

Docker

- [Docker的发展历程详述-腾讯云开发者社区-腾讯云\(tencent.com\)](#)
- 容器化已经形成大势
- Linux
 - 很有可能要对很多个发行版做适配，因为Linux不同发行版的寿命很长，企业也不会总是更新发行版
 - 采用独立于分发的包格式（distribution-independent package），将应用程序和必需依赖项捆绑在一起，比如snap，flatpak

Docker 提供了高度的可移植性，但不能说是100%的绝对可移植性。Docker 可移植性的核心在于容器镜像的设计理念——“一次构建，到处运行”(Build Once, Run Anywhere)，意味着开发者在一个环境中创建并测试的Docker镜像，理论上可以在任何安装了Docker环境且满足硬件兼容性的系统上运行。

Docker通过以下方式提高可移植性：

1. **标准化的容器格式**：Docker采用开放标准（OCI, Open Container Initiative）定义容器格式，确保不同运行时环境之间的兼容性。
2. **包含所有依赖**：Docker镜像包含了应用程序及其所有的依赖库、配置文件等，避免了因环境差异导致的问题。
3. **操作系统层虚拟化**：通过Linux内核的cgroups和namespaces等机制，在用户空间提供了进程和资源隔离，使得容器内的应用程序不依赖于底层主机的具体配置。

然而，Docker的可移植性并不是绝对无条件的：

- **底层兼容性**：虽然Docker可在多种Linux发行版以及Windows上运行，但仍然需要底层操作系统支持特定版本的内核功能。例如，某些高级网络或存储功能可能要求特定内核模块或驱动程序的支持。
 - 理论上linux宿主机启动不了windows容器，那么为什么windows和mac上能启动linux容器呢？因为采用了虚拟机技术，如windows上安装docker-desktop要求开启hyperv和安装wsl，安装后可以通过windows文件资源管理器进入docker-desktop虚拟机，其内部看起来就是一个比较标准的linux系统
- **硬件兼容性**：对于某些特定的硬件依赖或性能敏感的应用，容器无法模拟所有的硬件特性，可能导致在不同硬件环境下表现不一致。
- **API与服务依赖**：如果容器内部的应用依赖于外部的服务（比如数据库、消息队列等），这些服务在目标环境中必须可用且兼容。
- **安全性和合规性**：不同的环境可能存在不同的安全策略和合规要求，可能需要额外的配置或认证才能在目标环境中运行。

因此，虽然Docker极大提升了软件部署的可移植性，但在实际应用中，仍需考虑上述因素带来的潜在限制。不过，随着技术的不断进步和发展，Docker及容器生态系统的可移植性也在逐步改善。

docker两大支柱技术

Namespace

让进程/应用以为自己是独立的

- linux namespace用以实现进程间资源隔离，namespace内的进程对namespace外的进程没有感知
- linux内核里执行 `make menuconfig` 可以找到内核关于namespace的编译选项
- 容器内 `ps` 只能看到容器内的进程，但是在容器外 `ps` 可以看到所有容器的进程
- linux内核里，不同的network namespace可以视为不同的网卡，如何连接两张网卡？linux内核使用的是veth pair，相当于连接网卡的网线（veth = virtual Ethernet）
- 包含：[理解docker - namespace - 知乎\(zhihu.com\)](https://zhuanlan.zhihu.com/p/101111111)
 - pid namespace
 - network namespace
 - UTX namespace(unix time-sharing system)，允许每个docker容器有自己的主机名
 - user namespace
 - IPC namespace(inter process communication)
 - mount namespace，文件系统

CGroups(Control Groups)

