阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

马哥教育
IT人的高薪实战学院

4天实战

轻松玩转docker

实战总结一步到位

马哥教育官网：https://www.magedu.com

技术问题及课程答疑二维码

阿里云开发者“藏经阁”

海量电子手册免费下载

卷首语

1、docker 原理及在运维工作的地位和作用

运维工作进化论，docker、微服务、k8s 的联系，devops 和 docker 的关系，docker 的前世今生。

2、容器，镜像和仓库

容器和虚拟化，优势和劣势，底层核心；
容器除了 docker 还有什么其他选择？
docker 的安装及三大核心：容器、镜像、仓库。

3、docker 的实际运用

docker 的常用命令及注意事项，镜像的原理，dockerfile 的作用和应用，docker 的永久存储和网络通信。

4、docker 实际工作案例实现

搭建一台私有仓库，镜像仓库的上传和拉取，管理仓库，微服务的概念，用 docker 实现一个实际案例。

目录

docker 原理及在运维工作的地位和作用

演讲人：马哥教育

摘要：本文整理自 4 天 Docker 实战”负责人马哥教育，在“1024 创造营-的分享。本篇内容主要分为四个部分：

一、运维工作进化论：测试环境和生产环境
二、运维生产环境的发展
三、云原生技术栈的概念及技术
四、总结问答

视频链接：https://developer.aliyun.com/learning/course/892/detail/14271?spm=a2c6h.21258778.0.0.74b6db0d28oJyG

一、运维工作进化论：测试环境和生产环境

1、测试环境

1）LNMP 是一套技术组合

其中 L=Linux、N=Nginx、M=MySQL、P=PHP。

Nginx 服务不能处理动态请求。

静态请求：请求静态文件或者 html 页面，服务器上存在的 html 文件。

动态请求：动态页面上的内容存在于数据库中，根据用户发出的不同请求，其提供个性化的网页内容；大大降低网站维护的工作量。

当用户发起 http 请求，请求会被 Nginx 处理。如果是静态资源请求，Nginx 会直接返回。如果是动态请求，Nginx 通过 fastcgi 协议转交给后端的 PHP 程序处理。

2）一台服务器

3）更新用 ftp

FTP 有着极高的延时。从开始请求到第一次接收需求数据的时间非常长；并且会执行一些冗长的登录进程。

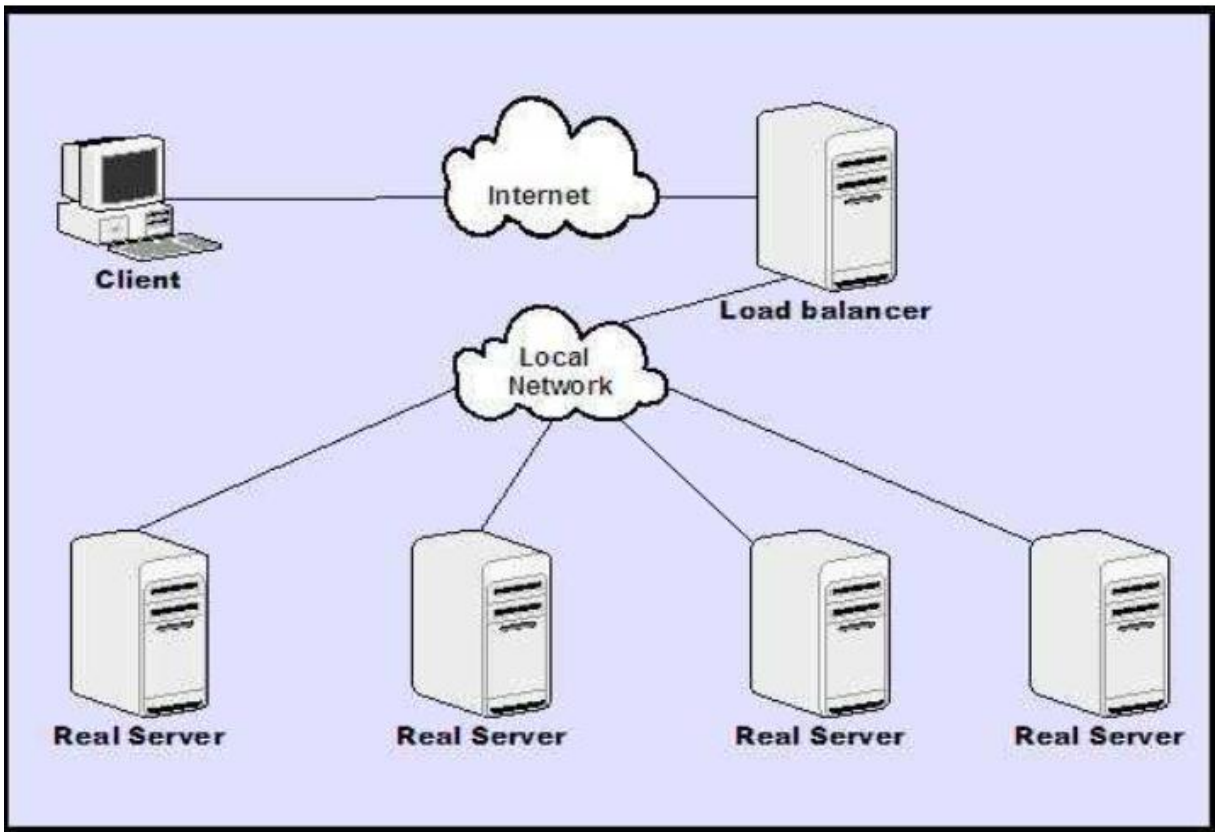
2、需要解决的问题

1）生产后有了一定程度的流量，需要运营和测试等部门给出数据，根据数据考虑冗余。

2）防止单点故障,保证业务稳定，做 HA 集群。

3）节约成本，使用公有云服务器，提前部署。

4）更新方式太过原始，测试环境使用 git。并搭建使用 gitlab 或 gogs。



3、原生产环境

1）多点集群单体应用。

多点集群服务一个单体应用。

2）公有云。

一种按使用量付费的模型，用户可以随时随地、便捷地、按需地从可配置的计算资源共享池中获取所需的计算资源（网络、服务器、存储、应用程序等服务）。

这些资源可以快速供给和释放，用户只需投入较少的管理工作。公有云：云计算运营商拥有超大规模基础设施，对外提供云服务。

3）git 更新

4、虚拟化和云计算

1）物理隔离

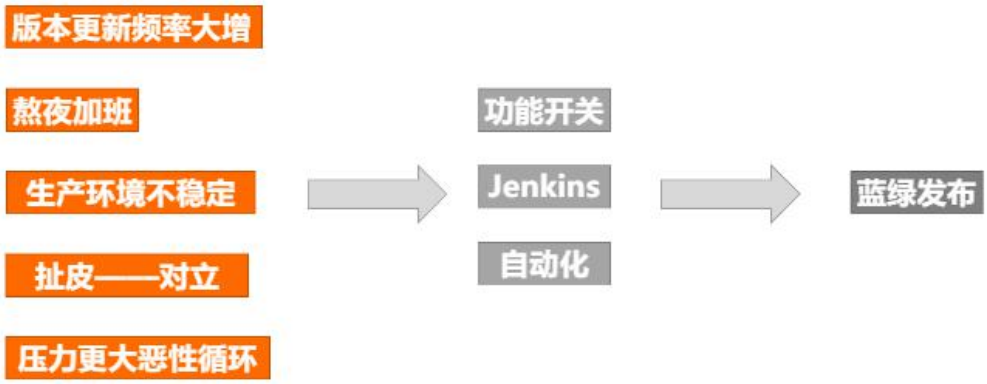
2005 年以前，大多数企业采用物理方法，将内网与外网隔离。从而避免入侵或信息泄露的技术手段。保证网络的保密性、安全性、完整性、防抵赖和高可用性。但其资源利用率极低，灵活性差，成本高。

2）虚拟化的技术

2008 年左右，企业开始使用虚拟化技术，通过硬件和软件，实现物理架构资源的重新整合利用。可以用一台物理机，通过虚拟化，划分为多套系统，在系统之内进行多方位隔离，隔离之后相当于多台服务器。大幅降低了 IT 硬件成本，减少资源浪费，提升了系统的稳定性和安全性。

3）云计算（概念）

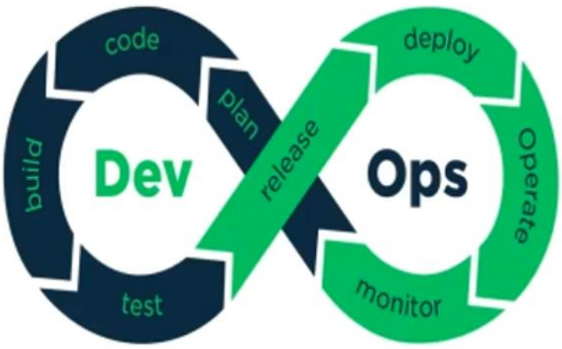
2018 年至今，大部分企业开始使用云计算技术。分布式计算技术透过网络将庞大的计算处理程序，自动分拆成无数个子程序。然后，交由多部服务器组成的系统进行搜寻、计算、分析之后，将处理结果回传给用户。



随着运维人员的版本更新频率增加，运维团队熬夜加班，各部门协调不稳定，导致工作人员压力更大，形成了恶性循环。

为了解决上述问题，开发人员做了功能开关。在独立的分支上开发新功能，全部开发测试完成之后，合并到主干。从而减少运维人员的工作压力。与此同时，运维人员通过脚本自动化和 Jenkins 程序，建立了持续集成和持续交付项目。

通过蓝绿发布的方式发布应用，减少发布过程中，服务停止的时间，从而进一步减少工作时间。



生产环境：

多点负载均衡单体应用。

混合云。

devops。

二、运维生产环境的发展

1、生产环境

1）多点负载均衡单体应用。

2）混合云。

提供既在公共空间又在私有空间中的服务。混合云把公用云模式与专用云模式结合在一起。混合云有助于提供所需的、外部供应的扩展。

3）devops。

原始的互联网公司工作模式是瀑布流，但用户越多，需求越大，公司的管理，人力成本都是问题。而如果更新间隔太慢，一定会导致用户的满意程度下滑。DevOps 的观念应运而生。所以 DevOps 是一个必然趋势，是一种方法，也是一种观念。

DevOps 打破开发人员和运维人员的壁垒，根据需求情况，把需求拆分成多个小需求，小步快跑大幅增加需求完成的频率。运用自动化和 CI/CD 的概念，运用工具，实现稳定、快速的版本更新上线。对运维人员的技术要求和经验大大提升。

问题1：资源利用率。

***一个虚拟机跑多个项目。治理、迁移、容灾?**

问题2：扩容不及时。

***脚本化启动50台虚拟机要多久？脚本化启动50台容器要多久？环境部署要多久？风险大不大?**

问题3：环境不一致导致的问题。

当一个虚拟机跑多个项目。服务器的治理，环境的迁移容灾会产生很多问题。比如资源利用率低，扩容不及时，环境不一致等。

生产环境：

多点负载均衡单体应用。

docker。

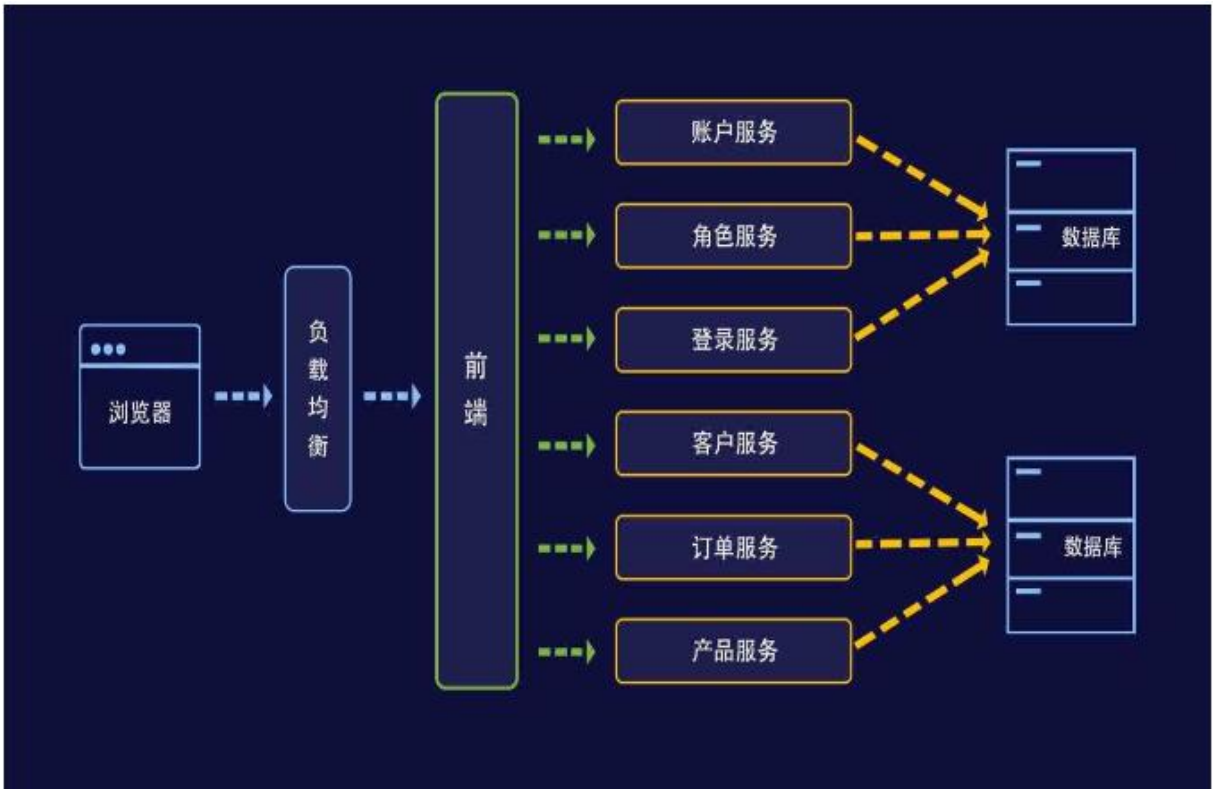
devops。

2、新型生产环境

生产环境主要分为：多点负载均衡单体应用，docker 和 devops。

Docker 于 2013 年初开源，基于 Linux 内核的 cgroup，namespace，以及 AUFS 类的 Union FS 等技术，对进程进行封装隔离，属于操作系统层面的虚拟化技术。容器就是一个技术类型，而 docker 是当下最主流的一种实现容器的方案。

其他方案包括：LXC，Mesos，RKT 等等。docker 或者容器和传统虚拟化最大的区别，就是虚拟化的封装是系统级的封装，docker 或者其他容器是进程级的封装。



3、微服务就是将前端拆成各个模块，然后连接到服务库。微服务需要跑多个容器，容器多又会涉及到通信、架构、伸缩、更新、监控等等问题。

如上图所示，游戏“王者荣耀”就是微服务架构，它主要有英雄技能，地图坐标，用户账号等多个模块，然后将其统一连接到数据库。

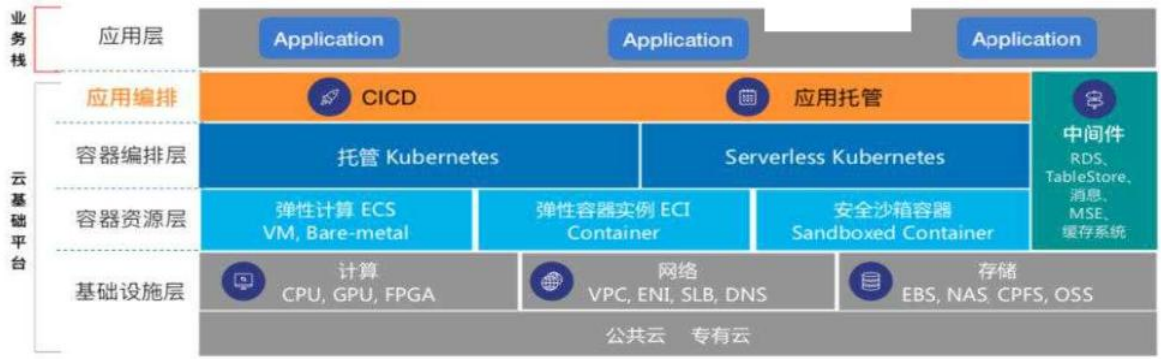
4、k8s－容器编排管理工具

K8s 的自愈功能是指重新启动失败的容器，在节点不可用时，替换和重新调度节点上的容器。

K8s 的弹性伸缩功能，通过监控容器的 cpu 的负载值，如果这个平均高于 80%，增加容器的数量，如果这个平均低于 10%，减少容器的数量。

K8s 的服务的自动发现和负载均衡，不需要修改应用程序来使用不熟悉的服务发现机制。Kubernetes 为容器提供了自己的 IP 地址和一组容器的单个 DNS 名称，并可以在它们之间进行负载均衡。

K8s 的滚动升级和一键回滚，Kubernetes 逐渐部署对应用程序或其配置的更改，同时监视应用程序运行状况，以确保它不会同时终止所有实例。 如果出现问题，Kubernetes 会恢复更改，利用日益增长的部署解决方案的生态系统。



三、云原生技术栈的概念及技术

1、云原生技术栈

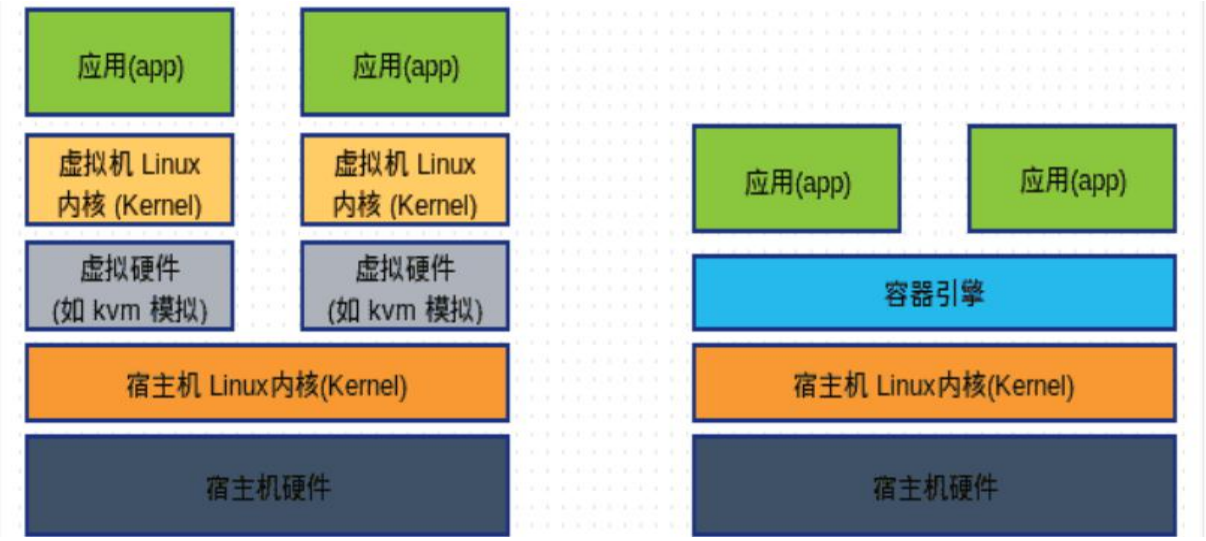
云原生是一个生态概念、是一线互联网公司发展到某个极端的必然选择。主要包含三大要素，即容器及编排管理、DevOps 和微服务。其主要技术模块有应用定义及部署，编排与管理，运行环境，配置等。

