深入存储驱动：...

本篇是第五部分"存储篇"的最后一篇，前两篇我主要为你介绍了 Docker volume 及其应用，本篇我将为你介绍 Docker 现在推荐的存储驱动 Overlay2。

本篇，我将为你介绍 Docker 现在推荐使用的存储驱动 Overlay2，在开始之前，你可以执行以下命令来查看 Docker 正在使用的存储驱动：

复制

```
(MoeLove) ➜  ~ docker info --format '{{.Driver}}'
overlay2
```

如果你看到的结果也是 Overlay2 说明你的 Docker 已经再用 Overlay2 存储驱动了。

你也可能会看到其他不同的结果，可以在启动 Docker Daemon 的时候，通过 `--storage-driver` 参数进行指定，也可以在 /etc/docker/daemon.json 文件中通过 storage-driver 字段进行配置。

目前对于 Docker v19.03 而言，你有以下几种存储驱动可供选择：

- BTRFS
- ZFS
- Overlay2
- AUFS
- Overlay
- Devicemapper
- VFS

但它们对于你使用的文件系统之类的都有不同的要求，且实现方式也不尽相同。我仍然以本篇的重点 Overlay2 存储驱动为例，它需要你使用 Linux 4.x 以上版本的内核，或者是对于 RHEL/CentOS 等需要使用 3.10.0-514 以上的内核（Docker 版本及内核兼容性选择，我在之前内容中已经介绍过了）。

同时，它支持你使用 ext4 的文件系统，或者增加了 `ftype=1` 的 XFS 文件系统。可以通过 `docker info` 进行得到文件系统相关的信息。

复制

```
# 省略了部分输出
(MoeLove) ➜  ~ docker info
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
```

存储驱动的作用

前面虽然已经聊了如何设置和检查当前在用的存储驱动，但尚未介绍为何一定要使用存储驱动，以及它的作用。

还记得我在镜像篇《构建镜像和分发》中为你介绍的 Docker 如何存储镜像相关的内容吗，如果忘了可以回头复习一下。

Docker 将容器镜像做了分层存储，每个层相当于包含着一条 Dockerfile 的指令。而这些层在磁盘上的存储方式，以及在启动容器时，如何组织这些层，并提供可写层，便是存储驱动的主要作用了。

另外需要注意的是：不同的存储驱动实现不同，性能也有差异，同时使用不同的存储驱动也会导致占用的磁盘空间有所不同。

同时：由于它们的实现不同，当你修改存储驱动后，可能会导致看不到原有的镜像、容器等，这是正常的，不必担心，切换回原先的驱动即可见。

OverlayFS

了解完前面的背景知识后，你也看到了我刚才列出的可用存储驱动中有两个 Overlay 和 Overlay2，其实 Overlay2 算是 Overlay 的升级版，这两个存储驱动所用的都是 OverlayFS。

Overlay 驱动是在 2014 年 8 月份首次进入 Docker 的，而 Overlay2 则是在 2016 年 6 月份被合并，并首次出现在 Docker 1.12 中的。它的出现是为了解决 Overlay 存储驱动可能早层 inode 耗尽的问题。

聊完 Overlay 和 Overlay2，我们将重点回归到 OverlayFS 上。

我们启动一个容器，以此为切入点来认识下 OverlayFS，注意：**以下内容使用 Linux 5.4.10 内核以及 Docker 19.03.5**，不同环境下可能结果略有差异。

复制

```
# 检查无在运行的容器和 Overlay 挂载
(MoeLove) ➜  ~ mount |grep overlay
(MoeLove) ➜  ~ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST

# 启动一个容器
(MoeLove) ➜  ~ docker run --rm -d alpine sleep 99999
caa9517ce0d799602735a30aaaaf123c07e07ff6e44c5a4b07e776af85780abe
(MoeLove) ➜  ~ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
caa9517ce0d7        alpine              "sleep 99999"       23 seconds ago      Up

# 检查 Overlay 挂载
(MoeLove) ➜  ~ mount |grep overlay
overlay on /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313b863d10f30ef447a3b
```

可以看到，在启动容器后，系统上多了一个 OverlayFS（Overlay）的挂载。注意看其中的几个内容。

1. 挂载点在：

复制

```
/var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313b863d10f30ef447a3b2f51ea9ece0
```

复制

```
(MoeLove) ➜  ~ sudo ls /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313b
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tm
```

其中的内容，看着很熟悉，是我们所启动容器根目录中的内容。为了验证这一说法，我在容器中新写一个文件：

复制

```
(MoeLove) ➜  ~ docker exec -it $(docker ps -ql) sh
/ # echo 'Hello Docker' > docker-course
```

再次查看此挂载点中的内容：

复制

```
(MoeLove) ➜  ~ sudo ls  /var/lib/docker/overlay2/22be5e4dc4541a60aa4f6de628c3
bin  dev  docker-course  etc  home  lib  media  mnt  opt  proc  root  run  sbi
(MoeLove) ➜  ~ sudo cat /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313
Hello Docker
```

