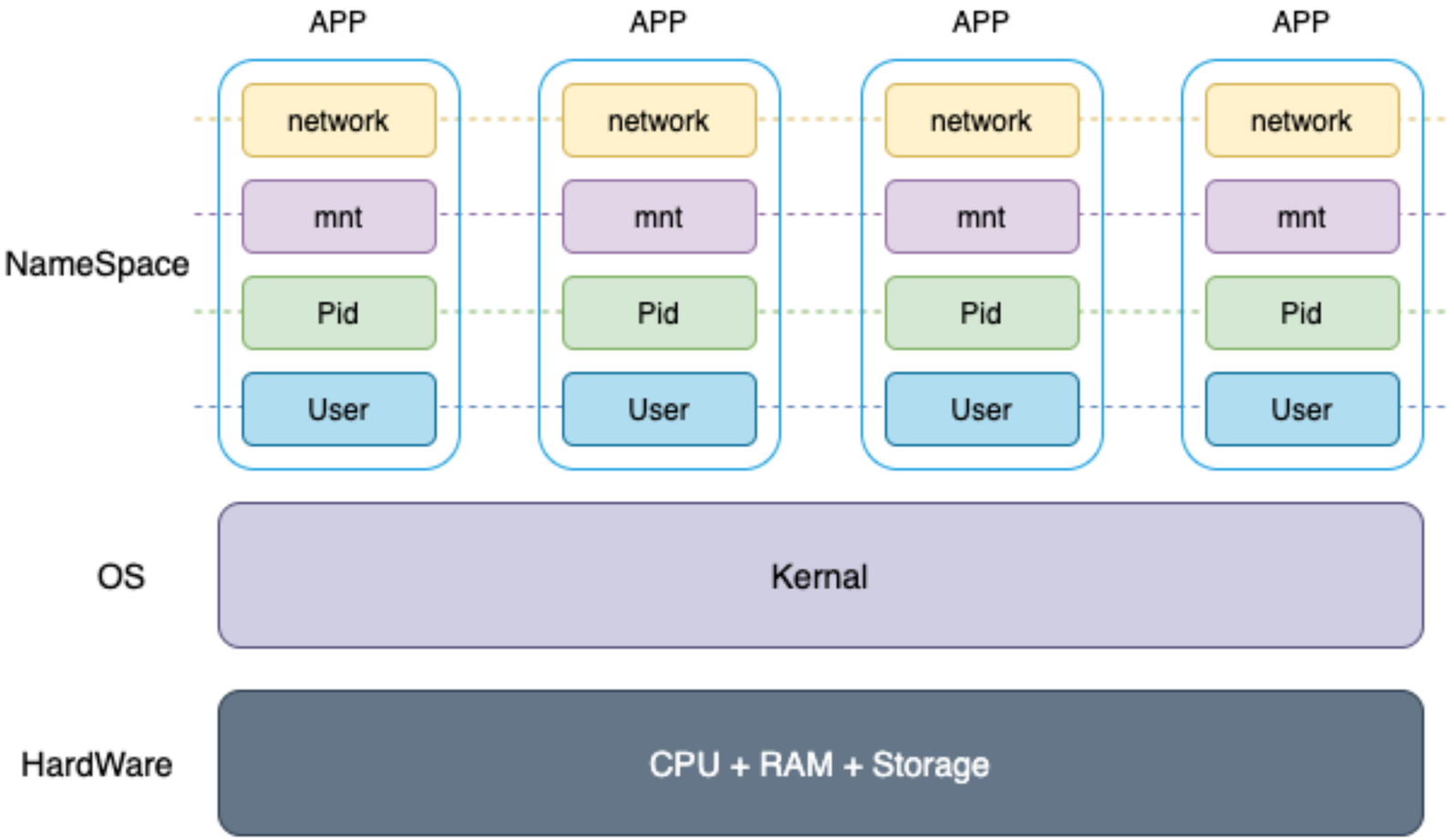
Docker freestyle

青铜

运行一个应用程序需要哪些资源?

How



- namespace 提供了全局资源隔离方法：
  - `Mount`: 隔离文件系统挂载点
  - `PID`: 隔离进程的ID
  - `Network`: 隔离网络资源
  - `User`: 隔离用户和用户组的ID
  - UTS: 隔离主机名和域名信息
  - IPC: 隔离进程间通信

容器定义

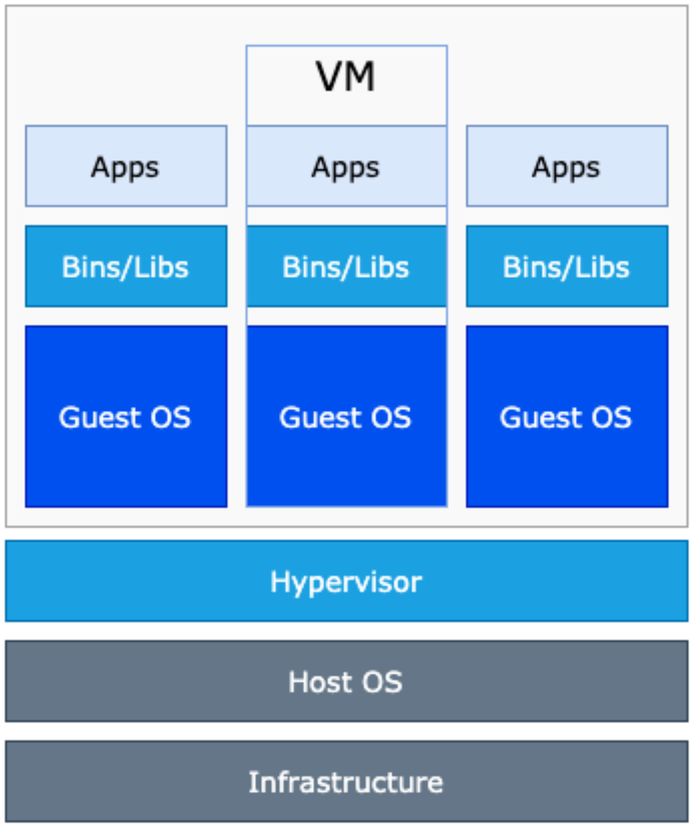
容器（container）是一种基于操作系统层虚拟化（OS level virtualization），为应用程序及其完整依赖提供了一个完全隔离的运行环境