在云原生时代，云原生、容器、devops 等一定是未来若干年的发展方向。容器不但解决了大型架构的发展瓶颈，而且取代了传统运维。docker 是云原生时代的基石和应用基础。学习 docker 需要一定程度的经验积累，所以现在企业对运维的要求越来越高。

2、容器

镜像和容器的关系，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。容器的实质是进程，容器进程运行于属于独立的命名空间。

容器存储层的生存周期和容器一样，当容器消亡时，容器存储层也随之消亡。因此，任何保存于容器存储层的信息都会随容器删除而丢失。



3、容器和虚拟化

虚拟机是一个主机模拟出多个主机，需要先拥有独立的系统。传统虚拟机，利用 hypervisor，模拟出独立的硬件和系统，在此之上创建应用。

docker 是在主机系统中建立多个应用及配套环境，把应用及配套环境独立打包成一个单位，是进程级的隔离。

表 1-1 Docker 容器技术与传统虚拟机技术的比较

特 性	容 器	虚 拟 机
启动速度	秒级	分钟级
性能	接近原生	较弱
内存代价	很小	较多
硬盘使用	一般为 MB	一般为 GB
运行密度	单机支持上千个容器	一般几十个
隔离性	安全隔离	完全隔离
迁移性	优秀	一般

如上图所示，容器的启动速度在秒级，虚拟机的启动速度在分钟级。容器的性能接近原生，而虚拟机的性能较弱。容器需要的内存较小，虚拟机需要的内存较多。总体来看，容器的整体性能要优于虚拟机。

容器和虚拟化

- 再次提高服务器资源利用率
- 重量更轻，体积更小。
- 匹配微服务的需求
- 保持多环境运行的一致性
- 快速部署迁移，容错高。

- 安全性较差
- 多容器管理存在难度。
- 稳定性较差
- 排错难度较大

容器相比虚拟化的优势在于，可以再次提高服务器的资源利用率，重量更轻，体积更小，能够匹配为服务的需求，保持多环境运行的一致性，快速部署迁移，且容错率高。

其劣势在于安全性相对较差，多容器管理有一定的难度，稳定性较差，排错难度较大。

4、Docker 底层技术主要包括 Namespaces，Cgroups 和 rootfs。

Namespace 的作用是访问隔离，Linux Namespaces 机制提供一种资源隔离方案。每个 Namespace 下的资源，对于其他 Namespace 下的资源都是不可见的。

Cgroup 主要用来资源控制，CPU\MEM\宽带等。提供的一种可以限制、记录、隔离进程组所使用的物理资源机制，实现进程资源控制。

rootfs 的作用是文件系统隔离。

四、总结问答

镜像可以看做是一个压缩包，里面包含所需要的应用和应用要的配置文件及底层的库、参数、环境变量等。容器是运行这个压缩包，实现具体功能的存在。

1、我们运行应用，庭层总是少不了系统环境的，而每个镜像都番要用，这样不会占用大量存储吗？

答：不会，docker 可以将一个基础系统锐像可以披多个锐像共用。这里可以代入调用和级存的概念。保证每个容器体积小，速度快，性能忧。

2、我们通过镜像后动容器去使用，更新文件是必然的事情，那镜像岂不是会被多个容器更改，造成冲突？

答：像采用了分层设计，启动容器后，镜像永远是只读属性。只不过在最上层加一层读写层（容器层），如果要对底层镜像的文件进行更改，读写层会复制一份镜像中的只读层进行写操作，这就是 Copy-On-Write。

3、我们想要在一个镜像（Centos7）上创建一个新镜像（Centos7+nginx），新镜像会复制层镜像吗？

答：不会，参考问题 1。

容器、镜像和仓库

演讲人：马哥教育

摘要：本文整理自 4 天 Docker 实战”负责人马哥教育，在“1024 创造营-的分享。本篇内容主要分为三个部分：

一、Docker 底层技术概述
二、Docker 版本及安装
三、Docker 的常用命令

视频链接：https://developer.aliyun.com/learning/course/892

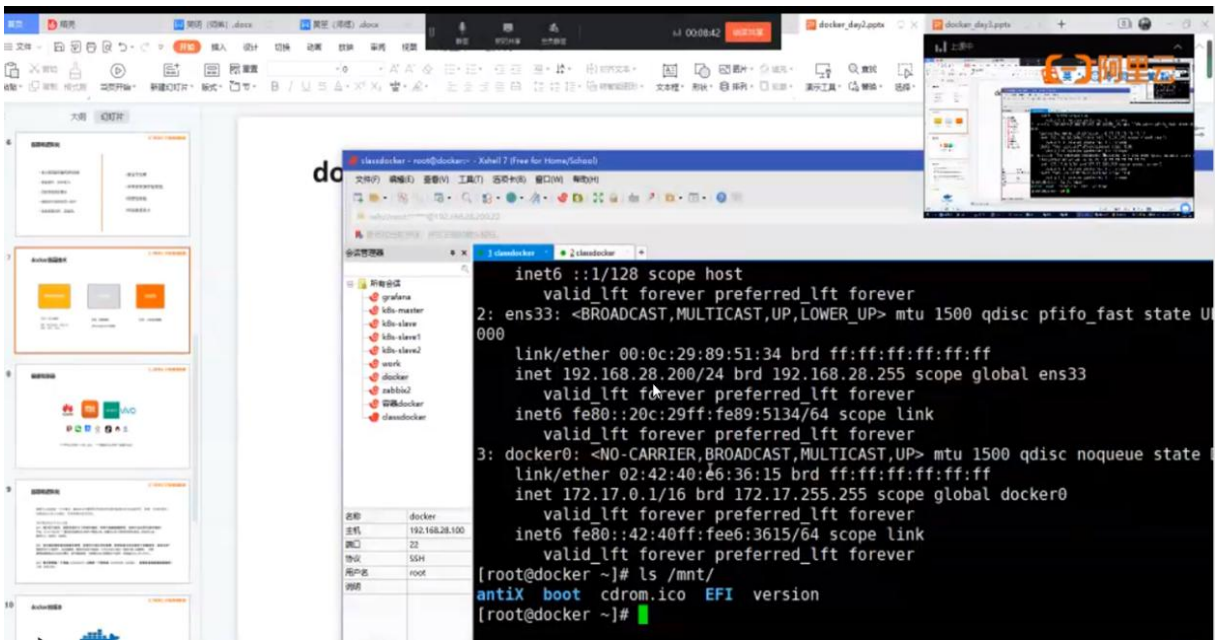
一、Docker 底层技术概述

Docker 底层技术主要包括 Namespaces，Cgroups 和 rootfs。

Namespace 的作用是访问隔离，Linux Namespaces 机制提供一种资源隔离方案。每个 Namespace 下的资源，对于其他 Namespace 下的资源都是不可见的。

Cgroup 主要用来资源控制，CPU\MEM\宽带等。提供的一种可以限制、记录、隔离进程组所使用的物理资源机制，实现进程资源控制。

rootfs 的作用是文件系统隔离。



接下来，访问隔离案例。首先，挂载镜像文件 iso，输入下列代码：

[root@docker ~] mount -t iso9660 zechen.iso /mnt
[root@docker ~] ls /mnt

然后，管理服务器，创建命名空间输入：[root@docker ~] unshare -m /bin/bash

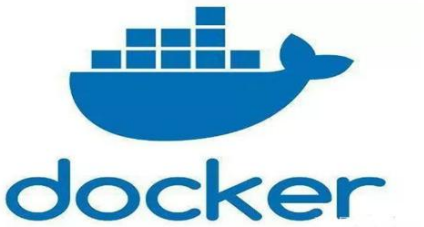
重新挂载输入：[root@docker ~] mount -t iso9660 -0 loop zechen.iso /mnt

重新查看输入：[root@docker ~] ls /mnt

通过这个案例，让大家感受命名空间的大致内容，理解隔离的作用，且每个容器都是隔离的。

阿里云 开发者训练营

docker的版本



docker官方提供2种版本，一个是docker企业版docker-EE，另外一个则是社区版docker-ce，
我们在学习或者测试环境使用docker-ce版本即可。

发布版本docker更新很快，现在最新的是：

二、Docker 版本及安装

docker 官方提供 2 种版本，即 docker 企业版 docker-EE 和社区版 docker-ce。在学习或测试环境时，使用 docker-ce 版本即可。进入官网就可以查看 docker 的版本发行时间和版本号。

阿里云 开发者训练营

docker的安装

docker常见的有3种安装方式，yum、rpm包、脚本。
我们采用相对简单但对各种环境比较友好的方式：（关防火墙和selinux）

```
#安装存储库拓展包
yum install -y yum-utils

#设定存储库
yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

#安装三个包
yum -y install docker-ce docker-ce-cli containerd.io

#验证
docker version   #查询docker版本信息
docker info #查询docker详细信息
docker run hello-world #运行第一个容器：helloworld
```

Docker 有 3 种安装方式：yum、rpm 包、脚本。我们采用相对简单但对各种环境比较友好的方式。

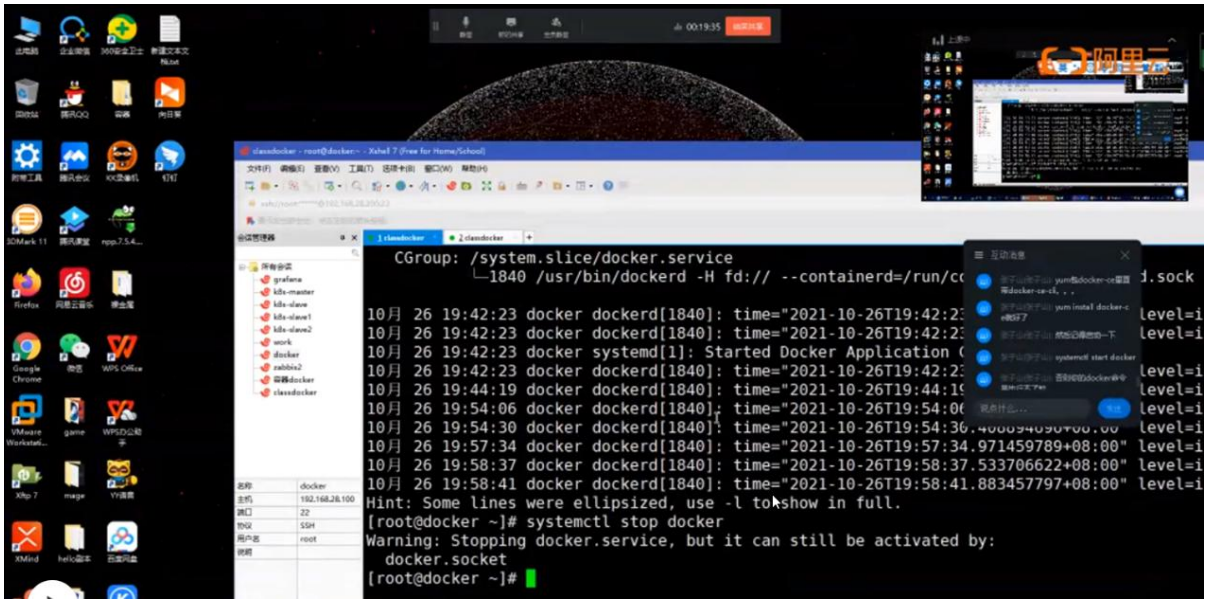
第一步，关闭防火墙和 selinux。检查 firewood，确定环境的镜像系统是马哥教育提供的。

第二步，安装存储库拓展包 yum install-y yum-utils。

第三步，设定存储库。

第四步，安装 yum-y install docker-ce，docker-ce-cli 和 containerd.io。其中，docker-ce 是 docker 的程序包；docker-ce-cli 是 docker 的工具包；containerd.io 是系统和 docker 的 api 的守护进程。

第五步，验证 docker version。查询 docker 版本信息如果有正确的打印或者正确的输出，说明安装成功。



docker 安装与启动的代码如下：

```
yum install -y epel-release
yum install docker-io        #安装 docker
 /etc/sysconfig/docker       #配置文件
chkconfig docker on          #加入开机启动
service docker start         #启动 docker 服务
```

#查看基本信息

docker version　　#查看 docker 的版本号，包括客户端、服务端等。

docker info　　#查看系统(docker)层面信息，包括 images，containers 等。

docker pull centos　　#下载

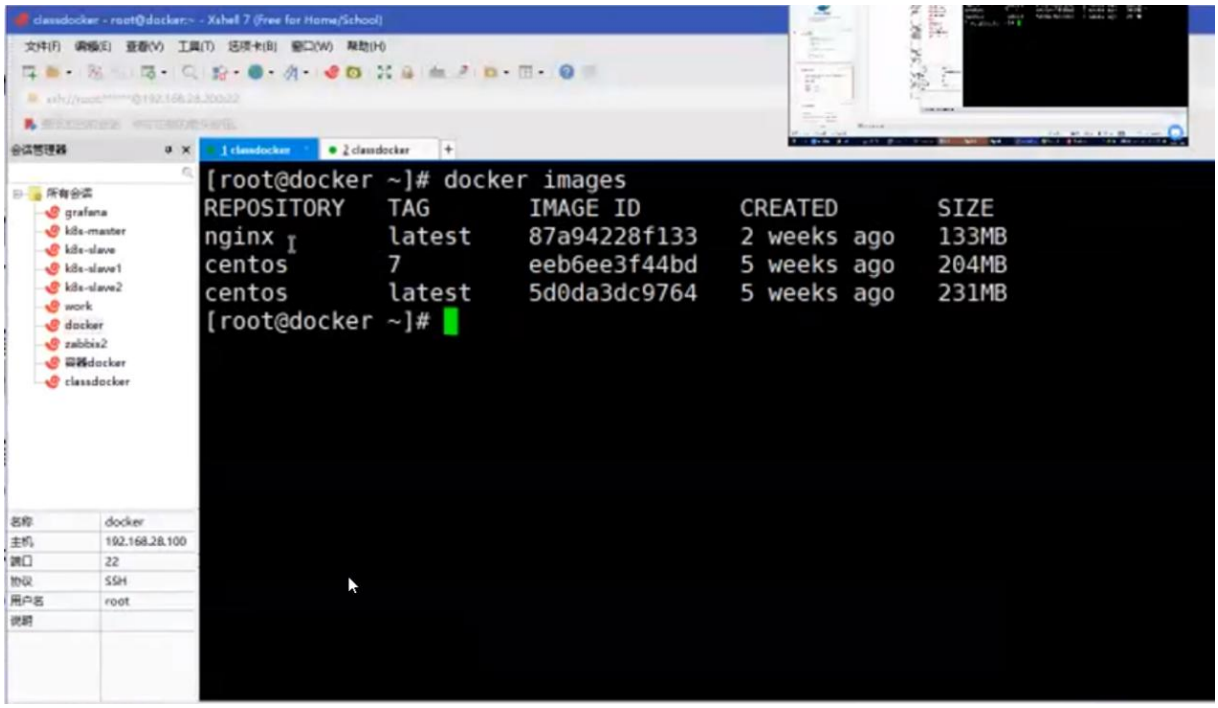
docker images [centos]　　#查看

docker run -i t centos /bin/bash

三、Docker 的常用命令

启动的 docker，镜像和容器分别是什么。可以理解，镜像就像 u 盘。u 盘可以在很多电脑上使用。

镜像相当于 u 盘；电脑相当于容器。镜像的来源一般有两种。即官方镜像仓库和企业级镜像仓库。这一章主要讲官方镜像仓库。



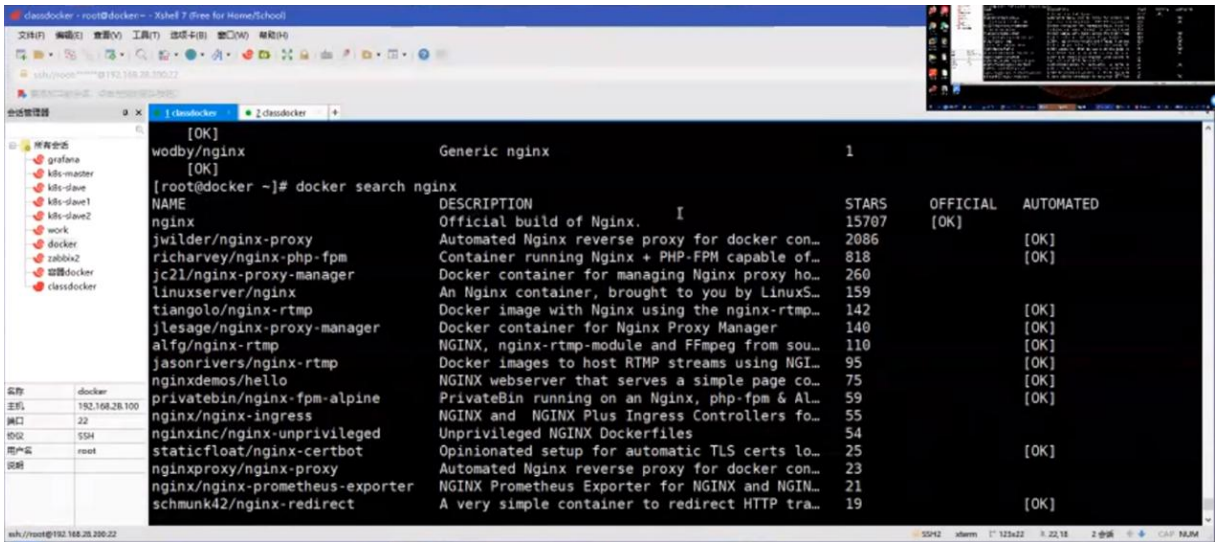
接下来，寻找官方镜像仓库的镜像并下载。需要的代码如下：

```
docker images      #查询本地镜像
docker search[images]      #查找镜像
docker pull[images:tag]      #下载镜像

# 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
```



