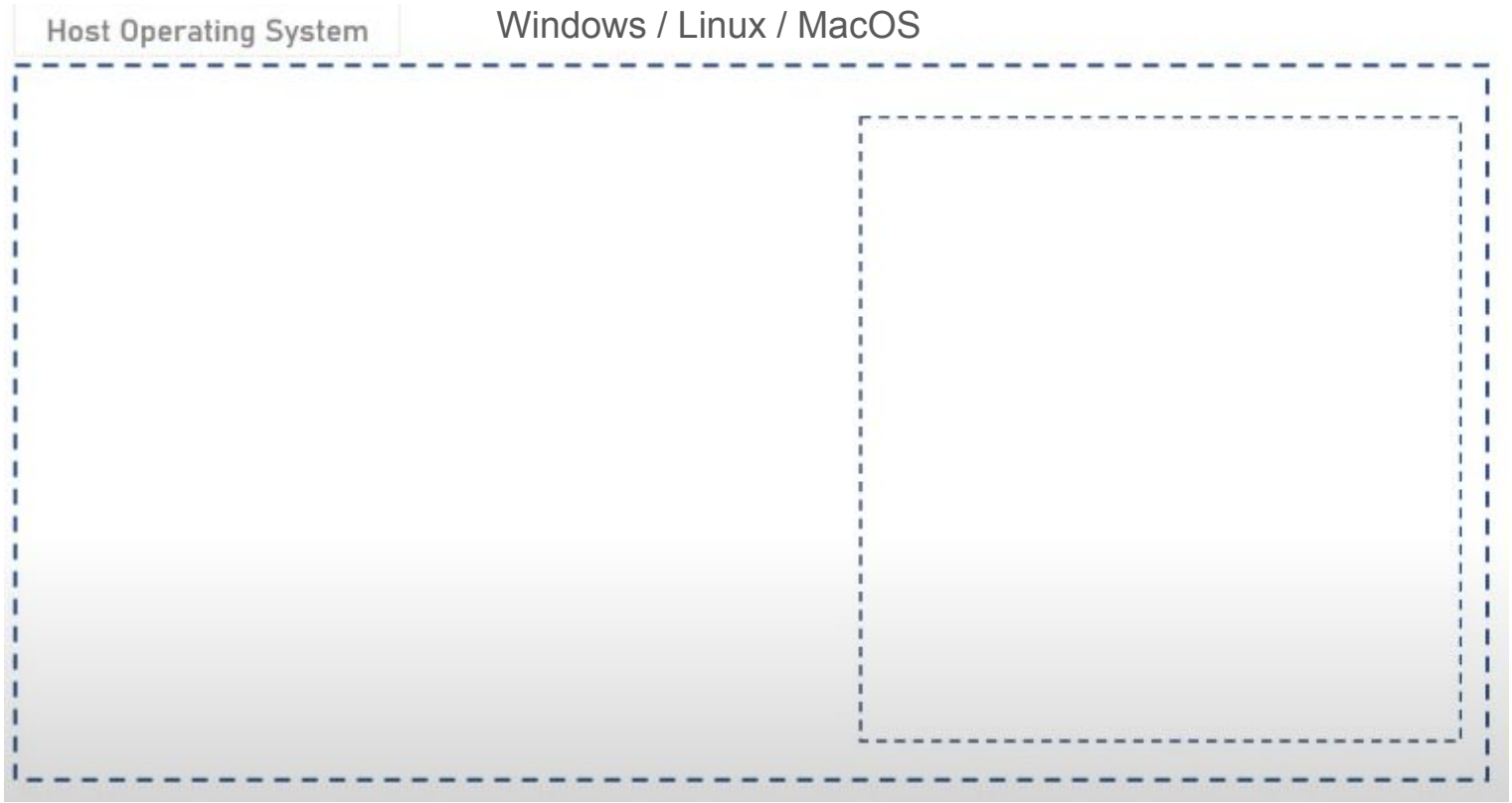
