

56 | 容器：大公司为保持创新，鼓励内部创业

2019-08-05 刘超

趣谈Linux操作系统

[进入课程 >](#)



讲述：刘超

时长 17:33 大小 16.08M



上一章，我们讲了虚拟化的原理。从一台物理机虚拟化出很多的虚拟机这种方式，一定程度上实现了资源创建的灵活性。但是你同时会发现，虚拟化的方式还是非常复杂的。这有点儿像，你去成立子公司，虽然说公司小，但毕竟是一些独立的公司，麻雀虽小，五脏俱全，因而就像上一章我们看到的那样，CPU、内存、网络、硬盘全部需要虚拟化，一个都不能偷懒。

那有没有一种更加灵活的方式，既可以隔离出一部分资源，专门用于某个进程，又不需要费劲周折的虚拟化这么多的硬件呢？毕竟最终我只想跑一个程序，而不是要一整个 Linux 系统。这就像在一家大公司搞创新，如果每一个创新项目都要成立一家子公司的话，那简直太麻烦了。一般方式是在公司内部成立一个独立的组织，分配独立的资源和人力，先做一段时间的内部创业。如果真的做成功了，再成立子公司也不迟。

在 Linux 操作系统中，有一项新的技术，称为容器，它就可以做到这一点。

容器的英文叫 Container，Container 的另一个意思是“集装箱”。其实容器就像船上的不同的集装箱装着不同的货物，有一定的隔离，但是隔离性又没有那么好，仅仅做简单的封装。当然封装也带来了好处，一个是打包，二是标准。

在没有集装箱的时代，假设我们要将货物从 A 运到 B，中间要经过三个码头、换三次船。那么每次都要将货物卸下船来，弄得乱七八糟，然后还要再搬上船重新摆好。因此在没有集装箱的时候，每次换船，船员们都要在岸上待几天才能干完活。

有了尺寸全部都一样的集装箱以后，我们可以把所有的货物都打包在一起。每次换船的时候，把整个集装箱搬过去就行了，几个小时就能完成。船员换船时间大大缩短了。这是集装箱的“打包”和“标准”两大特点在生活中的应用。

其实容器的思想就是要变成软件交付的集装箱。那么容器如何对应用打包呢？

我们先来学习一下集装箱的打包过程。首先，我们得有个封闭的环境，将货物封装起来，让货物之间互不干扰，互相隔离，这样装货卸货才方便。

容器实现封闭的环境主要要靠两种技术，一种是看起来是隔离的技术，称为 **namespace**（命名空间）。在每个 namespace 中的应用看到的，都是不同的 IP 地址、用户空间、进程 ID 等。另一种是用起来是隔离的技术，称为 **cgroup**（网络资源限制），即明明整台机器有很多的 CPU、内存，但是一个应用只能用其中的一部分。

有了这两项技术，就相当于我们焊好了集装箱。接下来的问题就是，如何“将这些集装箱标准化”，在哪艘船上都能运输。这里就要用到镜像了。

所谓**镜像**（Image），就是在你焊好集装箱的那一刻，将集装箱的状态保存下来。就像孙悟空说：“定！”，集装箱里的状态就被“定”在了那一刻，然后这一刻的状态会被保存成一系列文件。无论在哪里运行这个镜像，都能完整地还原当时的情况。

当程序员根据产品设计开发完毕之后，可以将代码连同运行环境打包成一个容器镜像。这个时候集装箱就焊好了。接下来，无论是在开发环境、测试环境，还是生产环境运行代码，都可以使用相同的镜像。就好像集装箱在开发、测试、生产这三个码头非常顺利地整体迁移，这样产品的发布和上线速度就加快了。

下面，我们就来体验一下这个 Linux 上的容器技术！

首先，我们要安装一个目前最主流的容器技术的实现 Docker。假设我们的操作系统是 CentOS，你可以参考<https://docs.docker.com/install/linux/docker-ce/centos/>这个官方文档，进行安装。

第一步，删除原有版本的 Docker。

复制代码

```
1 yum remove docker \
2     docker-client \
3     docker-client-latest \
4     docker-common \
5     docker-latest \
6     docker-latest-logrotate \
7     docker-logrotate \
8     docker-engine
```

第二步，安装依赖的包。

复制代码

```
1 yum install -y yum-utils \
2     device-mapper-persistent-data \
3     lvm2
```

第三步，安装 Docker 所属的库。

复制代码

```
1 yum-config-manager \
2     --add-repo \
3     https://download.docker.com/linux/centos/docker-ce.repo
4
```

