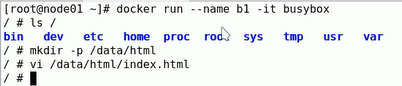




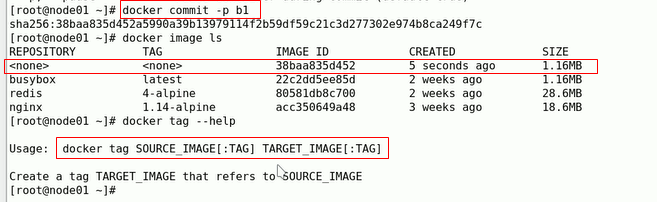
将当前在跑的容器（运行时产生了变化）打包成镜像并提交。

比如：在跑的busybox进程容器中 创建一些目录&文件，

在容器运行的状态下 用docker commit打包并提交镜像，之后用新镜像起容器时，里面便会包含之前那次创建的目录&文件



（另开命令终端）



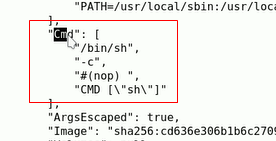
用docker tag命令可以给创建的容器标注上仓库名、一个或多个标签（版本等信息）



所以用docker rm [Repository name] 删除镜像时，若一个镜像（唯一的image ID）在一个tag被删后还有其他tag 那么该镜像并没有被真正删除（只是被untagged了一个，相当于删除了镜像的一个引用）

修改容器启动时默认运行的命令：

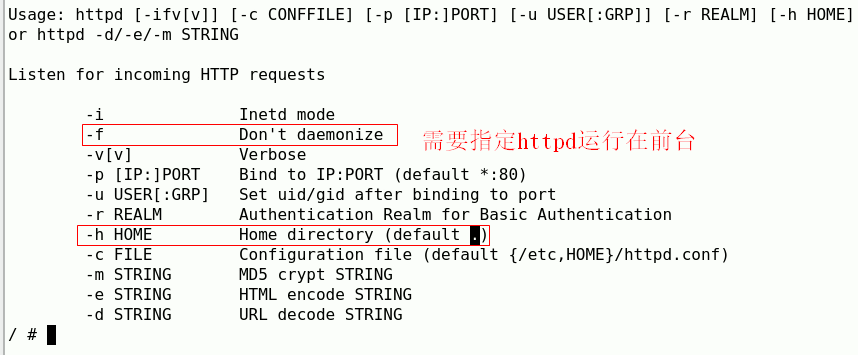
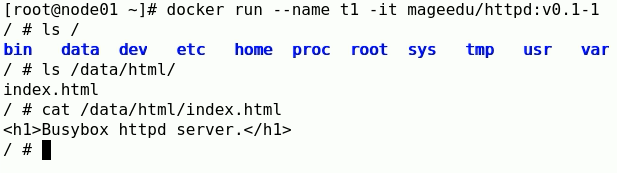
查看容器默认命令：docker inspect [Repository name] 看"Cmd" : ""



例如：busybox默认运行sh ;

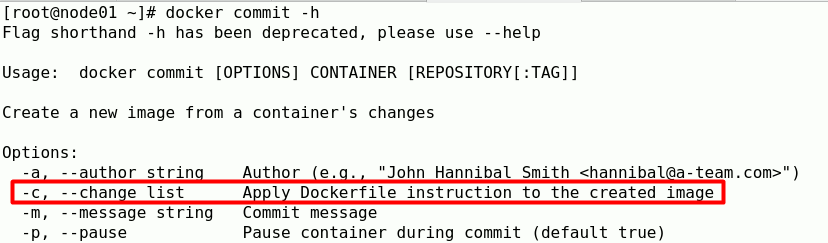
nginx默认运行"nginx -g daemon off"表示启动nginx进程并跑在前台

用存了个html网页的busybox容器打包一个镜像，用该镜像创一个容器并改变其默认命令，使其在启动时就用httpd命令打开网页



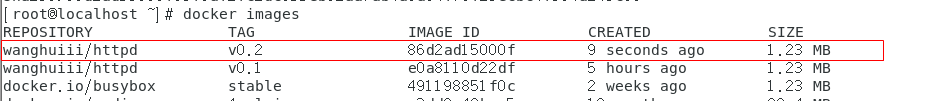
做法：

在使用docker commit打包镜像时修改

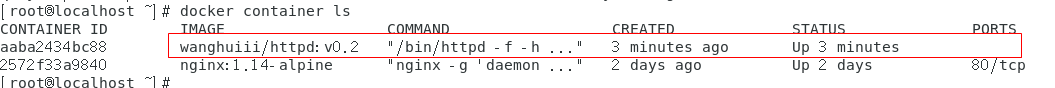




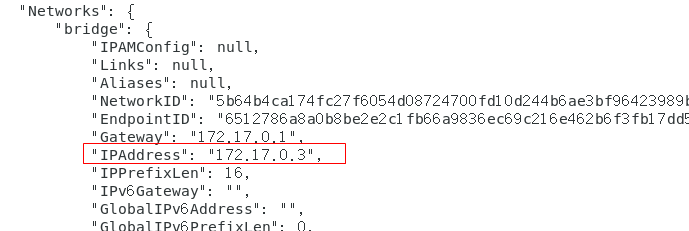
docker commit -c 'CMD ["/bin/httpd","-f","-h","/data/html"]' -p bb-httpd wanghuiii/httpd:v0.2