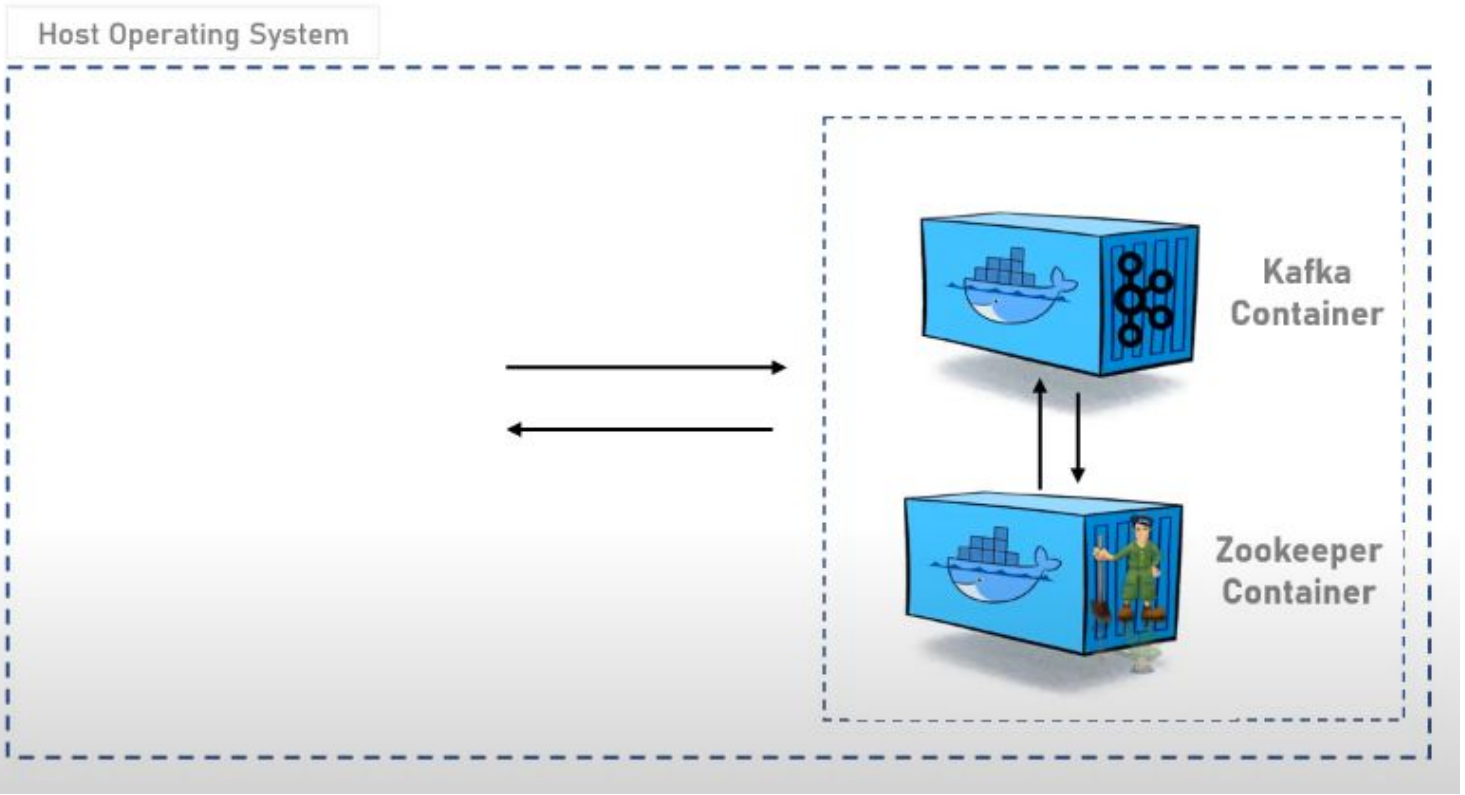
Docker Practice - 上

