第 2 章 企业级 Docker 虚拟化实战

Docker 是一款轻量级、高性能的虚拟化技术，是目前互联网使用最多的虚拟化技术。Docker 虚拟化技术的本质类似于集装箱机制。集装箱没有出现的时候，码头上有许多工人在搬运货物；集装箱出现以后搬运模式更加单一、更加高效，码头上更多的不是工人，而是集装箱。

将货物都打包在集装箱里面，可以防止货物之间相互影响。并且如果到了另外一个码头需要转运，有了集装箱以后，直接把它运送到另一个码头即可，完全可以保证里面的货物是整体的搬迁，并且不会损坏货物本身。

Docker 技术机制与集装箱类似。Docker 虚拟化是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，以实现虚拟化。

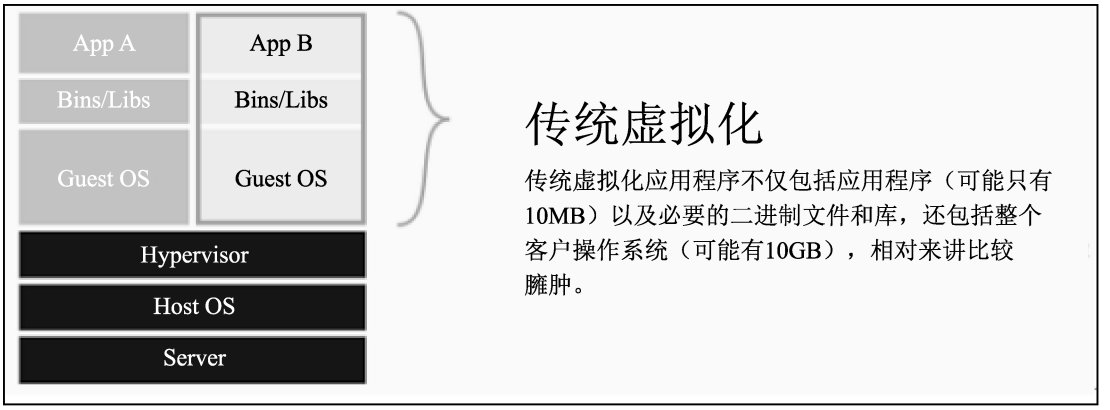
Docker 容器完全使用沙箱机制，相互之间不会有任何接口，几乎没有性能开销，可以很容易地在机器和数据中心中运行。最重要的是，它们不依赖于任何语言、框架或包括系统。

Docker 自开源后受到广泛的关注和讨论，以至于 dotCloud 公司后来都改名为 Docker Inc。

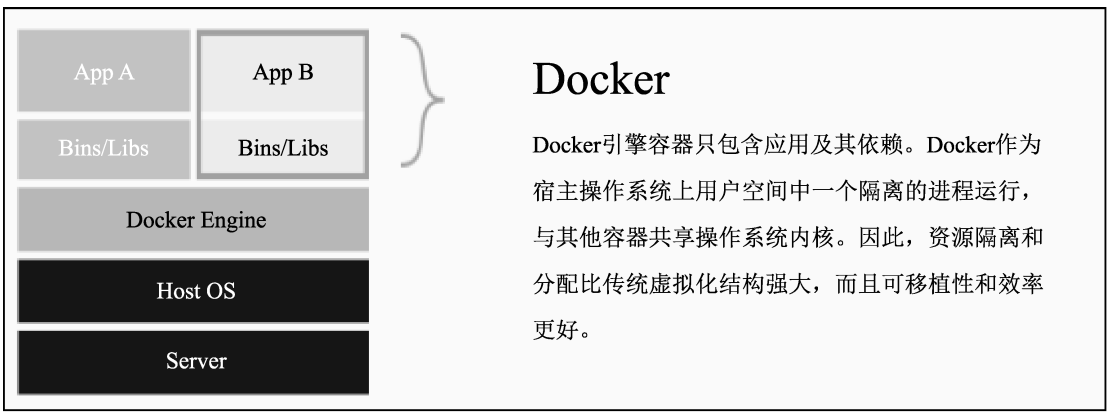
Redhat 已经在其 RHEL 6.5 中集中支持 Docker；Google 也在其 PaaS 产品中广泛应用，Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。

Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。

Docker 和传统虚拟化（KVM、XEN、Hyper-V、ESXI）结构的不同之处如图 2-1 所示。



（a）传统虚拟化结构



（b）Docker 虚拟化结构

图 2-1　Docker 和传统虚拟化结构的差异

（1）Docker 虚拟化技术概念总结如下。

Docker 虚拟化技术是在硬件的基础上，基于现有的操作系统层面上实现虚拟化，直接复用本地主机的操作系统，直接虚拟生成 Docker 容器。而 Docker 容器上部署相关的 App 应用（Apache、MySQL、PHP、Java）。

（2）传统虚拟化技术概念总结如下。

KVM、XEN、ESXI 等传统虚拟化（完全、半虚拟化）技术是在硬件的基础上，基于现有的操作系统实现虚拟化，但是不能复用本地主机的操作系统，而是必须虚拟出自己的 Guest OS 系统，然后在 Guest OS 系统上部署相关的 App 应用（如 Apache、MySQL、PHP、Java 等）。

Docker 虚拟化技术与传统虚拟化技术相比具有以下几种优点。

（1）操作启动快。

运行时性能可以获得极大提升，管理操作（启动、停止、开始、重启等）都是以 s 或 ms 为单位的。

（2）轻量级虚拟化。

可以拥有足够的“操作系统”，仅需添加或减少镜像即可。在一台服务器上可以部署 100～1000 个 Containers（容器）。但是利用传统虚拟化技术，虚拟 10～20 个虚拟机就不错了。

（3）开源免费。

Docker 虚拟化技术是开源的、免费的、低成本的，由现代 Linux 内核支持并驱动。注重轻量的 Container，可以在一个物理机上开启更多“容器”，注定比传统虚拟化技术便宜。

（4）前景及云支持。

Docker 越来越受欢迎，包括各大主流公司都在推动 Docker 的快速发展，因为其性能有很大的优势。随着 Go 语言越来越被人熟知，Docker 的使用也越来越广泛。

