SOFT130091.01

云原生软件技术

3. 容器化技术与Docker

复旦大学软件学院

沈立炜

shenliwei@fudan.edu.cn

你所能想到的应用部署遇到的困难?

- 如果你是一位运维工程师，需要部署各种风格迥异的应用，你觉得你遇到的问题有哪些?

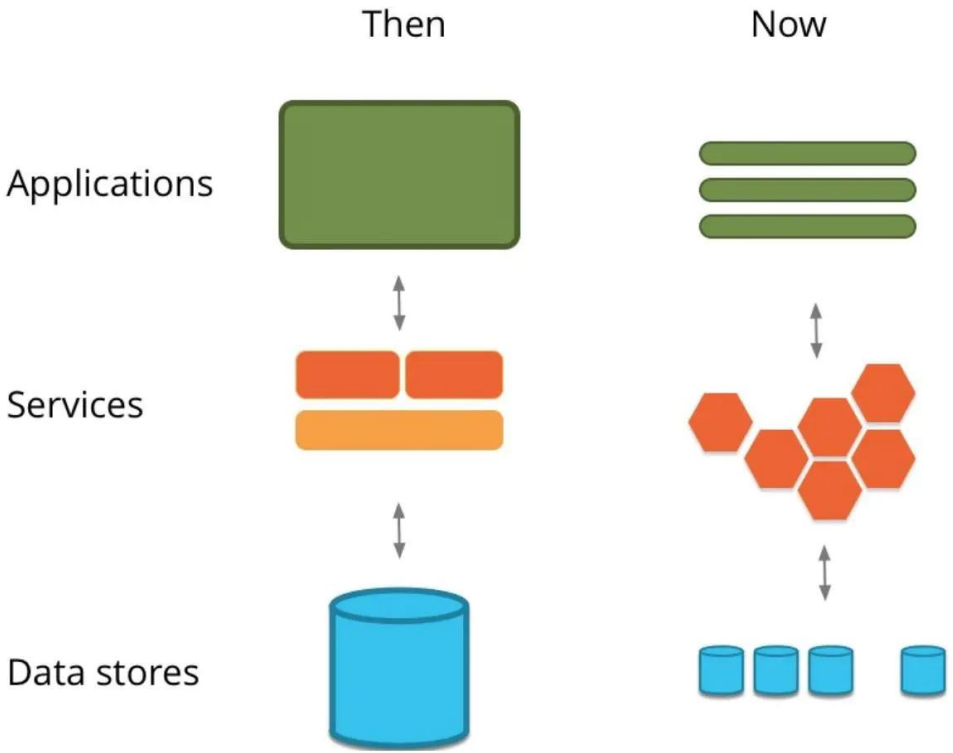
应用部署遇到的困难

开发和运行环境存在差异

复杂的环境配置

为什么要有容器

- 以前的应用程序
  - ✓ 几乎都是单体应用: 大系统, 多模块
  - ✓ 紧耦合: 内部调用
  - ✓ 不常变更: 需求稳定(改动成本高)
- 如今的应用程序
  - ✓ 解耦: 微服务/异步
  - ✓ 经常变更: 快速迭代
  - ✓ 动态创建和部署: 服务化
- 新架构的挑战
  - ✓ 多样化的技术栈
  - ✓ 需要动态创建机器
  - ✓ 很多活动组件
  - ✓ 运维人员需要管理复杂的架构



容器是什么

- 容器是一种打包应用及其运行环境的方式，为应用打包所有软件及其所依赖的环境，并且可以实现跨平台部署
- “容器”是一个黑盒，对于它的使用者来说
  - ✓ 无需关心里面有什么
    （只关注“容器”能做什么）
  - ✓ 有一套工具来管理黑盒
    （打包、运输、运行）
  - ✓ 减少了部署单元的数量，
    从而减少了花销
    （多个工具聚集在一个
    “容器”内）
  - ✓ 更容易管理多个环境
    （以“容器”为单位进行部署和管理）

容器技术的优点

- 轻量化
  - ✓ 相比于虚拟机，容器提供了更小的镜像，可以更快速地对容器进行构建和启动。
- 细粒度资源控制
  - ✓ 容器是一个沙箱运行进程，起到了细粒度管控资源的作用。在创建容器时，可以指定CPU、内存及I/O资源。在运行容器时强制执行这些资源限制，可防止容器占用其他资源。
- 高性能（资源利用率高）
  - ✓ 容器使用更轻量级的运行机制，是一种操作系统级别的虚拟化机制。由于容器是以进程形态运行的，因此其性能接近裸机的性能。

容器技术的优点

- 环境一致性
  - ✓ 容器实现了操作系统的解耦，它打包了整个操作系统，保证应用运行的本地环境和远端环境的高度一致性，从而保证一次容器打包可以到处运行，对跨平台、不同环境的应用部署有显著帮助。
- 管理便捷性
  - ✓ 使生命周期管理，包括迁移、扩展、运维等更加便捷。

