Docker 的前世今...

> 这是本专栏的第一部分：Docker 入门，共 3 篇，将带大家进入 Docker 的世界。首先了解 Docker 的发展历程，然后快速掌握 Docker 的基本使用；Docker 版本及内核兼容性选择是这部分的重点内容。大家如果在生产环境中需要使用 Docker 的话，建议重点关注这一篇。
>
> 下面我们就进入第一篇的内容。

Docker 在大多数人眼中几乎是容器（container）的代名词，即使是现在我也常会听到有人说“我在服务器上启动了 N 个 docker 在跑 XX 服务” 之类的话。

我们来看看为何 Docker 能成为容器的代名词，引领容器的时代。

容器技术的发展

chroot

在我看来，要梳理容器技术的发展，最早可以追溯到 1979 年，那时候 chroot 系统调用首次问世。

之后 1982 年 chroot 机制被移植到了 BSD 系统上，再后来便是我们所熟知的 Linux 系统上的 chroot 了。

我们有时会使用 chroot 改变某进程的根目录，使它不能访问该目录之外的其他目录。

这和我们在一个容器内的感觉很像了。事实上在几年前确实有人用一百多行的 bash 利用 chroot 写了一个模拟 Docker 创建容器的实现，称之为 bocker，有兴趣的读者可以去看看该项目的代码。

这里我们来介绍一个实际使用 chroot 创建隔离环境的例子：

比方说我们想创建一个 Debian 的隔离环境，那可以使用 chroot 将 Debian 的 rootfs 根文件系统作为新进程的根。至于 rootfs 如何获得，我们这里用 Docker 来完成，但这不是本节的重点，可以暂时忽略它，在以后的章节中我们会详细介绍。

复制

```
# 创建一个空文件夹
(MoeLove) ➜  ~ mkdir chroot-dir
(MoeLove) ➜  ~ cd chroot-dir
# 使用 Docker 来提取 Debian 的 rootfs
(MoeLove) ➜  chroot-dir docker save -o debian.tar debian:buster
(MoeLove) ➜  chroot-dir ls
debian.tar
(MoeLove) ➜  chroot-dir tar -xf debian.tar
(MoeLove) ➜  chroot-dir ls
098963abf3c3b87b8114ff67d164097dfac2d5659e39f9beb5604db91585f375.json  debian.tar
0f28619fe69181d3af529d56692f1362b7a7c8a6bf8dc9ab0d6d4f9ef9b0004d       manifest.js
(MoeLove) ➜  chroot-dir mkdir -p debian
(MoeLove) ➜  chroot-dir tar -C debian -xf 0f28619fe69181d3af529d56692f1362b7a7c8
(MoeLove) ➜  chroot-dir ls debian
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv
```

经过上述步骤，就拿到了 Debian 的 rootfs，接下来看看 chroot 的能力：

复制

```
# 使用 debian 文件夹作为容器的根
(MoeLove) ➜  chroot-dir sudo chroot debian /bin/bash -i
[sudo] tao 的密码：
root@localhost:/# whoami
root
root@localhost:/# cat /etc/os-release
PRETTY_NAME="Debian GNU/Linux 10 (buster)"
NAME="Debian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

此时我们就已经在一个“容器”内了，来看下使用这个“容器”能做些什么。

首先看看当前“容器”内的路由表：

复制

```
root@localhost:/# mkdir -p /sys
root@localhost:/# mount -t sysfs sys /sys
root@localhost:/# ip r
default via 192.168.0.1 dev wlp2s0 proto dhcp metric 600
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
192.168.0.0/24 dev wlp2s0 proto kernel scope link src 192.168.0.108 metric 600
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
```

另外，还可以将 /proc 也挂载进去：

复制