2.1 虚拟化技术实现方式

1）完全拟化技术

通过软件实现对操作系统的资源再分配，比较成熟。完全虚拟化代表技术有 KVM、ESXI 和 Hyper-V。

2）半虚拟化技术

通过代码修改已有的系统，形成一种新的可虚拟化的系统，并调用硬件资源去安装多个系统，整体速度上相对较高。半虚拟化代表技术为 XEN。

3）轻量级虚拟化

介于完全虚拟化技术和半虚拟化技术之间。轻量级虚拟化代表技术为 Docker。

2.2 Docker LXC 及 Cgroup 原理剖析

Docker 虚拟化技术结构体系最早为 LXC（Linux Container）和 AUFS（Another Union File System）结构组合。Docker 0.9.0 版本开始引入 LibContainer，可以视作 LXC 的替代品。

LXC 也是一种虚拟化的解决方案。和 KVM、XEN、ESXI 虚拟化技术基于硬件层面虚拟化不

同，LXC 是基于内核级的虚拟化技术，Linux 操作系统软件服务进程之所以能够相互独立，且系统能够控制每个服务进程的 CPU 和内存资源，也是得益于 LXC 容器技术。

0.9.0 版本之后的 Docker 虚拟化技术在 LXC 基础上进一步封装，比 LXC 技术更完善，并提供了一系列完整的功能。在 Docker 虚拟化技术中，LXC 主要负责资源管理，AUFS 主要负责镜像管理，而 LXC 又包括 Cgroup、NameSpace、Chroot 等组件，并通过 Cgroup 进行资源管理。从资源管理结构体系上来看，Docker、LXC、Cgroup 三者的关系如下：

Cgroup 在最底层落实资源管理，LXC 在 Cgroup 上封装了一层，Docker 又在 LXC 封装了一层。要深入掌握 Docker 虚拟化技术，需要了解负责资源管理的 Cgroup 和 LXC 相关概念和用途。Docker、LXC、Cgroup、AUFS 结构如图 2-2 所示。

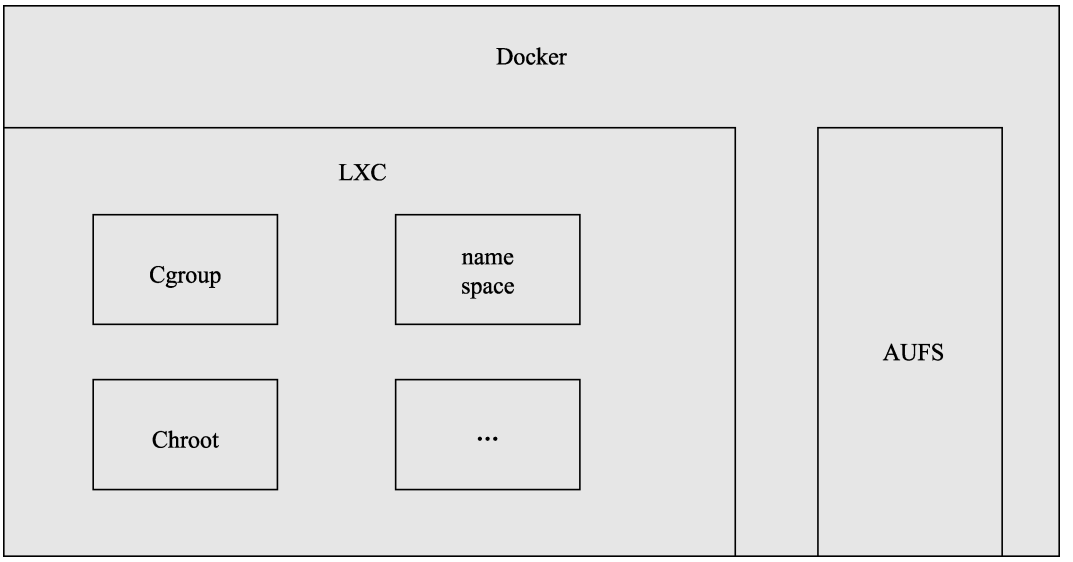


图 2-2　Docker 虚拟化内部结构

Cgroup 又名 Control group，是 Linux 内核提供的一种可以用于限制、记录、隔离进程组的物理资源（CPU、Memory、I/O、NET）的机制。

Cgroup 最初由 Google 的工程师提出，后来被整合进 Linux 内核。Cgroup 也是 LXC 为实现虚拟化所使用的资源管理手段。没有 Cgroup，就没有 LXC；没有 LXC，就没有 Docker。

Cgroup 最初的目标是为资源管理提供一个统一的框架，既整合现有的 Cpuset 等子系统，也为未来开发新的子系统提供接口。现在的 Cgroup 适用于多种应用场景，从单个进程的资源控制，到实现操作系统层次的虚拟化（OS Level Virtualization）。

LXC 可以提供轻量级的虚拟化，以便隔离进程和资源，且不需要提供指令解释机制及全虚拟化的其他复杂性。容器有效地将由单个操作系统管理的资源划分到相互独立的组中，以更好地在组间平衡相互冲突的资源使用需求。

LXC 建立在 Cgroup 基础上，可以理解为 LXC=Cgroup+ namespace + Chroot + veth +用户态控制脚本。

LXC 利用内核的新特性（Cgroup）提供用户空间的对象，用来保证资源的隔离和对应用或系统的资源控制。

典型的 Linux 文件系统由 bootfs 和 rootfs 两部分组成，bootfs（boot file system）主要包含 Bootloader 和 Kernel。Bootloader 主要用于引导加载 Kernel，当 Kernel 被加载到内存中后，bootfs 就被卸载。rootfs（root file system）包含的就是典型 Linux 系统中的/dev、/proc、/bin、/etc 等目录和文件，如图 2-3 所示。