容器技术的缺点

- 安全隔离是容器技术的一大弊端
  - ✓容器只是运行在宿主机上的一种特殊进程，因此多个容器之间共享的还是同一台宿主机的操作系统内核，大大增加了安全攻击面。

容器与虚拟机

容器

虚拟机

容器与虚拟机

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

容器的核心技术

namespace
命名空间

cgroup
控制组

rootfs
根文件系统

namespace

- namespace是Linux内核的一项功能，用于对内核资源进行隔离，使得一组进程只能看到与自己相关的一部分资源，而另一组进程看到另一组资源。
- 这种隔离机制确保了处于不同namespace的进程拥有独立的全局系统资源，改变一个namespace中的系统资源只会影响当前namespace里的进程，对其他namespace中的进程没有影响。

namespace

namespace	系统调用参数	隔离内容	内核版本
UTS	CLONE_NEWUTS	主机名和域名	2.6.19
IPC	CLONE_NEWIPC	信号量、消息队列和共享内存	2.6.19
PID	CLONE_NEWPID	进程编号	2.6.24
Network	CLONE_NEWNET	网络设备、网络栈、端口等	2.6.29
Mount	CLONE_NEWNS	挂载点（文件系统）	2.4.19
User	CLONE_NEWUSER	用户和用户组	3.8

这 6 项资源隔离分别对应 6 种系统调用，通过传入上表中的参数，调用 clone() 函数来完成。

cgroup

- 虽然容器通过namespace实现了隔离，但是它在宿主机上还是被看作一个普通的进程与其他所有进程之间保持平等关系。
- 这意味着，虽然容器进程表面上被隔离了，但是它所能够使用的资源（CPU、内存等）却是可以随时被宿主机上的其他进程（或其他容器）占用，这显然不是一个“沙盒”应该表现出来的合理行为。
- 这个缺陷可以通过Linux内核中用来为进程设置资源限制的cgroup来弥补。

cgroup

- cgroup(Control Groups)是Linux内核的一项功能，用于限制、记录和隔离一个进程组的物理资源（如CPU、内存、磁盘I/O等）使用。
- cgroup确保容器在运行时不会过度消耗宿主机的资源，从而提高系统的稳定性和安全性。

cgroup的主要功能

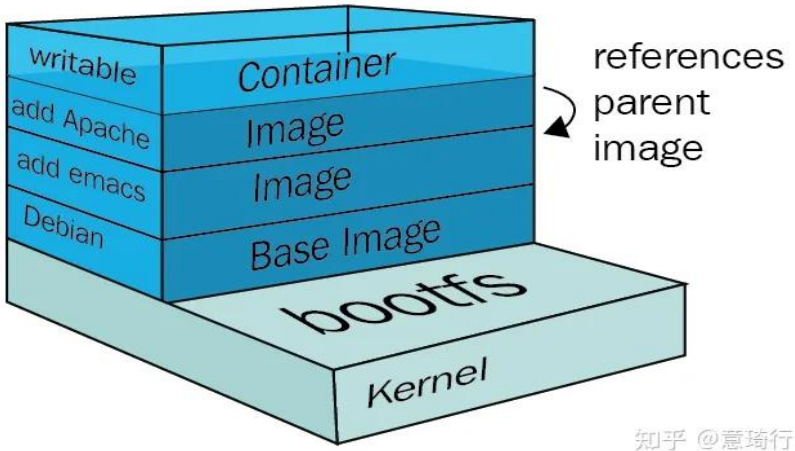
- 资源限制
  - ✓ CPU 限制：通过设置 CPU 使用率的上限，确保容器不会占用过多的 CPU 资源。例如，可以限制一个容器最多只能使用 50% 的 CPU 资源。
  - ✓ 内存限制：通过设置内存使用上限，防止容器占用过多内存。如果容器超过设定的内存限制，内核会触发 OOM（Out Of Memory）杀手进程，终止该容器的进程。
  - ✓ 磁盘 I/O 限制：通过限制磁盘 I/O 的读写速度，确保容器不会对磁盘进行过度的读写操作。
- 资源优先级
  - ✓ CPU 优先级：可以设置不同容器的 CPU 优先级，确保高优先级的容器在资源竞争中优先获得 CPU 时间片。
  - ✓ 内存优先级：可以设置不同容器的内存优先级，确保高优先级的容器在内存紧张时优先分配内存。