容器 VS 虚拟机

虚拟机提供了硬件级别虚拟化，用来共享主机物理资源。

虚拟机需要单独的操作系统，硬件也是虚拟化的，需要启动管理程序管理虚拟机程序。

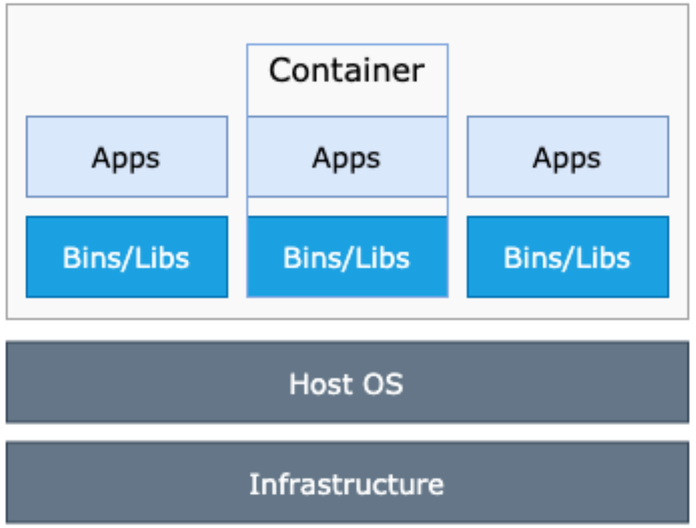
虚拟机适合搭建完整运行环境



容器通过操作系统虚拟化共享宿主机操作系统。

容器非常轻量级，提供了一套可移植，一致性的应用程序操作环境。

虚拟机适合运行单独的应用程序



容器 VS 虚拟机

特性	容器	虚拟机
虚拟程度	操作系统级虚拟	硬盘级虚拟
隔离性	部分隔离	完全隔离
启动时间	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	效能较慢
硬盘使用	单击支持近千个容器	一般为几个，不超过几十个
可移植性	方便	困难