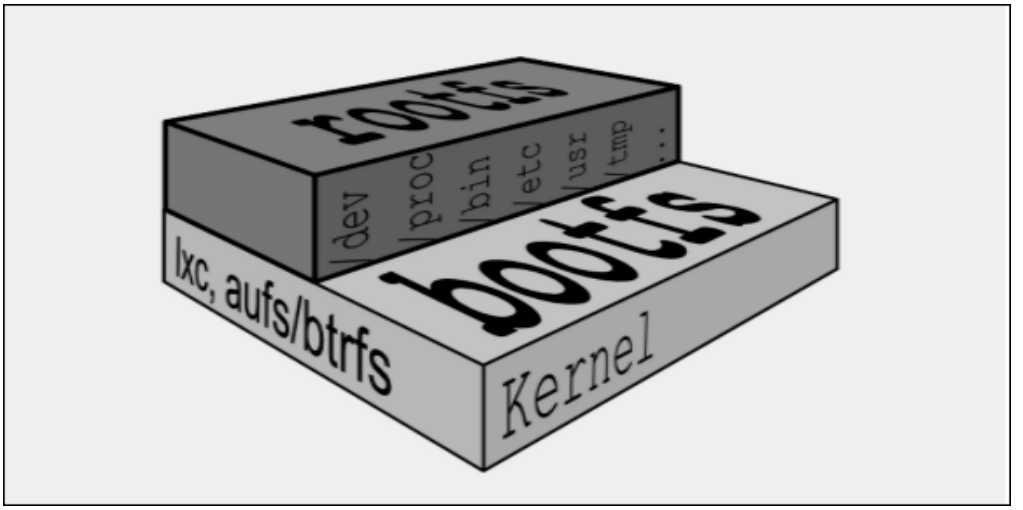


图 2-3　bootfs 和 rootfs 结构图

Docker 容器的文件系统最早是建立在 AUFS 基础上的。AUFS 是一种 UnionFS，简单来说就是支持将不同的目录挂载到同一个虚拟文件系统下，并实现一种 layer 的概念。由于 AUFS 未能加入 Linux 内核，考虑到兼容性问题，加入了 devicemapper 的支持。

Docker 虚拟化磁盘文件系统，默认使用 devicemapper 方式。Docker 虚拟化目前支持 6 种文件系统：AUFS、Btrfs、Device Mapper、OverlayFS、ZFS、VFS，如图 2-4 所示。

Technology（技术）	Storage driver name（存储驱动程序名）
OverlayFS	`overlay`
AUFS	`aufs`
Btrfs	`btrfs`
Device Mapper	`devicemapper`
VFS	`vfs`
ZFS	`zfs`

图 2-4　Docker 虚拟化支持的文件系统

2.3　AUFS 简介

AUFS 将挂载到同一虚拟文件系统下的多个目录分别设置成 read-only（只读）、read-write（读写）以及 whiteout-able 权限，对 read-only 目录只能读，而写操作只能实施在 read-write 目录中。重点在于，写操作是在 read-only 上的一种增量操作，不影响 read-only 目录。

挂载目录时要严格按照各目录之间的这种增量关系，将被增量操作的目录优先于在它基础上增量操作的目录挂载，待所有目录挂载结束后，继续挂载一个 read-write 目录，如此便形成了一种层次结构。

传统的 Linux 加载 bootfs 时会先将 rootfs 设为 read-only，在系统自检之后将 rootfs 从 read-only 改为 read-write，然后就可以在 rootfs 上进行写和读的操作了。但 Docker 的镜像却不是这样，它在 bootfs 自检完毕之后并不会把 rootfs 的 read-only 改为 read-write，而是利用 union mount（UnionFS 的一种挂载机制）将一个或多个 read-only 的 rootfs 加载到之前的 read-only 的 rootfs 层之上。

在加载了这么多层的 rootfs 之后，仍然让它看起来只像是一个文件系统，在 Docker 的体系里把 union mount 的这些 read-only 的 rootfs 叫作 Docker 的镜像。但此时的每一层 rootfs 都是 read-only 的，还不能对其进行操作。当创建一个容器，也就是将 Docker 镜像进行实例化，系统会在一层或多层 read-only 的 rootfs 之上分配一层空的 read-write 的 rootfs。一个完整的容器文件系统层级结构如图 2-5 所示。

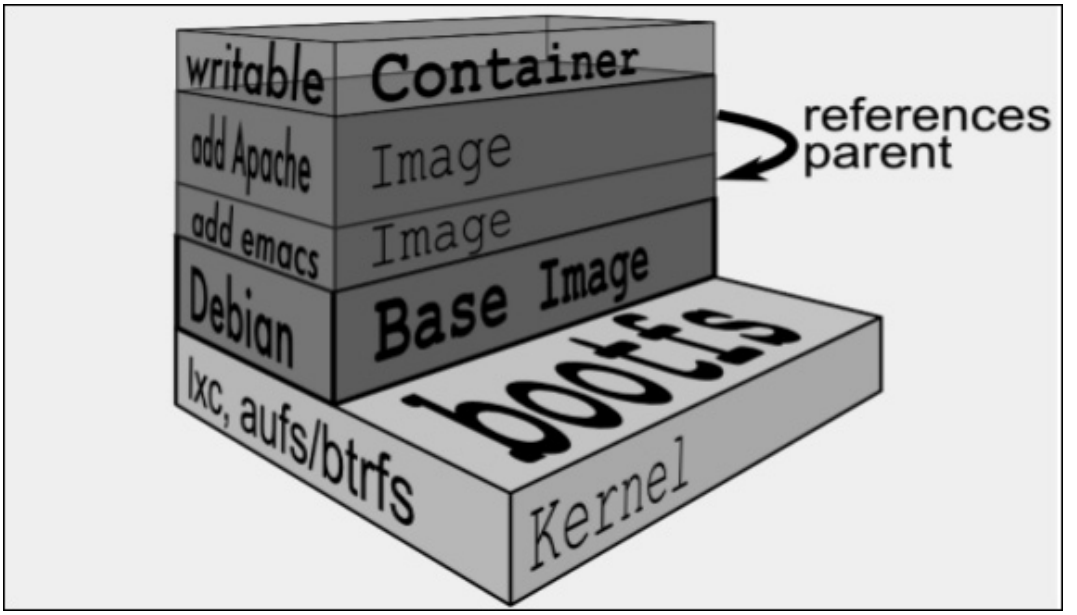


图 2-5　Docker 容器文件系统结构

2.4　Device Mapper 文件系统简介

Device Mapper 是 Linux 2.6 内核中支持逻辑卷管理的通用设备映射机制，它为实现用于存储资源管理的块设备驱动提供了一个高度模块化的内核架构。

Device Mapper 的内核体系架构如图 2-6 所示。

Device Mapper 在内核中通过一个一个模块化的 Target driver 插件实现对 I/O 请求的过滤或者重新定向等工作，当前已经实现的 Target driver 插件包括软 RAID、软加密、逻辑卷条带、多路径、镜像、快照等，图 2-6 中 linear、mirror、snapshot、multipath 表示的是 Target driver。

Device Mapper 进一步体现了 Linux 内核设计中策略和机制分离的原则，将所有与策略相关的工作放到用户空间完成，内核中主要提供完成这些策略所需要的机制。

Device Mapper 用户空间相关部分主要负责配置具体的策略和控制逻辑，比如逻辑设备和哪些物理设备建立映射、怎么建立这些映射关系等，而具体过滤和重定向 I/O 请求的工作由内核中相关代码完成。因此，整个 Device Mapper 机制由两部分组成——内核空间的 Device Mapper 驱动、用户空间的 Device Mapper 库及其提供的 Dmsetup 工具。