第四步，安装 Docker。

```
1 yum install docker-ce docker-ce-cli containerd.io
```

第五步，启动 Docker。

```
1 systemctl start docker
```

Docker 安装好之后，接下来我们就来运行一个容器。

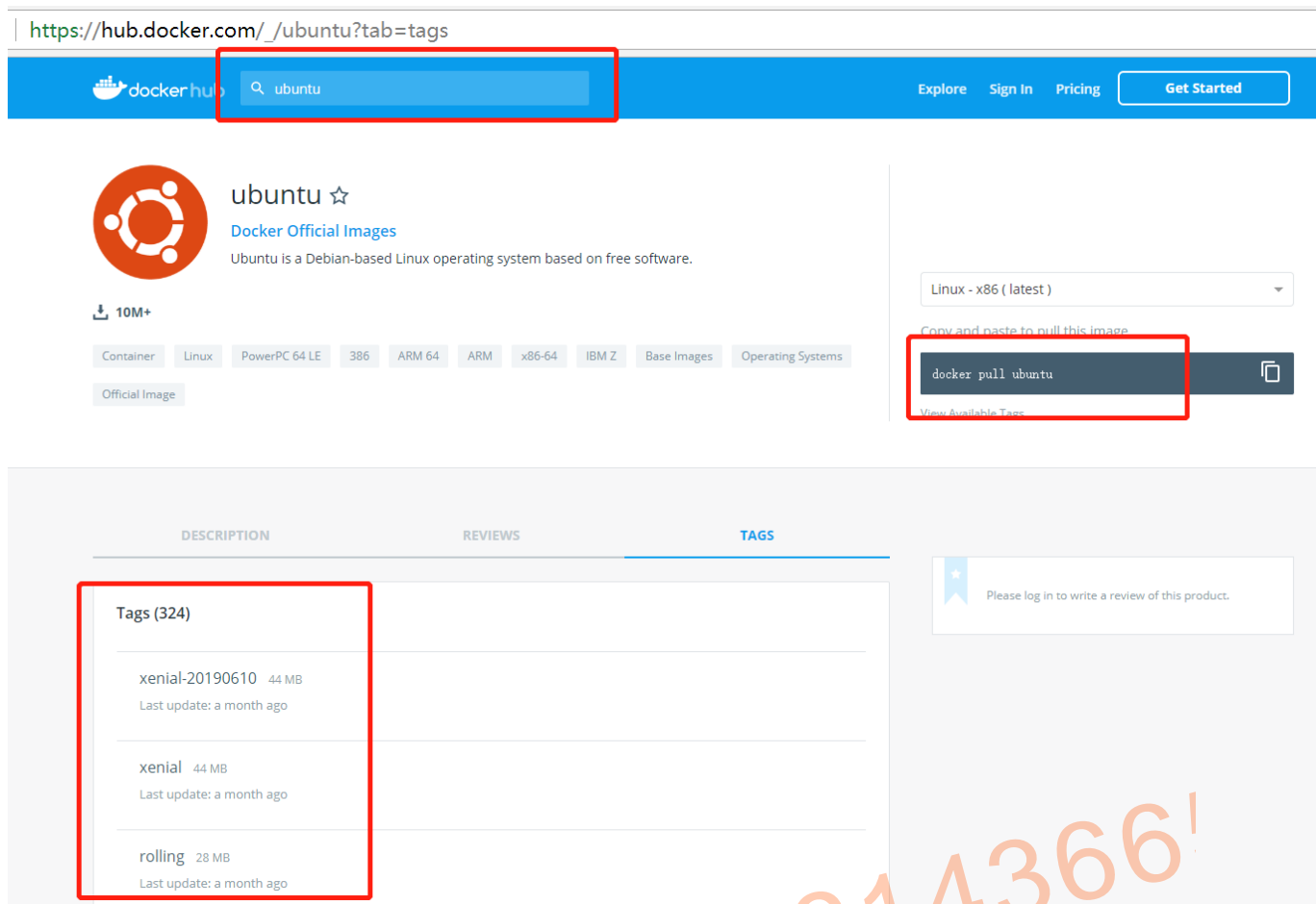
就像上面我们讲过的，容器的运行需要一个镜像，这是我们集装箱封装的那个环境，在 <https://hub.docker.com/> 上，你能找到你能想到的几乎所有环境。