容器核心架构

Namespace 对不同容器资源进行隔离

User

User Namespace 提供了对访问用户的隔离，
容器内部提供了一个虚拟的 ROOT 用户，
能访问容器内部所有文件资源权限，
但无法修改宿主机文件

PID

将容器内部进程与外部环境进行隔离，
pid namespace 是一个嵌套关系父子关系，
pid 1 代表系统初始化进程信息，是所有进程的祖先节点

Mount

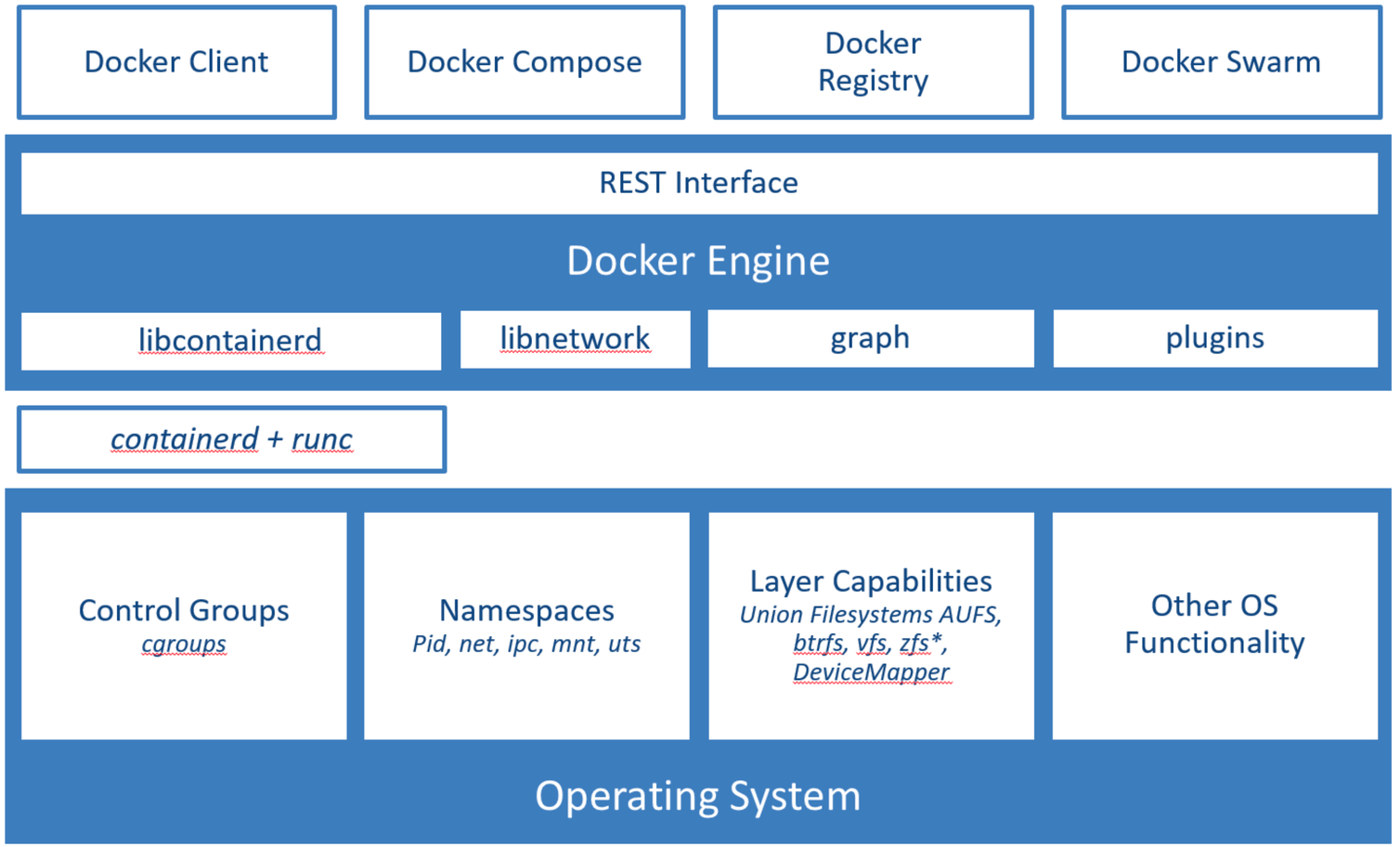
Mount 是对容器文件系统进行隔离：
1. BOOT filesystem: kernal
2. ROOT filesystem: ubuntu/centOS/debian
3. User fileSystem: /home/user
4. App fileSystem: jenkins/mysql/java/nginx

Network

容器在内部创建了一个虚拟的网络环境，将容器网络与宿主机隔离。

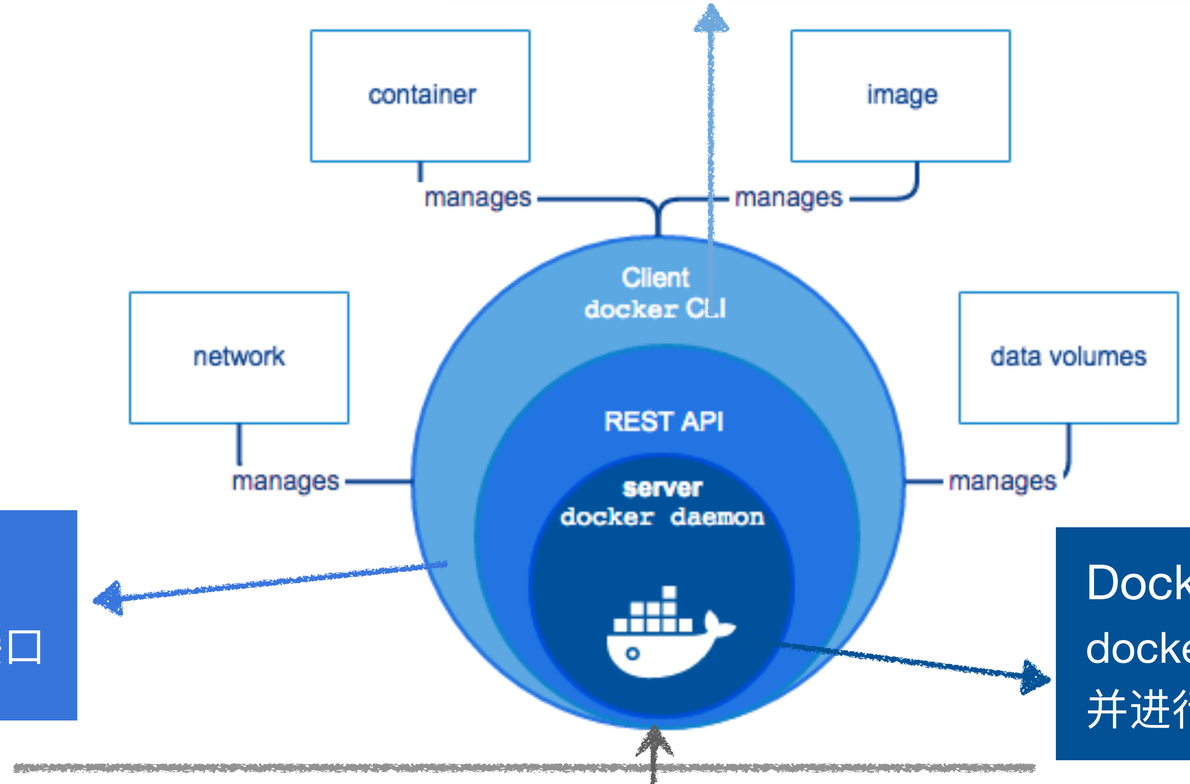
Docker

- Docker 基于 linux 容器技术，进行了进一步的封装，从文件系统、网络互联到进程隔离
- build once, run everywhere



Docker 产品架构

Docker CLI：
docker 命令行工具，用于和 Docker daemon 进行交互，管理image、container、network、volumes等实体