cgroup的主要功能

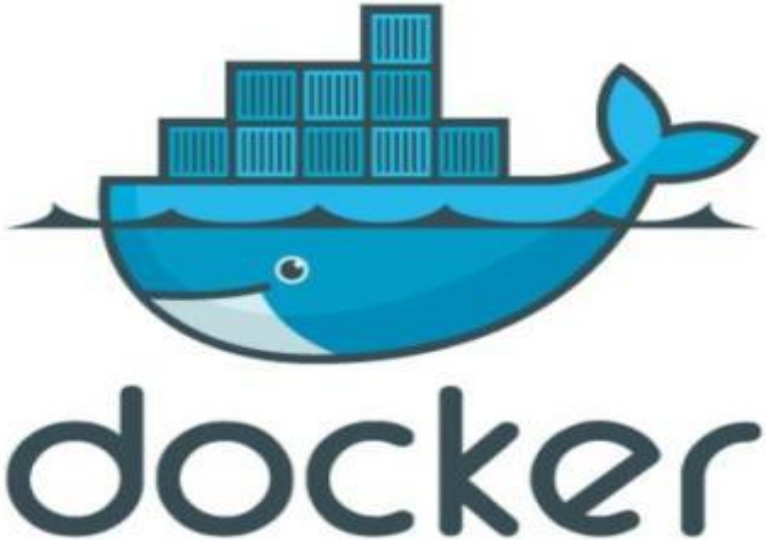
- 资源统计
  - CPU 使用统计：记录容器的 CPU 使用情况，包括使用时间、使用率等信息。
  - 内存使用统计：记录容器的内存使用情况，包括已用内存、可用内存等信息。
  - 磁盘 I/O 统计：记录容器的磁盘 I/O 使用情况，包括读写次数、读写速度等信息。
- 资源隔离
  - CPU 隔离：通过将 CPU 核心分配给不同的容器，确保容器之间的 CPU 资源隔离。
  - 内存隔离：通过限制容器的内存使用，确保容器之间的内存资源隔离。
  - 磁盘 I/O 隔离：通过限制容器的磁盘 I/O 速度，确保容器之间的磁盘资源隔离。

rootfs

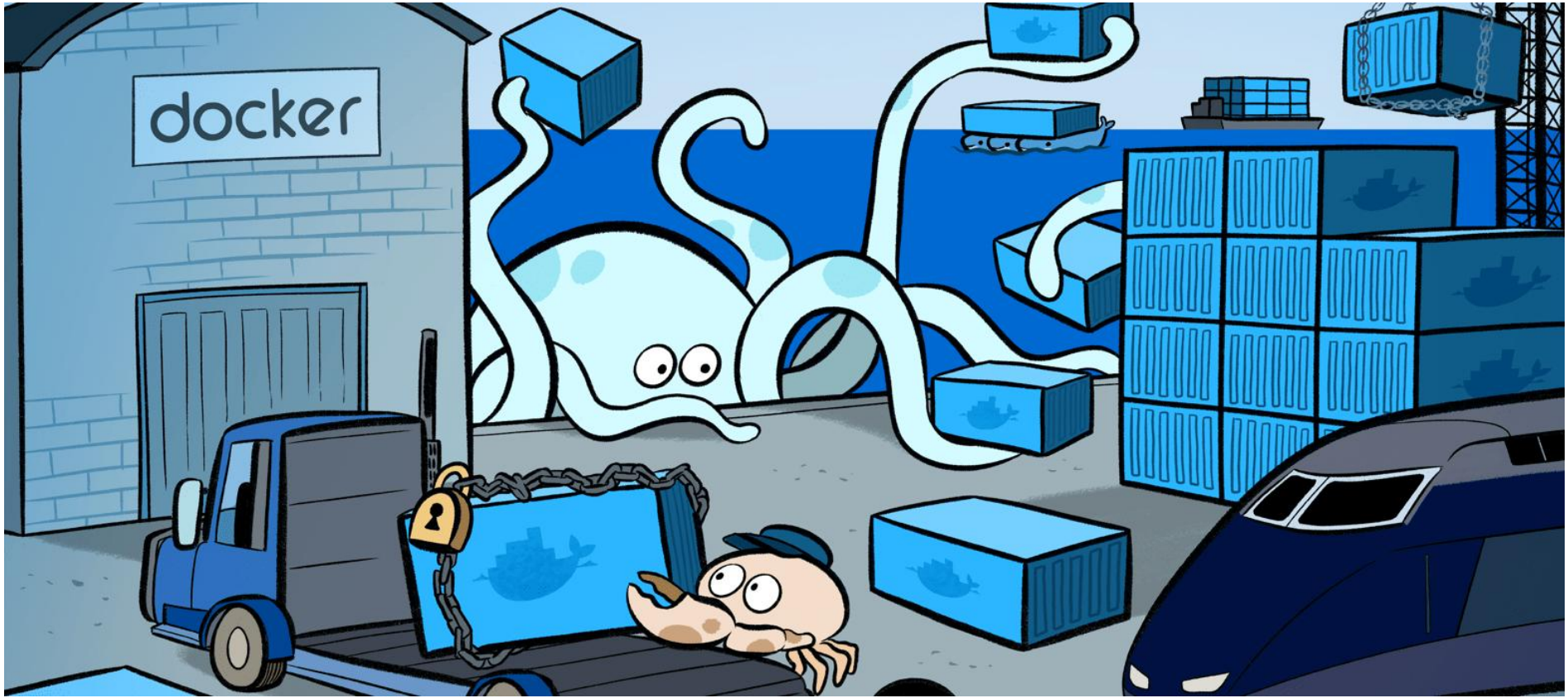
- rootfs是容器技术中的一个核心概念，它代表容器运行时的根文件系统。
- rootfs包含了容器运行所需的所有文件和目录，包括系统库、应用程序、配置文件等。通过 rootfs，容器可以实现环境的一致性，确保应用程序在不同的环境中都能以相同的方式运行。
- rootfs 只是一个操作系统所包含的文件、配置和目录，并不包括操作系统内核。同一台机器上的所有容器，都共享宿主机操作系统的内核。
- 容器中的文件系统是哪儿来的？实际上在构建镜像时打包进去，然后容器启动时挂载到了根目录下。