最基础的环境就是操作系统。

咱们最初讲命令行的时候讲过，每种操作系统的命令行不太一样，就像黑话一样。有时候我们写一个脚本，需要基于某种类型的操作系统，例如，Ubuntu 或者 CentOS。但是，Ubuntu 或者 CentOS 不同版本的命令也不一样，需要有一个环境尝试一下命令是否正确。

最常见的做法是有几种类型的操作系统，就弄几台物理机。当然这样一般人可玩不起，但是有了虚拟机就好一些了。你可以在你的笔记本电脑上创建多台虚拟机，但是这个时候又会有另一个苦恼，那就是，虚拟机往往需要比较大的内存，一般一台笔记本电脑上无法启动多台虚拟机，所以做起实验来要经常切换虚拟机，非常麻烦。现在有了容器，好了，我们可以在一台虚拟机上创建任意的操作系统环境了。

比方说，你可以在 <https://hub.docker.com/> 上搜索 Ubuntu。点开之后，找到 Tags。镜像都有 Tag，这是镜像制作者自己任意指定的，多用于表示这个镜像的版本号。



如果仔细看这些 Tags，我们会发现，哪怕非常老版本的 Ubuntu，这里面都有，例如 14.04。如果我们突然需要一个基于 Ubuntu 14.04 的命令，那就不需要费劲去寻找、安装一个这么老的虚拟机，只要根据命令下载这个镜像就可以了。

复制代码

```
1 # docker pull ubuntu:14.04
2 14.04: Pulling from library/ubuntu
3 a7344f52cb74: Pull complete
4 515c9bb51536: Pull complete
5 e1eabe0537eb: Pull complete
6 4701f1215c13: Pull complete
7 Digest: sha256:2f7c79927b346e436cc14c92bd4e5bd778c3bd7037f35bc639ac1589a7acfa90
8 Status: Downloaded newer image for ubuntu:14.04
```

下载完毕之后，我们可以通过下面的命令查看镜像。


复制代码

```
1 # docker images
2 REPOSITORY          TAG                 IMAGE ID           CREATED            SIZE
3 ubuntu              14.04             2c5e00d77a67      2 months ago      188MB
```

有了镜像，我们就可以通过下面的启动一个容器啦。


启动一个容器需要一个叫 `entrypoint` 的东西，也就是入口。一个容器启动起来之后，会从这个指令开始运行，并且只有这个指令在运行，容器才启动着。如果这个指令退出，整个容器就退出了。

因为我们想尝试命令，所以这里 `entrypoint` 要设置为 `bash`。通过 `cat /etc/lsb-release`，我们可以看出，这里面已经是一个老的 Ubuntu 14.04 的环境。

 复制代码

```
1 # docker run -it --entrypoint bash ubuntu:14.04
2 root@0e35f3f1fbc5:/# cat /etc/lsb-release
3 DISTRIB_ID=Ubuntu
4 DISTRIB_RELEASE=14.04
5 DISTRIB_CODENAME=trusty
6 DISTRIB_DESCRIPTION="Ubuntu 14.04.6 LTS"
```


如果我们想尝试 `centOS 6`，也是没问题的。

 复制代码

```
1 # docker pull centos:6
2 6: Pulling from library/centos
3 ff50d722b382: Pull complete
4 Digest: sha256:dec8f471302de43f4cfcf82f56d99a5227b5ea1aa6d02fa56344986e1f4610e7
5 Status: Downloaded newer image for centos:6
6
7 # docker images
8 REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
9 ubuntu              14.04       2c5e00d77a67     2 months ago    188MB
10 centos              6           d0957ffdf8a2     4 months ago    194MB
11
12 # docker run -it --entrypoint bash centos:6
13 [root@af4c8d598bdf /]# cat /etc/redhat-release
14 CentOS release 6.10 (Final)
```

除了可以如此简单地创建一个操作系统环境，容器还有一个很酷的功能，就是镜像里面带应用。这样的话，应用就可以像集装箱一样，到处迁移，启动即可提供服务。而不用像虚拟机那样，要先有一个操作系统的环境，然后再在里面安装应用。