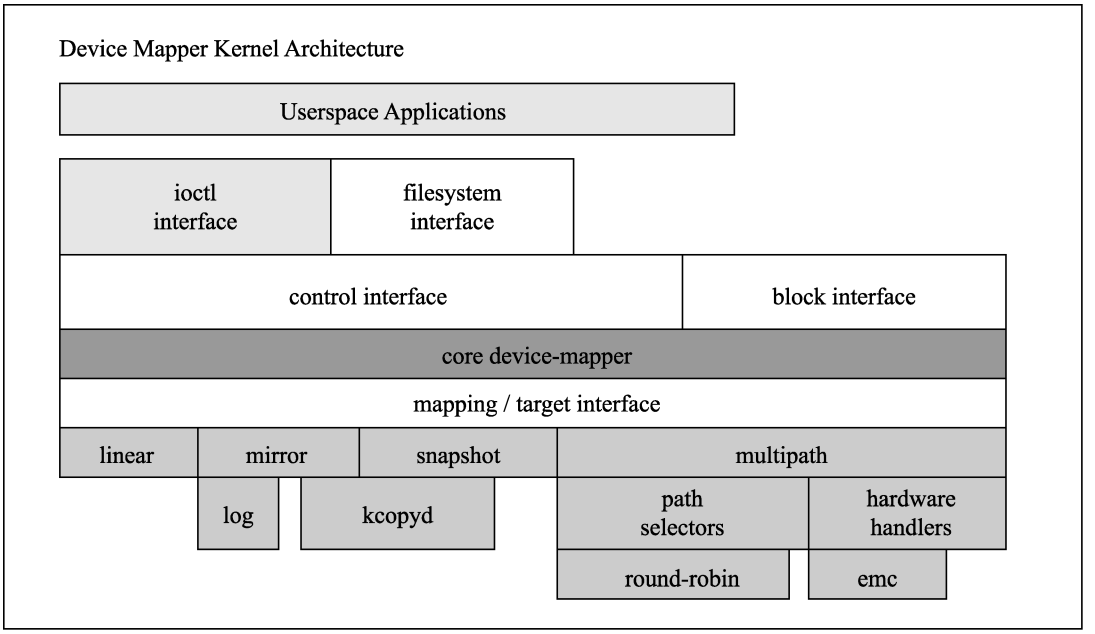


图 2-6　Device Mapper 内核架构

2.5　OverlayFS 简介

OverlayFS 是目前使用比较广泛的层次文件系统，是一种类似 AUFS 的堆叠文件系统，于 2014 年正式合并入 Linux 3.18 主线内核。OverlayFS 文件系统实现简单，且性能良好，可以充分利用不同或相同 Overlay 文件系统的 Page Cache，具有上下合并、同名遮盖、写时复制等特点。

Page Cache 也称为页缓冲或文件缓冲，由多个磁盘块构成，大小通常为 4KB，在 64 位系统上为 8KB，构成的几个磁盘块在物理磁盘上不一定连续，文件的组织单位为一页，也就是一个 Page Cache 大小，文件读取是由外存上不连续的几个磁盘块到 Buffer Cache，组成 Page Cache，然后供给应用程序。

Page Cache 用于在 Linux 读写文件时缓存文件的逻辑内容，从而加快对磁盘上映像和数据的访问。加速对文件内容的访问，Buffer Cache 缓存文件的具体内容——物理磁盘上的磁盘块，这是加速对磁盘的访问。

Docker 虚拟化 Overlay 存储驱动利用了很多 OverlayFS 特性构建和管理镜像与容器的磁盘结构。从 Docker 1.12 起，Docker 也支持 Overlay2 存储驱动。与 Overlay 相比，Overlay2 在 Inode 优化上更加高效。但 Overlay2 驱动只兼容 Linux Kernel 4.0 及以上的版本。

OverlayFS 加入 Linux Kernel 主线后，在 Linux Kernel 模块中的名称从 OverlayFS 改名为 Overlay。在实际用中，OverlayFS 代表整个文件系统，而 Overlay/Overlay2 表示 Docker 的存储驱动。

在 Docker 虚拟化技术中，OverlayFS 将一个 Linux 主机中的两个目录组合起来，一个在上，一个在下，对外提供统一的视图。这两个目录就是层，将两个层组合在一起的技术称作联合挂载（Union Mount）。在 OverlayFS 中，上层的目录称作 Upper Dir，下层的目录称作 Lower Dir，对外提供的统一视图称作 Merged，如图 2-7 所示。

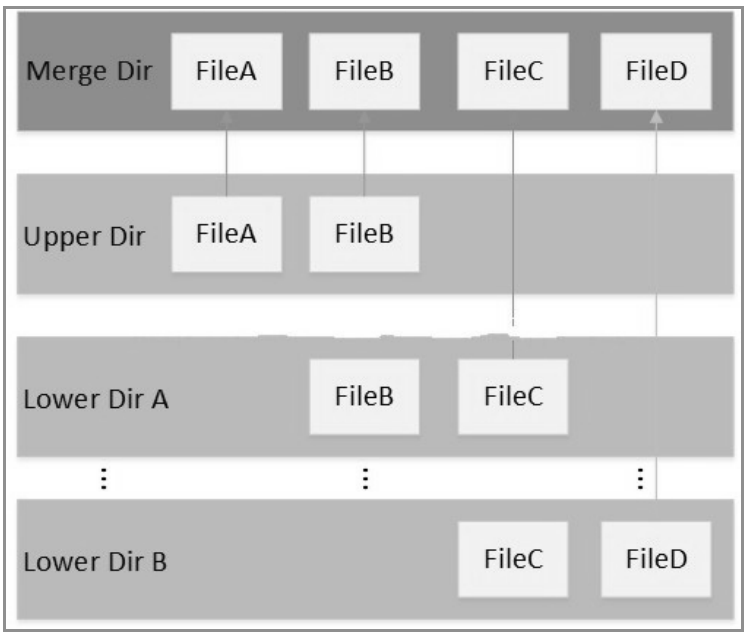


图 2-7　OverlayFS 结构图

当需要修改一个文件时，使用 Copy On Write（写时复制）将文件从只读的 Lower Dir 复制到可写的 Upper Dir 进行修改，结果也保存在 Upper Dir 层。在 Docker 中，底下的只读层是 Image，可写层是 Container。

如果镜像层和容器层可以有相同的文件，则 Upper Dir 中的文件将覆盖 Lower Dir 中的文件。Docker 镜像中的每一层并不对应 OverlayFS 中的层，而是/var/lib/docker/overlay 中的一个文件夹，文件夹以该层的 UUID 命名，然后使用硬链接将下层的文件引用到上层，在一定程度上可以节省磁盘空间。