如上图所示，官方镜像仓库一共有五列。其中，第一列是名字。筛选之后，注意显示 offical 的镜像，就是官方镜像。

首先，查看本地已有镜像。然后，查找需要的镜像，并找到官方版本的镜像。最后，进行下载即可。

阿里云 开发者训练营

docker常用命令

利用镜像直接创建容器：
docker run +参数 [images:tag] 启动命令

#查看现有容器
docker ps -a (加了该选项可以查出未启动的容器)

#指定容器名字
docker run --name [name] [image:tag]

#利用镜像直接创建容器
docker run -d --name nginx_1 nginx:latest

#镜像用可交互的方式创建容器
docker run -itd --name nginx_1 nginx:latest

#创建容器并暴露端口
docker run -itd -p 8800:80 --name nginx_1 nginx:latest

#进入容器
docker exec -it [容器ID] 命令

#如何启动和停止容器
docker start/stop [容器ID]

#如何删除已停止的容器
docker rm [容器ID]

#如何删除所有已停止的容器
docker rm `docker ps -aq`

容器的使用及命令如下：

使用镜像创建容器

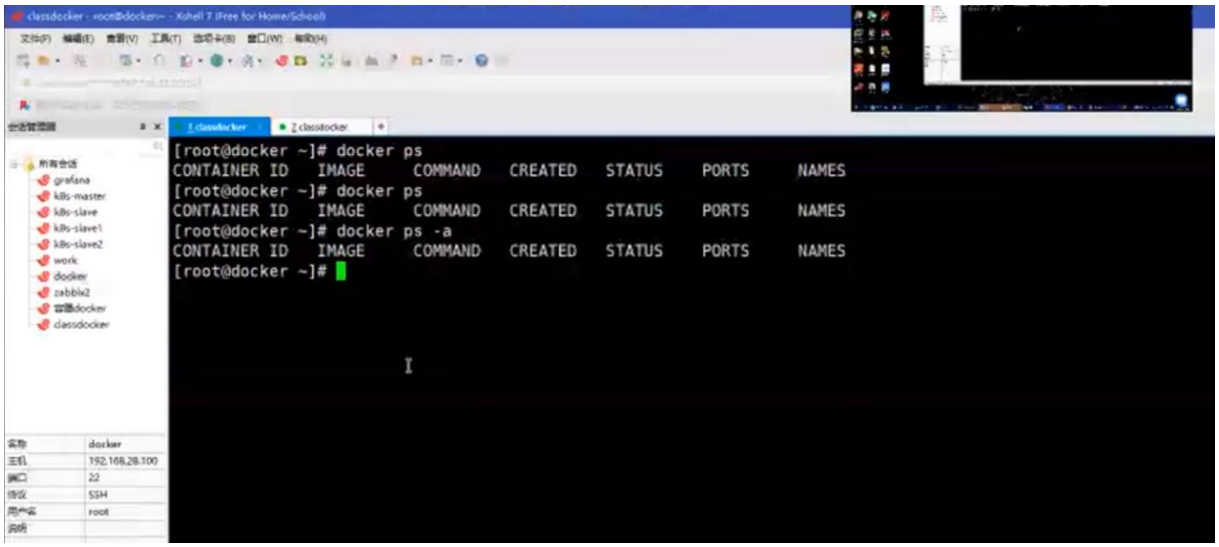
docker run -i -t sauloal/ubuntu14.04

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

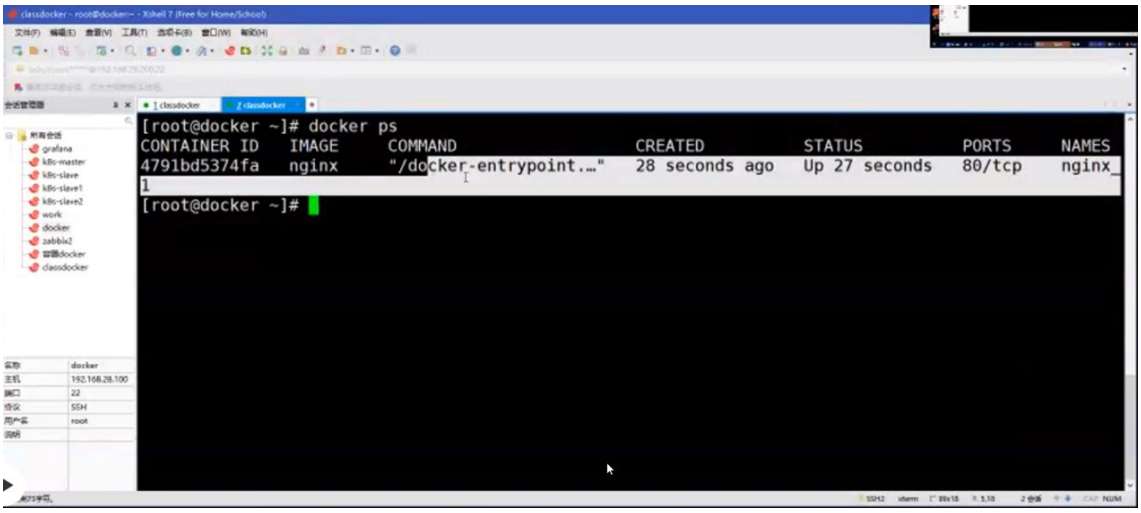
docker start/stop/restart <container> #：开启/停止/重启 container

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

#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 -aq` #：删除所有的 container

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

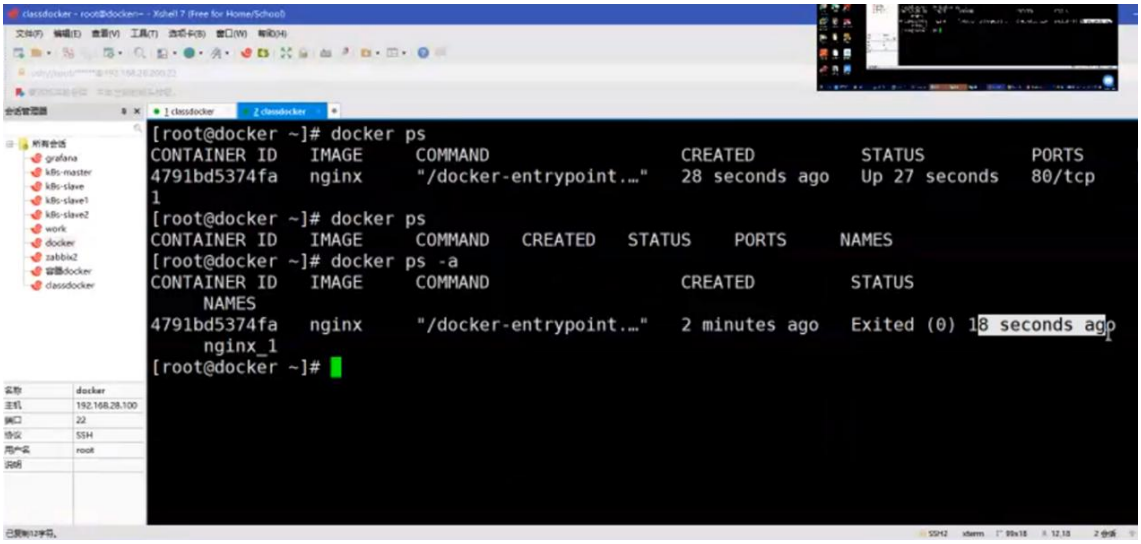
创建容器基础的命令格式。

利用镜像直接创建容器:

docker run+参数[images:tag] 启动命令

#查看现有容器

docker ps -a(加了该选项可以查出未启动的容器)



#指定容器名字

docker run --name[name][image:tag]

#利用镜像直接创建容器

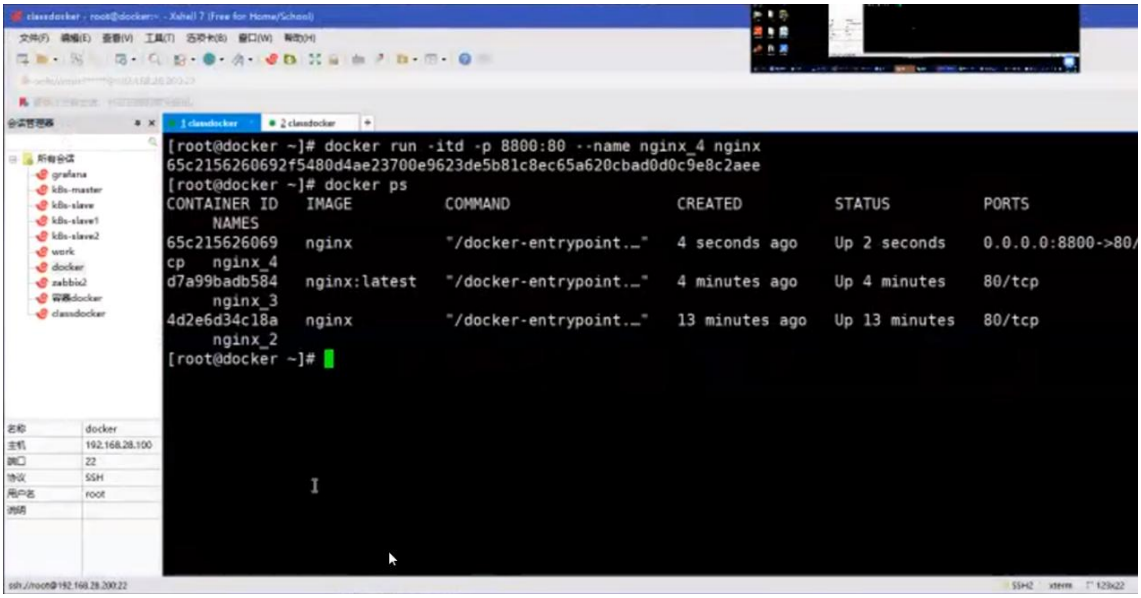
docker run -d --name nginx_1 nginx:latest

#镜像用可交互的方式创建容器

docker run-id--name nginx_1 nginx:latestt 以交互模式运行容量 t 为容器重新分配一个为输入终端

#创建容器并暴露端口

docker run-itd-p 8800:80--name nginx_1 nginx:latest



如果有用户想要访问虚拟机或者宿主机的容器，只需要访问 8800 端口。通过内网的形式，找到容器里的 8800 端进行访问。

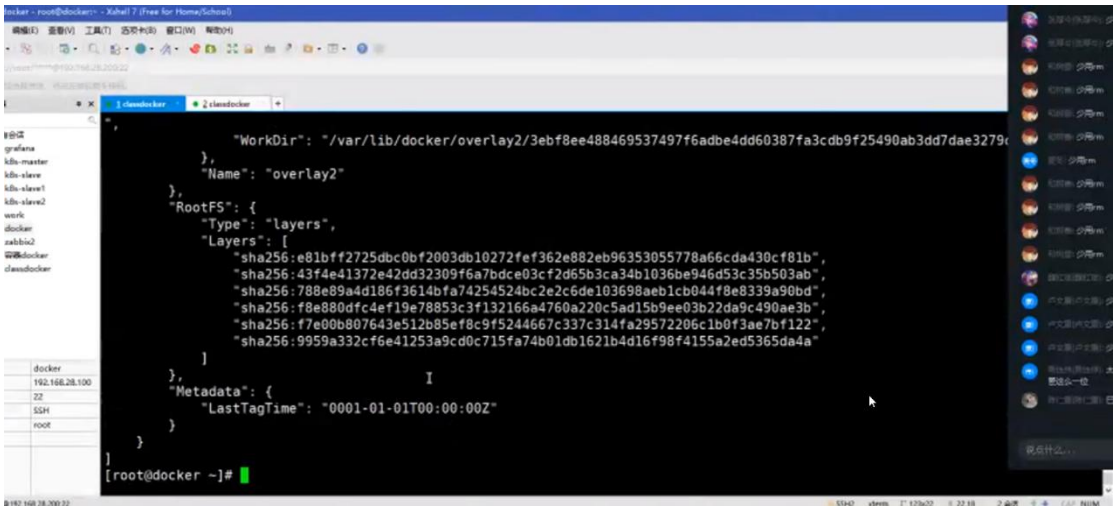
接下来，对容器进行验证。首先进入容器，输入 docker exec -it[容器 ID]命令。其中，-it 不是一条命令，是两个选项。

-it 是以交互模式运行容器，交互模式运行容器。-t 为容器，重新分配一个伪终端输入。

分配一个伪终端，并且以交互的模式运行。如果-d 启动不成功，尝试用-itd 启动。启动和停止容器时，输入：docker start/stop [容器 ID]

docker 的常用基础命令如下：

#查看镜像或容器的详细信息:
docker inspect[容器 ID/镜像名:tag]
#给镜像添加一一个软链接并改名和标签:
docker tag[oldname:tag]
[newname:tag]



#删除镜像:

docker rmi 镜像:tag/镜像 ID

#容器和宿主机之间文件复制

docker cp [文件目录容器 ID:内部路径]

docker cp [容器 ID:内部路径文件目录]

docker 的实际运用

演讲人：马哥教育

摘要：本文整理自“4 天 Docker 实战”，在“1024 创造营”中的分享。本篇内容主要分为四个部分：

一、docker 基础命令（下）
二、dockerfile
三、docker 是否可以被替代?
四、docker 的前世今生

视频链接：https://developer.aliyun.com/learning/course/892

一、docker 基础命令（下）

这是 docker 基础命令下半部分的学习。

从天池基础镜像中获取

pythondocker pull 天池镜像：标签；

docker 构建镜像

docker?build?-t?registry.cn-shanghai.aliyuncs.com/target:myversion

docker 构建镜像并指定 DockerFile

docker?build?-f?./dockerfile?-t?registry.cn-shanghai.aliyuncs.com/target:myversion

设置默认工作目录

add . /目录名称

镜像启动后-----执行

sh run.sh,我们目前使用 cmd [“sh”,“run.sh”]

查看镜像或容器的详细信息:

docker inspect [容器 ID/镜像名:tag]

给镜像添加一个软链接并改名和标签:

docker tag [oldname:tag]

[newname:tag]

删除镜像:

docker rmi 镜像:tag/镜像 ID

容器和宿主机之间文件复制

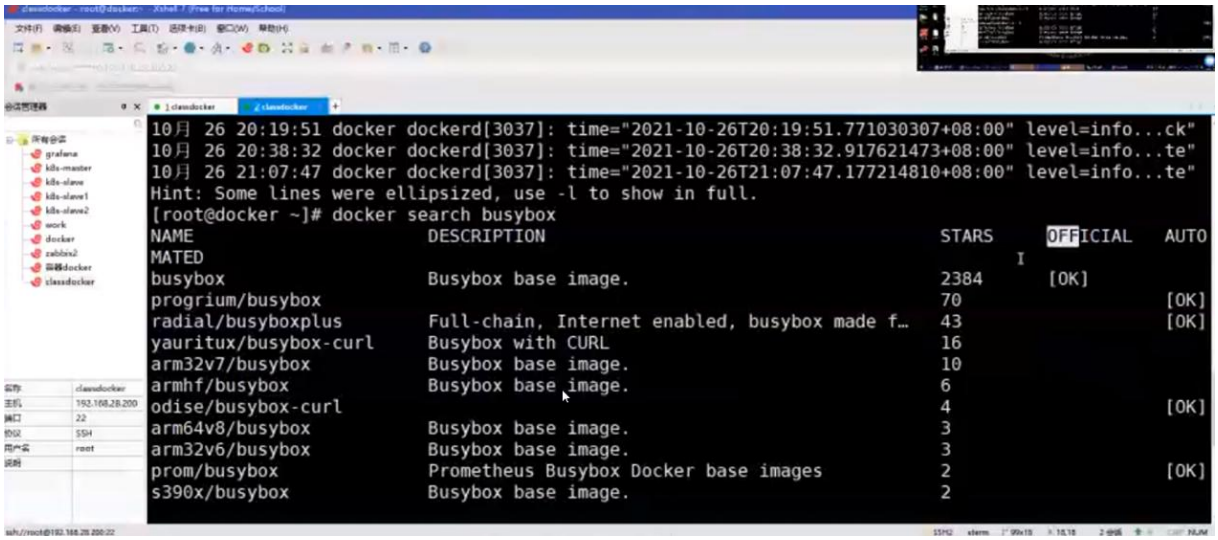
docker cp [文件目录 容器 ID:内部路径]

docker cp [容器 ID:内部路径 文件目录]