Docker



Docker



Docker

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications.

The main components are: the Docker Engine, a portable, lightweight runtime and packaging tool, and Docker Hub

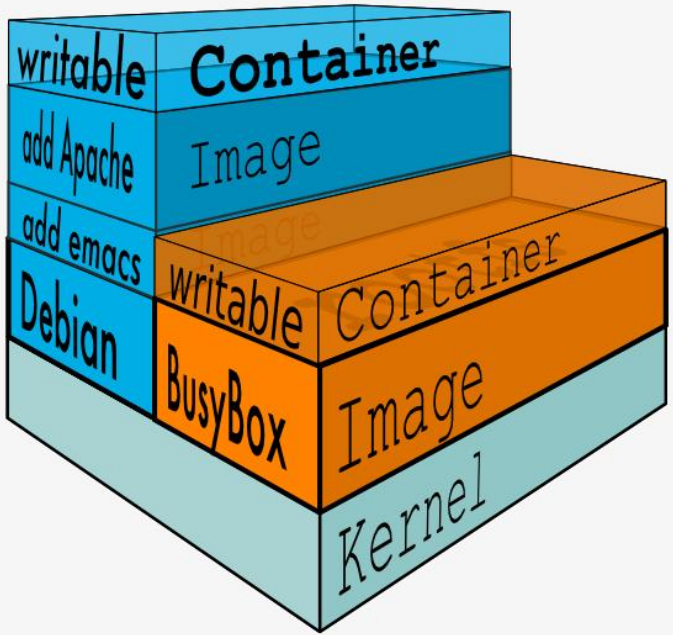
http://docker.com/whatisdocker

Docker

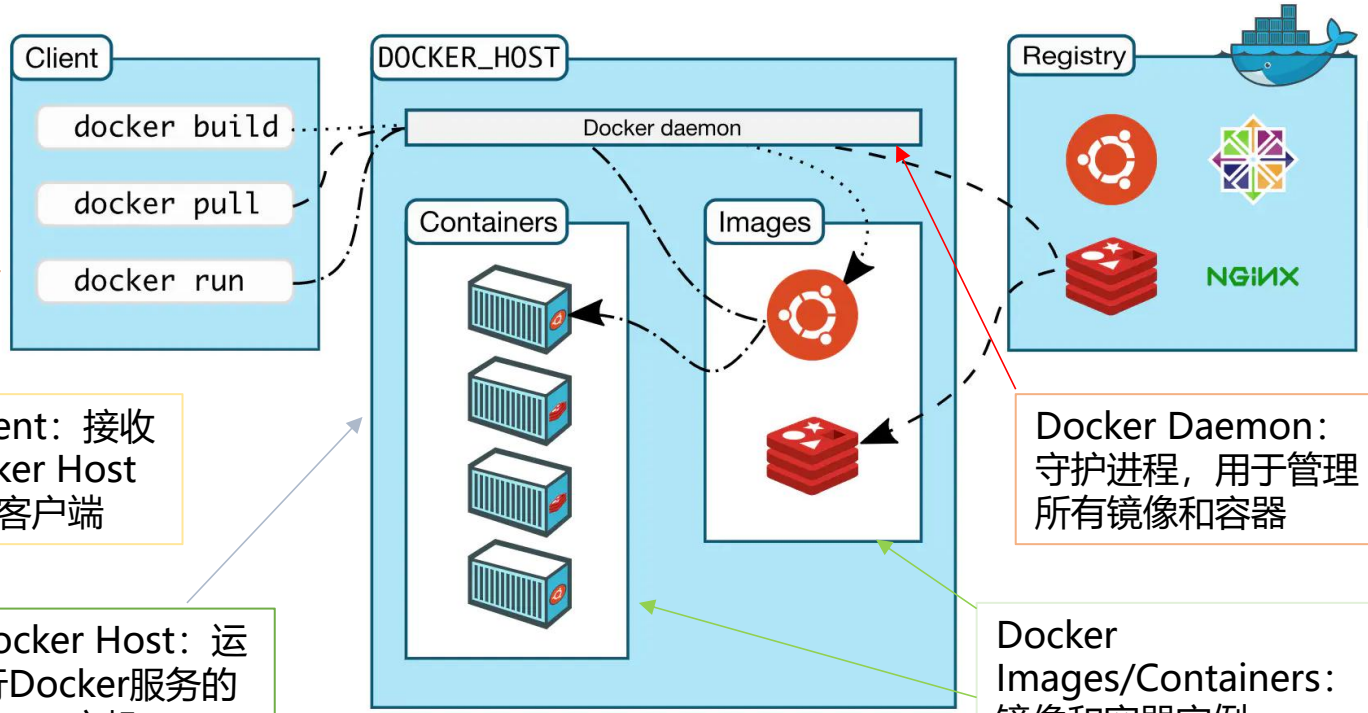
- Docker是容器服务中的一种
  - ✓ 比Docker更早的容器技术没有提供标准化的应用运行环境，所以一直没有变成主流。而Docker做到这一点。
- Docker是开源的应用容器引擎
  - ✓ dotCloud公司开源了基于XLC（Linux Container）的应用容器引擎，称为Docker
- Docker是一个用于开发、交付和运行应用程序的开发平台
  - ✓ 使用Go语言，遵从Apache2.0开源协议
  - ✓ 提供了在容器中打包和运行应用程序的功能
  - ✓ 在给定物理机或虚拟机上可运行多个Docker容器