容器和镜像的层与 OverlayFS 的 Upper Dir、Lower Dir、Merged Dir 之间的对应关系，如图 2-8 所示。

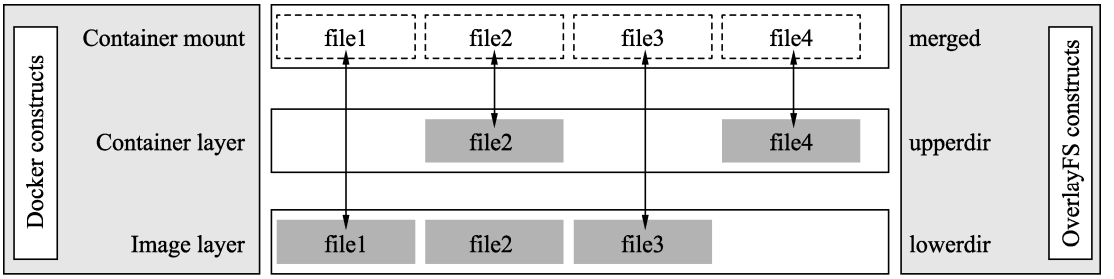


图 2-8　OverlayFS 文件系统结构

2.6　为什么使用 Docker

Docker 在以下几方面具有较大的优势。

（1）更快速的交付和部署。

Docker 在整个开发周期内都可以完美地辅助开发者实现快速交付。Docker 允许开发者在装有应用和服务本地容器做开发。可以直接集成到可持续开发流程中。

开发者可以使用一个标准的镜像构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器部署代码。Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。Docker 容器很轻很快，容器的启动时间是秒级的，可有效节约开发、测试、部署的时间。

（2）高效的部署和扩容。

Docker 容器几乎可以在任意平台上运行，包括物理机、虚拟机、公有云、私有云、个人计算机、服务器等。这种兼容性可以让用户把应用程序从一个平台直接迁移到另外一个平台。

Docker 的兼容性和轻量特性可以很轻松地实现负载的动态管理。可以快速扩容或方便地下线应用和服务，这种速度趋近实时。

（3）更高的资源利用率。

Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要建 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。

（4）更简单的管理。

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量

的方式被分发和更新，从而实现自动化且高效的管理。

2.7 Docker 镜像、容器、仓库

熟悉了 Docker 虚拟化简介、组件和工作原理之后，还需要掌握 Docker 虚拟化镜像原理、引擎架构等知识。

Docker 虚拟化技术有三个基础概念：Docker 镜像、Docker 容器、Docker 仓库。

（1）Docker 镜像。

Docker 虚拟化最基础的组件为镜像，类似常见的 Linux ISO 镜像。但是 Docker 镜像是分层结构的，由多个层级组成，每个层级分别存储软件实现某个功能。Docker 镜像是静止的、只读的，不能对镜像进行写操作。

（2）Docker 容器。

Docker 容器是 Docker 虚拟化的产物，也是最早在生产环境中使用的对象。Docker 容器的底层是 Docker 镜像，是基于镜像运行，并在镜像最上层添加一层容器层之后的实体。容器层是可读、可写的，容器层如果需用到镜像层中的数据，可以通过 JSON 文件读取镜像层中的软件和数据，对整个容器进行修改。写操作只能作用于容器层，不能直接对镜像层进行写操作。

（3）Docker 仓库。

Docker 仓库是用于存放 Docker 镜像的地方。Docker 仓库分为两类，分别是公共仓库（Public）和私有仓库（Private）。国内和国外有很多默认的公共仓库，对外开放，免费或者付费使用。企业测试环境和生产环境推荐自建私有仓库，私有仓库的特点为安全、可靠、稳定、高效，能够根据自身的业务体系进行灵活升级和管理。

2.8 Docker 镜像原理剖析

完整的 Docker 镜像可以支撑一个 Docker 容器的运行，在 Docker 容器运行过程中主要提供文件系统数据支撑。Docker 镜像是分层结构的，由多个层级组成，每个层级分别存储各种软件实现某个功能。Docker 镜像作为 Docker 中最基本的概念，有以下几个特性。