Overview



Overview


Host Operating System
Kafka
Container
Zookeeper
Container

容器技术原理

- chroot
- Namespace

  Namespace 是 Linux 内核的一项功能，对内核资源进行隔离，使得容器中的进程都可以在单独的命名空间中运行，并且只可以访问当前容器命名空间的资源。Namespace 可以隔离进程 ID、主机名、用户 ID、文件名、网络访问和进程间通信等相关资源。

  Docker 主要用到以下五种命名空间: 1) pid namespace：用于隔离进程 ID。2) net namespace：隔离网络接口，在虚拟的 net namespace 内用户可以拥有自己独立的 IP、路由、端口等。3) mnt namespace：文件系统挂载点隔离。4) ipc namespace：信号量,消息队列和共享内存的隔离。4) uts namespace：主机名和域名的隔离。

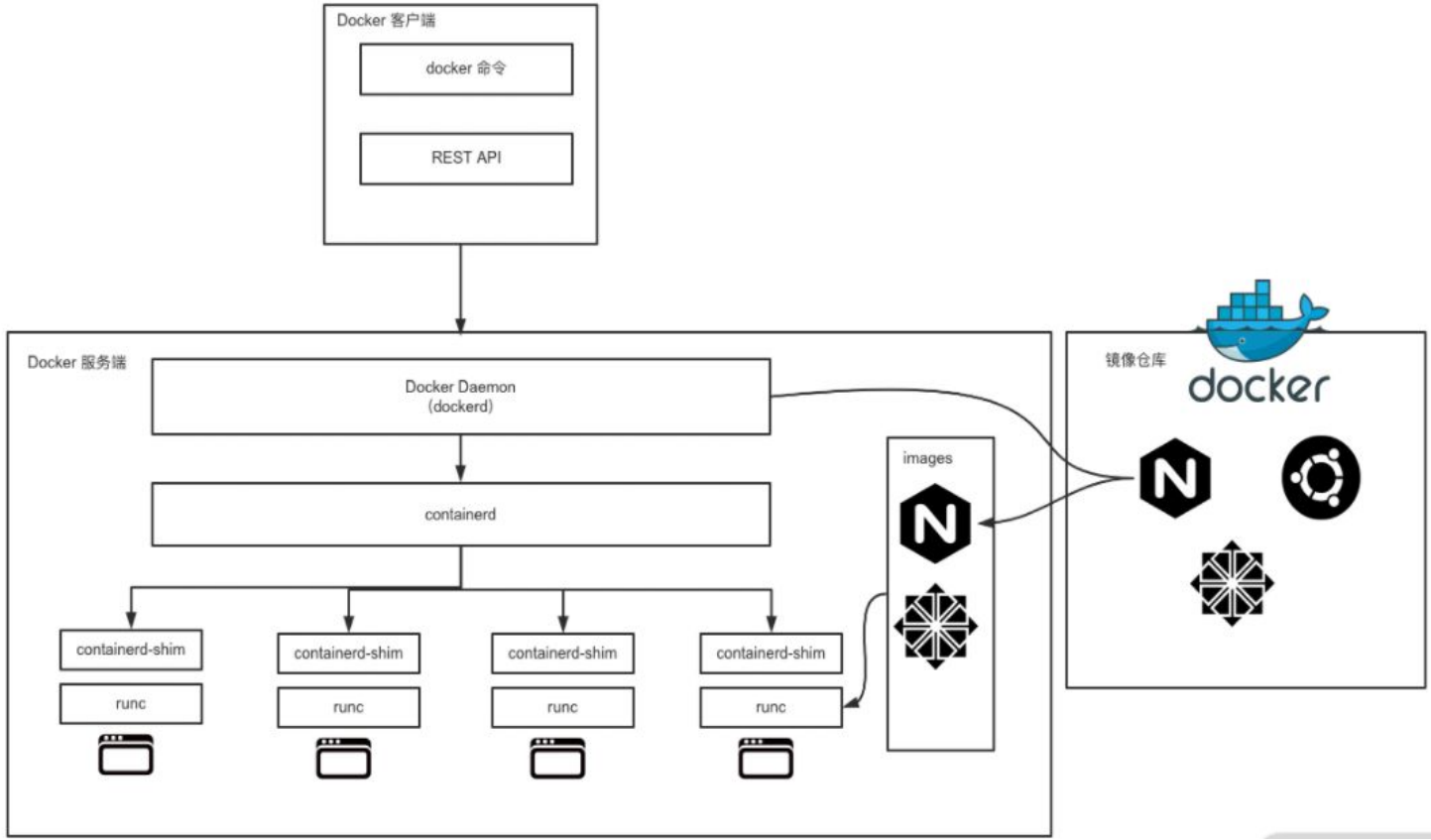
- Cgroups

  Cgroups 是 Linux 内核的另一项功能，可以限制和隔离进程的资源使用情况（CPU、内存、磁盘 I/O、网络等）。在容器的实现中，Cgroups 通常用来限制容器的 CPU 和内存等资源的使用。