UnionFS

- UnionFS 是一种为 Linux 操作系统设计的用于把多个文件系统“联合”到同一个挂载点的文件系统服务
- 每一个镜像层都是建立在另一个镜像层之上的，同时所有的镜像层都是只读的，只有每个容器最顶层的容器层才可以被用户直接读写，所有的容器都建立在一些底层服务（Kernel）上，包括命名空间、控制组、rootfs 等等，这种容器的组装方式提供了非常大的灵活性，只读的镜像层通过共享也能够减少磁盘的占用。



Docker架构



Registry(Hub)：
镜像仓库

Docker Client：接收命令和Docker Host进行交互的客户端

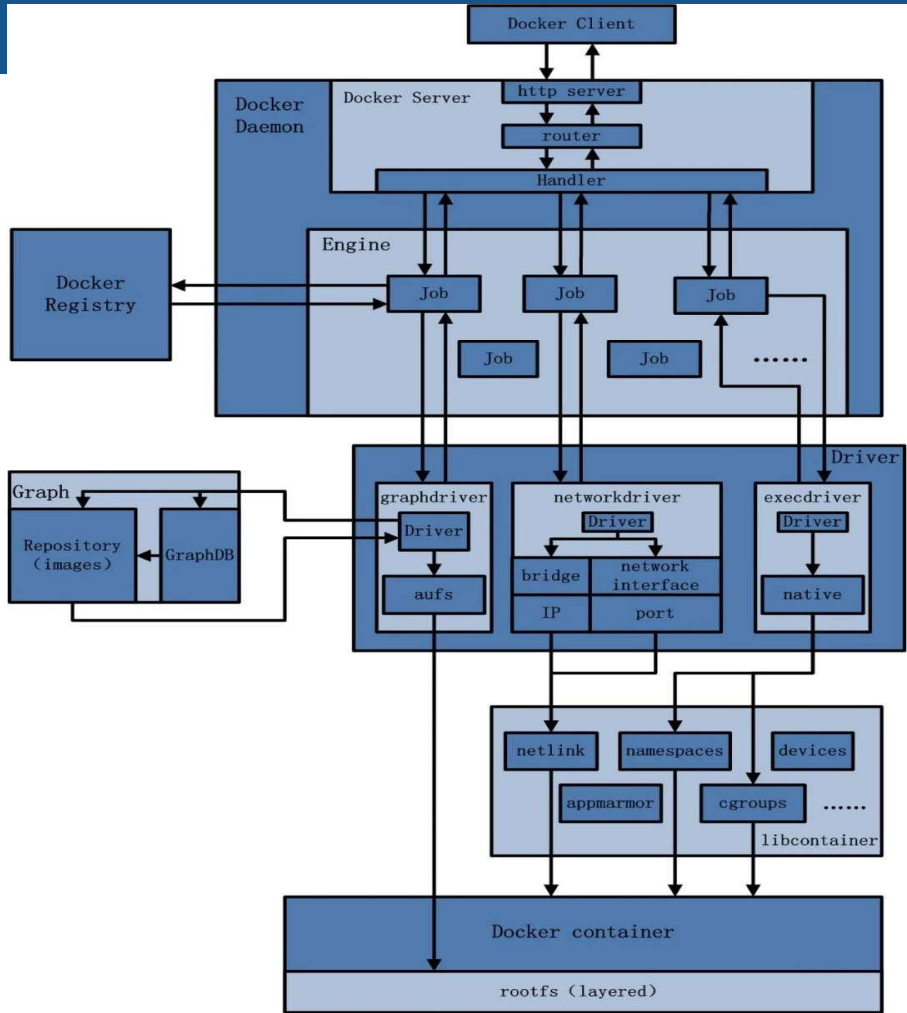
Docker Daemon：
守护进程，用于管理所有镜像和容器

Docker Host：运行Docker服务的主机

Docker Images/Containers：
镜像和容器实例

Docker机制

- 用户使用Docker Client与Docker Daemon建立通信，并发送请求给后者。
- Docker Daemon作为Docker架构中的主体部分，首先提供Docker Server的功能使其可以接受Docker Client的请求。
- Docker Engine执行Docker内部的一系列工作，每一项工作都是以一个Job的形式的存在。
- Job的运行过程中，当需要容器镜像时，则从Docker Registry中下载镜像，并通过镜像管理驱动 Graphdriver将下载镜像以Graph的形式存储。