REST API：
docker 客户端和守护进程交互接口

Docker daemon：
docker 守护进程，接收 REST API 指令并进行后台交互

containerD：
container runtime 守护进程，负责管理容器生命周期，镜像传输和存储，容器执行和监控

containerD

runC

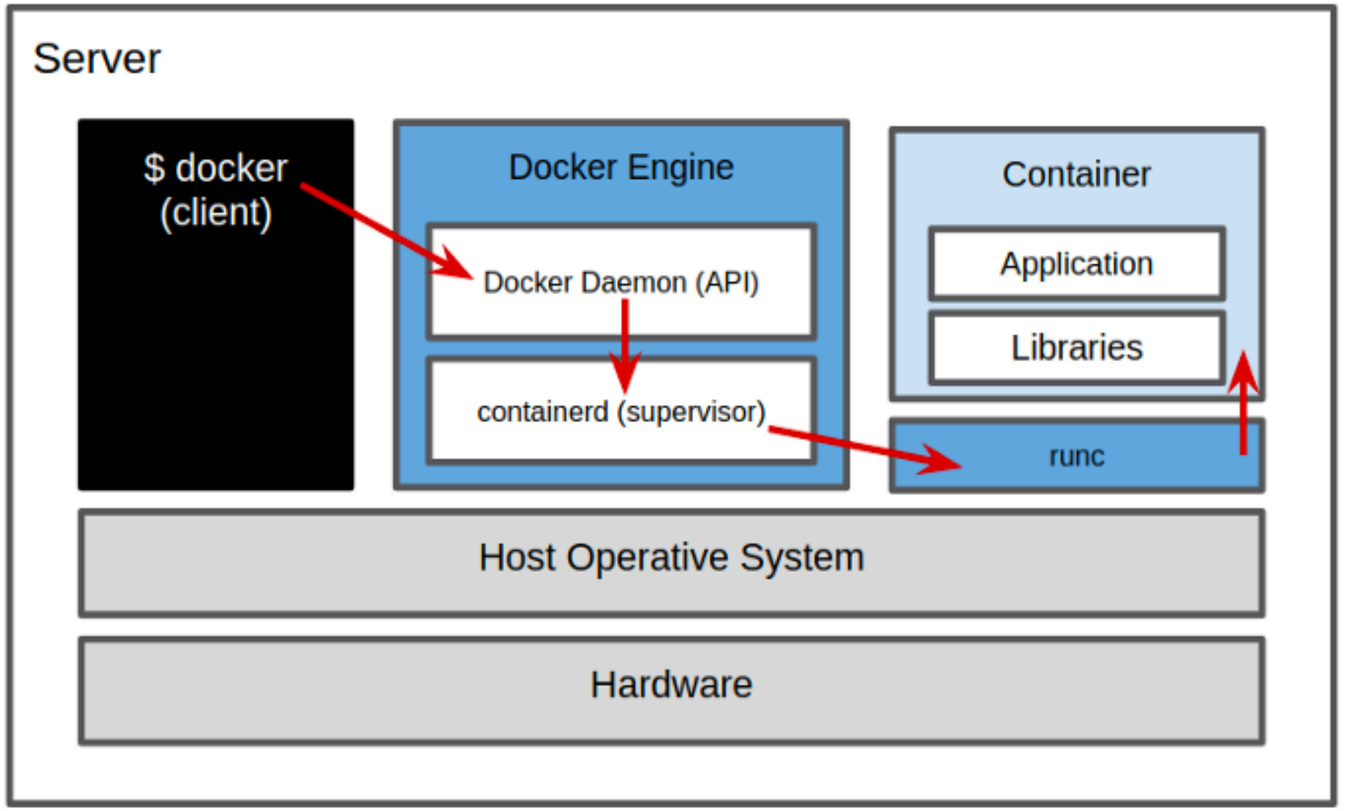
runC：
OCI 标准容器运行时规范

Docker 容器

- Docker 基于 linux 容器技术，进行了进一步的封装，从文件系统、网络互联到进程隔离
- Docker 启动容器做了什么?

```
$ docker run -itd —rm jenkins
```

  - 1. pull 镜像
  - 2. create 容器配置
  - 3. 启动容器阶段
    - 创建 USER NameSpace
    - 创建 PID NameSpace
    - 创建 MNT NameSpace
    - 创建 NET NameSpace
  - 4. 创建根进程 pid=1