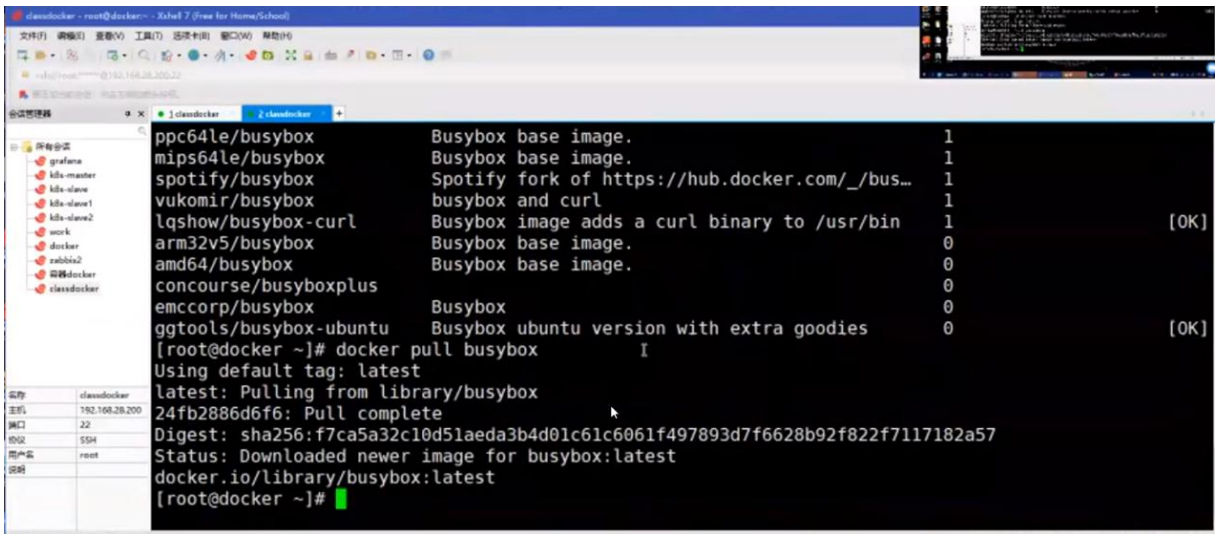
二、dockerfile

1、案例一

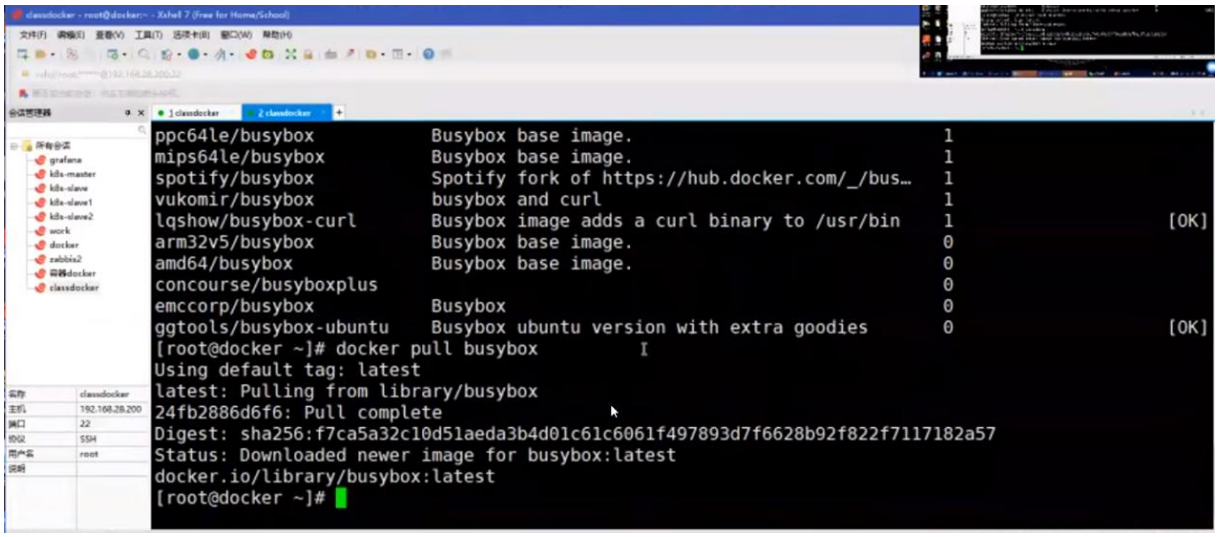
1）搜索 busybox 的镜像，并确保为官方镜像。



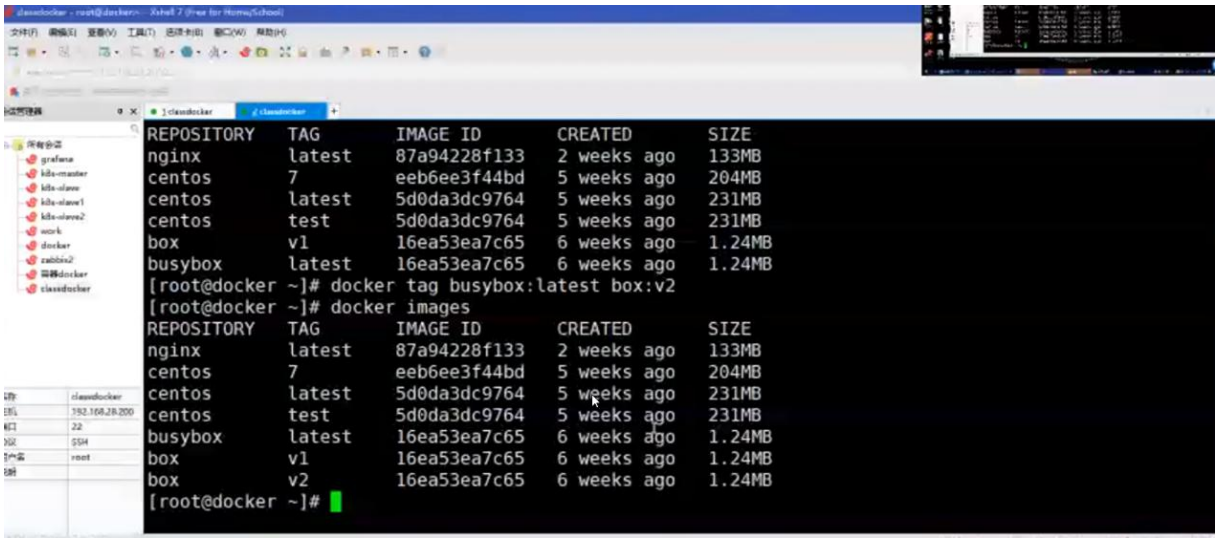
2）docker 下载该镜像 busybox 的镜像。



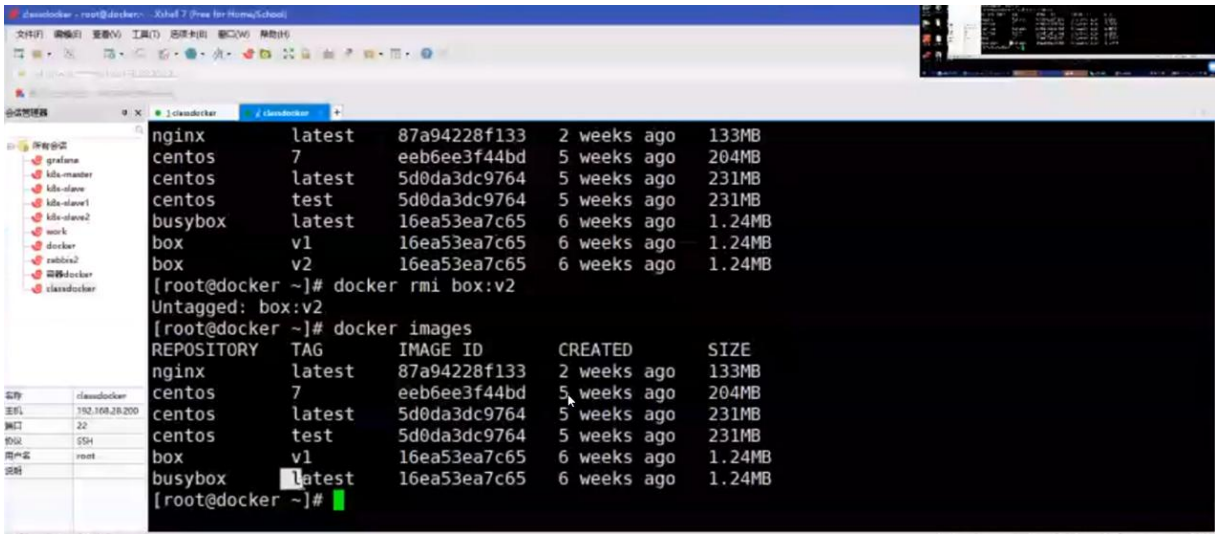
3）给镜像创建一个软链接并改名 box:v1。



4）再创建一个软链接改名为 box:v2。



5）删除 box:v2 镜像。



操作流程

1）启动虚拟机，检查防火墙和 Slinux 是否关闭

[root@docker ~] systemctl status firewalld

[root@docker ~] getenforce 0

2）检查 docker 是否启动

[root@docker ~] systemctl status docker

3）查看 busybox 镜像

[root@docker ~] docker search busybox

4）下载 busybox 镜像并检查是否成功

[root@docker ~] docker pull busybox

[root@docker ~] docker images

5）给镜像创建一个软链接并改名 box:v1 并检查是否成功

[root@docker ~] docker images

[root@docker ~] docker tag busybox:latest box:v1

查看 ImageID,如果一样则软连接成功

6）再创建一个软链接改名为 box:v2。

[root@docker ~] docker tag busybox:latest box:v2

(7)删除 box:v2 镜像

[root@docker ~] docker rmi box:v2

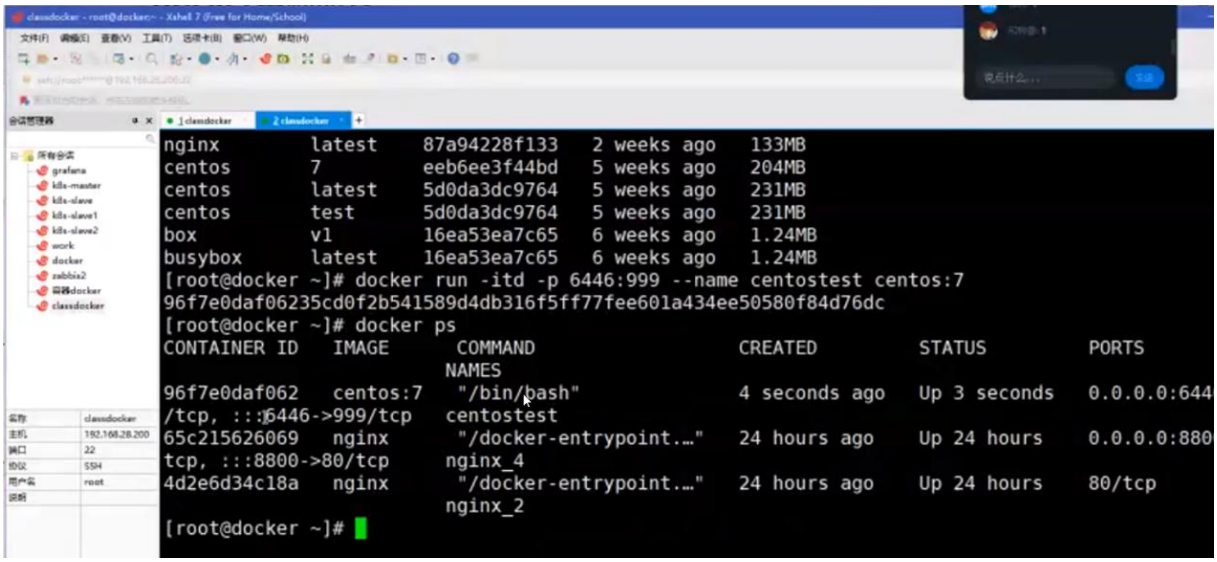
至此，第一个案例就此完成。

2、案例二

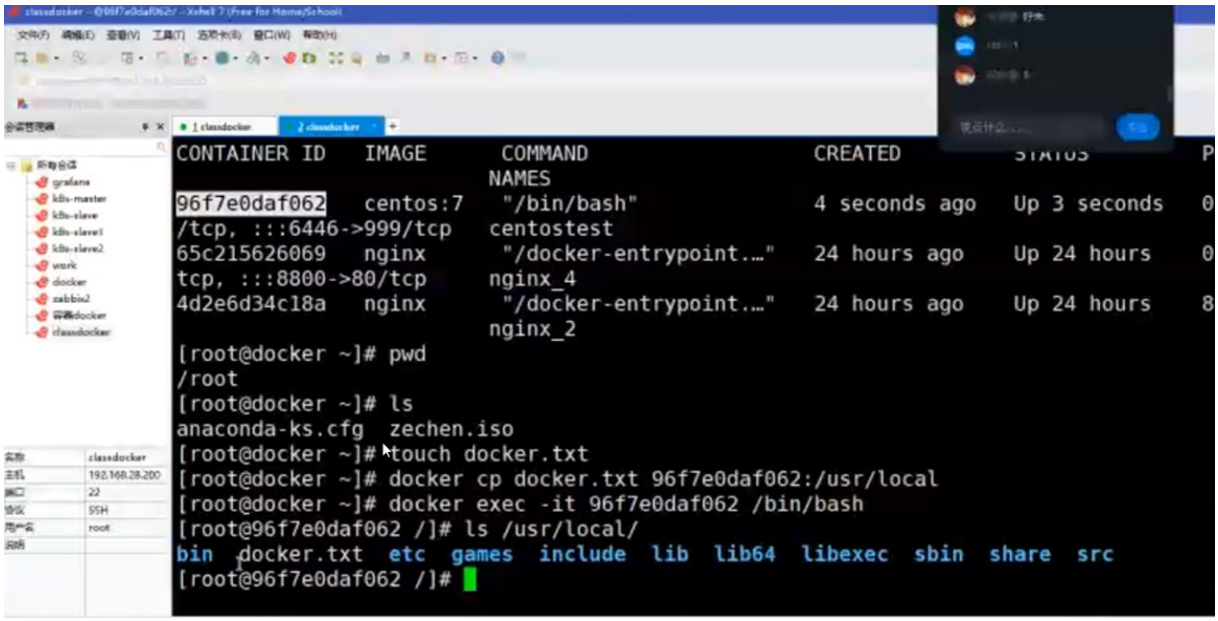
[root@docker ~]

(没有文件自行创建)

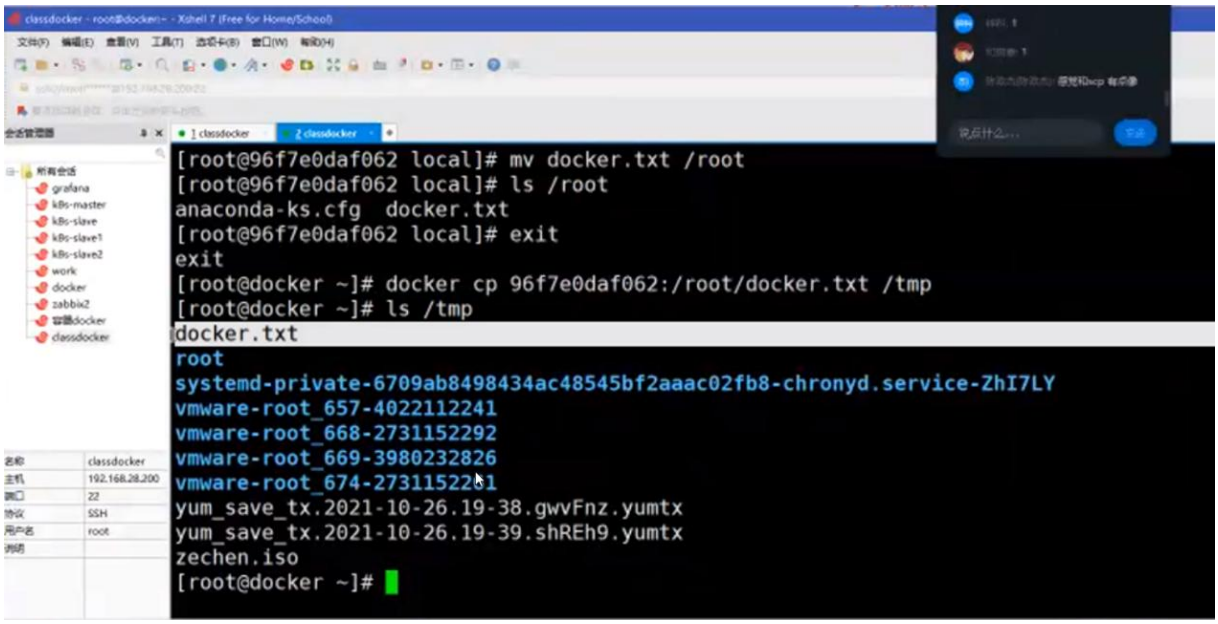
1）下载 centos:7 镜像。并用该镜像启动容器，命名为 centostest,暴露 6446 宿主机端口，映射容器内 999 端口。



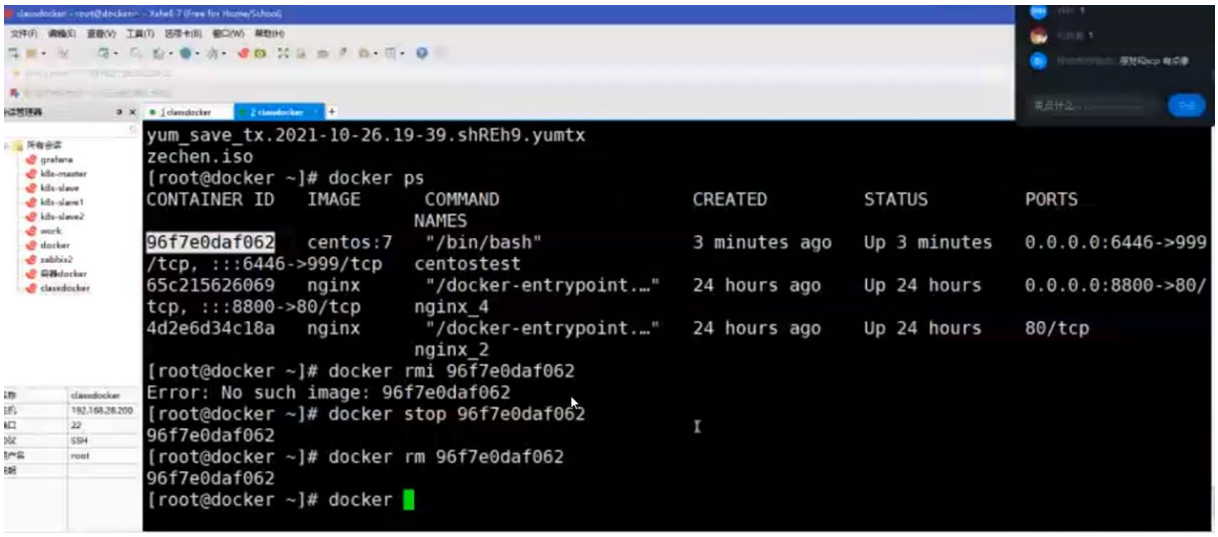
2）将宿主机/root/docker.txt 复制到容器内/user/local/。



3）再从该容器内的/root/docker.txt 文件复制到宿主机/tmp。



4）删除该容器。



操作流程

1）用该镜像启动容器，命名为 centostest,暴露 6446 宿主机端口，映射容器内 999 端口

[root@docker ~] docker run -itd -p 6449:999 --name centostest centos:7

参数解释：

-itd:固定写法

-p:暴露的端口号

--name:修改名字

3）查看容器是否启动成功

[root@docker ~] docker ps

4）将宿主机/root/magegood.txt 复制到容器内/usr/local/

[root@docker ~] touch docker.txt

[root@docker ~] docker cp docker.txt 96f7e0daf062:/usr/local

验证

[root@docker ~] docker exec -it 96f7e0daf062 /bin/bash

[root@docker ~] ls /usr/local

5）再从该容器内的/root/docker.txt 文件复制到宿主机/tmp

[root@docker tmp] ls /tmp

[root@docker tmp] docker cp 96f7e0daf062 :/root/docker.txt /tmp

6）删除该容器并检查

删除前需要停掉

[root@docker ~] docker stop 96f7e0daf062

[root@docker ~] docker rmi 96f7e0daf062

[root@docker ~] docker ps

注意：使用 search 搜索 centos:7 是找不到官方版本的

[root@docker ~] docker search centos:7

[root@docker ~] docker search centos tag 即可搜索官方的

三、dockerfile 是否可以被替代?

