Docker：概述

郭宁@mobike

2018-08-23

目录

Docker 有什么用：解决什么问题

Docker Containerization Unlocks the Potential for Dev and Ops

Freedom of choice, agile operations and integrated security
for legacy and cloud-native applications

How Docker Works for You

Developers

Tooling that is simple to use, yet powerful and delivers a great user experience so you can focus on what you love - writing great code.

→ Get Started

IT Operations

Docker delivers an enterprise-ready container platform to deploy and run applications in a way that makes the best sense for your customers and business.

→ Get Started

Business Leader

Docker provides a platform to drive your digital transformation by accelerating new innovation while dramatically driving down your existing IT costs.

→ Get Started

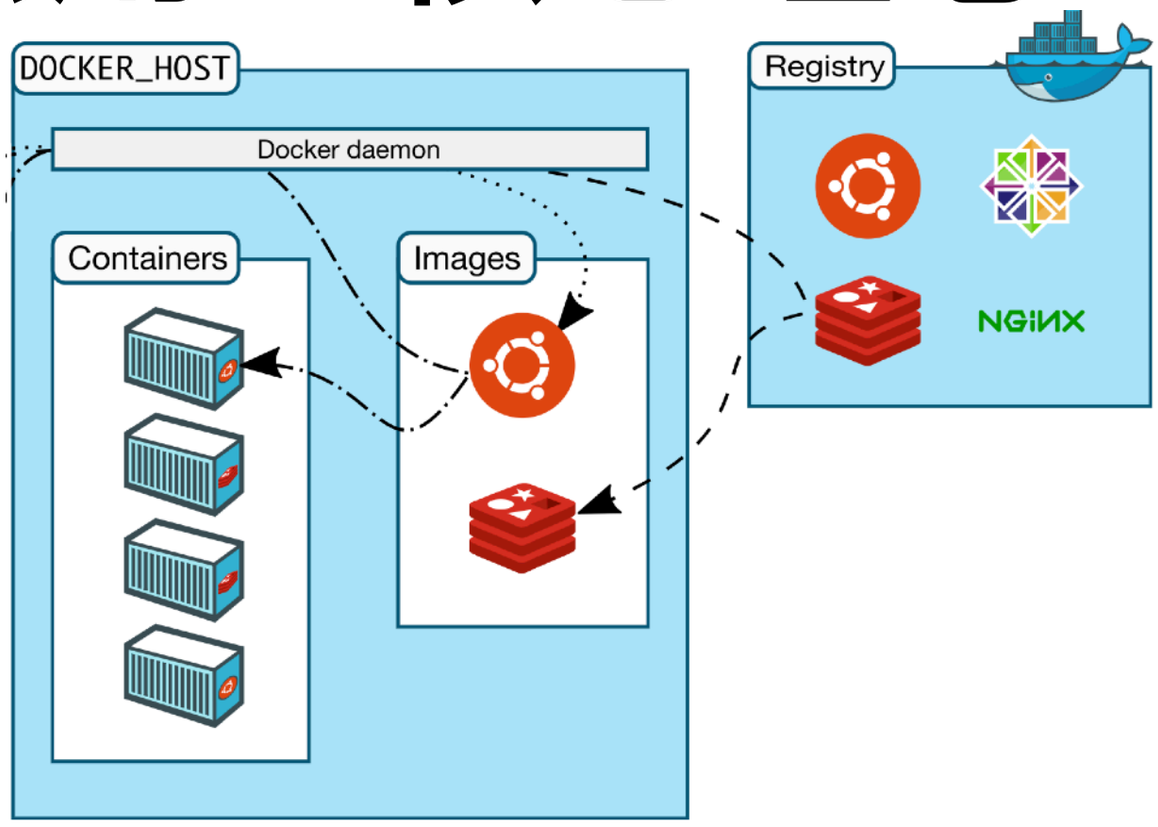
Docker 有什么用：解决什么问题

- 解决的核心问题：
  - **效率**

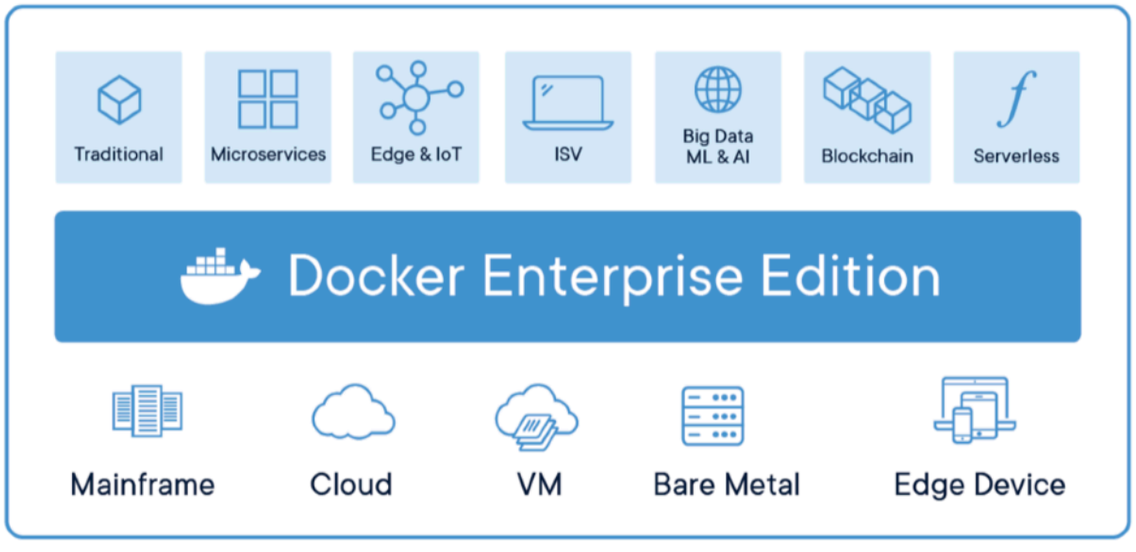
- 「**开发**」和「**运维**」的**效率**
  - **单个环境**：机器准备、环境准备、基础服务准备等（硬件 → 操作系统 → 业务服务）
    - **快速部署环境**：环境的**多样性**，不同物理机 or 云主机环境下，快速构造服务的运行环境，部署多种组件
    - **高效管理环境**：服务的**高效伸缩**、**服务监控**以及**自动拉起**（只能感知进程，无法感知业务）
  - **多环境之间**：
    - 环境之间灵活隔离：隔离 or 耦合