（1）镜像是分层的，每个镜像都由一个或多个镜像层组成。

（2）可通过在某个镜像加上一定的镜像层得到新镜像。

（3）通过编写 Dockerfile 或基于容器 Commit 实现镜像制作。

（4）每个镜像层拥有唯一镜像 ID，Docker 引擎默认通过镜像 ID 识别镜像。

（5）镜像在存储和使用时，共享相同的镜像层，在 PULL 镜像时，已有的镜像层会自动跳过下载。

（6）每个镜像层都是只读，即使启动成容器，也无法对其进行真正的修改，修改只会作用于最上层的容器层。一个完整的 Docker 容器系统如图 2-9 所示。

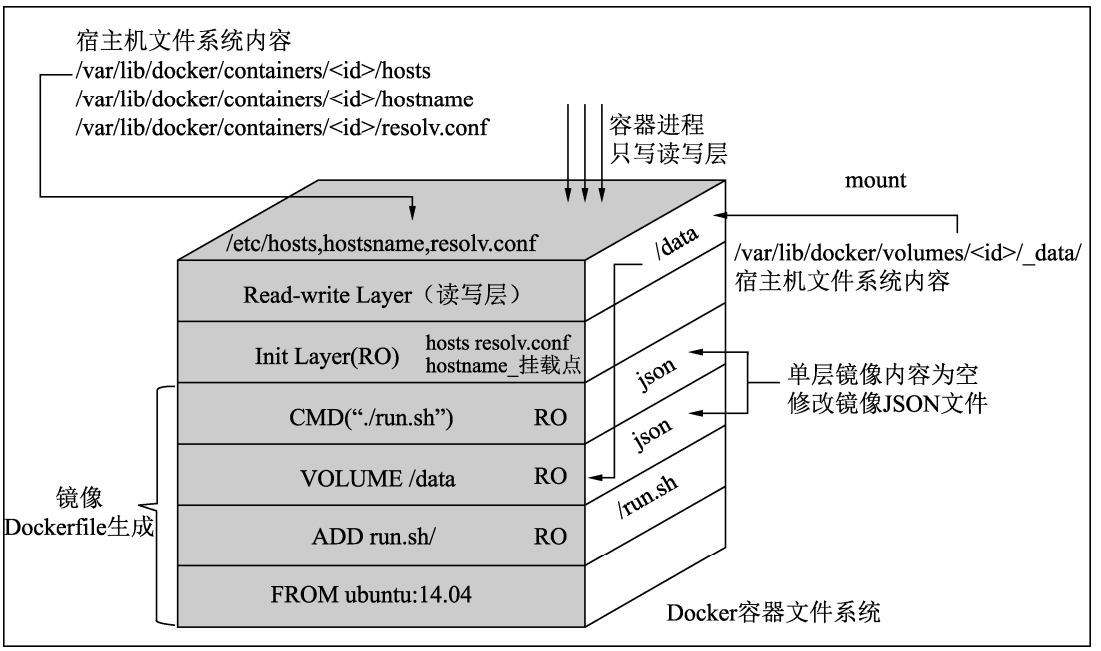


图 2-9　Docker 容器系统结构

Docker 容器是一个或多个运行进程，而这些运行进程将占有相应的内存、CPU 计算资源、虚拟网络设备及文件系统资源。Docker 容器所占用的文件系统资源，则通过 Docker 镜像的镜像层文件提供。基于每个镜像的 JSON 文件，可以通过解析 Docker 镜像的 JSON 的文件获知应该在这个镜像之上运行什么样的进程，应该为进程配置什么样的环境变量，而 Docker 守护进程实现了从静态向动态的转变。

Docker 虚拟化引也是一个 C/S（Client/Server）结构的应用，如图 2-10 所示。

Docker 虚拟化完整体系，包括以下几个组件。

（1）Docker Server 是一个常驻进程。

（2）REST API 实现了 Client 和 Server 之间的交互协议。

（3）Docker CLI 实现容器和镜像的管理，为用户提供统一的操作界面。

（4）Images 为容器提供了统一的软件、文件底层存储。

（5）Container 是 Docker 虚拟化的产物，直接作为生产使用。

（6）Network 为 Docker 容器提供完整网络通信。

（7）Volume 为 Docker 容器提供额外磁盘、文件存储对象。

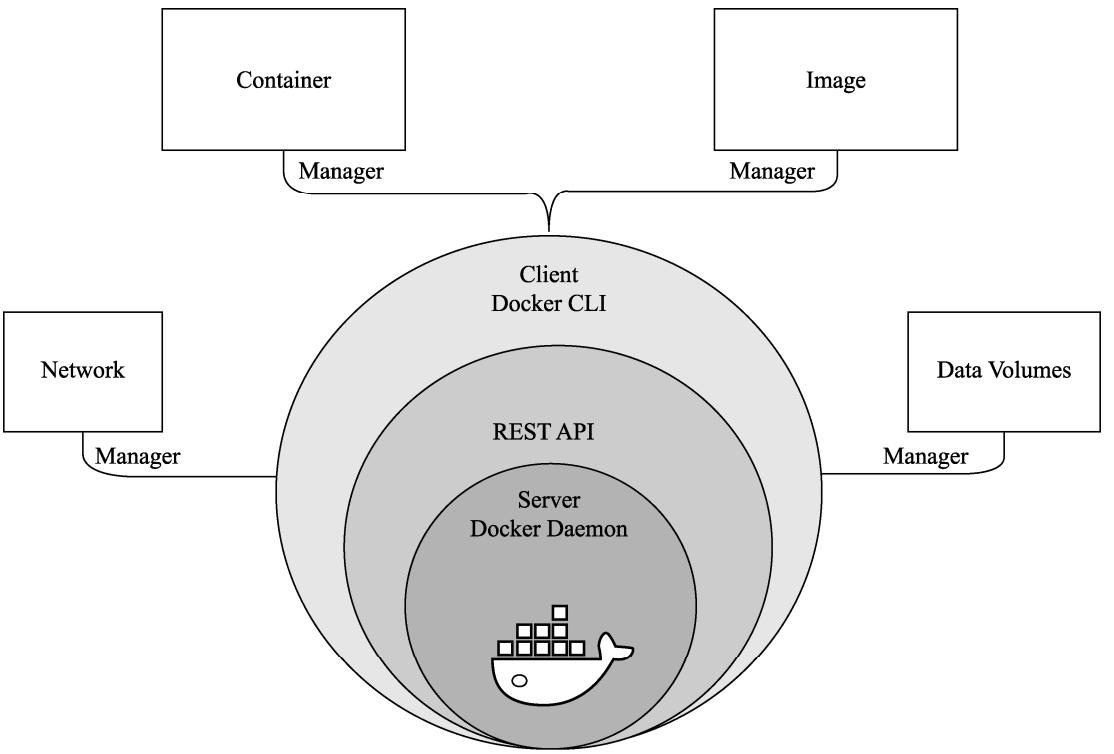


图 2-10　Docker C/S 引擎结构

Docker 使用 C/S 架构，Client 通过接口与 Server 进程通信实现容器的构建、运行和发布。Client 和 Server 可以运行在同一台集群，也可以通过跨主机实现远程通信，架构如图 2-11 所示。