```
root@localhost:/# mkdir -p /proc
root@localhost:/# mount -t proc proc /proc
# 这里是随意找了一个进程进行查看
root@localhost:/# ls -al  /proc/31730/ns/pid
lrwxrwxrwx. 1 1000 1000 0 Jul 29 16:47 /proc/31730/ns/pid -> 'pid:[4026531836]'
```

可以看到在这个“容器”内，可以访问到主机上的进程信息，这表示没有任何的进程隔离，带来的危险是我们甚至可以在这个“容器”内杀掉主机上的进程，或者通过“容器”来攻击主机。

为了能更好的解决这个问题，接下来出现了另一个技术：Namespace。

Namespace

Namespace 是在 2002 年由 Linux 2.4.19 开始加入内核的特性，它 的主要作用是做了一层抽象和隔离，使得在 namespace 中的进程 / 进程组可以看起来拥有自己的独立资源，具体的“资源”表现形式取决于给它赋予了哪些 namespace。

随着 2013 年 Linux 3.8 中 user namespace 的引入，对于我们现在所熟知的容器所需的全部 namespace 就都实现了：mnt、pid、net、ipc、uts、user 和 cgroup 对于这些 Namespace 和 Docker 的关系，我们在以后的章节中都会深入学习，这里先对 Namespace 做下介绍，便于大家了解容器技术的发展。

我们可以通过三个系统调用直接操作 Namespace，这三个系统调用分别是：

- clone，可以通过传递不同 namespace 的标志来为新的（子）进程指定其所属的 namespace；
- unshare，允许一个进程（或线程）取消当前与其他进程（或线程）共享的执行上下文；
- setns，进入文件描述符指定的 namespace。

知道了这些基础知识后，我们回到前面“容器”的内容中。

我们在前面 chroot 的例子中看到没能做到进程隔离，现在来试试看用 namespace 完成该需求。

复制

```
(MoeLove) ➜  ~ sudo unshare -fp --mount-proc -n
[sudo] tao 的密码：
[root@localhost]/home/tao# ps -a
  PID TTY          TIME CMD
    1 pts/15   00:00:00 zsh
   33 pts/15   00:00:00 ps
```

这里很明显，我们当前所在进程的 PID 为 1 并且看不到宿主机上的其他进程，达到了基础的隔离效果。

cgroups 和 LXC

另一条关键的时间线在 2008 年，cgroups 进入 Linux 2.6.24 后，基于它并且瞄准容器世界的一个项目诞生了。

Linux Container （LXC）结合了 namespace 和 cgroups 等技术，目标就是要创造出运行在 Linux 系统中，并且隔离性良好的容器环境。

LXC 的发布在 2008 年，但值得注意的是 cgroups 最初是由 Google 的工程师开发的，最早的记录是在 2006 年，事实上当时 Google 确实也在做类似的容器化项目。

Docker 的发展

时间一晃而过，就到了 2013 年的 PyCon 上，在这次大会上 Docker 正式面世。而它当时其实也只是构建在 LXC 之上的一个工具，屏蔽掉了 LXC 的使用细节，让用户可以一句 `docker run` 命令行便创建出自己的容器环境。

同时，它允许用户将容器环境打包成为一个 Docker 镜像进行分发，这也大大降低了用户使用的门槛。Docker 镜像分发可以说是 Docker 成功的一个关键要素了。

另一个关键要素，我认为是开源生态，Docker 在首次亮相之后不久，就完全开源了，吸引了来自世界各地开发者的关注和积极贡献。

2014 年 Docker 发布 1.0 正式进入生产就绪的状态。在此之前它也将 LXC 逐步从它的底层移除，换成了自己实现的 libcontainer，幸运的是我也在 0.9 版本时开始了我的 Docker 之路。

此后 Docker 便成为了风靡技术界的新热潮。

再后来也出现过 Swarm Mesos 和 Kubernetes 等众多容器编排系统争夺市场份额的情况，但是随着时间的推移，Kubernetes 成为了事实上的标准，国内外各个公司也都在推进 Kubernetes 的落地和实施。而 Docker 作为容器运行时，也正是极其关键的一环。

总结

从最先的 chroot 到后来的 namespace 和 cgroups 等，再到后来的 LXC 和 Docker 等技术的出现，这都是在容器技术领域的一种探索和前进。

当然，我这里只列出了我认为最核心和关键的内容，事实上当时技术圈除了这条主干之外，也有着很多支线剧情，比如从各类 PaaS （Platform as a Service）厂商的竞争到全面拥抱 Docker，这些内容也从侧面来反映出了 Docker 的火热。