Docker 有什么用：核心理念

- **核心理念**：
  - **标准化**
  - 减少多样性，**降低复杂度**，提升效率

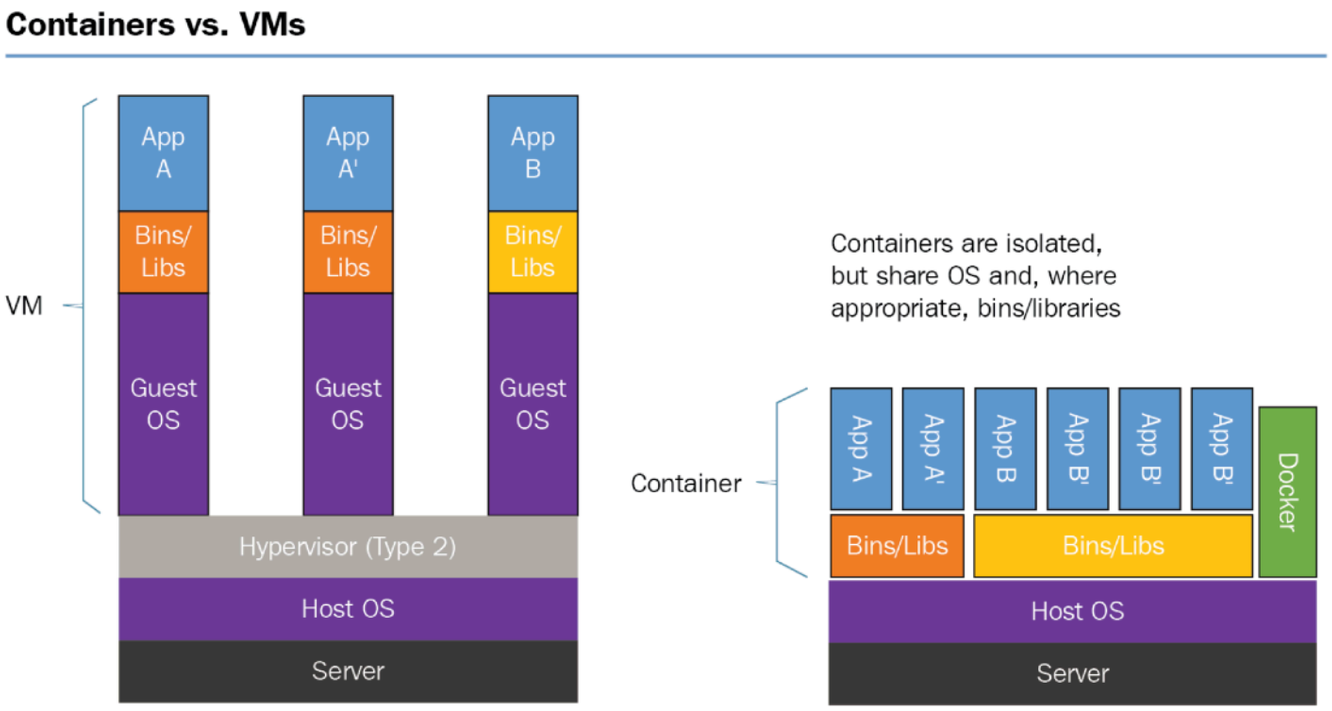


- **具体**：
  - **统一**：
    - 虚拟层之上，**统一暴露接口**
    - **一个镜像**：涵盖 OS 和 APP 层逻辑
  - **集中**：集中**管理**镜像、**分发**镜像



Docker 有什么用：之前方案对比

- Docker 之前，如何解决上面问题的：
  - **Docker 容器**：
    - 直接利用系统内核，进行资源隔离、限额，没有虚拟 OS 等额外的虚拟层，系统效率损耗小
  - **VM**：
    - 效率损耗：虚拟硬件，虚拟 OS（完整的 OS 视图）
    - 存在复杂、独立的管理程序 Hypervisor，耗用系统资源



