Dcoker—底层实现

Docker—底层实现

docker 底层的核心技术包括 Linux 上的命名空间（Namespaces）、控制组（Control groups）、Union 文件系统（Union file systems）容器格式（Container format）。

Docker—底层实现

我们知道，传统的虚拟机通过在宿主主机中运行 hypervisor 来模拟一整套完整的硬件环境提供给虚拟机的操作系统。虚拟机系统看到的环境是可限制的，也是彼此隔离的。 这种直接的做法实现了对资源最完整的封装，但很多时候往往意味着系统资源的浪费。 例如，以宿主机和虚拟机系统都为 Linux 系统为例，虚拟机中运行的应用其实可以利用宿主机系统中的运行环境。

Docker—底层实现

我们知道，在操作系统中，包括内核、文件系统、网络、PID、UID、IPC、内存、硬盘、CPU 等等，所有的资源都是应用进程直接共享的。 要想实现虚拟化，除了要实现对内存、CPU、网络IO、硬盘IO、存储空间等的限制外，还要实现文件系统、网络、PID、UID、IPC等等的相互隔离。 前者相对容易实现一些，后者则需要宿主机系统的深入支持。

Docker—底层实现

随着 Linux 系统对于命名空间功能的完善实现，程序员已经可以实现上面的所有需求，让某些进程在彼此隔离的命名空间中运行。大家虽然都共用一个内核和某些运行时环境（例如一些系统命令和系统库），但是彼此却看不到，都以为系统中只有自己的存在。这种机制就是容器（Container），利用命名空间来做权限的隔离控制，利用 cgroups 来做资源分配。

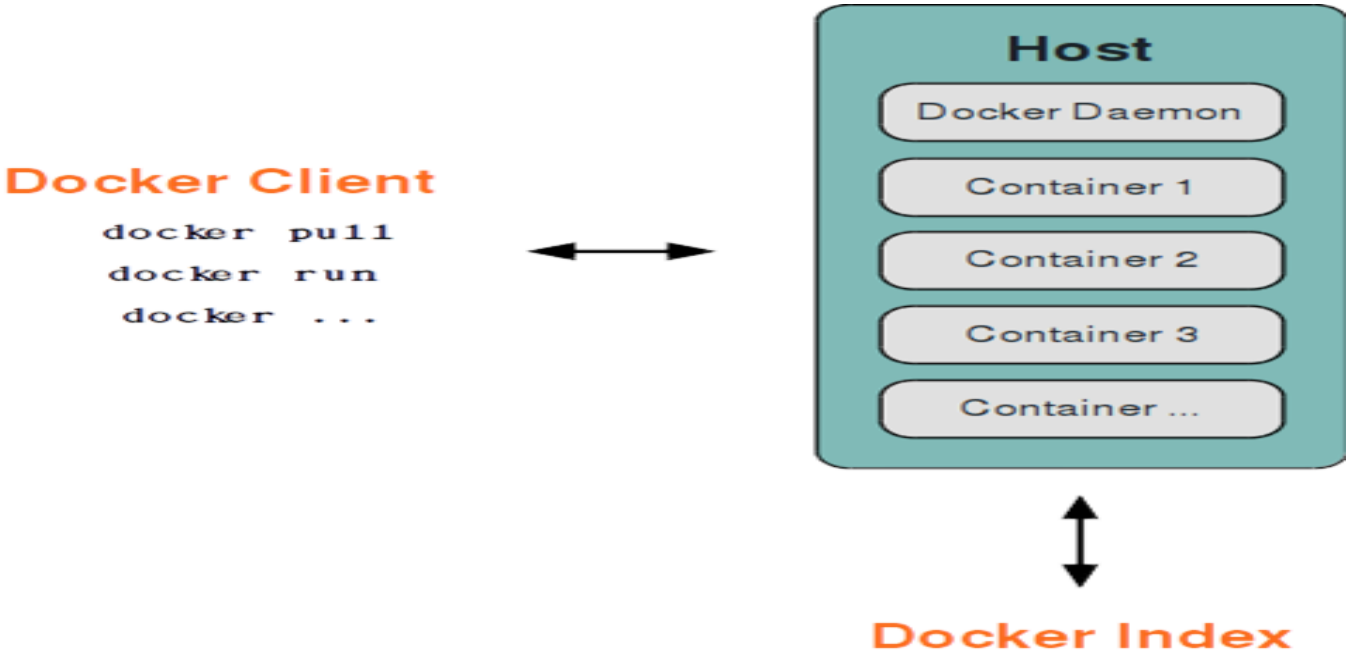
Docker—底层实现

- 基础架构

Docker 采用了 C/S架构，包括客户端和服务端。 Docker daemon 作为服务端接受来自客户的请求，并处理这些请求（创建、运行、分发容器）。 客户端和服务端既可以运行在一个机器上，也可通过 socket 或者 RESTful API 来进行通信。