- UnionFS

  一种通过创建文件层进程操作的文件系统，因此，联合文件系统非常轻快。Docker 使用联合文件系统为容器提供构建层，使得容器可以实现写时复制以及镜像的分层构建和存储。常用的联合文件系统有 AUFS、Overlay 和 Devicemapper 等。

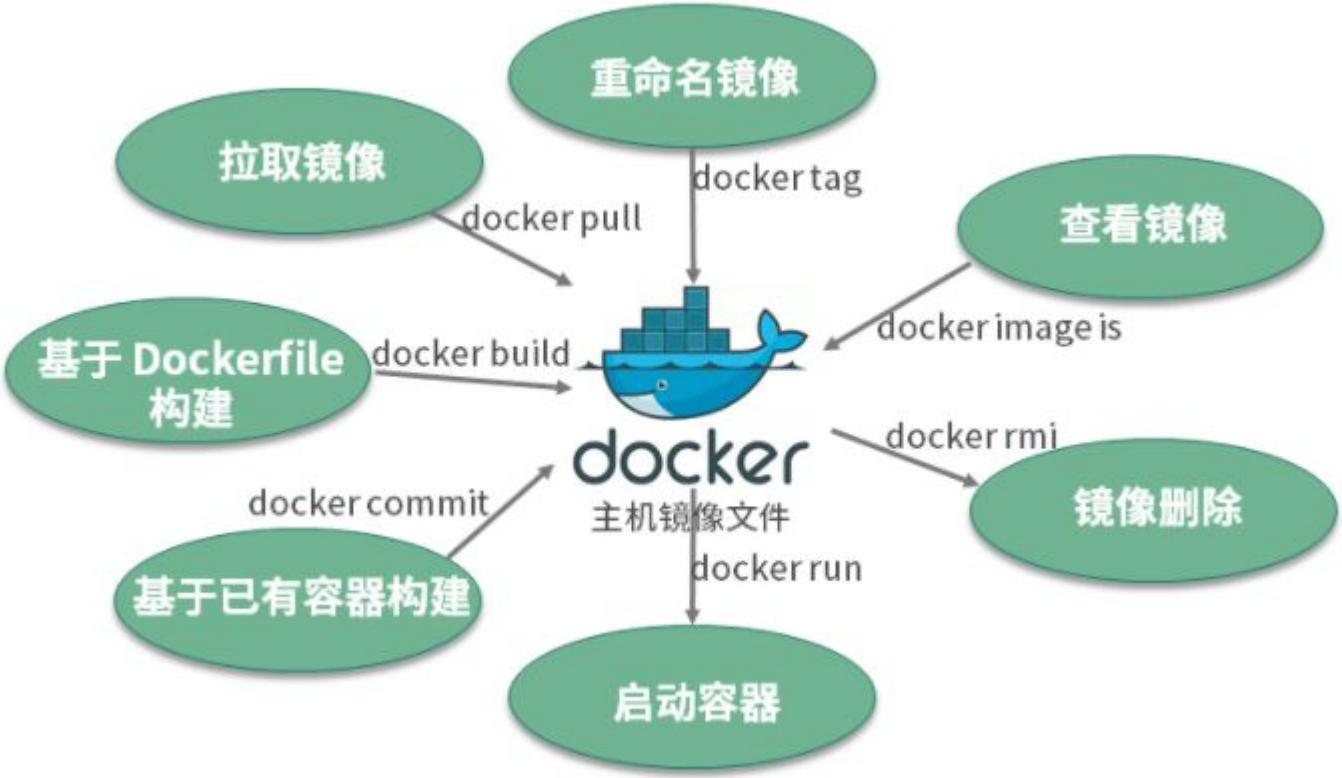
Docker 架构图


Docker 客户端
docker 命令
REST API
Docker 服务端
Docker Daemon
(dockerd)
containerd
containerd-shim
runc
containerd-shim
runc
containerd-shim
runc
containerd-shim
runc
images
镜像仓库
docker