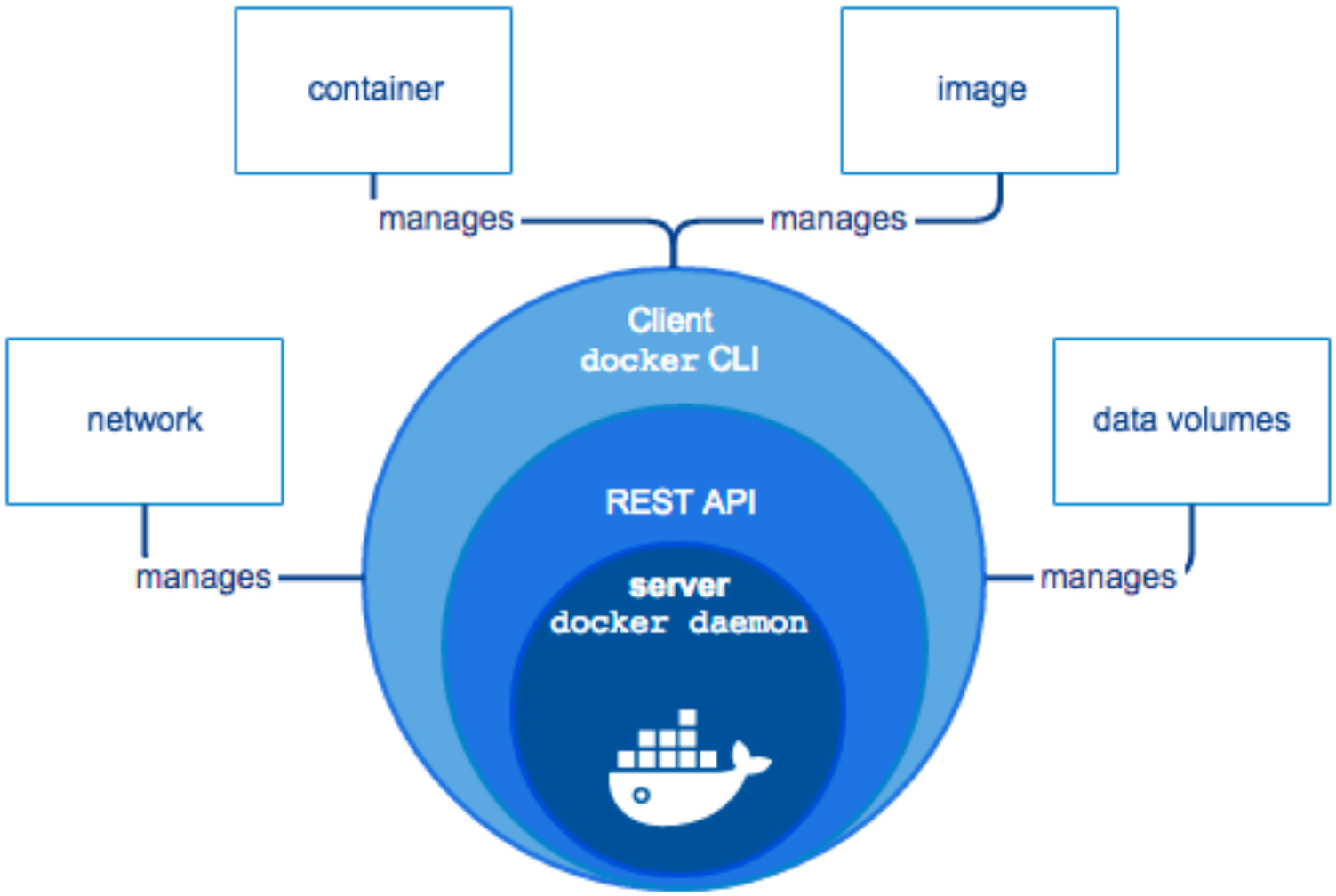
目录

Docker 技术原理：技术架构