我们举一个最简单的应用的例子，也就是 nginx。我们可以下载一个 nginx 的镜像，运行起来，里面就自带 nginx 了，并且直接可以访问了。

 复制代码

```
1 # docker pull nginx
2 Using default tag: latest
3 latest: Pulling from library/nginx
4 fc7181108d40: Pull complete
5 d2e987ca2267: Pull complete
6 0b760b431b11: Pull complete
7 Digest: sha256:48cbeee0cb0a3b5e885e36222f969e0a2f41819a68e07aeb6631ca7cb356fed1
8 Status: Downloaded newer image for nginx:latest
9
10 # docker run -d -p 8080:80 nginx
11 73ff0c8bea6e169d1801afe807e909d4c84793962cba18dd022bfad9545ad488
12
13 # docker ps
14 CONTAINER ID      IMAGE          COMMAND                  CREATED           STATUS
15 73ff0c8bea6e      nginx         "nginx -g 'daemon of..." 2 minutes ago    Up
16
17 # curl http://localhost:8080
18 <!DOCTYPE html>
19 <html>
20 <head>
21 <title>Welcome to nginx!</title>
```

这次 nginx 镜像运行的方式和操作系统不太一样，一个是 -d，因为它是一个应用，不需要像操作系统那样有交互命令行，而是以后台方式运行，-d 就是 daemon 的意思。


另外一个就是端口 -p 8080:80。容器这么容易启动，每台机器上可以启动 N 个 nginx。大家都监听 80 端口，不就冲突了吗？所以我们要设置端口，冒号后面的 80 是容器内部环境监听的端口，冒号前面的 8080 是宿主机上监听的端口。

一旦容器启动起来之后，通过 docker ps 就可以查看都有哪些容器正在运行。

接下来，我们通过 curl 命令，访问本机的 8080 端口，可以打印出 nginx 的欢迎页面。

docker run 一下，应用就启动起来了，是不是非常方便？nginx 是已经有人打包好的容器镜像，放在公共的镜像仓库里面。如果是你自己开发的应用，应该如何打包成为镜像呢？

因为 Java 代码比较麻烦，我们这里举一个简单的例子。假设你自己写的 HTML 的文件就是代码。

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Welcome to nginx Test 7!</title>
5     <style>
6       body {
7         width: 35em;
8         margin: 0 auto;
9         font-family: Tahoma, Verdana, Arial, sans-serif;
10      }
11    </style>
12  </head>
13  <body>
14    <h1>Test 7</h1>
15    <p>If you see this page, the nginx web server is successfully installed and
16    working. Further configuration is required.</p>
17    <p>For online documentation and support please refer to
18    <a href="http://nginx.org/">nginx.org</a>.<br/>
19    Commercial support is available at
20    <a href="http://nginx.com/">nginx.com</a>.</p>
21    <p><em>Thank you for using nginx.</em></p>
22  </body>
23 </html>
```

那我们如何将这些代码放到容器镜像里面呢？要通过 Dockerfile，Dockerfile 的格式应该包含下面的部分：

FROM 基础镜像

RUN 运行过的所有命令

COPY 拷贝到容器中的资源

ENTRYPOINT 前台启动的命令或者脚本

按照上面说的格式，可以有下面的 Dockerfile。

[📄 复制代码](#)

```
1 FROM ubuntu:14.04
2 RUN echo "deb http://archive.ubuntu.com/ubuntu trusty main restricted universe multiver:
3 RUN echo "deb http://archive.ubuntu.com/ubuntu trusty-updates main restricted universe r
4 RUN apt-get -y update
5 RUN apt-get -y install nginx
6 COPY test.html /usr/share/nginx/html/test.html
7 ENTRYPOINT nginx -g "daemon off;"
```

将代码、Dockerfile、脚本，放在一个文件夹下，以上面的 Dockerfile 为例子。

[📄 复制代码](#)

```
1 [nginx]# ls
2 Dockerfile  test.html
```

现在我们编译这个 Dockerfile。

[📄 复制代码](#)

```
1 docker build -f Dockerfile -t testnginx:1 .
```

编译过后，我们就有了一个新的镜像。

[📄 复制代码](#)


```
1 # docker images
2 REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
3 testnginx           1                  3b0e5da1a384       11 seconds ago    221MB
4 nginx               latest             f68d6e55e065       13 days ago       109MB
5 ubuntu              14.04             2c5e00d77a67       2 months ago      188MB
6 centos              6                 d0957ffdf8a2       4 months ago      194MB
```

接下来，我们就可以运行这个新的镜像。

[📄 复制代码](#)

```
1 # docker run -d -p 8081:80 testnginx:1
2 f604f0e34bc263bc32ba683d97a1db2a65de42ab052da16df3c7811ad07f0dc3
3 # docker ps
4 CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
5 f604f0e34bc2        testnginx:1        "/bin/sh -c 'nginx -..."   2 seconds ago      Up                  8081->8081
6 73ff0c8bea6e        nginx              "nginx -g 'daemon of..."   33 minutes ago     Up
```

我们再来访问我们在 nginx 里面写的代码。

 复制代码