1、概述

dockerfile 可以理解为一个制作镜像的脚本，但远没有脚本复杂。他根据某种格式自定义内容，就可以快速创建出需求的镜像。

docker 容器启动的时候在最上层挂载了一个可写层，比如说我在容器里面创建一个文件，这个文件是存放在可写层的，这时候容器要是销毁了，那么我们对容器的一些写入操作也就没了，我这个文件也会随着容器销毁而销毁了，我们要是想要保存我们对容器的一些写入操作的话，可以使用 commit 命令然后将容器制作成一个镜像，这样下次 run 起来该镜像的时候，我们之前的写入操作就还存在了。

除了使用 commit 方式制作镜像，还有一种方式就是编写 dockerfile ，然后使用 build 命令来制作镜像了。

2、dockerfile 的规则

格式：
是注释
指令建议要大写，内容小写。

执行顺序：
docker 是按照 dockerfile 指令顺序依次执行的，也就是说从上到下。

3、指令

1）FROM：底层镜像（如系统）

这个 FROM 指令是 dockerfile 的第一个指令，指定了基础镜像，后面的所有指令都是运行在该基础镜像环境上的 MAINTAINER，该指令是描述的维护者信息。
底层的系统镜像用的是什么，使用 from 指定，绝大部分情况都用的底层有个系统或者基础的环境用的是什么，对 docker 不熟练使用 centos 镜像即可。

2）RUN：构建时容器内运行的命令

RUN 指令用于在容器中执行命令。我们常用来安装基础软件。
镜像安装软件依赖包都可以放在 RUN 中。

3）COPY：复制 docker 目录中的文件到镜像中

COPY 指令类似 ADD 指令，但是 ADD 指令范围更广些，ADD 能够自动解压文件，能够访问网络资源，而 COPY 指令做不到。

非目录需要重新指定，放在目录中非常便利，属于好的一种习惯，值得推荐使用目录。

4）ADD：复制 docker 目录中的文件到镜像中。（过程可以解压）

ADD 指令是用来将宿主机某个文件或目录放到（复制）容器某个目录下面。

官方不推荐 ADD,高级复制功能，需求不精准，推荐使用 COPY。

5）EXPOSE：声明开放端口

EXPOSE 指令用于暴露容器里的端口，我们在 3.5 里面演示过了，nginx 暴露的端口是 80，但是启动容器的时候需要指定宿主机端口来映射暴露的端口。需要暴露多个端口的话可以使用多个 EXPOSE，也可以一个 EXPOSE 指令后面跟多个端口，端口之间用空格隔开。

声明不是变更，变更使用-p 构建容器时候使用。

6）ENV：设置环境变量

ENV 指令是用于设置环境变量的。

底层环境变量需要需提前设置。

7）CMD：容器启动时执行的命令，最多只能执行一条

CMD 指令是你在容器启动的时候帮你运行的命令，而 RUN 这个指令是构建镜像的时候帮你运行的命令。

容器启动时执行命令，最多执行一条。

8）WORKDIR：声明工作目录。类似 cd

WORKDIR 是指下面的指令都在 WORKDIR 指定目录下面工作，这个与 linux 里面的 cd 差不多。

切换目录使用 WORKDIR。

做容器轻量级最好，比较小就很好，使用一条命令就不要使用两条命令。

4、dockerfile 步骤

1）创建 dockerfile 目录，名字可以自定义。

在大目录里面创建各种小目录和文档，比较有层次。

2）在该目录中编辑 dockerfile。

3）用 COPY 或 ADD，需要把被 COPY 文件提前放到 dockerfile 目录中。

4）官方推荐用 COPY，如果需要过程中解压，用 ADD。

5）CMD 字段，需要用[“命令”，“选项 1”]这种格式书写。

强制格式写法，必须使用[“命令”，“选项 1”]，类似于 python 的数组。

6）CMD 字段只能写一条命令，且这条命令尽量是前台执行的命令。如果是后台命令，这条命令结束，docker 就会自动关闭。

区别：前台执行命令和后台执行命

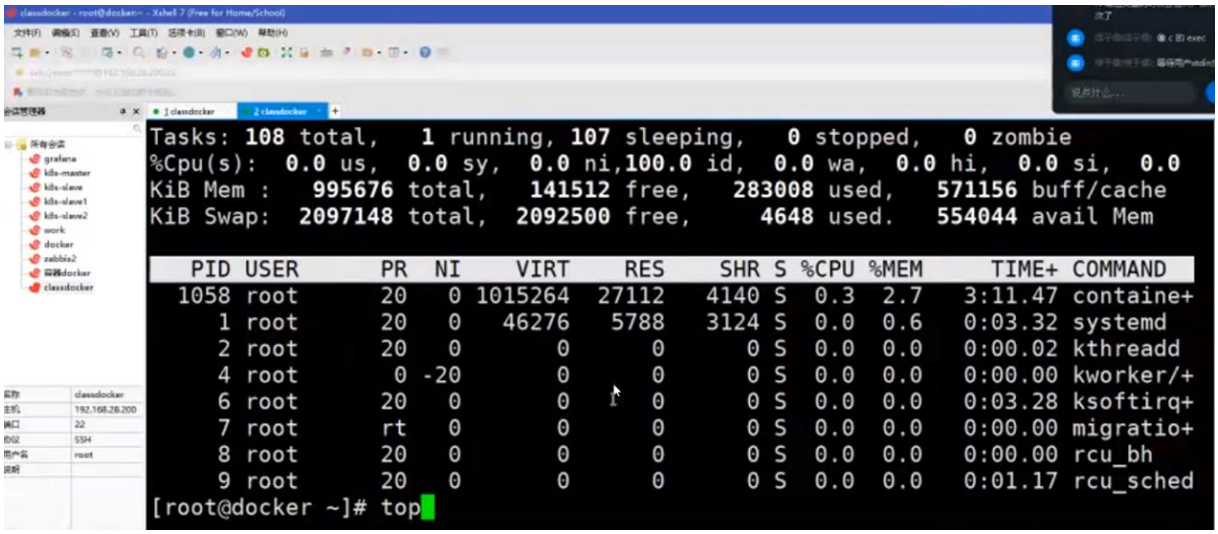
1）后台命令：

当在前台运行某个作业时，终端被该作业占据；而在后台运行作业时，它不会占据终端。可以使用命令把作业放到后台执行。当在后台执行命令时，可以继续使用你的终端做其他事情。比如 cd/tmp 作为主进程，fork 出来 shell 进程，一旦执行这条后台命令，会顶掉 PID 为 1 的进程，整个 docker 就挂掉了。

注意：经常操作的 shell 命令都属于后台命令

2）前台命令：

CMD 属于前台命令没有特殊操作会一致出现。



四、docker 的前世今生

1、dockerfile 操作测试

1）用 dockerfile 构建镜像：

docker build -t（设置要构建镜像的名字,最后要加.）

创建基础镜像

docker 提供了两种方法来创建基础镜像

一种是通过引入 tar 包的形式，

另外一种是通过一个空白的镜像来一步一步构建

scratch 是 docker 保留镜像，镜像仓库中的任何镜像都不能使用这个名字，使用 FROM scratch 表明我们要构建镜像中的第一个文件层。

如果我们有一个 Linux 下可执行的二进制文件，可以构建一个简单的镜像，仅执行这个二进制。构建的过程很简单，执行如下命令。

$ docker build -t chello:0.1 .

通过 docker images 命令可以本地的镜像。

$ docker images

2）再运行镜像：

docker run -itd -p 8800:80 [image:tag]

docker 中的容器运行在操作系统中，共享了操作系统的内核。对于在 Mac、Windows 平台下，则是基于 Linux 虚拟机的内核。

而 Linux 内核仅提供了进程管理、内存管理、文件系统管理等一些基础的管理模块。除此之外，我们还需要一些 Linux 下的管理工具，包括 ls、cp、mv、tar 以及应用程序运行依赖的一些包。因此我们就需要首先构建一个 Minimal 的操作系统镜像，在此基础上构建 Python 环境，再构建应用镜像。这样就实现了镜像文件分层，今后如果我们需要更新 Python 版本，那么只需要对这一层进行更新就可以。

3）查看镜像：

docker ps -a 确认是启动状态

docker 的镜像实际上由一层一层的文件系统组成，这种层级的文件系统就是上文说到的 UnionFS。在 docker 镜像的最底层是 bootfs。这一层与我们典型的 Linux/Unix 系统是一样的，包含 boot 加载器和内核。

当 boot 加载完成之后整个内核就都在内存中了，此时内存的使用权已由 bootfs 转交给内核，此时系统也会卸载 bootfs。docker 在 bootfs 之上的一层是 rootfs（根文件系统）。rootfs 就是各种不同的操作系统发行版，比如 Ubuntu，Centos 等等。docker 核心技术与实现原理 这篇文章，作者阅读了 rootfs 的规范，指出构建 rootfs 一些必须的文件夹。

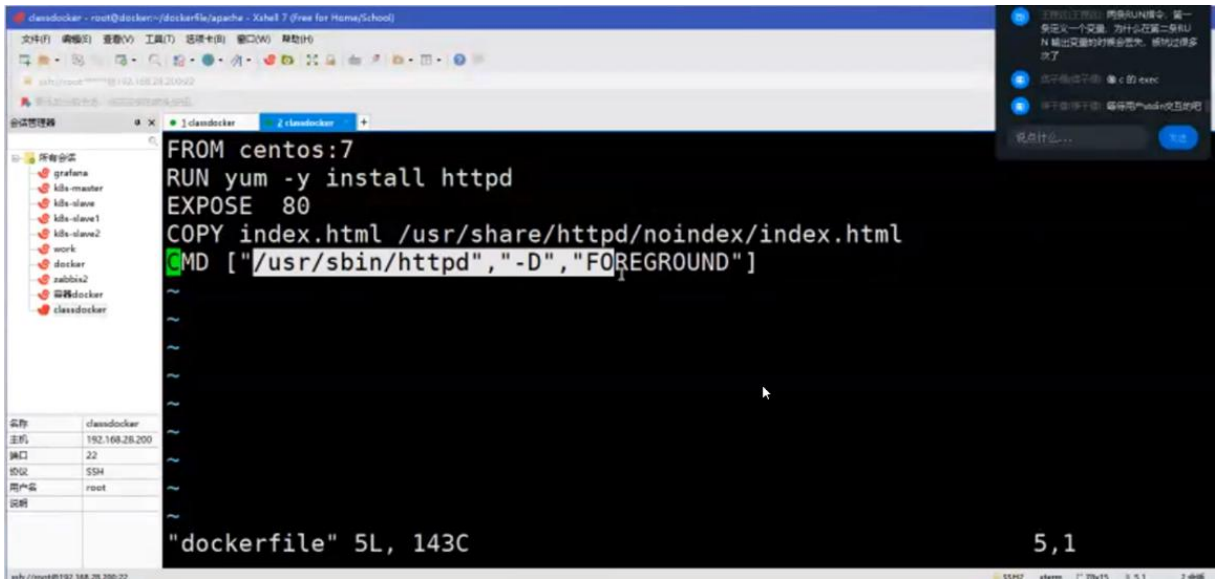
4）测试：

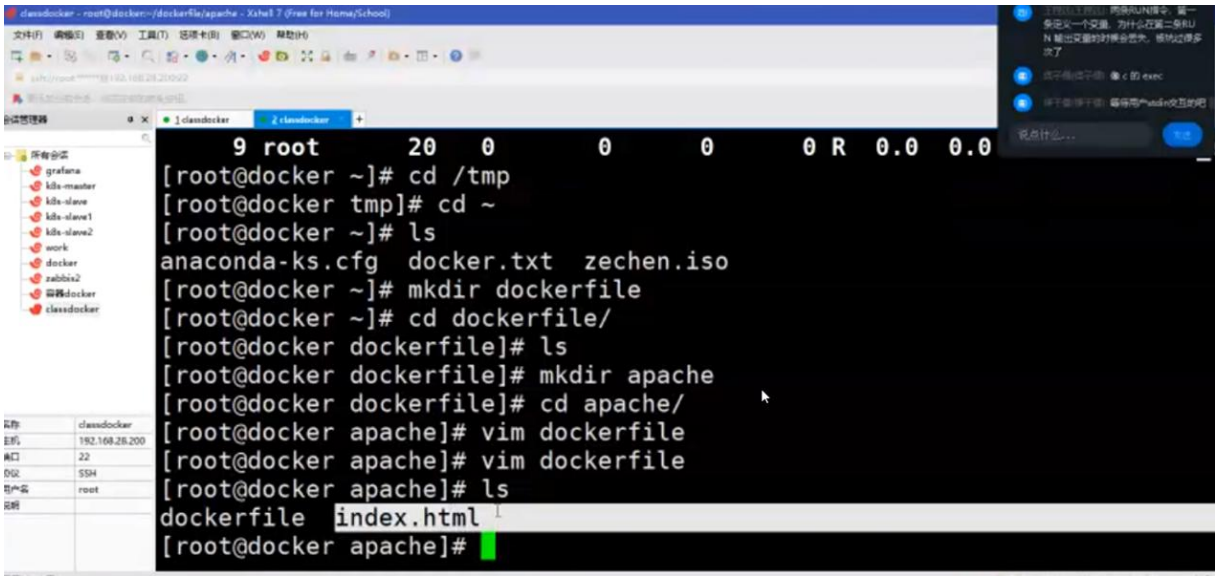
宿主机用浏览器访问 IP+port 看是否启动成功。

2、dockerfile 案例 1

1）用 dockerfile 创建并启动一个 centos 的 apache 镜像。指定自定义内容。

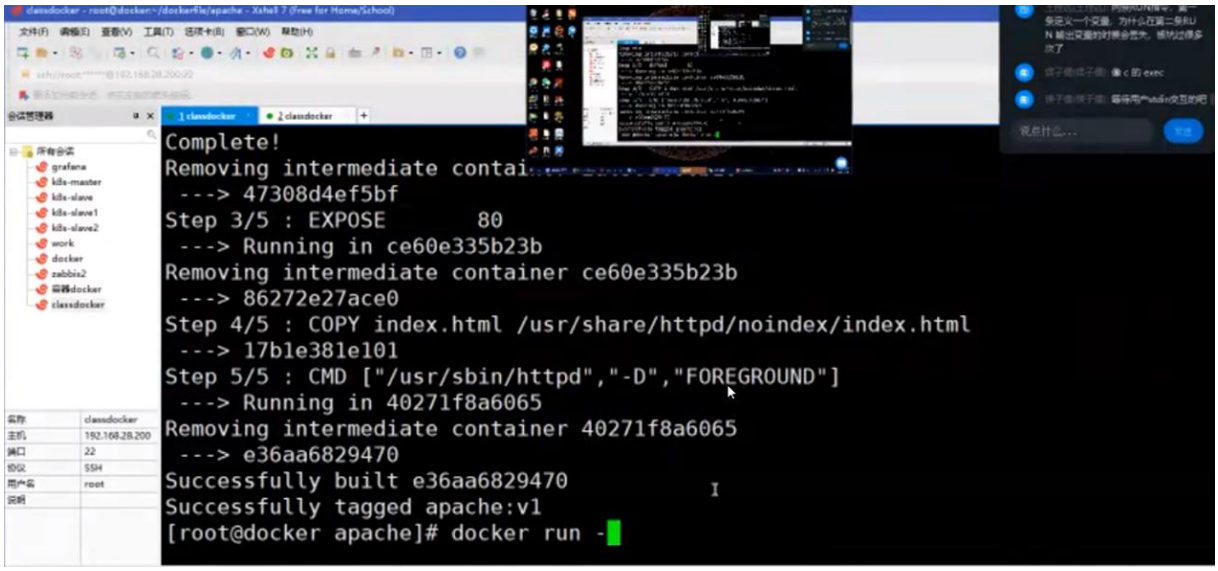
```
FROM centos:7
RUN yum -y install httpd
EXPOSE 80
COPY index.html /usr/share/httpd/noindex/index.html
CMD ["/usr/sbin/httpd","-D","FOREGROUND"]
```





2）构建镜像

[root@docker apache]# docker build -t apache :v1 .



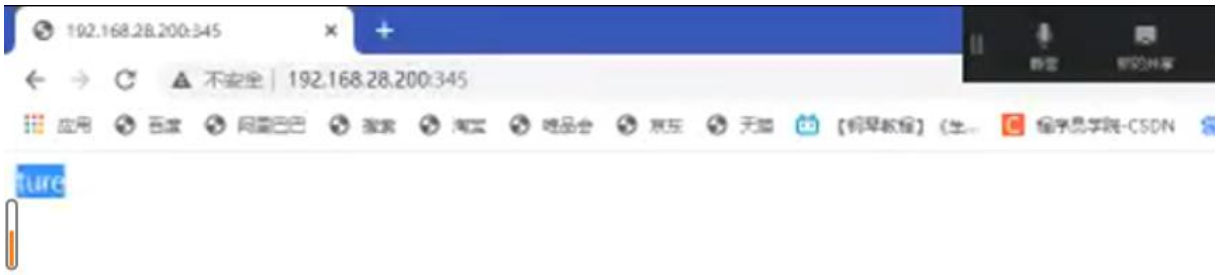
看到 successfully 即安装成功。

3）验证

[root@docker apache] docker run -itd -p 345:80 apache:v1

[root@docker apache] docker ps

通过访问宿主机 ip 端口，查看 true，防火墙关闭。



3、dockerfile 案例 2

1）用 dockerfile 创建并启动一个 centos7 的 nginx 镜像。

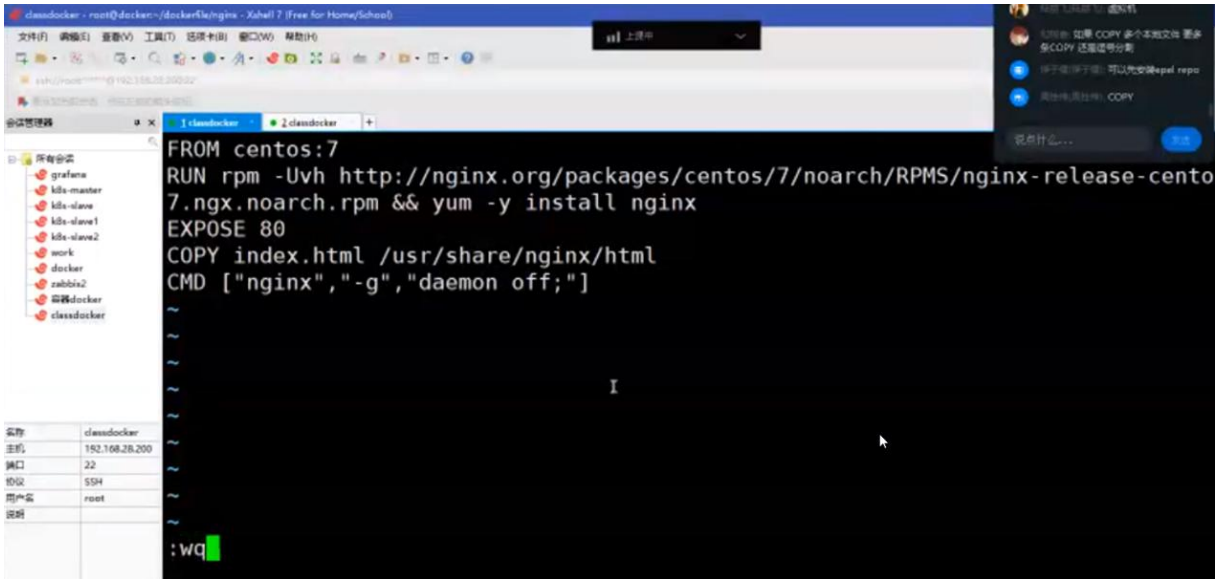
FROM centos:7

RUNrpm-Uvhhttp://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx.noarch.rpm