在物理上让进程/应用独立

- 如果其中的某一个容器正在执行 CPU 密集型的任务，那么就会影响其他容器中任务的性能与执行效率，导致多个容器相互影响并且抢占资源。如何对多个容器的资源使用进行限制就成了解决进程虚拟资源隔离之后的主要问题，而 Control Groups（简称 CGroups）就是能够隔离宿主机上的物理资源，例如 CPU、内存、磁盘 I/O 和网络带宽。
- 本质上来说，cgroups是内核附加在程序上的一系列钩子hook，通过程序运行时对资源的调度触发相应的钩子以实现资源追踪和限制
- 包含[一文带你搞懂 Docker 容器的核心基石 Cgroups - 知乎\(zhihu.com\)](https://zhuanlan.zhihu.com/p/101111111)
 - subsystem，如利用CFS(completely fair scheduler)，控制CPU的使用
 - hierarchy

docker pull

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

- tag比较面向人类，不同的打包可以用同一个tag
- digest面向机器，唯一性比tag强
- 都是用来解决下载哪个镜像

docker镜像层的组织：

- 树形结构
- 类似git commit历史

步骤：

1. 解析用户输入镜像名称，解析出要下载的镜像地址
2. 向registry发送请求

总结：代码 + Dockerfile = 镜像层 + manifest list + config

- 镜像层：每层代表一个Dockerfile中的命令
- manifest list:
 - `docker manifest` 是 Docker 的一个命令，它提供了一种方便的方式来管理不同操作系统和硬件架构的 Docker 镜像。通过 `docker manifest`，用户可以创建一个虚拟的 Docker 镜像，其中包含了多个实际的 Docker 镜像，每个实际的 Docker 镜像对应一个不同的操作系统和硬件架构
 - `docker manifest` 命令本身并不执行任何操作。为了操作一个 `manifest` 或 `manifest list`，必须使用其中一个子命令
 - `manifest` 可以理解为是一个 JSON 文件，单个 `manifest` 包含有关镜像的信息，例如层 (layers)、大小 (size) 和摘要 (digest) 等
 - `manifest list` 是通过指定一个或多个（理想情况下是多个）镜像名称创建的镜像列表（即上面所说的虚拟 Docker 镜像）。可以像普通镜像一样使用 `docker pull` 和 `docker run` 等命令来操作它。`manifest list` 通常被称为「多架构镜像」。

docker build

1. 定位dockerfile
2. 根据dockerfile的指令
 1. 每一条指令都创建一个临时容器
 2. 执行命令后commit出一个镜像层
 3. 删除临时容器（输出中的hash即临时容器的id）
3. docker镜像层删除文件机制：删除一个下层文件时，仅在读写层中创建一个whiteout文件来标志文件不存在，因此在dockerfile中写删除指令时，镜像层大小不变
 - 真正的删除：同层操作，用 `&&`，装完立刻删