```
1 [root@deployer nginx]# curl http://localhost:8081/test.html
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <title>Welcome to nginx Test 7!</title>
```

看，我们的代码已经运行起来了。是不是很酷？

其实这种运行方式又更加酷的功能。

第一就是持续集成。

想象一下，你写了一个程序，然后把它打成了上面一样的镜像。你在本地一运行 docker run 就把他运行起来了。接下来，你交给测试的就不是一个“程序包 + 配置 + 手册”了，而是一个容器镜像了。测试小伙伴同样通过 docker run 也就运行起来了，不存在“你这里跑的起来，他那里跑不起来的情况”。测试完了再上生产，交给运维的小伙伴也是这样一个镜像，同样的运行同样的顺畅。这种模式使得软件的交付效率大大提高，可以一天发布多次。

第二就是弹性伸缩。

想象一下，你写了一个程序，平时用的人少，只需要 10 个副本就能够扛住了。突然有一天，要做促销，需要 100 个副本，另外 90 台机器创建出来相对比较容易，用任何一个云都可以做到，但是里面 90 个副本的应用如何部署呢？一个个上去手动部署吗？有了容器就方便多了，只要在每台机器上 docker run 一下就搞定了。

第三就是跨云迁移。

如果你不相信任何一个云，怕被一个云绑定，怕一个云挂了自己的应用也就挂了。那我们想一想，该怎么办呢？你只能手动在一个云上部署一份，在另外一个云上也部署一份。有了容器了之后，由于容器镜像对于云是中立的，你在这个云上 docker run，就在这个云上提供服务，等哪天想用另一个云了，不用怕应用迁移不走，只要在另外一个云上 docker run 一下就解决了。

到现在，是不是能够感受到容器的集装箱功能了，这就是看起来隔离的作用。

你可能会问，多个容器运行在一台机器上，不会相互影响吗？如何限制 CPU 和内存的使用呢？

Docker 本身提供了这样的功能。Docker 可以限制对于 CPU 的使用，我们可以分几种的方式。

Docker 允许用户为每个容器设置一个数字，代表容器的 CPU share，默认情况下每个容器的 share 是 1024。这个数值是相对的，本身并不能代表任何确定的意义。当主机上有多个容器运行时，每个容器占用的 CPU 时间比例为它的 share 在总额中的比例。

Docker 为容器设置 CPU share 的参数是 `-c --cpu-shares`。

Docker 提供了 `--cpus` 参数可以限定容器能使用的 CPU 核数。

Docker 可以通过 `--cpuset` 参数让容器只运行在某些核上