RUN yum -y install nginx

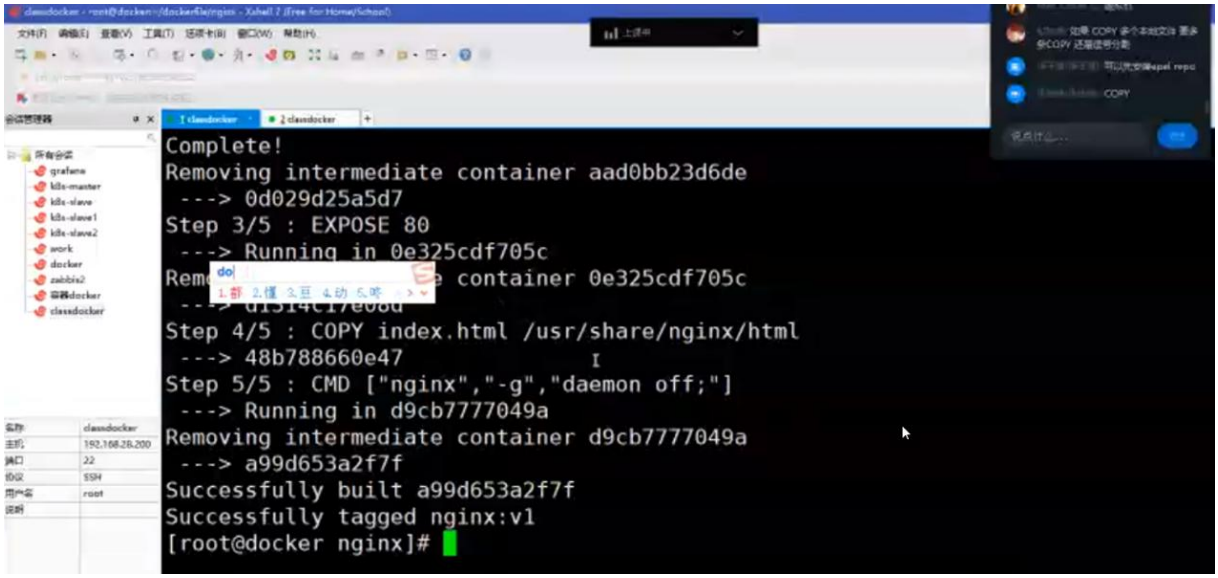
EXPOSE 80

CMD ["nginx","-g","daemon off;"]



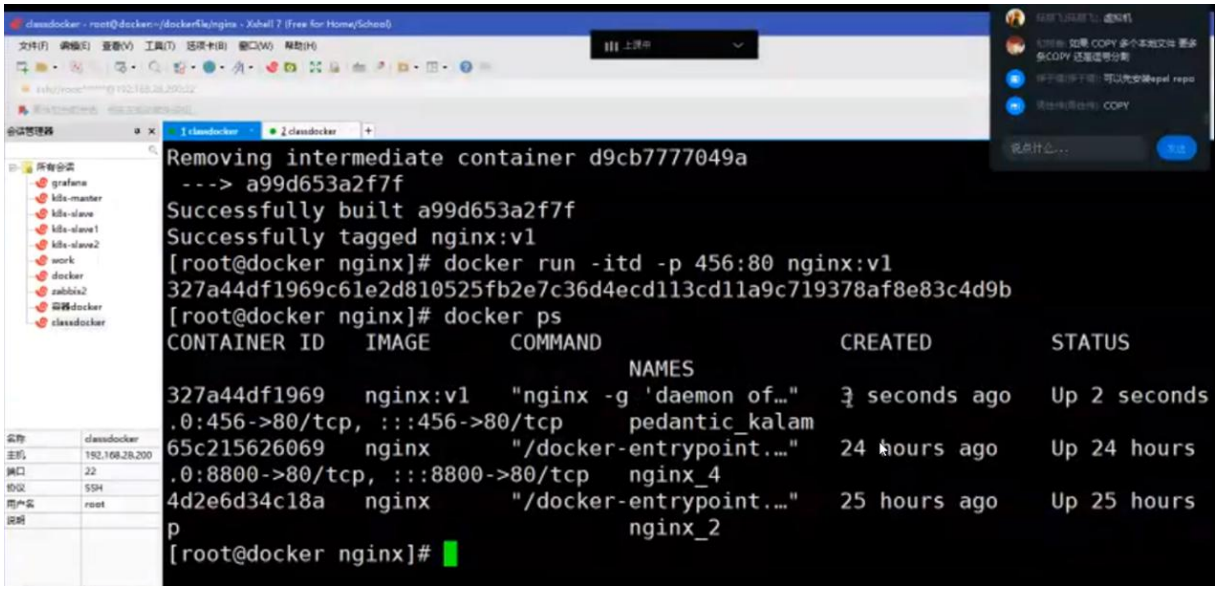
2）创建

[root@docker nginx]#docker bulid -t nginx:v1 .



3）创建容器

[root@dockernginx]# docker run -itd -p 456:80



4）验证

通过访问宿主机的 ip+端口，查看 nginx，防火墙关闭。

docker 实际工作案例实现

演讲人：马哥教育

摘要：本文整理自“4 天 docker 实战“，在”1024 创造营“的分享。本篇内容主要分为三个部分。

一、数据持久化
二、Barbor 仓库
三、微服务

视频链接：https://developer.aliyun.com/learning/course/892

一、数据持久化

我们什么情况下要做数据持久化呢？一定是在做容器之前先预判好哪些文件是要永久存储的，而不会跟着容器的生命周期而消失。

比如说配置文件、日志文件、缓存文件或者应用数据等等。数据初始化有三种类型：第一种volumes，这个是最推荐的，也是最好的一种方式，第二种是 bind—mount，第三种是 tmpfs。

数据持久化方式一：

存于主机文件系统中的某个区域，由 Docker 管理（/var/lib/docker/volumes/ ）。非 Docker 进程不应该修改这些数据。你可以把它当做容器数据持久化专门的磁盘分区。

特点：

1：volumes 是Docker 中持久化数据的最好方式。因为和容器解耦度更高。
2：多个容器可以同时访问一个 volumes 。
3：远程主机或非本地常用该方式。（架构）
4：首先需要创建。之后再进行使用。

使用：

docker volume create test1
docker run -itd -p 880 0:80 -v test1:/ usr/share/nginx/html nginx:v1

方式一：volumes

它是官方比较推荐也是大型的集群比较常见的一种方式。可以理解为在自己的宿主机或者云端或者在某一个区域创建一块磁盘专门去存放容器里的数据或文件。把这个容器里边的数据或者文件还有目录等都规划好，再去启动容器。在老一些的版本里边首先必须要去创建 volumes，否则是没有办法创建成功的。新版本好像不写命令也可以创建成功，具体可以查询官方关于 volumes 的文档。

使用：

docker volume create test1

docker run -itd -p 8800:80 -v test1:/usr/share/nginx/html nginx:v1

创建：docker volume create

删除某个卷：docker volume rm 卷名

删除所有未使用的卷：docker volume prune

列出所有卷：docker volume ls

查看某个卷的信息：docker volume inspect 卷名

挂载到容器：-v 或–volume。如果是 Docker17.06 或更高：推荐使用–mount。（同 bind mount）

挂载类型：key 为 type，value 为 bind、volume 或 tmpfs

挂载源：key 为 source 或 src，对于命名卷，value 为卷名，对于匿名卷，则忽略

容器中的挂载点：key 为 destination、dst 或 target，value 为容器中的路径

读写类型：value 为 readonly，没有 key

volume-opt 选项，可以出现多次。比如 volume-driver=local,volume-opt=type=nfs,…

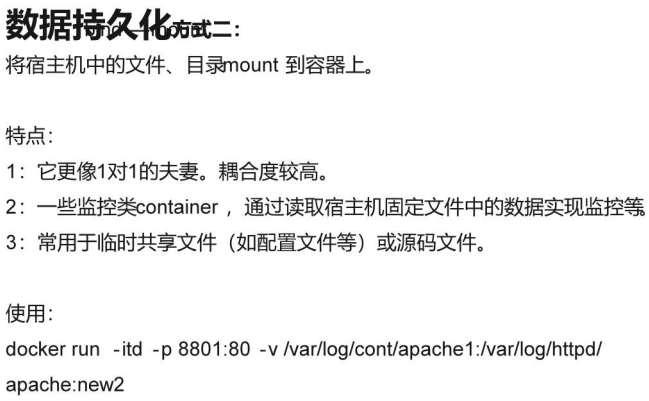
第一个域：对于命名卷，为卷名；匿名卷，则忽略，此时会创建匿名卷

第二个域：容器中的挂载点

第三个域：可选参数，由’,'隔开，如 ro

-v 或—volume：由 3 个域组成，’:’ 分隔

—mount：由多个’,’ 隔开的键值对组成



方式二：bind—mount

将宿主机中的文件、目录 mount 到容器上。宿主机、container 之间共享宿主机文件系统。这种持久化方法更导致 container 与宿主机的耦合过于紧密，所以不推荐使用。

使用：

docker run -itd -p 8801:80 -v /var/log/cont/apache1:/var/log/httpd/ apache:new2

挂载到容器：-v 或—volume。如果是 Docker17.06 或更高：推荐使用—mount。（同 volumes）

-v 或–volume：由 3 个域组成，’:'分隔

第一个域：对于命名卷，为卷名；匿名卷，则忽略，此时会创建匿名卷

第二个域：容器中的挂载点

第三个域：可选参数，由’,'隔开，如 ro

–mount：由多个’,'隔开的键值对=组成：

挂载类型：key 为 type，value 为 bind、volume 或 tmpfs

挂载源：key 为 source 或 src，value 为主机中文件或目录的路径

容器中的挂载点：key 为 destination、dst 或 target，value 为容器中的路径

读写类型：value 为 readonly，没有 key

- bind-propagation 选项：key 为 bind-propagation，value 为 rprivate、private、rshared、shared、rslave 或 slave
- 一致性选项：value 为 consistent、delegated、cached。这个选项仅仅适用于 Docker for Mac
- –mount 不支持 z 和 Z（这个不同于-v 和—volume）

数据持久化方式一：

将数据存于宿主机内存中。

特点：

1：仅限Linux系统。

2：较少用，常见用于对于访问有大量读写，或安全层面考虑。

3：不会持久化，生命周期限于容器启动时。

4：多容器无法共享同一个tmpfs

使用：

```
docker run  -itd  -- name tmptest  -- tmpfs /  root  nginx:latest
```

方式三：tmpfs

将数据存于宿主机内存中。docker 可将用户名与密码等敏感数据保存在某个数据库中，当启动需要访问这些敏感数据的 container 或者 service 时，docker 会在宿主机上创建一个 tmpfs，然后将敏感数据从数据库读出写到 tmpfs 中，再将 tmpfs mount 到 container 中，这样能保证数据安全。当容器停止运行时，则相应的 tmpfs 也从系统中删除。

docker run -itd --name tmptest --tmpfs /root nginx:latest

二、harbor 仓库

阿里云 开发者训练营

harbor仓库

是由VMware公司开源的企业级的Docker Registry 管理项目，
相比docker 官方拥有更丰富的权限权利和完善的架构设计，适用大规模
docker 集群部署提供仓库服务。

作用：
1：企业级镜像仓库，根据企业需求创建的镜像，可以通过harbor 实现
管理和使用。
2：主要特点功能包括：基于角色的访问控制，基于镜像的复制策略，
图形化用户界面，审计管理等。

官网：
https://github.com/goharbor/harbor/releases

镜像获取有两种方式：

第一种方式是从官方的镜像仓库获取，search 命令从官方镜像查找已有的镜像。

第二种方式是企业级镜像仓库，公司最常用的是 harbor。

阿里云 开发者训练营

harbor仓库

Harbor是由VMware公司开源的企业级的Docker Registry管理项目，
相比docker官方拥有更丰富的权限权利和完善的架构设计，适用大规模
docker集群部署提供仓库服务。

作用：
1：企业级镜像仓库，根据企业需求创建的镜像，可以通过harbor实现
管理和使用。
2：主要特点功能包括：基于角色的访问控制，基于镜像的复制策略，
图形化用户界面，审计管理等。

官网：
https://github.com/goharbor/harbor/releases

harbor 的作用：

1：企业级镜像仓库，根据企业需求创建的镜像，可以通过 harbor 实现管理和使用。

2：主要特点功能包括：基于角色的访问控制，基于镜像的复制策略，图形化用户界面，审计管理等。

安装：

安装并启动 docker

安装 docker-compose1.18 以上版本。

（docker 单机容器编排工具）

- wget

https://github.com/docker/compose/releases/download/1.29.2/

docker-compose-Linux-x86_64

#从官网下载 docker-compose 二进制文件

- chmod +x docker-compose-Linux-x86_64

#添加执行权限

- mv docker-compose-Linux-x86_64 /usr/bin

#移动到系统指令目录下

tar -xvf harbor-offline-installer-v2.3.1.tgz

#解压

- cp harbor.yml.tmpl harbor.yml

#复制官方配置文件案例。

- vim harbor.yml

#修改配置文件中 hostname 和 https 字段。

Vim/etc/docker/daemon.json(信任 harbor 端 IP)

{

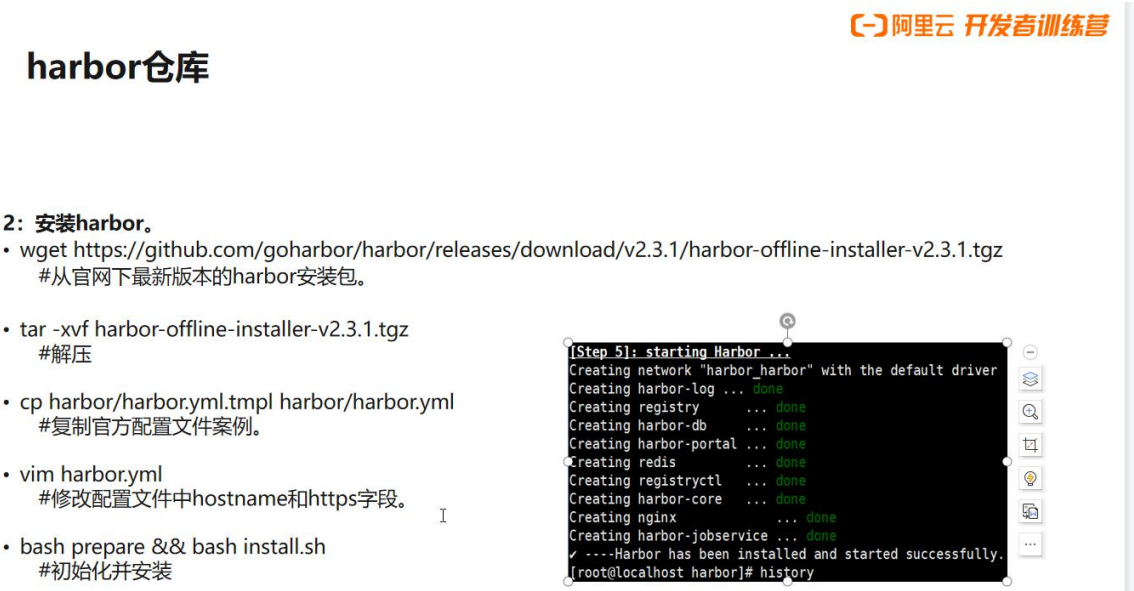
“Registry-mirrors”:[“https//f9dk003m.mirror.aliyuncs.cm”],

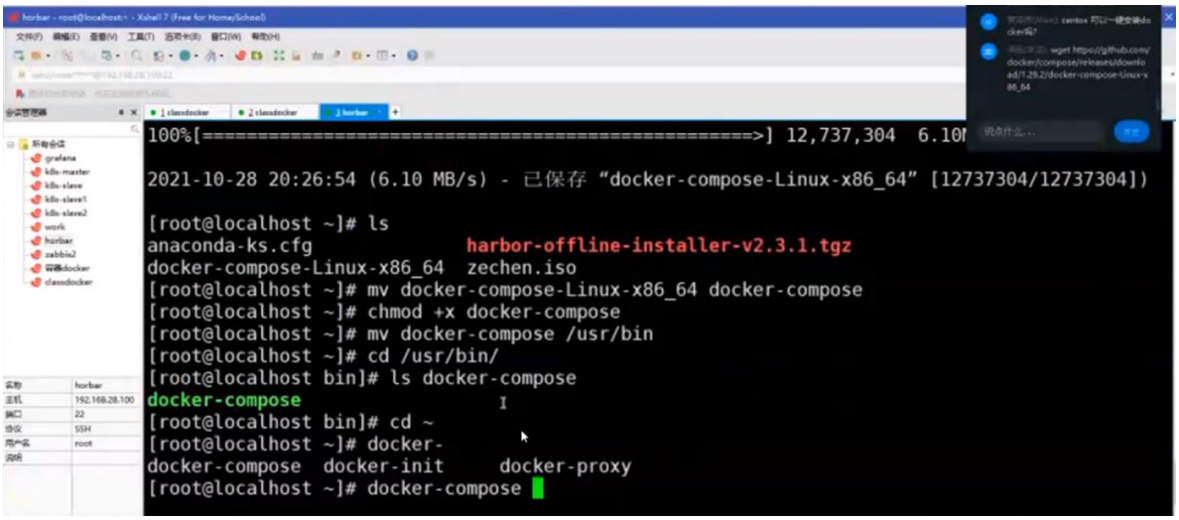
“insecure-regidtries”:[“192.168.28.100]

}(重启 docker)