由图 2-11 可以清晰地看出 Docker 虚拟化整个生态体系。

（1）基于 Docker Client 客户端–Rest API–操作 Docker Daemon。

（2）Docker Daemon 部署至 Docker 宿主机（通常为硬件物理机）。

（3）基于 Docker pull 可以从 Registry 仓库获取各种镜像至 Docker Host（主机）。

（4）基于 Docker run 可以通过获取的镜像启动 Docker Container（容器）；

（5）基于 Docker build 可以构建满足企业需求的各种 Docker Image（镜像）。

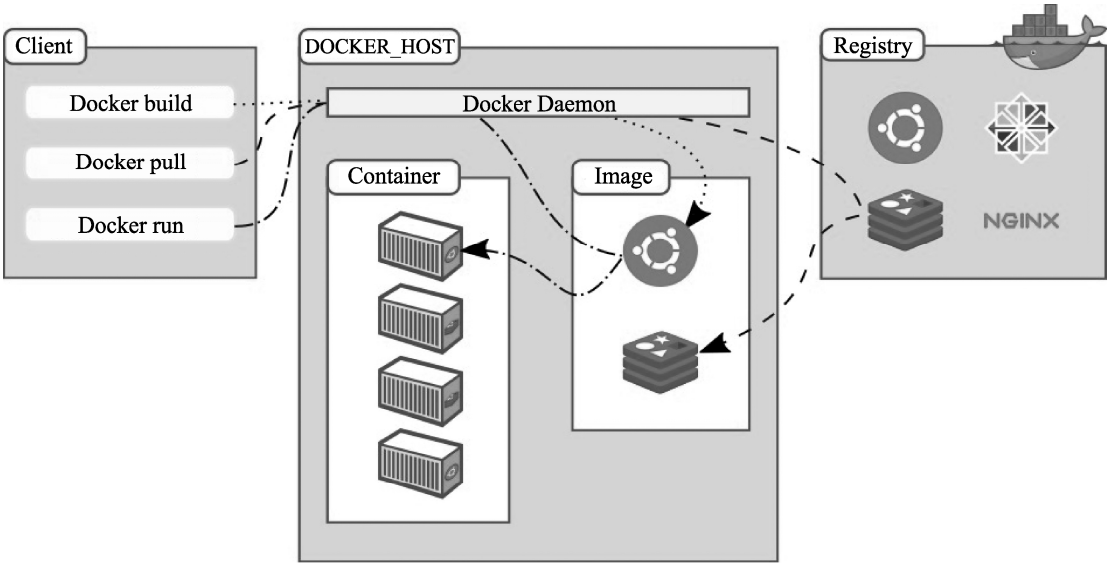


图 2-11　Docker 虚拟化拓扑

2.9　CentOS 7.x（7.0+）Linux Docker 平台实战

掌握了 Docker 虚拟化概念和原理之后，最重要的就是要在生产环节中落地 Docker。Docker 虚拟化平台最早期只支持 Linux 操作系统，现在最新版 Windows 操作系统也支持 Docker 虚拟化。

本章节将选择不同的发行版本构建 Docker 虚拟化平台。Linux 操作系统主流发行版本包括 Red Hat Linux、CentOS、Ubuntu、SUSE Linux、Fedora Linux 等。以下简要介绍即将部署 Docker 虚拟化平台的两个系统：CentOS 和 Ubuntu。

Docker 官方要求 Linux 内核版本在 3.8+以上，生产环境中推荐使用 3.10+的 Linux 内核版本。Docker 从 1.13 版本起，采用时间线的方式作为版本号。Docker 版本现在基于 YY.MM，分为社区版（Community Edition）和企业版（Enterprise Edition）。社区版是免费提供给个人开发者和小型团体使用的，而企业版会提供额外的收费服务。

社区版按照 Stable 和 Edge 两种方式发布，每个季度更新 Stable 版本，如 17.06、17.09，每个月份更新 Edge 版本，如 17.09、17.10。

虚拟化和 Docker 虚拟化技术本质的用途：为了最大化地利用高配物理机的资源，提高硬件

设备服务器的资源利用率，淘汰一些老、旧服务器，对老、旧服务器进行资源的重组、重用，满足企业快速发展，虚拟化落地实施硬件设备应尽量使用高配物理机资源，参考配置如下。

① 服务器品牌：Dell R730、R820。

② CPU 配置：Intel 至强 E5-2600 系列。

③ MEM 配置：ECC DDR3 256GB。

④ DISK 配置：SAS 12TB（最大支持 24TB）。

⑤ 网络配置：Intel 四端口千兆网卡/双端口万兆网卡。

（1）安装步骤和命令如下：

```
#安装国内阿里源
wget  -P /etc/yum.repos.d/ http://mirrors.aliyun.com/docker-ce/linux/
centos/docker-ce.re po
#安装 Docker-CE 版本
yum install docker-ce* -y
#检查 Docker 版本是否安装
rpm -qa|grep -E "docker"
#启动 Docker 引擎服务
service docker restart
systemctl restart docker.service
#查看 Docker 服务进程
ps -ef|grep docker
```

（2）安装完成，如图 2-12 所示。

```
[root@www-jfedu-net ~]# > #安装Epel扩展源;
-bash: syntax error near unexpected token `newline'
[root@www-jfedu-net ~]# yum install epel-release -y
#安装Docker-CE版本;
yum install docker* -y
#查Docker版本是否安装;
rpm -qa|grep -E "docker"
#启动Docker引擎服务;
Loaded plugins: fastestmirror, priorities
service docker restart
Repository base is listed more than once in the configurat
Repository updates is listed more than once in the configu
```

图 2-12　Docker 安装图解

（3）查看启动进程，如图 2-13 所示。

（4）查看 Docker 基础信息，如图 2-14 所示。

（5）从 Docker 仓库下载 Nginx 镜像，如图 2-15 所示。

```
[root@www-jfedu-net ~]# #启动Docker引擎服务;
[root@www-jfedu-net ~]# service docker restart
Redirecting to /bin/systemctl restart docker.service
[root@www-jfedu-net ~]# systemctl restart docker.service
[root@www-jfedu-net ~]# #查看Docker服务进程;
[root@www-jfedu-net ~]# ps -ef|grep docker
root     25711     1  3 19:57 ?        00:00:00 /usr/bin/d
ibexec/docker/docker-runc-current --default-runtime=docker
userland-proxy-path=/usr/libexec/docker/docker-proxy-curre
current --seccomp-profile=/etc/docker/seccomp.json --selin
rification=false --storage-driver overlay2
root     25716 25711  0 19:57 ?        00:00:00 /usr/bin/d
ker/libcontainerd/docker-containerd.sock --metrics-interva
```

图 2-13　Docker 服务进程

```
[root@www-jfedu-net ~]# docker info|more
  WARNING: You're not using the default seccomp profile
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
```

图 2-14　Docker 基础信息

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# docker pull docker.io/nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
a5a6f2f73cd8: Extracting 6.652 MB/22.49 MB
a5a6f2f73cd8: Pull complete
67da5fbcb7a0: Pull complete
e82455fa5628: Pull complete
Digest: sha256:31b8e90a349d1fce7621f5a5a08e4fc519b634f7
Status: Downloaded newer image for docker.io/nginx:late
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
```

图 2-15　Docker 下载 Nginx 镜像

2.10　CentOS 8.x（8.0+）Linux Docker 平台实战

（1）基于 CentOS 8.x Linux 操作系统，从零开始构建一套 Docker 虚拟化平台，使用二进制

Tar 包方式，部署的方法和步骤如下：