Docker 会限制容器内存使用量，下面是一些具体的参数。

`-m --memory`：容器能使用的最大内存大小。

`--memory-swap`：容器能够使用的 swap 大小。

`--memory-swappiness`：默认情况下，主机可以把容器使用的匿名页 swap 出来，你可以设置一个 0-100 之间的值，代表允许 swap 出来的比例。

`--memory-reservation`：设置一个内存使用的 soft limit，如果 docker 发现主机内存不足，会执行 OOM (Out of Memory) 操作。这个值必须小于 `--memory` 设置的值。

`--kernel-memory`：容器能够使用的 kernel memory 大小。

-oom-kill-disable：是否运行 OOM (Out of Memory) 的时候杀死容器。只有设置了 -m，才可以把这个选项设置为 false，否则容器会耗尽主机内存，而且导致主机应用被杀死。

这就是用起来隔离的效果。

那这些看起来隔离和用起来隔离的技术，到内核里面是如何实现的呢？我们下一节仔细分析。

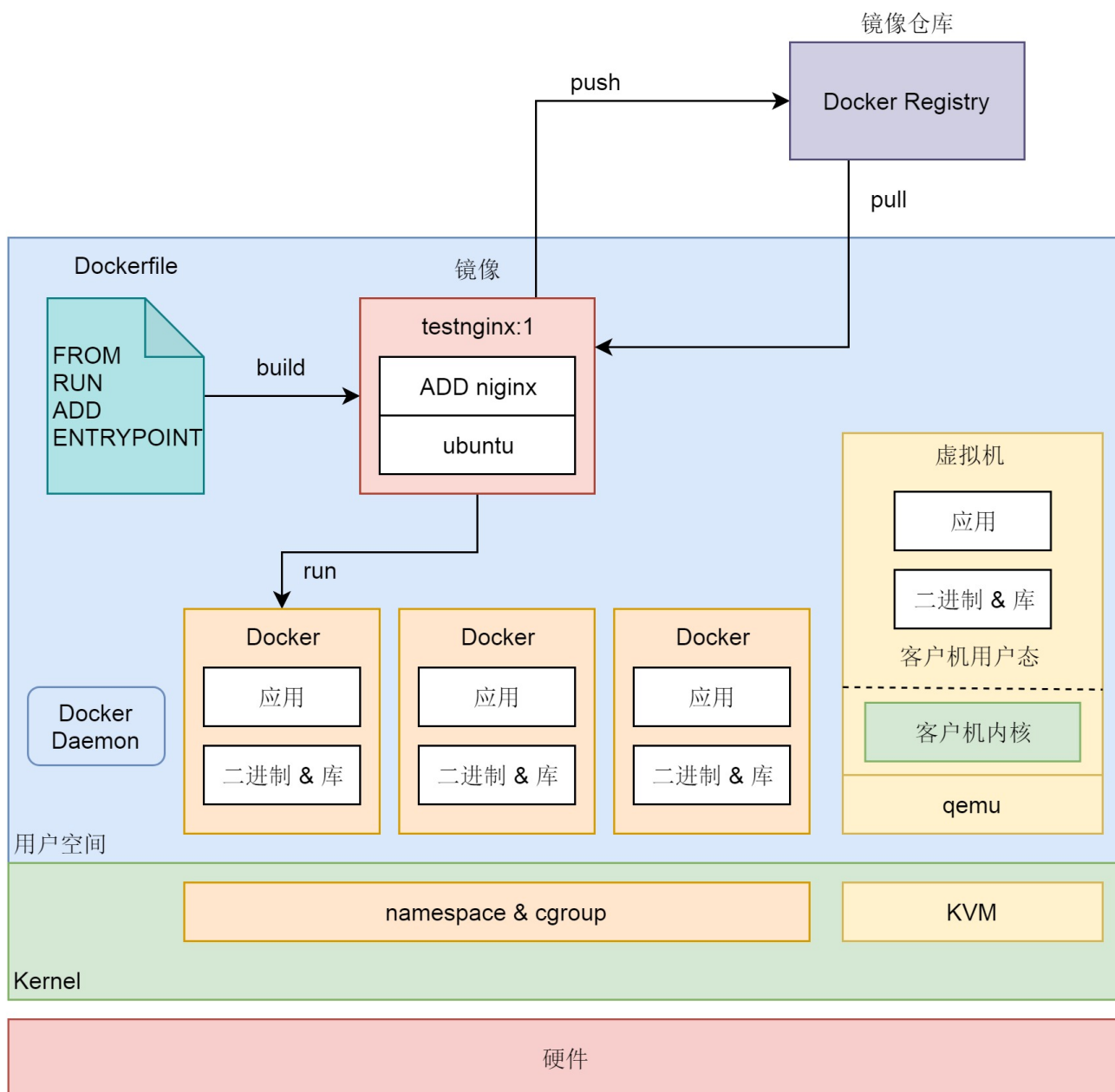
总结时刻

这里我们来总结一下这一节的内容。无论是容器，还是虚拟机，都依赖于内核中的技术，虚拟机依赖的是 KVM，容器依赖的是 namespace 和 cgroup 对进程进行隔离。

为了运行 Docker，有一个 daemon 进程 Docker Daemon 用于接收命令行。

为了描述 Docker 里面运行的环境和应用，有一个 Dockerfile，通过 build 命令称为容器镜像。容器镜像可以上传到镜像仓库，也可以通过 pull 命令从镜像仓库中下载现成的容器镜像。

通过 Docker run 命令将容器镜像运行为容器，通过 namespace 和 cgroup 进行隔离，容器里面不包含内核，是共享宿主机的内核的。对比虚拟机，虚拟机在 qemu 进程里面是有客户机内核的，应用运行在客户机的用户态。



课堂练习

请你试着用 Tomcat 的容器镜像启动一个 Java 网站程序，并进行访问。

欢迎留言和我分享你的疑惑和见解，也欢迎收藏本节内容，反复研读。你也可以把今天的内容分享给你的朋友，和他一起学习和进步。

趣谈 Linux 操作系统

像故事一样的操作系统入门课

刘超

网易杭州研究院

云计算技术部首席架构师



新版升级：点击「👤 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 55 | 网络虚拟化：如何成立独立的合作部？

精选留言

写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。