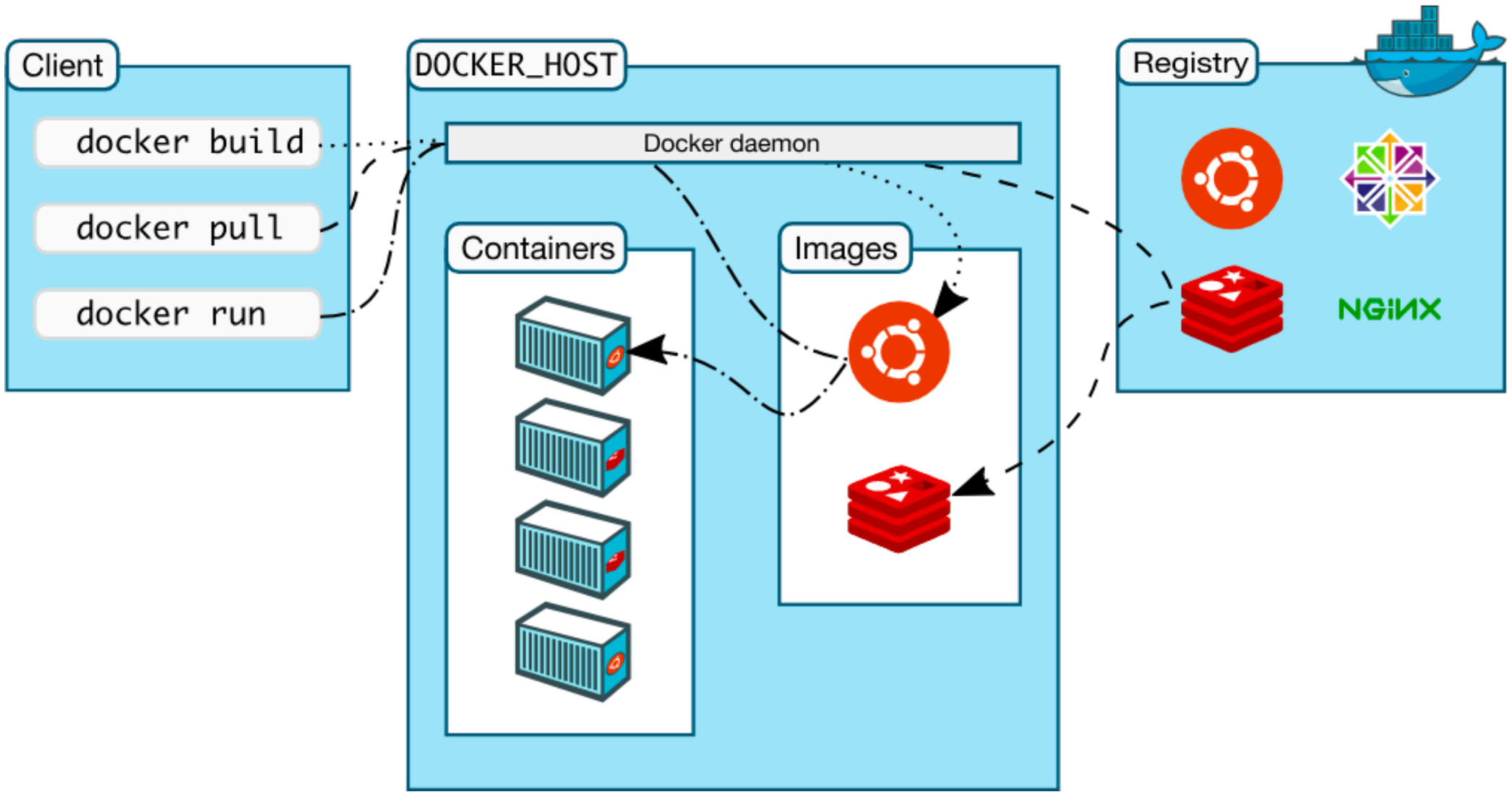
- **Docker Engine**（Docker 核心引擎），**客户端/服务器** 模式。