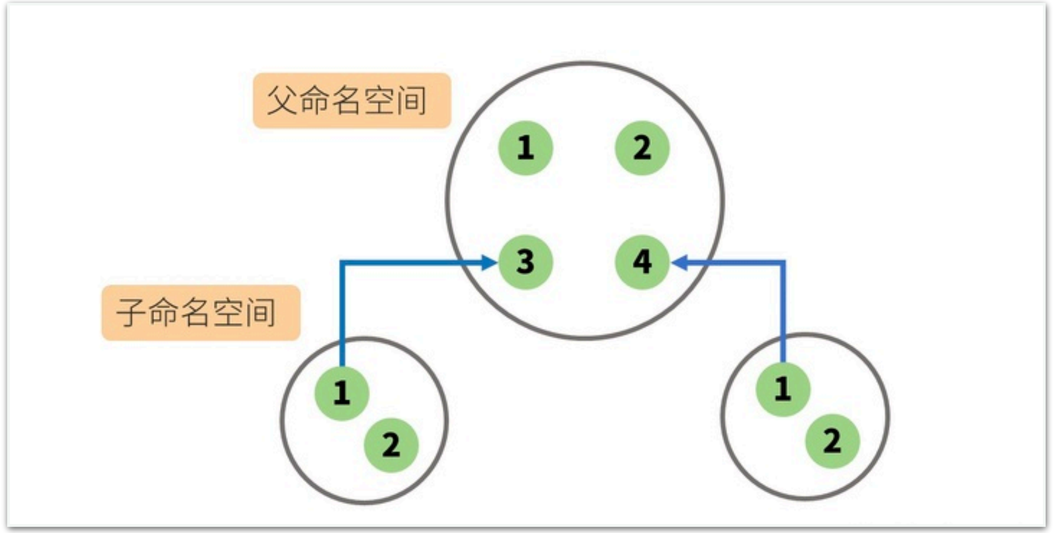


如何调试容器

- 启动以后的容器是一个黑盒，那么如何能进入容器内部呢?
  - exec
    - 在容器内部执行一个可执行程序
    - 生成一个新进程
    - 退出当前进程，容器不会停止
  - attach
    - 如果当前容器通过 /bin/bash 启动，attach 命令可以连接到容器 bash
    - 基于已有pid 1 进程，不会创建新进程
    - 退出当前进程，容器也会一并中止
- 在外部访问已创建的容器?
  - docker inspect
  - docker0 bridge
  - docker -p 端口转发

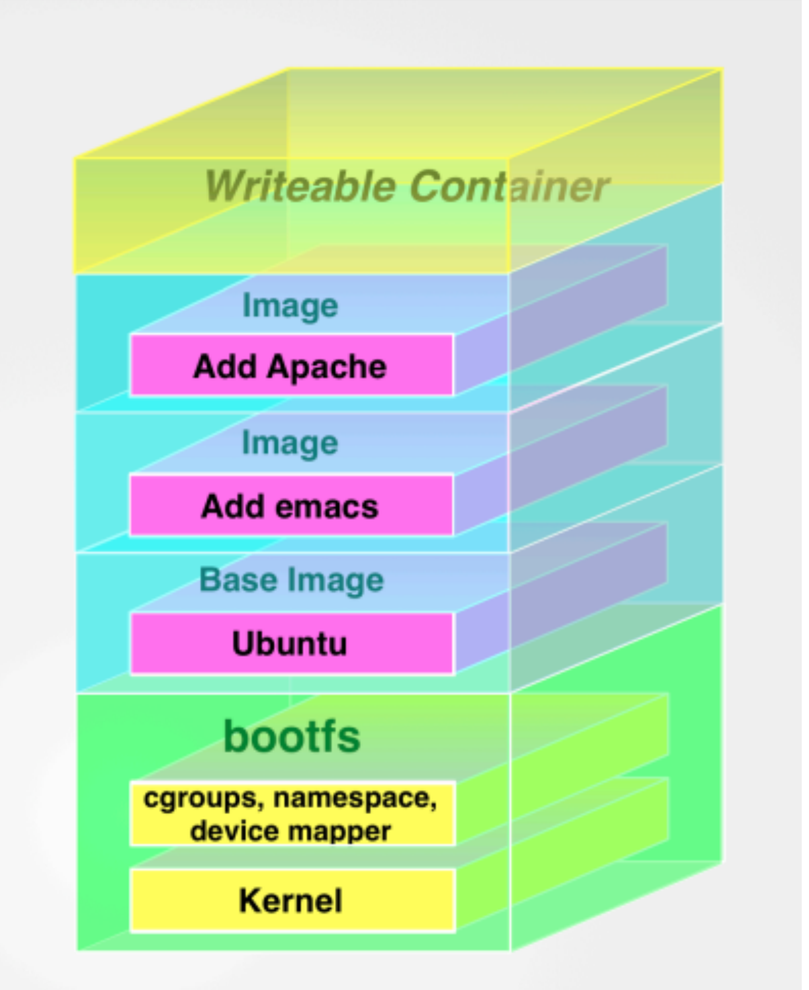
PID 1 的魔法

- PID 1 在 linux 系统中代表系统启动的初始进程（init process），系统其它进程都为其子进程
  - 物理机：systemd
  - 虚拟机：systemd
  - 容器: anything（bash/java/python）,取决于镜像启动命令
- docker logs
  - 打印容器内进程 PID 为 1 的 STDOUT
- docker top
  - 查看容器中正在运行的进程信息（PID：1），并输出为宿主机的进程 id



Docker 镜像

- 镜像是是一个只读模板，用于创建容器。镜像分层（layers）构建，而定义这些层次的文件叫 **Dockerfile**
- 镜像就是分层的文件系统，镜像通过挂载到容器生效
  - ROOT fileSystem：ubuntu/centOS/debian
  - USER fileSystem：/home/user
  - App fileSystem：jenkins/python/java
- 镜像一层一层往上叠加，内容为增量更新，为了支持写操作，镜像最上层为可读写层，其他层面都为只读层
- 底层镜像层可以被多个容器共享，可反复读取与启动