- 当需要为Docker创建网络环境时，通过网络管理驱动Networkdriver创建并配置Docker容器网络环境。
- 当需要限制Docker容器运行资源或执行用户指令等操作时，则通过Execdriver来完成。
- Libcontainer是一项独立的容器管理包，Networkdriver以及Execdriver都是通过Libcontainer来实现对容器进行的具体操作。

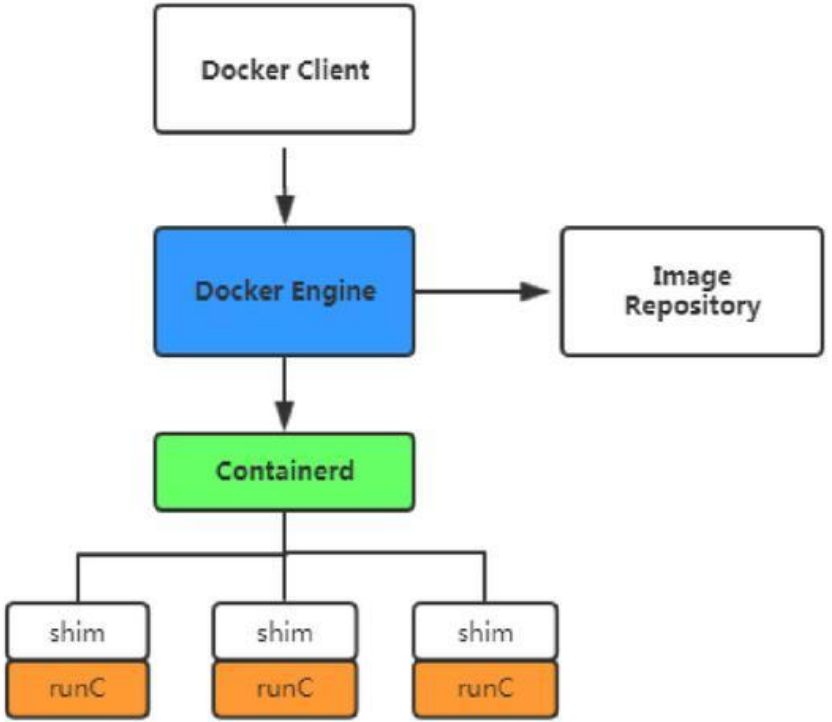


Docker容器运行时

- libcontainer
  - ✓libcontainer库可与Linux内核进行交互，用于容器生命周期管理，包括创建和管理隔离的容器环境
- runC
  - ✓随着Docker内部架构日渐复杂，Docker把底层容器管理部分单独剥离出来作为一个底层容器运行时，称为runC。
  - ✓runC中包含了原先的libcontainer库，可以独立于Docker引擎，其目标是使标准容器随处可用
  - ✓runC专注于容器实现，功能涉及环境隔离、资源限制、容器安全等，后来被捐赠给OCI。
- containerd
  - ✓containerd作为上层容器运行时，负责容器生命周期管理，如镜像传输和存储、容器运行和监控、底层存储、网络附件管理等
  - ✓containerd向上为Docker守护进程提供了gPRC接口，屏蔽底层细节，向下通过containerd-shim操作runC，使得上层Docker守护进程和底层容器运行时相互独立