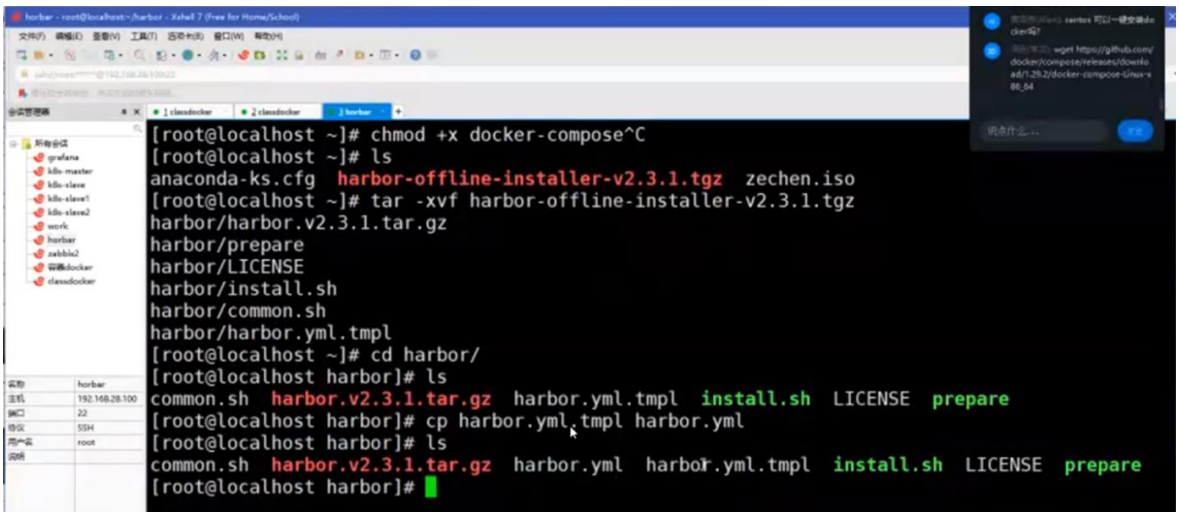
- bash prepare && bash install.sh

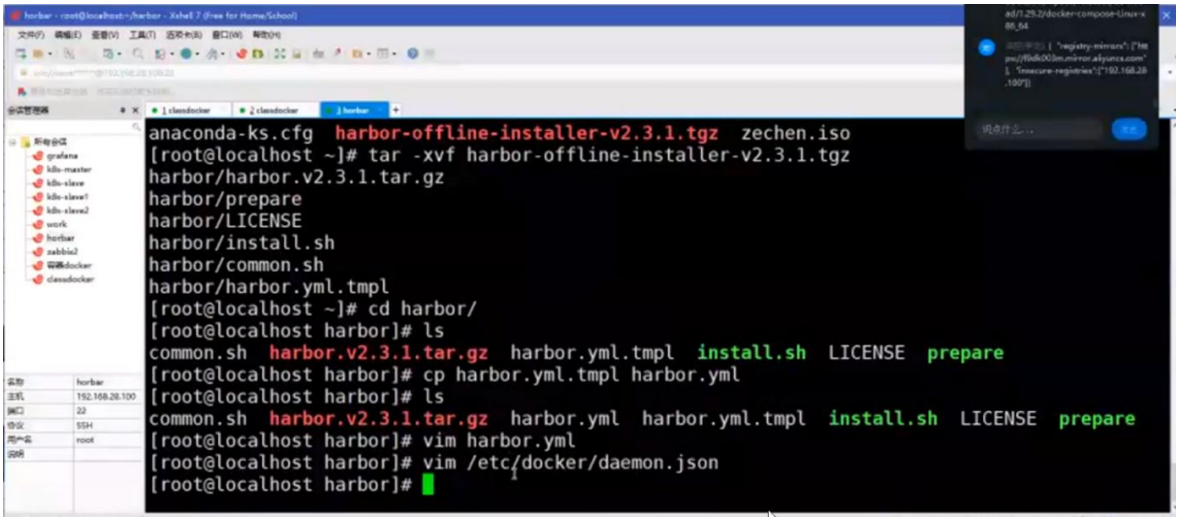
#初始化并安装





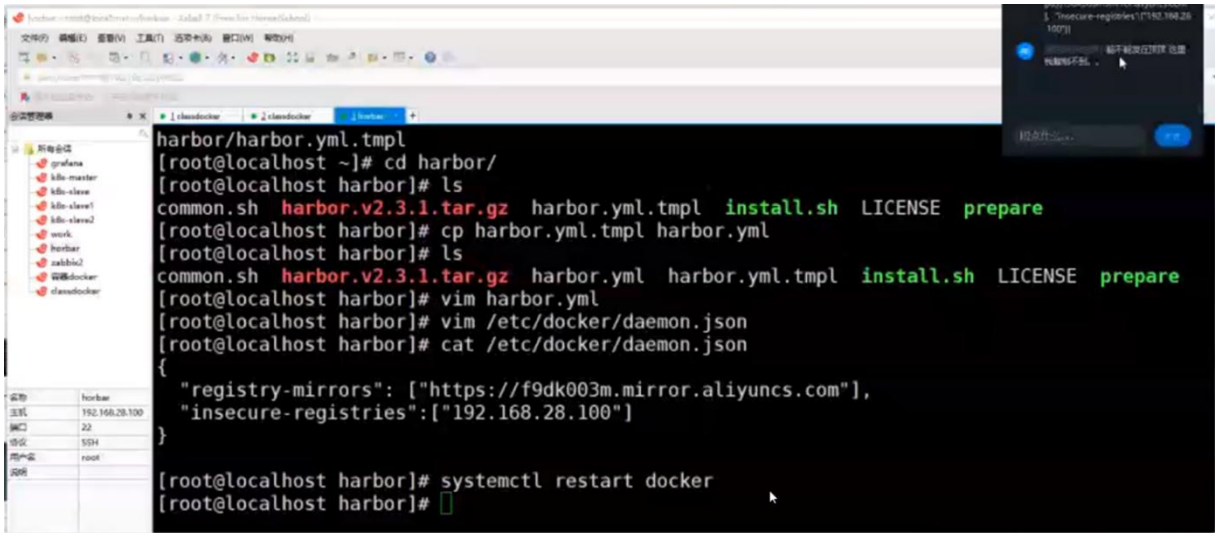
compose 是 docker 的单机容器编排工具。





https 注释，帮助我们实现安全的目的。

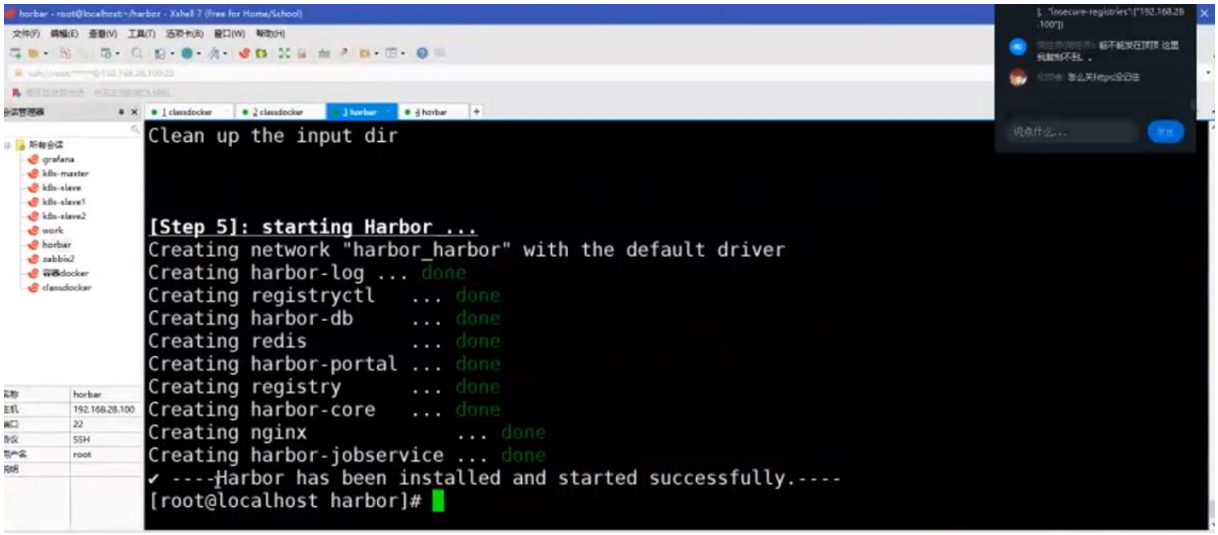
重启 docker。



执行，

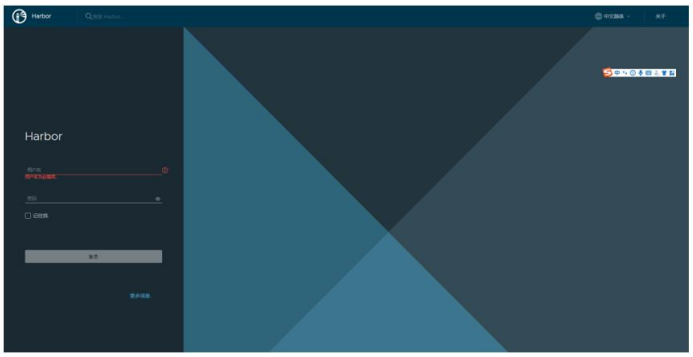
保存退出，

出现 Harbor has been installedand started successfully，安装成功。



阿里云 开发者训练营

harbor仓库



用户名：admin

密码：Harbor12345

阿里云 开发者训练营

harbor仓库

使用harbor

一：web页面中操作新建项目、新建人员并设置权限、匹配人员和项目。

二：将本地刚刚做的镜像推送到harbor共用项目仓库中。

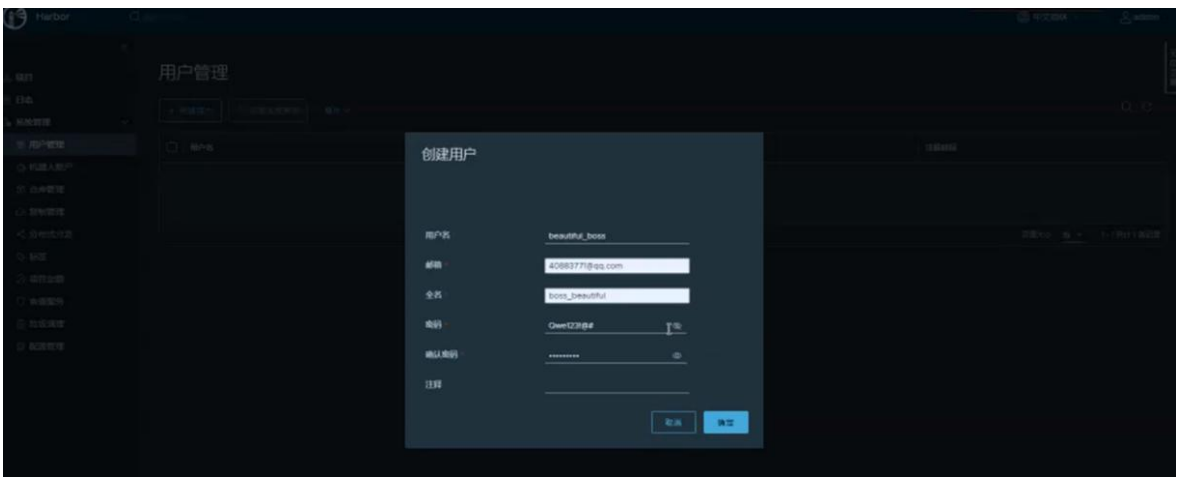
- docker tag apache:new1 192.168.28.131/library/apache:new1
- vim /etc/docker/daemon.json （信任harbor端IP）

```
{
  "registry-mirrors": ["https://f9dk003m.mirror.aliyuncs.com"],
  "insecure-registries":["192.168.28.131"]
}
```

- docker login 192.168.28.131
- docker push 192.168.28.131/library/apache:new1

三：如何从harbor上边拉取镜像到服务器?

应用



1）新建项目后，创建用户，新建成员名称要和创建用户名字一样。

2）将本地刚刚做的镜像推送到 harbor 共用项目仓库中。

docker tag apache:new1 192.168.28.100/library/apache:new1

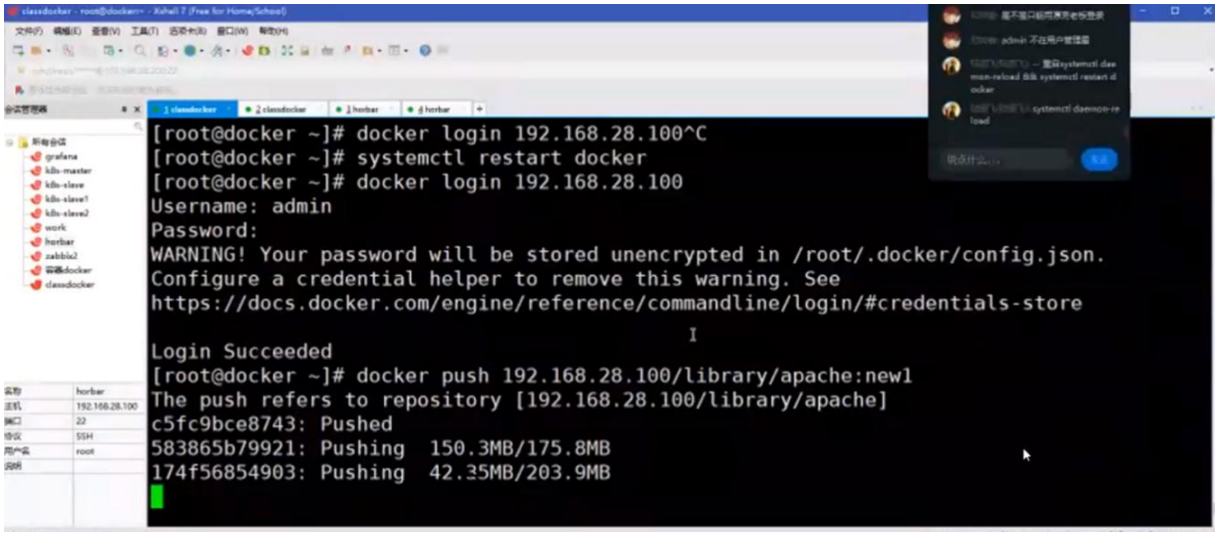
使用 harbor•vim /etc/docker/daemon.json

（信任 harbor 端 IP）

```
{
  "registry-mirrors": ["https://f9dk003m.mirror.aliyuncs.com"],
  "insecure-registries":["192.168.28.200"]
}
```

docker login 192.168.28.100

docker push 192.168.28.100/library/apache:new1



三、微服务

微服务是一种架构的模式，是单体的应用。单体应用拆分成各个模块，充分的去解耦，独立构建部署，互相之间可以协调配合。

微服务的特点:颗粒度小，且专注一件事情;单独的进程;轻量级的通信机制;松耦合，独立部署。

微服务具备四个特性，复杂性，隐匿性，异变性，配合性。

1、复杂性

困境：系统会随着业务的发展，增加越来越多的特性和功能，使得系统复杂到没有一个人能全面理解，没有一个人敢去修改原有的功能或代码。

微服务：微服务提出以业务边界作为模块划分原则，每个模块独立进程，一个业务由很多独立的小业务组合而成，系统也是由独立的小系统组合而成，这样的好处是每个小系统都很容易理解，一个大系统可以根据业务组合微服务，或者逐渐发展独立的微服务。

代价：微服务为了降低单个微服务的复杂性，导致整体系统的复杂性急剧增加。微服务之间的连接更加复杂，网络通讯不可靠和性能损耗，协议匹配，接口对接和转换，版本协作，微服务注册和发现，编排和调度，分布式业务和数据一致性等复杂性都是单体架构不需要考虑的。

2、隐匿性

困境：软件在没有应用到业务之前，各种信息和思考大多在每个人的脑海里，很难完全呈现和想象出来，客户只是知道自己想要更多新英雄，但并不知道要具体什么样的英雄；开发设计人员知道怎么做英雄，但又不能完全理解用户需求；运维知道部署服务和上线，但又不能完全理解业务逻辑。单体架构中的各个模块隐藏在大系统的页面下，在交付之前对外界是不可见的。导致没有人能看清全貌，各自都把事情做到最好，但组合起来却不是客户想要的东西。

微服务：微服务架构并没有改变软件开发过程中的隐匿性，而是通过缩短从需求到交付这段软件开发周期，减少隐匿时间，来降低软件工程总体的隐匿性。

代价：组织必须要具备自动部署持续交付能力。假如一个系统上线需要 3 小时进行部署，如果我们要持续部署，每天都部署一次，那就需要每天拿出 3 小时做部署，这个成本是不能接受的。

3、耦合性

困境：假如系统从零开始做的话，头三个月开发会比较慢，因为需要搭建和熟悉一些开发、测试、部署基础设施，随着基础设施和公共组件的完善，接下的半年到一年开发会加速，但是再往后开发速度又会逐渐降低，因为那些一开始提高开发效率的接口、共享表、依赖组件都变成了复杂网络缠绕在一起，变成了所谓的牵一发而动全身，改一行代码都不知道会影响到什么地方。

微服务：微服务系统的耦合性问题总是在一个可控的范围内。比如微服务独立数据库表结构，那我们根据业务需要改表结构的时候就不需要去考虑会不会影响到其他业务，因为其他业务和这个表结构完全没关系。

代价：单纯从接口协同一致上来说，微服务架构比较糟糕。单体架构的接口之间配合是相同的编程语言，基本上在编译时就能发现错误，而微服务的接口往往是远程服务，验证增加难度。

4、易变性

困境：比如某个模块的流量突然增加，或者需要大内存，单体架构只能为极少的模块增加整个系统的计算资源，又因为增加整个系统的计算资源成本很高，实施时间长，导致性能需求迟迟不能得到满足。

微服务：微服务要求独立进程，可以完全根据需耩定制不同类型的计算资源，更精细化分类的利用内存、IO、CPU。因为小，可以更快水平扩展响应性能需求变化。更关键是，微服务小，强调独立业务价值。小团队直接面对客户需求做决策，所有信息和想法在小范围内快速交流，业务价值流动更容易可见，更快速的响应变化。后期可以弹性扩容。

代价：微服务架构需要改变组织结构小团队充分制授权、业务交付模式。对于传统组织而言，这点是最难的，尤其是大公司往往采用层级组织架构。

总结：

1、单体应用的困难，是现在普遍的困境。新的团队有了理念后，会在万丈高楼平地起时就采用该架构，避免单体应用困境。
2、微服务是趋势，现在某些大公司难以实现只是时间问题。除非它不需要拓展业务和变更业务。
3、18 年前后 k8s 的出现，短短三年就被世界各互联网公司所接受和使用，就足以说明微服务的重要性。对运维人员和开发人员来说，这是个大的挑战和机遇。
4、遇到风口，不进则退。

实现：

我们最后做一个案例，nginx 和 php-fpm 实现 php 网页正常运行。这个案例学通，之后就可以根据该案例知识，做更复杂的架构。该案例也是运维微服务需要基础掌握的核心内容。
有几个问题需要解决：

1、一个容器只能有一个启动命令（CMD），但这里启动两个服务 。
2、多个容器之间如何通信？
3、php-fpm 只允许本机通信，多个容器通信需自私做？
4、nginx 和 php-fpm 需要访问同一个网页文件，怎么解决？

php-fpm 镜像

创建镜像准备：

vim /etc/php-fpm.d/www.conf
listen=0.0.0.0:9000 监听 IP 和端口。

;listen.allowed_clients=127.0.0.1 允许连接的 FastCGI 客户的 ipv4 地址列表。

mkdir /root/php-fpm
cp /etc/php-fpm.d/www.conf/root/php-fpm/
写 php-fpm 的 dockerfile
vim /root/ php-fpm/dockerfile
FROM centos:7
RUN yum –y install php-fpm
EXPOSE 9000
COPY www.conf /etc/ php-fpm.d/www.conf
CMD [“/usr/sbin/ php-fpm”,”—nodaemonize”]