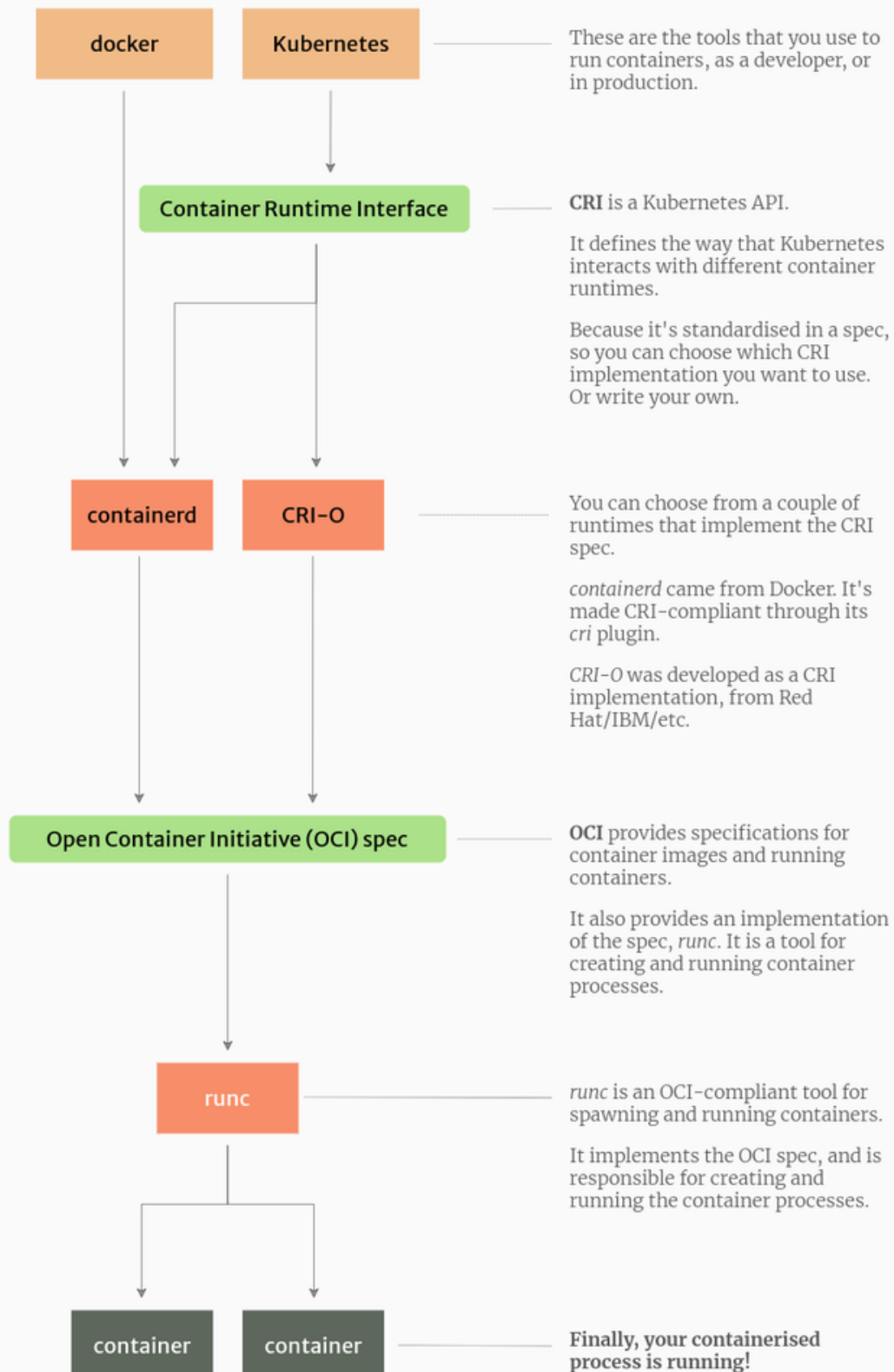
docker镜像大小只增不减

runc

runc提供了通用的容器运行时管理工具，无论上层是什么容器管理平添（docker，k8s）

- docker使用的也是runc
- go和c语言实现
- [Docker, containerd, CRI, CRI-O, OCI, runc 分不清? 看这一篇就够了 - 知乎 \(zhihu.com\)](#)

Docker, Kubernetes, OCI, CRI-O, containerd & runc: How do they work together?



容器监控

- free 和第三方工具无法处理 /proc 文件系统隔离不全的问题
- 容器内得到的信息有宿主机的信息
- 方案
 - 修改容器内 /proc，兼容已有的监控工具，如使用LXCFS用户态文件系统
 - 面向容器的监控方案，如k8s resource metric pipeline

容器网络

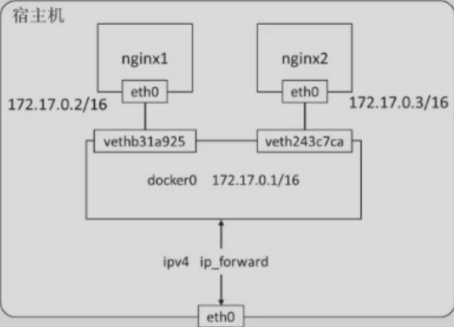
[Docker之十八：libnetwork 插件化网络功能-CSDN博客](#)

[NAT模式、路由模式、桥接模式的区别 - 知乎 \(zhihu.com\)](#)

bridge网络模式分析：demo环境介绍

- 假设一台Ubuntu虚拟机
- 当前已经通过如下命令启动两个nginx容器：
 - `docker run -d --rm -p 5000:80 nginx:alpine`
 - `docker run -d --rm -p 5001:80 nginx:alpine`
- nginx1容器的IP地址为172.17.0.2/16，其eth0设备的另一端是vethb31a925，“插在”docker0上
- nginx2容器的IP地址为172.17.0.3/16，其eth0设备的另一端是veth243c7ca，也“插在”docker0上
- docker0是Docker daemon创建的Linux bridge设备，它是一个工作在二层的交换机，使得连接在上面的nginx1/2两个容器在同一个局域网内，可以直接通信。同时它内部还有个网卡接口，IP地址为172.17.0.1/16
- 一旦vethb31a925和veth243c7ca这样的虚拟网卡“插在”网桥docker0上，就会被“剥夺”调用网络协议栈处理数据包的资格，降为网桥上的一个端口，因此也不必配置IP。例如，veth243c7ca收到数据包，不会通过网络协议栈处理它，而是直接到达veth pair的另一端：nginx2的eth0上
- 图中最下方的eth0是虚拟机网卡，IP地址为192.168.171.41/20，跟容器不在同一个网段