Docker容器运行时

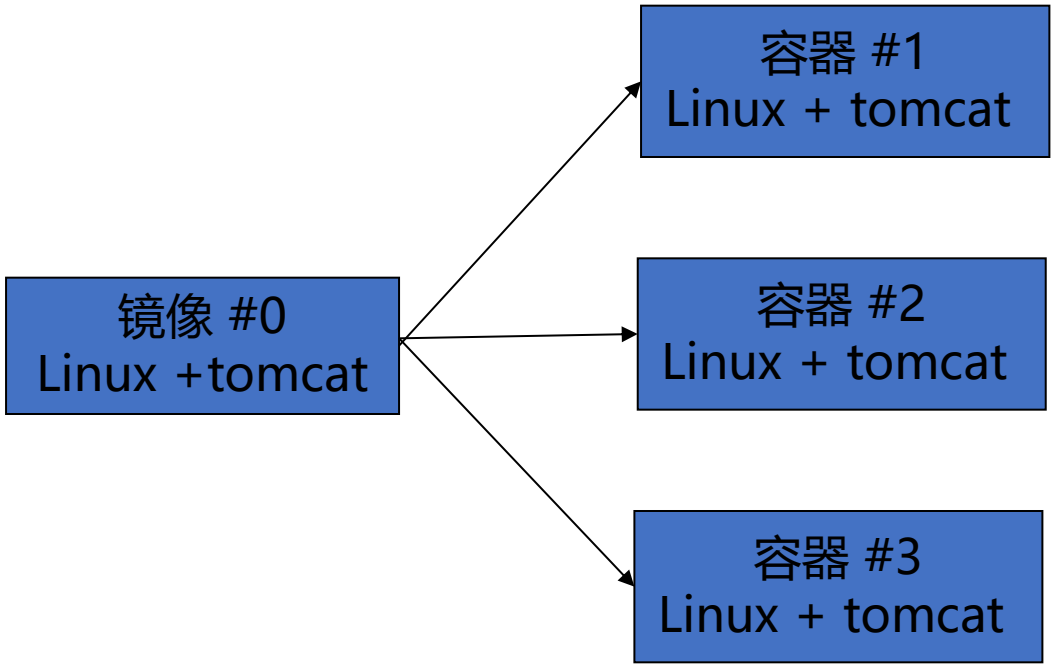
- Docker引擎创建容器并将其传递给containerd
- containerd调用containerd-shim
- containerd-shim使用runC来运行容器
- 容器运行时（runC）在容器启动后退出



镜像

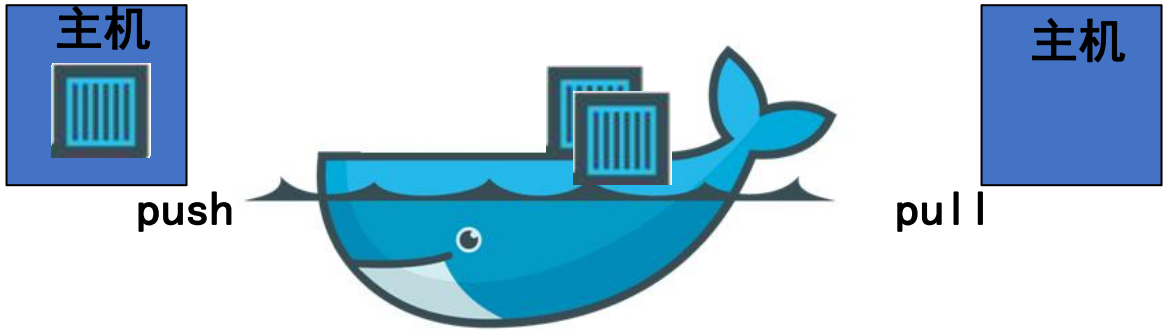
- Docker镜像解决了打包所存在的根本性问题
  - ✓应用打包、部署最大的问题是必须为每种语言、每个框架，甚至每个版本的应用都维护一个打好的包。
  - ✓在一个环境中运行地很好的应用，要在另一个环境中运行起来，需要做很多修改和配置工作。
- Docker镜像是一个压缩包，直接由一个完整的操作系统的所有文件和目录构成，即包含了这个应用运行所需的所有依赖，这个压缩包中的内容与本地开发和测试环境中的操作系统是完全一致的。
- 将程序作为镜像可以随处部署和运行，开发、测试和生产环境彻底统一，还能进行资源管控和虚拟化。