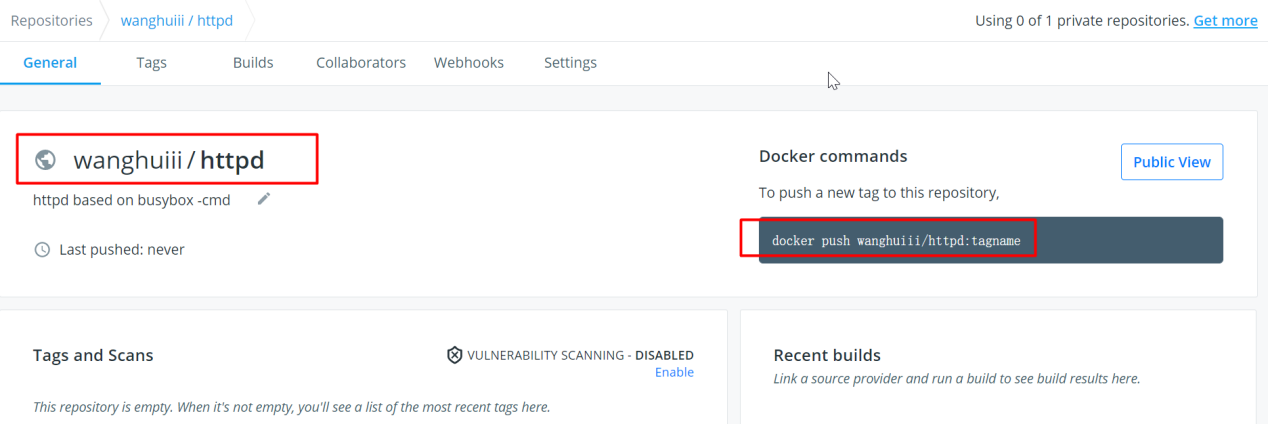


docker inspect cmd-httpd

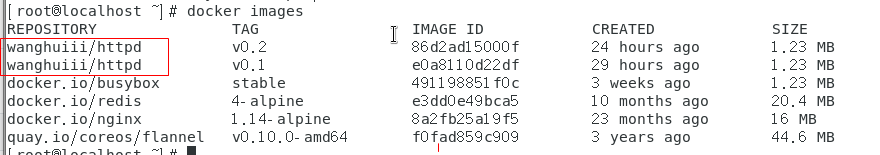




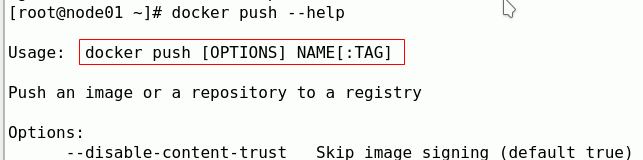
在dockerhub中创建镜像仓库



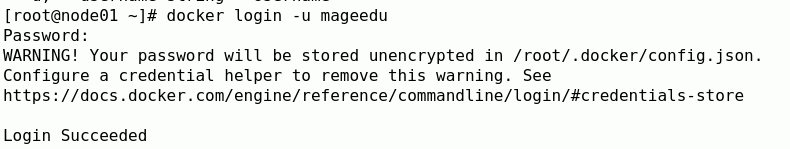
被推送的镜像Repository要和创建的仓库一样



用docker image push向docker hub推送镜像，



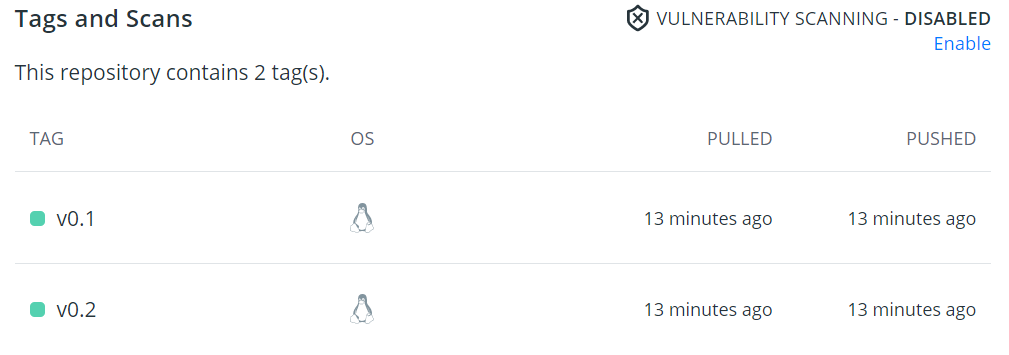
需要先登录







docker hub上就有了

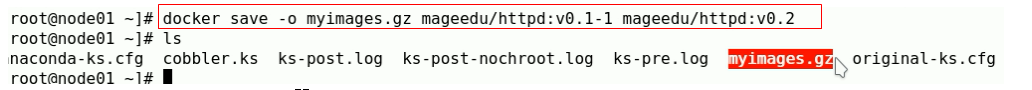


从其他主机上获取镜像的方法：

（1）docker pull 其他主机将镜像传到docker hub上

（2）

先在提供镜像的机器上docker save -o [打包压缩成文件] [要压缩的镜像们]



将压缩文件传到目标机器上



在目标机器上docker load -i [压缩文件名]





docker加载安装包：

docker load -i grafana-7.3.4.linux-amd64.tar

如果报错

open /var/lib/docker/tmp/docker-import-258307467/grafana-7.3.4/json: no such file or directory

则改为：

cat grafana-7.3.4.linux-amd64.tar.gz | docker import - grafana-7.3.4