核心概念

- 镜像Image
- 容器Container
- 仓库Repo
- Open Container Initiative

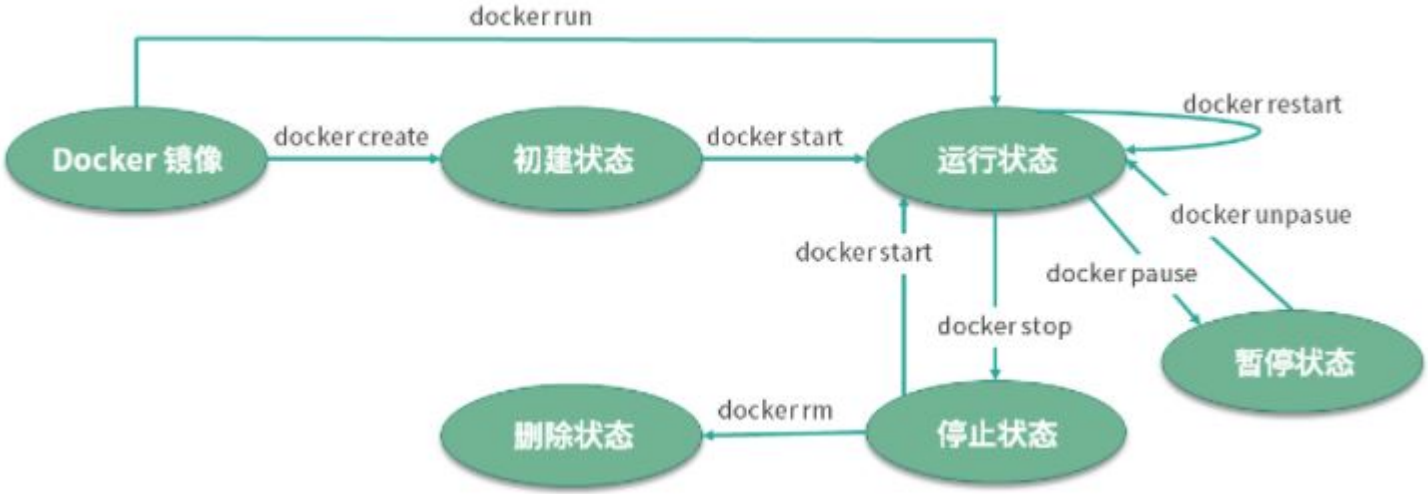
镜像操作


重命名镜像
拉取镜像
docker tag
docker pull
查看镜像
docker image is
基于 Dockerfile
构建
docker build
docker rmi
docker
主机镜像文件
镜像删除
docker commit
docker run
基于已有容器构建
启动容器

容器组成


容器
容器层，允许修改镜像的整个副本
容器层（layer）
可读写
镜像层（layer）
镜像层（layer）
镜像层（layer）
只读

容器的生命周期



created：初建状态
running：运行状态
stopped：停止状态
paused： 暂停状态
deleted： 删除状态

通过docker create命令生成的容器状态为初建状态，初建状态通过docker start命令可以转化为运行状态，运行状态的容器可以通过docker stop命令转化为停止状态，处于停止状态的容器可以通过docker start转化为运行状态，运行状态的容器也可以通过docker pause命令转化为暂停状态，处于暂停状态的容器可以通过docker unpause转化为运行状态 。处于初建状态、运行状态、停止状态、暂停状态的容器都可以直接删除。

Repo仓库

- Registry vs Repository
- 公共镜像仓库 - Docker Hub
- 私有镜像仓库
  - Distribution
  - Harbor
  - Elastic Container Registry

Dockerfile

- Dockerfile 本身是一个文本文件，方便存放在代码仓库做版本管理，可以很方便地找到各个版本之间的变更历史；
- 过程可追溯，Dockerfile 的每一行指令代表一个镜像层，根据 Dockerfile 的内容即可很明确地查看镜像的完整构建过程；
- 屏蔽构建环境异构，使用 Dockerfile 构建镜像无须考虑构建环境，基于相同 Dockerfile 无论在哪里运行，构建结果都一致。
- 生产实践中一定优先使用 Dockerfile 的方式构建镜像