镜像 rule

- 同一个镜像文件可以创建多个容器，区别就是最上层的读写层，容器启动之后的文件变更变更记录都写在读写层上，读写层随着容器的生命周期而存在。
- 当有文件冲突时，上层（upper layer）优先级高于下层（down layer）的优先级。
- 镜像的增量更新基于 linux 联合文件系统（Union File System）实现
- Copy on Write

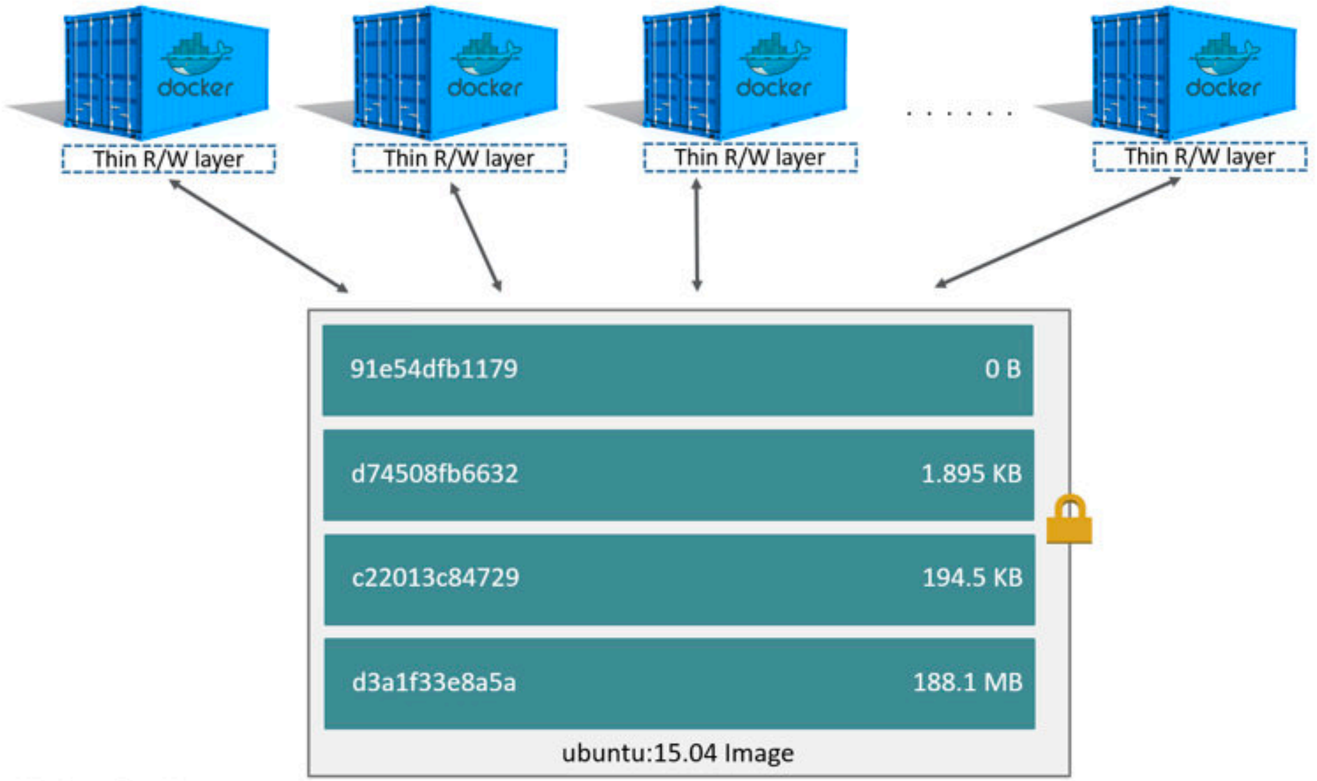
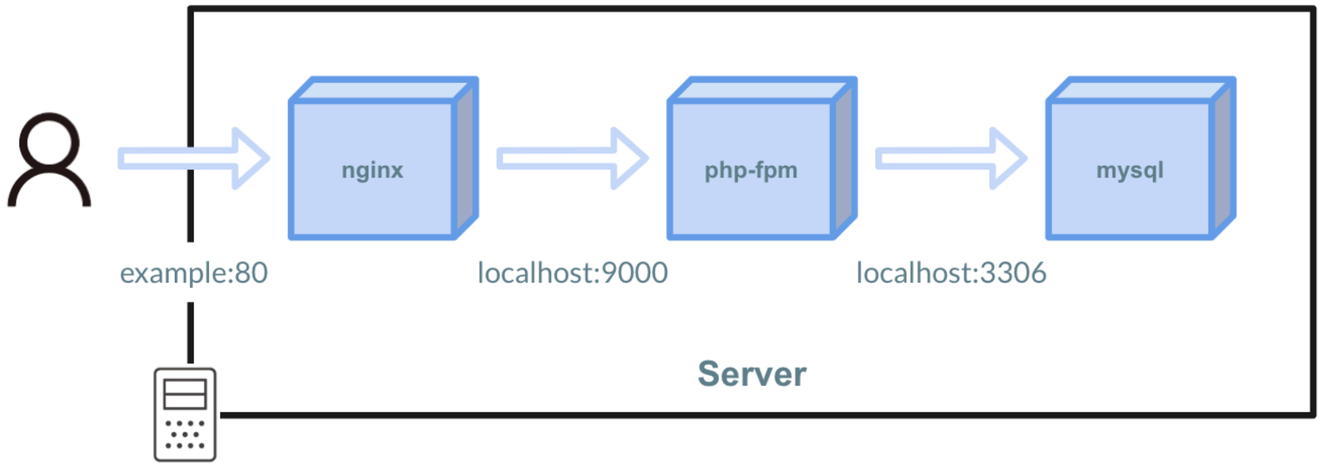