```
#从官网下载 Docker 软件包
ls -l docker-19.03.9.tgz
#通过 Tar 工具对其解压缩（-x 表示 extract 解压,-z gzip 表示压缩格式,-v verbose 表示
#详细显示,-f file 表示文件属性）
tar -xzvf docker-19.03.9.tgz
#创建 Docker 程序部署目录/usr/local/docker/
mkdir -p /usr/local/docker/
#将解压后的 Docker 程序移动至部署目录
\mv docker/* /usr/local/docker/
#查看 Docker 程序是否部署成功
ls -l /usr/local/docker/
#创建 Docker 用户和组,同时将 Docker 部署目录加入 PATH 环境变量
useradd -s /sbin/nologin docker -M
cat>>/etc/profile<<EOF
export PATH=\$PATH:/usr/local/docker/
EOF
#使其 PATH 环境变量生效
source /etc/profile
#启动 Docker 引擎服务
nohup /usr/local/docker/dockerd &
#查看 Docker 进程状态
ps -ef|grep -aiE docker
#查看 Docker 版本信息
/usr/local/docker/docker version
docker version
```

（2）根据以上 Docker 平台部署指令，Docker 平台部署成功，查看其版本信息，如图 2-16 所示。

```
[root@www-jfedu-net ~]# /usr/local/docker/docker version
Client: Docker Engine - Community
 Version:           19.03.8
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        afacb8b7f0
 Built:             Wed Mar 11 01:22:56 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.8
```

图 2-16　Docker 版本查看

2.11　Ubuntu（16.04+）Linux Docker 平台实战

（1）安装步骤和命令如下：

```
#更新 apt 源
apt-get update
#apt 源使用 HTTPS 以确保软件下载过程中不被篡改。需要添加使用 HTTPS 传输的软件包以及
#CA 证书
apt-get install \
apt-transport-https \
ca-certificates \
curl \
software-properties-common
#默认 apt 访问国外源,网络非常慢,此处建议使用国内源,需要添加软件源的 GPG 密钥
curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo
apt-key add -
#curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
#add -
#向 source.list 中添加 Docker 软件源
add-apt-repository \
"deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu
$(lsb_release -cs) stable"
#官方源
#add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/
#ubuntu $(lsb_release -cs) stable"
```

（2）Docker 安装操作方法和步骤如下：

```
#更新 apt 软件包缓存
apt-get update
#基于 apt-get 安装 docker-ce 社区版本
apt-get install docker-ce
#查 Docker 版本是否安装
dpkg -s docker-ce
#启动 Docker 引擎服务
service docker restart
#查看 Docker 服务进程
ps -ef|grep docker
```

（3）安装完成，如图 2-17 所示。

```
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for systemd (229-4ubuntu13) ...
Setting up pigz (2.3.1-2) ...
Setting up aufs-tools (1:3.2+20130722-1.1ubuntu1) ...
Setting up cgroupfs-mount (1.2) ...
Setting up containerd.io (1.2.0-1) ...
Setting up docker-ce-cli (5:18.09.0~3-0~ubuntu-xenial) .
Setting up docker-ce (5:18.09.0~3-0~ubuntu-xenial) ...
update-alternatives: using /usr/bin/dockerd-ce to provid
Processing triggers for libc-bin (2.23-0ubuntu5) ...
Processing triggers for systemd (229-4ubuntu13) ...
Processing triggers for ureadahead (0.100.0-19) ...
```

图 2-17　Ubuntu 安装 Docker

（4）查看启动进程和 Docker 基础信息，如图 2-18 所示。

```
root@www-jfedu-net:~# ps -ef|grep docker
root      25420      1  2 22:27 ?        00:00:00 /usr/bin/doc
root      25539  20763  0 22:28 pts/0    00:00:00 grep --color
root@www-jfedu-net:~#
root@www-jfedu-net:~# docker info|more
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 18.09.0
Storage Driver: overlay2
 Backing Filesystem: extfs
```

图 2-18　Ubuntu 查看 Docker 信息

（5）从 Docker 仓库下载 Nginx 镜像，如图 2-19 所示。

```
root@www-jfedu-net:~#
root@www-jfedu-net:~# docker pull docker.io/nginx
Using default tag: latest
latest: Pulling from library/nginx
a5a6f2f73cd8: Pull complete
67da5fbcb7a0: Pull complete
e82455fa5628: Pull complete
Digest: sha256:31b8e90a349d1fce7621f5a5a08e4fc519b634f
Status: Downloaded newer image for nginx:latest
root@www-jfedu-net:~# docker images
REPOSITORY          TAG                 IMAGE ID
nginx               latest              e81eb098537d
root@www-jfedu-net:~#
```

图 2-19　Docker 下载 Nginx 镜像

2.12　Docker 仓库源更新实战

Docker 默认连接的国外官方镜像，根据网络情况不同，通常访问时快时慢，大多时候获取速度非常慢，为了提升效率可以自建仓库或先修改为国内仓库源，提升拉取镜像的速度。

Docker 可以配置的国内镜像有很多，例如 Docker 中国区官方镜像、阿里云、网易蜂巢、DaoCloud 等，这些都是国内比较快的镜像仓库。

从国外官网下载 Docker Tomcat 镜像，访问速度慢，如图 2-20 所示。

```
[root@www-jfedu-net ~]# docker pull tomcat
Using default tag: latest
Trying to pull repository docker.io/library/tomcat ...
latest: Pulling from docker.io/library/tomcat
cc1a78bfd46b: Downloading 457.9 kB/45.32 MB
6861473222a6: Pulling fs layer
7e0b9c3b5ae0: Downloading 1.747 MB/4.336 MB
ae14ee39877a: Waiting
8085c1b536f0: Waiting
6e1431e84c0c: Waiting
ca0e3df5a1fd: Waiting
d2cb611ced6c: Waiting
268dc3e43e66: Waiting
79a7e8d254c7: Waiting
5c848af92738: Waiting
789b92e37607: Waiting
```

图 2-20　Docker 下载 Tomcat 镜像

Docker 镜像修改方法为，在命令行输入 vim /etc/docker/daemon.json，执行如下命令：

```
cat>/etc/docker/daemon.json<<EOF
{
"registry-mirrors":["https://registry.docker-cn.com"]
}
EOF
service docker restart
```

重启 Docker 服务即可。修改仓库地址为国内仓库后，获取镜像速度非常快，如图 2-21 所示。

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# service docker restart
Redirecting to /bin/systemctl restart docker.service

[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# systemctl restart docker.service
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# docker pull tomcat
Using default tag: latest
Trying to pull repository docker.io/library/tomcat ...
latest: Pulling from docker.io/library/tomcat
cc1a78bfd46b: Downloading  37.2 MB/45.32 MB
6861473222a6: Downloading 7.439 MB/10.77 MB
7e0b9c3b5ae0: Download complete
ae14ee39877a: Download complete
8085c1b536f0: Download complete
6e1431e84c0c: Download complete
ca0e3df5a1fd: Downloading  10.5 MB/122.1 MB
d2cb611ced6c: Waiting
```

图 2-21　Docker 下载 Tomcat 镜像