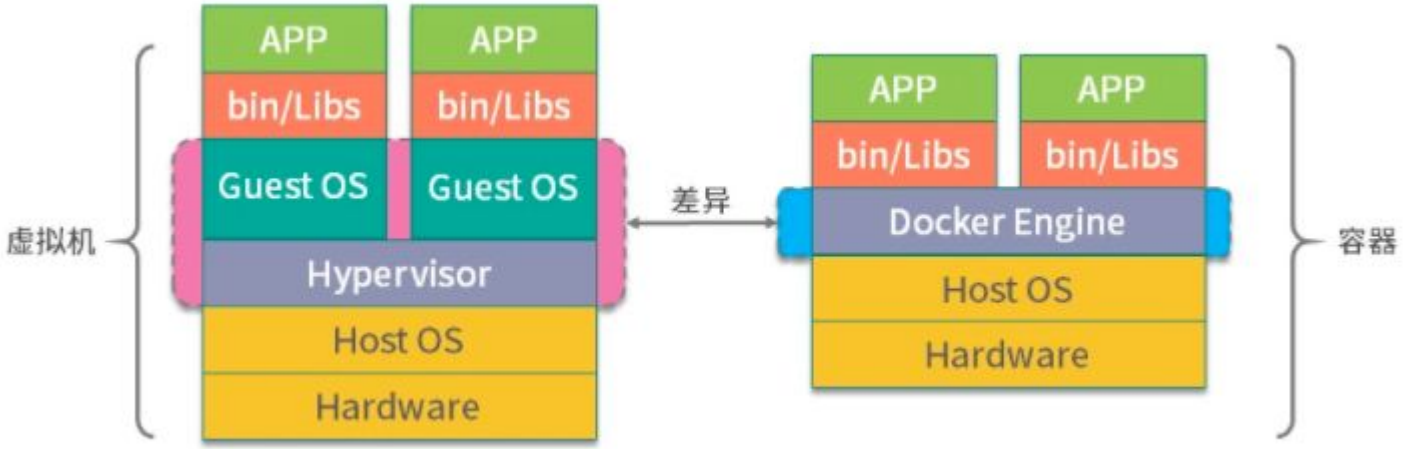
Dockerfile example

```
# syntax=docker/dockerfile:1

FROM node:12-alpine

RUN apk add --no-cache python2 g++ make

WORKDIR /app

COPY . .

RUN yarn install --production

CMD ["node", "src/index.js"]

EXPOSE 3000
```

Dockerfile书写原则

- 单一职责
- 提供注释信息
- 保持容器最小化
- 合理选择基础镜像
- 使用.dockerignore文件
- 镜像层之间尽量使用构建缓存
- 尽量减少镜像层数
- 正确设置时区

Docker安全



Docker容器的安全问题	解决办法
Docker作为一款容器引擎，本身也会存在一些安全漏洞（权限提升，信息泄漏）	使用Docker最新版本就可以得到更好的安全保障
Images存在安全漏洞，仓库漏洞，用户程序漏洞	在私有镜像仓库中安装image安全扫描组件，对上传的镜像进行检查，通过与CVE数据库对比，发现漏洞通就阻止上传并Alert
Linux内核只使用NameSpace，隔离不够，某些关键内容没有被完全隔离	宿主机内核尽量安装最新安全补丁；使用Linux的Capabilities划分权限；使用SELinux，AppArmor，GRSecurity等安全组件；每个容器都要限制资源使用
所有容器贡献主机内核 - 攻击者利用一些特殊手段导致内核崩溃，造成主机宕机从而影响主机上的服务	使用安全容器如Kata Containers

Docker监控

监控原理：

Docker 是基于 Namespace、Cgroups 和UnionFS实现的。其中 Cgroups 不仅可以用于容器资源的限制，还可以提供容器的资源使用率。无论何种监控方案的实现，底层数据都来源于 Cgroups。

Cgroups 的工作目录为/sys/fs/cgroup，/sys/fs/cgroup目录下包含了 Cgroups 的所有内容。Cgroups包含很多子系统，可以用来对不同的资源进行限制。例如对CPU、内存、PID、磁盘 IO等资源进行限制和监控。

- docker stats
- cAdvisor
- sysdig
- Prometheus
- …

容器编排Orchestration

- Docker Compose
- Docker Swarm
- Kubernetes

Next about Docker

- Docker Practice - 中，optional，Linux环境下容器化后实现CICD (Docker + Jenkins + Gitlab)
- Docker Practice - 下，optional，容器的底层实现原理和各种关键技术