Image layer

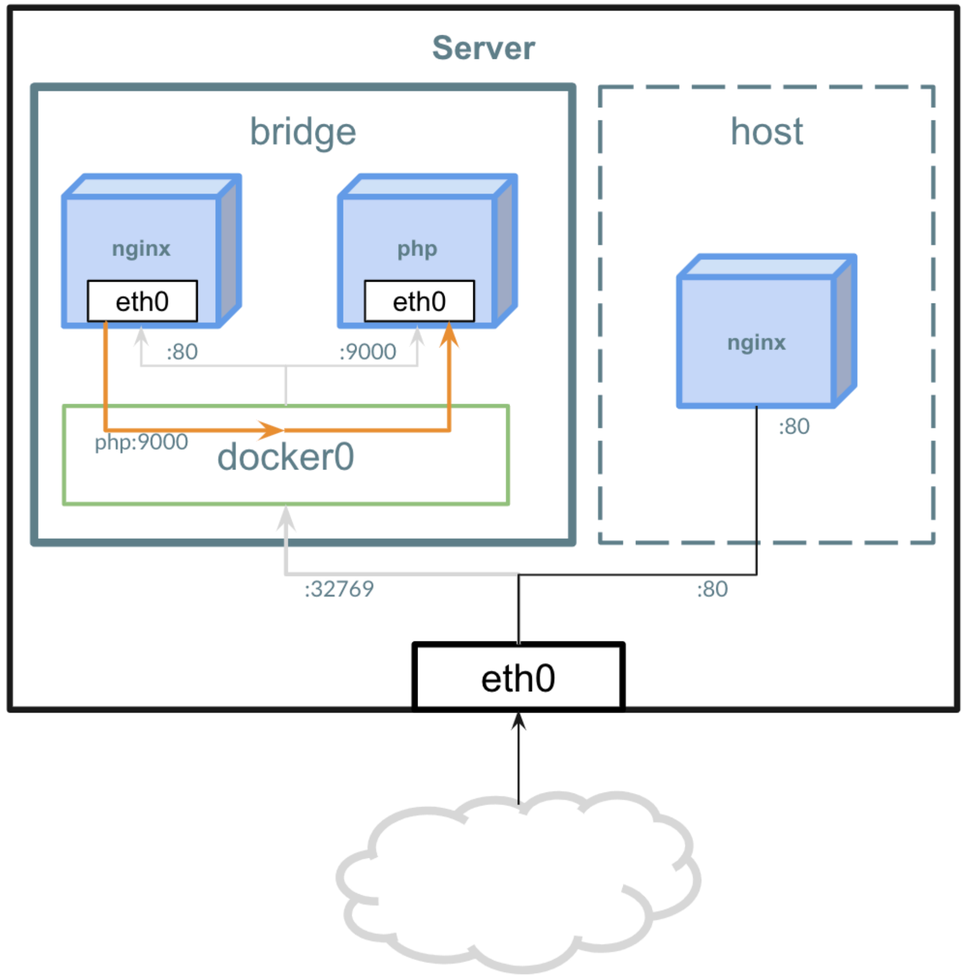
- Image Layer 分为两类：Data Layer 和 Meta Layer。
- Data Layer：
  - 静态的文件系统（dirs/files）
  - 存储目录：/var/libs/docker/overlay2
- Meta Layer：
  - 动态镜像元数据
  - 存储目录：/var/libs/docker/image
  - ENV
  - Volume
  - User
  - Port
  - History

Docker network



- docker container 一般推荐最佳实践是**一个容器运行一个单独的进程**
- 当多个容器进程间通信时，可以通过 docker 自带的 network 模块实现通信
- docker network 分为三类驱动：
  - bridge: docker 默认驱动类型，基于 linux network namespace 实现网络隔离
  - host: 基于主机 eth0 网卡驱动，**不具备网络隔离**
  - none: 关闭容器网络连接

bridge network



- Docker 安装时默认启动 **docker0** 网桥，会给容器分配一个单独的 ip，网段为127.0.0.1。
- 每次创建容器时，docker 会分享一个空闲的 ip 给容器的 etho 网卡，在同一个网段之内实现通信
- 容器可以通过端口映射，对宿主机开放访问
- docker 也可以通过自定义网络，具备DNS 解析 ip 功能，直接容器名称加入这个网络

```
# 创建自定义网络
$ docker network create my-net

# 将容器连接到统一网络
$ docker run -itd —net=my-net
```