1）Php-fpm 创建镜像准备

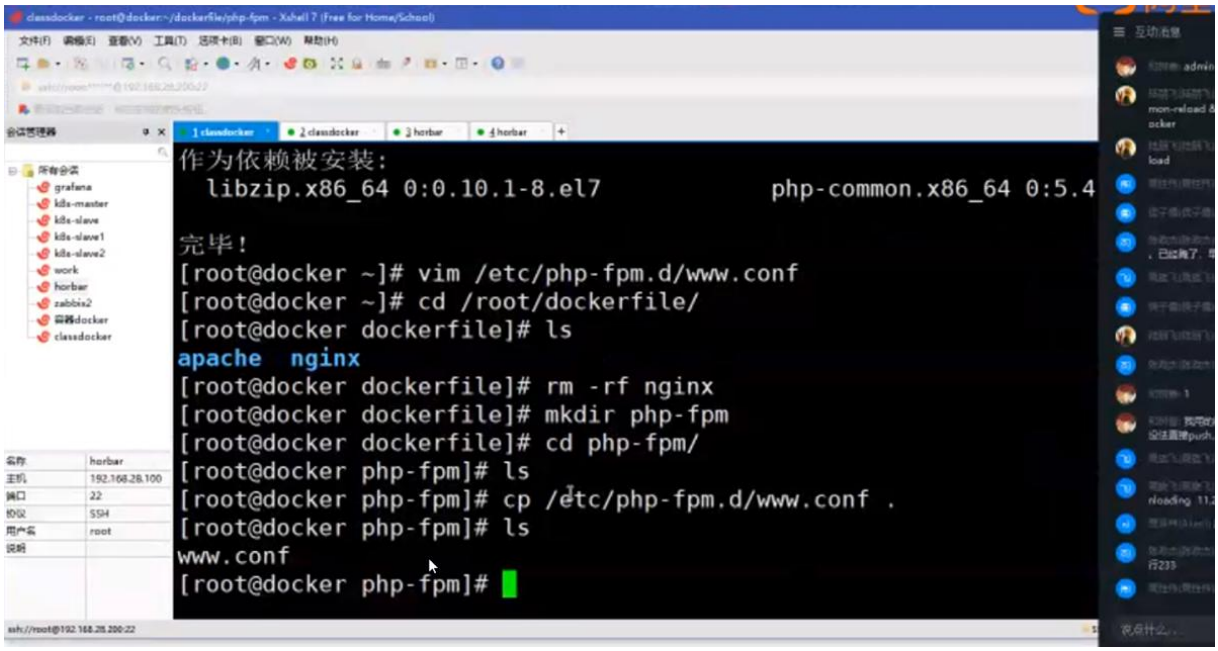
vim/etc/php-fpm.d/www.conf

listen = 0.0.0.0:9000 监听 IP 和端口。

;listen.allowed_clients = 127.0.0.1

mkdir /root/php-fpm

cp /etc/php-fpm.d/www.conf /root/php-fpm/



2）php-fpm 的 dockerfile

vim /root/php-fpm/dockerfile

FROM centos:7

RUN yum -y install php-fpm

EXPOSE 9000

COPY www.conf /etc/php-fpm.d/www.conf

CMD ["/usr/sbin/php-fpm","--nodaemonize"]

3）nginx 创建镜像准备

vim /etc/nginx/conf.d/default.conf

改成 php 配置

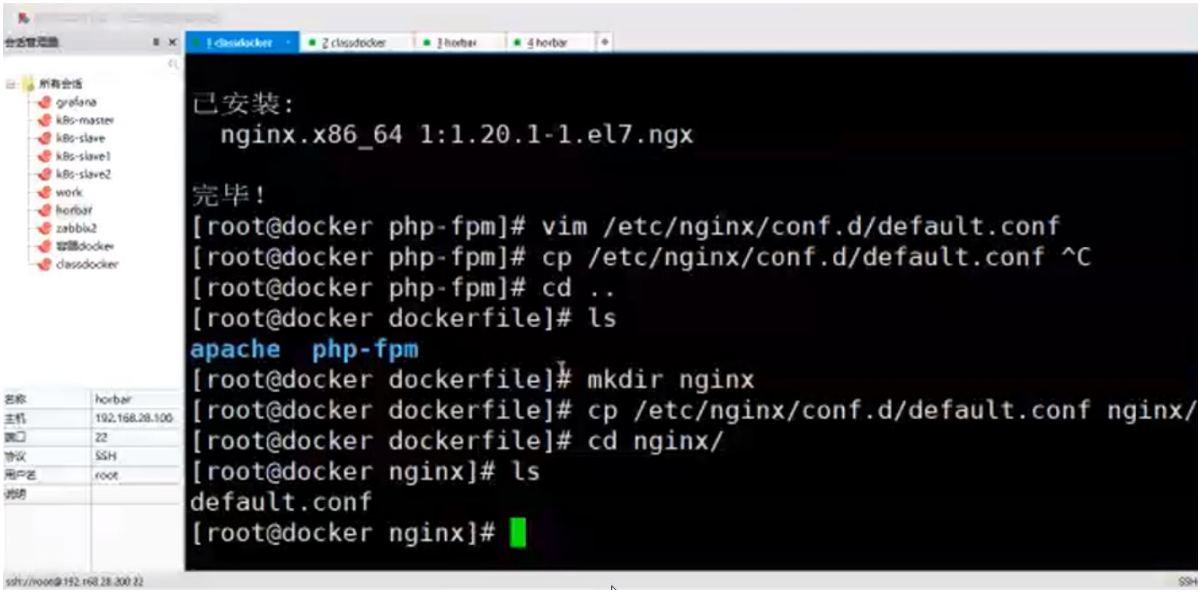
mkdir /root/nginx

cp /etc/nginx/conf.d/default.conf /root/nginx

mkdir /root/nginxweb

vim /root/nginxweb/index.php

<?php echo '<p>true</p>'; ?>



4）写 nginx 的 dockerfile

vim /root/nginx/dockerfile

FROM centos:7

RUN rpm -Uvh

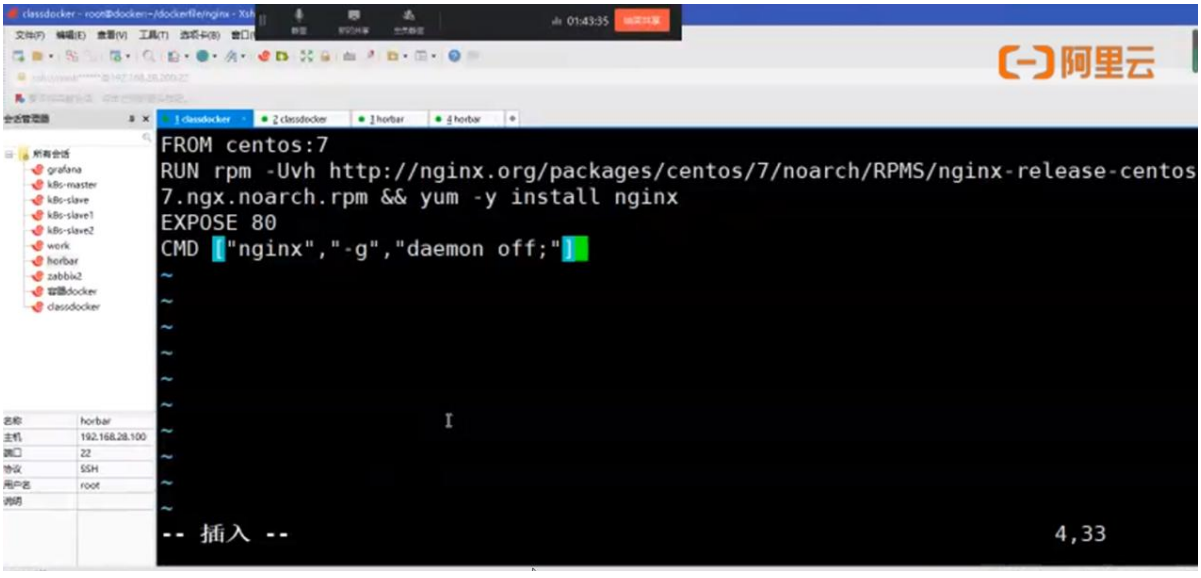
http://nginx.org/packages/centos/7/noarch/RPMS/

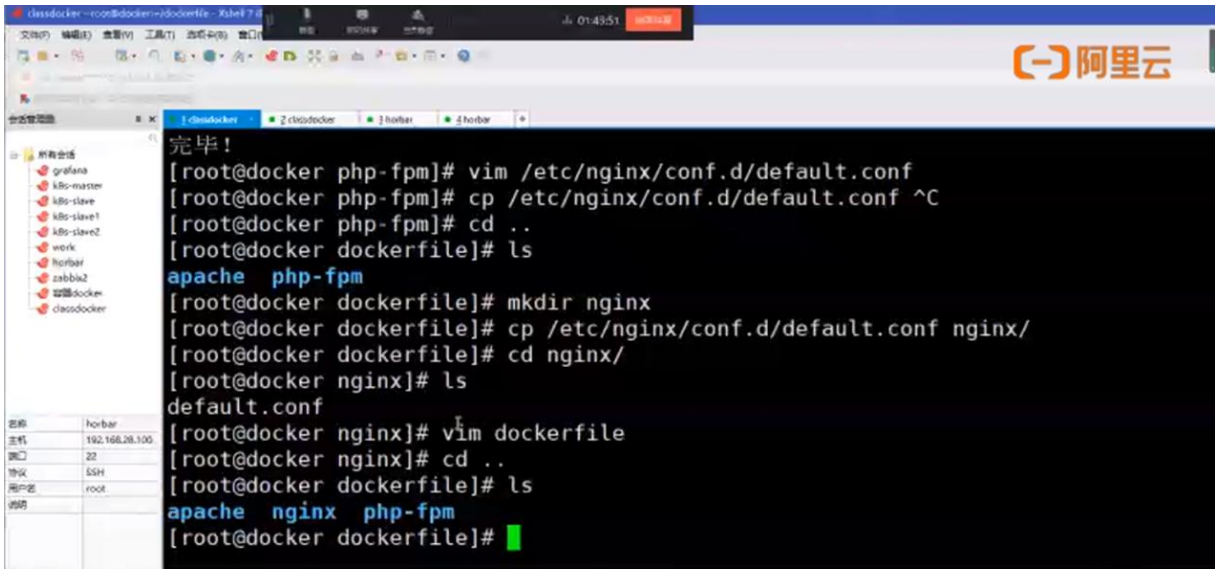
nginx-release-centos-7-0.el7.ngx.noarch.rpm

RUN yum -y install nginx

EXPOSE 80

CMD ["nginx","-g","daemon off;"]





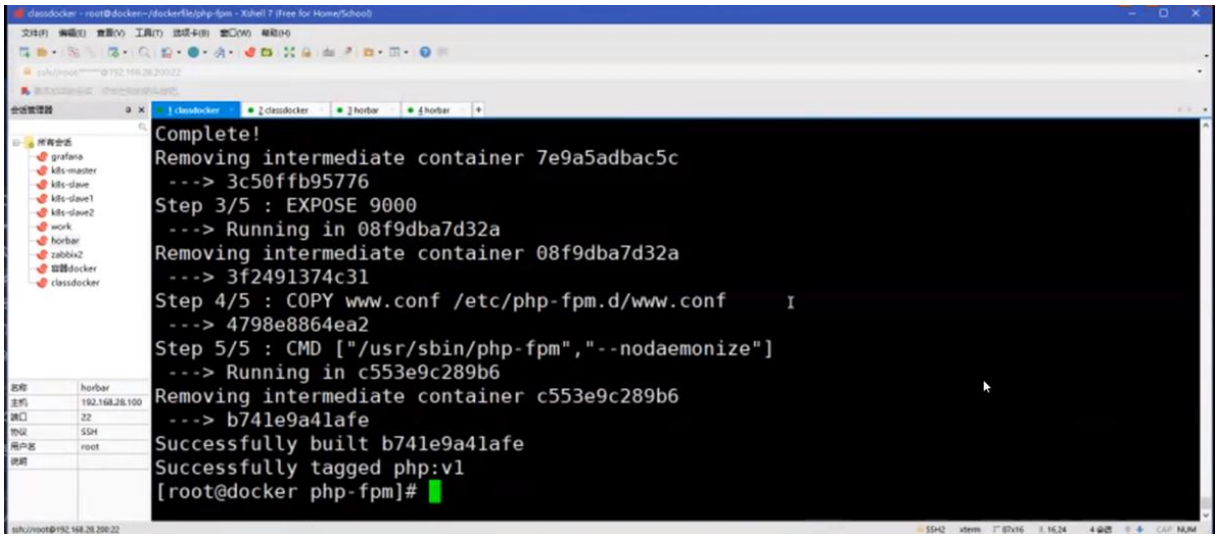
5）构建镜像

cd /root/nginx

docker build -t nginx:v1 .

cd /root/php-fpm

docker build -t php-fpm:v1 .



6）启动容器

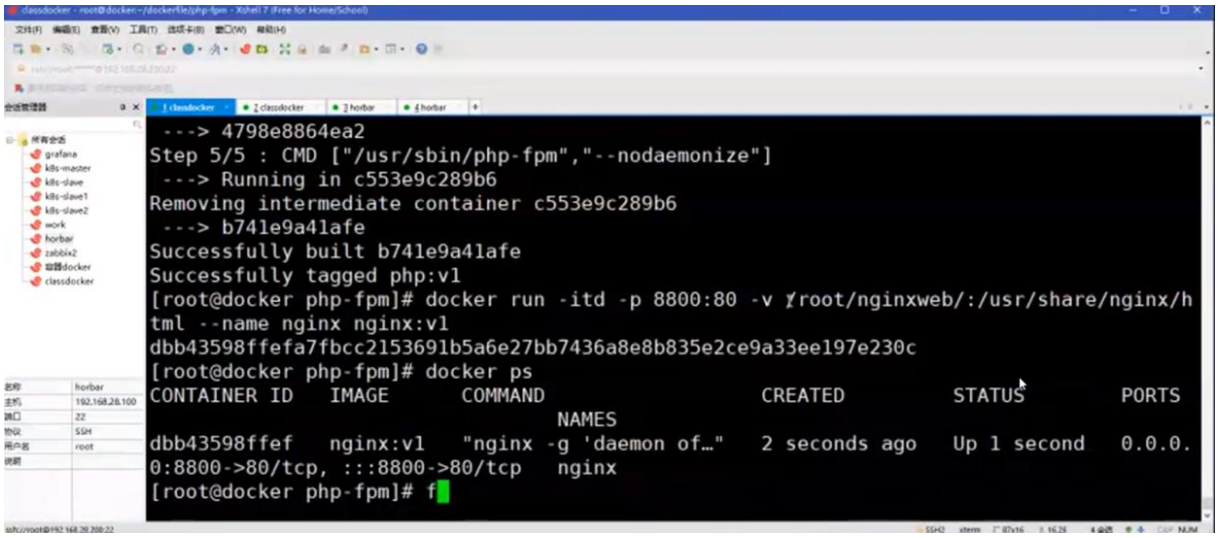
docker run -itd -p 8800:80 -v /root/nginxweb/:/usr/share/nginx/html

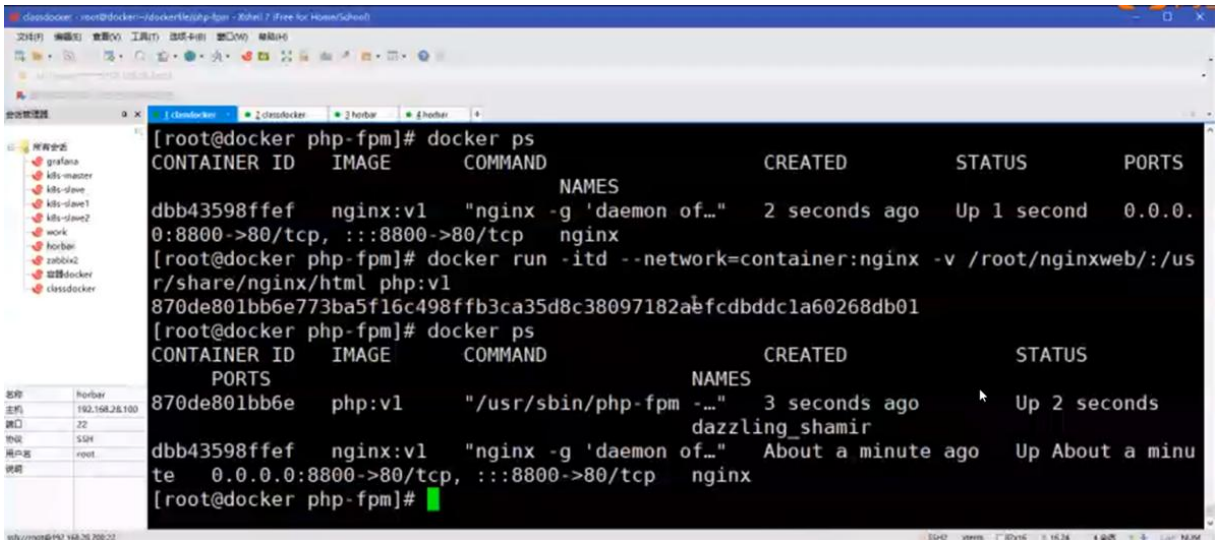
--name nginx nginx:v1

docker run -itd --network=container:nginx

-v /root/nginxweb/:/usr/share/nginx/html

php-fpm:v1




[root@docker php-fpm]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
dbb43598ffef nginx:v1 "nginx -g 'daemon of…" 2 seconds ago Up 1 second 0.0.0.
0:8800->80/tcp, :::8800->80/tcp nginx
[root@docker php-fpm]# docker run -itd --network=container:nginx -v /root/nginxweb/:/us
r/share/nginx/html php:v1
870de801bb6e773ba5f16c498ffb3ca35d8c38097182aefcdbddc1a60268db01
[root@docker php-fpm]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
870de801bb6e php:v1 "/usr/sbin/php-fpm -…" 3 seconds ago Up 2 seconds
dazzling_shamir
dbb43598ffef nginx:v1 "nginx -g 'daemon of…" About a minute ago Up About a minu
te 0.0.0.0:8800->80/tcp, :::8800->80/tcp nginx
[root@docker php-fpm]#

验证，

直接网页打开即可。