可以看到刚才写的内容已经在这个挂载点的目录中了。

2. lowerdir：这是 OverlayFS 中必要的目录。

这个 lowerdir 中包含两个目录，这是使用了内核对 OverlayFS multi layer 特性的支持，我们分别查看下其中内容：

复制

```
(MoeLove) ➜  ~ sudo ls -a /var/lib/docker/overlay2/l/5OO3RLRXHJPEH3IFEXNCTO4F
.  ..  dev  .dockerenv  etc
(MoeLove) ➜  ~ sudo ls -a /var/lib/docker/overlay2/l/UVA7IR67ZZTN2BNTKCZ7T6HU
.  ..  bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv
```

这两个目录，是不是看着很熟悉？

是的，它们就是我们所启动容器根目录中的大部分内容。为什么说是大部分内容呢？当我们查看其中的内容时，你也会发现它们的内容也并不完整。比如我们刚才新写入的 docker-course 文件，或者当我们查看 etc 目录下的文件，你也会发现其中都只是常规系统 /etc 目录下的部分内容。

复制

```
(MoeLove) ➜  ~ sudo ls /var/lib/docker/overlay2/l/5OO3RLRXHJPEH3IFEXNCTO4PY5/
hostname  hosts  mtab  resolv.conf
(MoeLove) ➜  ~ sudo ls /var/lib/docker/overlay2/l/UVA7IR67ZZTN2BNTKCZ7T6HUWU/
alpine-release  fstab     init.d       modprobe.d      mtab         passwd
apk             group     inittab      modules         network      periodic
conf.d          hostname  issue        modules-load.d  opt          profile
crontabs        hosts     logrotate.d  motd            os-release   profile.d
```

3. upperdir 是另一个重要的目录，我们来看看其中的内容：

复制

```
(MoeLove) ➜  ~ sudo ls -a /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d3
.  ..  docker-course  root
```

我们发现这个目录中包含着刚才创建的 docker-course 文件。同时，其中也包含一个 root 目录，这个目录便是我们默认使用的 root 用户的家目录。

如果去查看其中的内容，也会发现刚才我们执行命令的历史记录。

4. workdir 这个目录和 upperdir 在同一个父目录下，查看其内容发现里面只有一个 work 目录。

```
(MoeLove) ➜  ~ sudo ls -a /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d3
.  ..  work
```

看完以上的介绍，想必你已经发现了它们之间的部分联系，在此之前，我们在额外看一个目录，那就是 upperdir 和 workdir 以及挂载点共同的父目录：

```
(MoeLove) ➜  ~ sudo ls  /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313b863
diff  link  lower  merged  work
```

你会发现这个目录下的内容就比较直观了。我们刚才已经看了其中 diff、merged 和 work 目录的内容了，现在看看 lower 中的内容吧：

```
(MoeLove) ➜  ~ sudo cat /var/lib/docker/overlay2/f4356a8f14342008fc298bf3d313b863
l/5OO3RLRXHJPEH3IFEXNCTO4PY5:l/UVA7IR67ZZTN2BNTKCZ7T6HUWU
```

我们发现，lower 文件中的内容是以 `:` 分隔的两个 lowerdir 的目录名称。

至此，我们可以得到以下结论：

- lower 是基础层，可以包含多个 lowerdir；
- diff 是可写层，即挂载时的 upperdir，在容器内变更的文件都在这一层存储；
- merged 是最终的合并结果，即容器给我们呈现出来的结果。

Overlay2

经过前面对 Docker 启动容器后挂载的 OverlayFS 的介绍后，Overlay2 的工作流程想必你也就比较清楚了。

将镜像各层作为 lower 基础层，同时增加 diff 这个可写层，通过 OverlayFS 的工作机制，最终将 merged 作为容器内的文件目录展示给用户。

你可能会有疑问，如果只是这样简单的组织，会不会有什么限制呢？答案是肯定的，当然有限制，我们可以通过 Overlay2 的代码来看：

复制

```
// daemon/graphdriver/overlay2/overlay.go#L423
func (d *Driver) getLower(parent string) (string, error) {
// 省略部分内容
    if len(lowers) > maxDepth {
        return "", errors.New("max depth exceeded")
    }
}
```

可以看到其对 lower 的深度有硬编码的限制，当前硬编码的限制是 128。如果你在使用的过程中遇到这个错误，那表示你超过了最大深度限制，你就需要找些办法来减少层级了。

总结

本篇，我为你介绍了 OverlayFS 及 Overlay2 存储驱动相关的内容。通过实际启动容器生成的相关目录来介绍 Overlay2 的工作流程，想必通过这种方式能更易理解。

这一篇也是“存储篇”的最后一篇内容，下一篇我们将进入“安全篇”的学习，掌握与 Docker 安全相关的核心知识点。