详见《Docker容器与容器云》3.8节：Docker网络管理



k8s的CNI标准已经成为主流，docker的CNM失去了云厂商和网络方案的支持

现状

- [技术干货 | Docker和Containerd的区别，看这一篇就够了 - 知乎 \(zhihu.com\)](#)
- docker其实就是通过containerd来管理容器的
- k8s其实已经摆脱了对docker的依赖，可以直接通过cri调containerd，资源消耗更小 [Docker 被 K8S 抛弃了！不要慌！分分钟转型 Containerd - 知乎 \(zhihu.com\)](#)

Windows docker

- 携程主推
 - 携程是.net应用大户，早期携程整个应用架构都放在.net上
 - 平台想转去java，去分享java的红利，但是不可能重写整个应用

- .net目前90%的应用都跑在虚拟机上，但是其自身虚拟机粒度太粗，扩容慢
- windows有两种容器
 - windows server container: `docker run --isolation=process ...`
 - hyperv: `docker run --isolation=hyperv ...`
 - hyperv隔离度更好，运行速度比虚拟机快，但是申请资源和获取资源时比windows server container稍慢
- 问题
 - windows基础镜像 `windowsservercore` 太大
 - 没有现成的监控日志方法
 - 难以管理大规模容器

reference

- [.NET Framework框架详解 - 知乎 \(zhihu.com\)](#)
- [.NET 7.0+WebAPI 后端架构实战 - 知乎 \(zhihu.com\)](#)

docker to moby

- docker公司叫docker，公司的产品叫docker，公司的开源项目也叫docker
- docker这个名字不能随便用了
- docker公司的ceo决定区分开产品docker和开源项目docker，开源项目以后将叫做moby；而公司会基于moby构建社区版产品docker和企业版docker
- 公司希望moby可以将标准组件库像乐高一样组建成定制的容器框架
 - LinuxKit: 一个工具包，以一个yaml文件创建为应用场景定制的最小化linux发行版，构建一个操作系统+应用的完整镜像（轻量化linux内核大约35MB）
 - 可能在iot场景上用LinuxKit的比较多
 - 一般应该是改产品的名字或者公司名字，改开源项目的名字，一是导致更多人会去不自觉地使用公司产品，二是伤害了开源社区的感情

Serverless

[It's the Future. Hey, my boss said to talk to you — I... | by Paul Biggar | CircleCI | Medium](#)

So I just need to split my simple CRUD app into 12 microservices, each with their own APIs which call each others' APIs but handle failure resiliently, put them into Docker containers, launch a fleet of 8 machines which are Docker hosts running CoreOS, "orchestrate" them using a small Kubernetes cluster running etcd, figure out the "open questions" of networking and storage, and then I continuously deliver multiple redundant copies of each microservice to my fleet. Is that it?

-Yes! Isn't it glorious?

I'm going back to Heroku.

- Serverless：云服务可以让我们选择合适配置的虚拟的云主机，然后按时、按月、按年的收费，但不管你使用与否，费用都在那里，不增不减。Serverless 则可以让你只在使用时，根据使用时间、内存占用等一些你具体使用的东西来收费，不管你使用的频率高低都可以承受，就像水和电，用了多少就是多少费用。
- [前端工程化：Docker、k8s、Serverless - 知乎\(zhihu.com\)](#)

docker使用和dockerfile

鉴于本人老docker用户了，这部分就不写了，列一下常用的命令

- `docker exec`
- `docker images`
- `docker ps -a`
- `docker logs`
- `docker tag`
- `docker rm` / `docker rmi`
- `docker system prune` / `docker image prune`