从同一个镜像启动多个容器



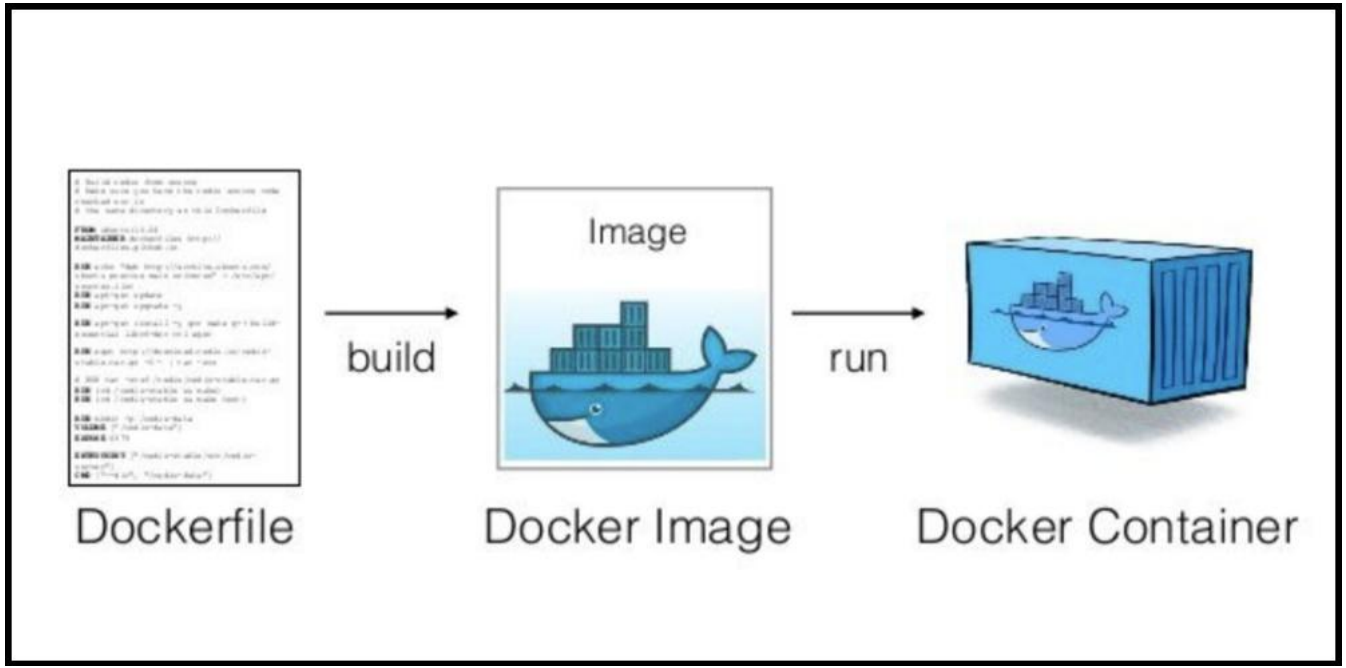
镜像仓库及仓库注册服务器

- 仓库是集中存放镜像文件的场所
- 仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签
- 仓库分为公开仓库（Public）和私有仓库（Private）两种形式
- push 镜像到仓库，从仓库pull下镜像



Dockerfile

- Dockerfile是docker镜像的描述文件，最基础原始的镜像文件都是通过dockerfile文件构建而来



Dockerfile模板指令

- **FROM**：指定基础镜像，必备的指令，并且必须是第一条指令。
- **MAINTAINER**：该镜像的维护者和点子邮件。比如：MAINTAINER xxxx "xxxxx@qq.com"
- **ENV**：设置环境变量。比如：EVN key1=value1 key2=value2...
- **RUN**：在新镜像内部执行的命令，比如安装一些软件、配置一些基础环境，可使用\来换行。比如：RUN yum install -y mysql-server
- **COPY**：将主机的文件复制到镜像文件中，如果目标位置不存在会自动创建。比如：COPY application.yml /etc/resources
- **ADD**：和COPY一样，但是ADD会加上解压操作
- **EXPOSE**：暴露镜像的端口供主机做映射，可以暴露多个端口。比如：EXPOSE 8080
- **WORKDIR**：在构建镜像时，指定镜像的工作目录，之后的命令都是基于此工作目录，如果不存在，则会创建目录，而且在进入容器时，会默认定位到该路径下。比如：WORKDIR /usr/local
- **VOLUME**：用来向基础镜像中添加数据卷比如 VOLUME /root/mydata /root/condata
- **CMD**：容器启动时需要执行的命令。比如 CMD /bin/bash

原文链接：https://blog.csdn.net/Zz1366/article/details/130780845

Dockerfile示例

FROM tomcat:8

-- 该镜像时基于tomcat:8镜像构建

WORKDIR /usr/local/tomcat

--设置当前基础路径，也是后续命令的相对路径

COPY test.war ./webapps

--将工程war包复制到tomcat的webapps路径下

EXPOSE 8080

--对外暴露8080端口，也就是tomcat访问路径

RUN ./bin/startup.sh

--设置容器启动时命令，即启动tomcat

Docker基本命令

- 下载image
- $docker pull image_name

- 列出镜像列表;
- $docker images

- 在容器中运行"echo"命令，输出"hello word"
- $ docker run image_name echo "hello word"

Docker基本命令

- 列出当前所有正在运行的container
- $docker ps

- 利用dockerfile建立新的镜像
- $docker build -t image_name Dockerfile_path

- 发布docker镜像
- $docker push new_image_name

小结

- 容器的定义、优点和缺点
- 容器的核心技术
- Docker的定义、架构
- Docker容器运行时
- Docker镜像、Dockerfile
- Docker基本命令

SOFT130091.01
云原生软件技术

End

3. 容器化技术与Docker