Docker—底层实现

Docker daemon 一般在宿主主机后台运行，等待接收来自客户端的消息。 Docker 客户端则为用户提供一系列可执行命令，用户用这些命令实现跟 Docker daemon 交互。



Docker—底层实现

- 命名空间—Namespaces

命名空间是 Linux 内核一个强大的特性。每个容器都有自己单独的命名空间，运行在其中的应用都像是在独立的操作系统中运行一样。命名空间保证了容器之间彼此互不影响。

Docker—底层实现

- pid 命名空间

不同用户的进程就是通过 pid 命名空间隔离开的，且不同命名空间中可以有相同 pid。所有的 LXC 进程在 Docker 中的父进程为Docker进程，每个 LXC 进程具有不同的命名空间。同时由于允许嵌套，因此可以很方便的实现嵌套的 Docker 容器。

Docker—底层实现

- net 命名空间

有了 pid 命名空间, 每个命名空间中的 pid 能够相互隔离，但是网络端口还是共享 host 的端口。网络隔离是通过 net 命名空间实现的， 每个 net 命名空间有独立的 网络设备, IP 地址, 路由表, /proc/net 目录。这样每个容器的网络就能隔离开来。Docker 默认采用 veth 的方式，将容器中的虚拟网卡同 host 上的一 个Docker 网桥 docker0 连接在一起。

Docker—底层实现

- ipc 命名空间

容器中进程交互还是采用了 Linux 常见的进程间交互方法(interprocess communication - IPC), 包括信号量、消息队列和共享内存等。然而同 VM 不同的是，容器的进程间交互实际上还是 host 上具有相同 pid 命名空间中的进程间交互，因此需要在 IPC 资源申请时加入命名空间信息，每个 IPC 资源有一个唯一的 32 位 id。

Docker—底层实现

- mnt 命名空间

类似 chroot，将一个进程放到一个特定的目录执行。mnt 命名空间允许不同命名空间的进程看到的文件结构不同，这样每个命名空间 中的进程所看到的文件目录就被隔离开了。同 chroot 不同，每个命名空间中的容器在 /proc/mounts 的信息只包含所在命名空间的 mount point

Docker—底层实现

- uts 命名空间

UTS("UNIX Time-sharing System") 命名空间允许每个容器拥有独立的 hostname 和 domain name, 使其在网络上可以被视作一个独立的节点而非 主机上的一个进程。

Docker—底层实现

- user 命名空间

每个容器可以有不同的用户和组 id, 也就是说可以在容器内用容器内部的用户执行程序而非主机上的用户。

Docker—底层实现

控制组--Cgroups

控制组（cgroups）是 Linux 内核的一个特性，主要用来对共享资源进行隔离、限制、审计等。只有能控制分配到容器的资源，才能避免当多个容器同时运行时的对系统资源的竞争。

控制组技术最早是由 Google 的程序员 2006 年起提出，Linux 内核自 2.6.24 开始支持。

控制组可以提供对容器的内存、CPU、磁盘 IO 等资源的限制和审计管理。

Docker—底层实现

- 联合文件系统

联合文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。

Docker—底层实现

联合文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

另外，不同 Docker 容器就可以共享一些基础的文件系统层，同时再加上自己独有的改动层，大大提高了存储的效率。

Docker—底层实现

Docker 中使用的 AUFS（AnotherUnionFS）就是一种联合文件系统。AUFS 支持为每一个成员目录（类似 Git 的分支）设定只读（readonly）、读写（readwrite）和写出（whiteout-able）权限, 同时 AUFS 里有一个类似分层的概念, 对只读权限的分支可以逻辑上进行增量地修改(不影响只读部分的)。

Docker 目前支持的联合文件系统种类包括 AUFS, btrfs, vfs 和 DeviceMapper。