容器运维

docker logs

打印容器内 PID 为 1 的进程 STDOUT

```
$ docker logs jenkins
```

docker stats

查看容器资源使用信息

```
$ docker stats jenkins
```

docker top

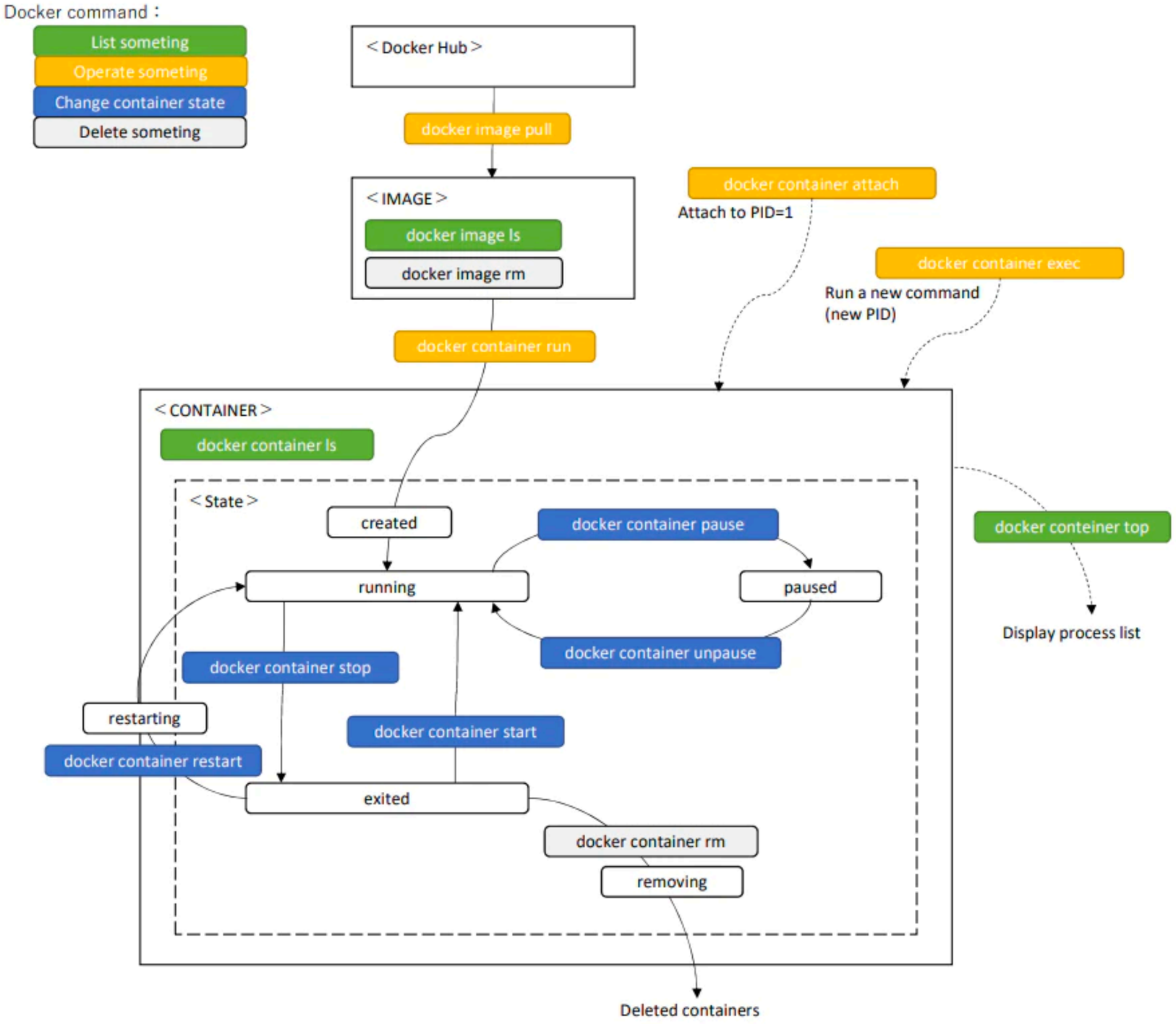
展示容器主进程在宿主机下的进程信息

```
$ docker ps
$ docker logs jenkins
```

docker events

获取docker daemon 实时日志信息

```
$ docker events
```


Docker command：
List someting
Operate someting
Change container state
Delete someting
＜Docker Hub＞
docker image pull
＜IMAGE＞
docker image ls
docker image rm
docker container run
docker container attach
Attach to PID=1
docker container exec
Run a new command
(new PID)
＜CONTAINER＞
docker container ls
＜State＞
created
docker container pause
running
paused
docker container unpause
docker container stop
restarting
docker container start
docker container restart
exited
docker container rm
removing
docker conteiner top
Display process list
Deleted containers

总结

Namespace 提供隔离环境

- User Namespace
- Network NameSpace
- PID NameSpace
- Mount NameSpace

虚拟机和容器

- 虚拟机基于硬件虚拟化提供了完整隔离环境
- 容器基于操作系统虚拟化，轻量、可移植开发环境

docker 容器

- 基于 NameSpace 提供单独隔离环境
- 通过 exec 和 attach 可以进入容器调试
- 神奇的 pid 1

docker 镜像

- 镜像是分层的文件系统，用于创建容器
- 镜像支持增量更新，下层镜像只读，最上层可读写
- 镜像文件分为静态文件系统和元文件信息

docker 网络

- docker 推荐一个容器对应一个单独进程
- docker 通过虚拟网卡实现同一网段通信
- 通过端口转发实现与宿主机通信

资源监控（Monitor）

- docker logs 打印主进程 STDOUT
- docker top 获取宿主机中容器主进程信息
- docker stats 获取容器资源使用日志
- docker events 实时监控 daemon 日志

THANKS
Q&A