- 整体分为 2 部分：
  - **Client**：**客户端**，提供命令接口的客服端（CLI, Command Line Interface），例如 **docker** 命令
  - **Server**：**服务器**，分为 2 部分
    - **对外接口**：**REST API**，接收 Client 的命令，管理容器
    - **后台进程**：后台进程 **dockerd**，实现功能，包括创建并管理容器、镜像，以及网络、磁盘。

Docker 技术原理：技术架构

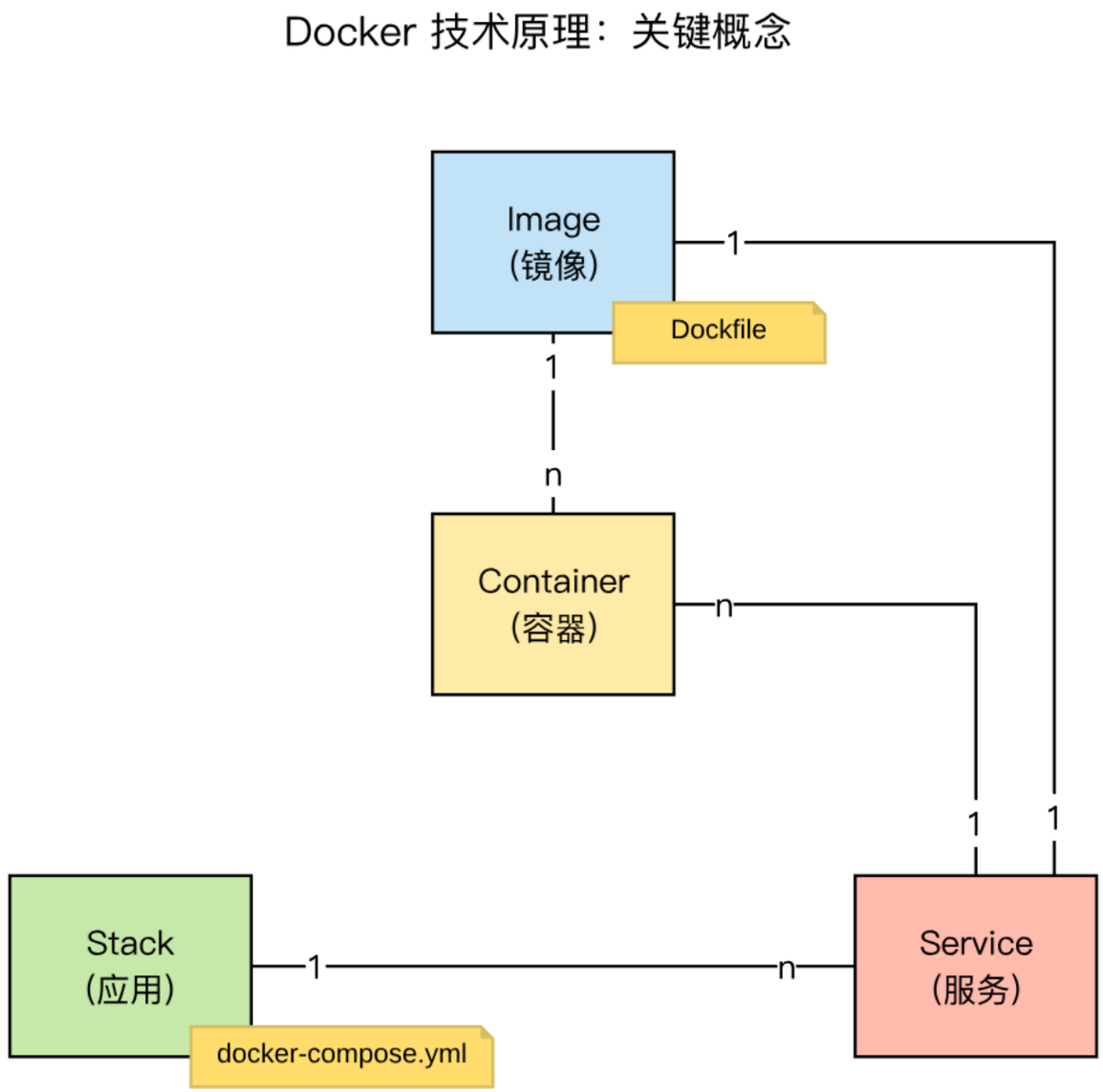


Docker 技术原理：技术架构



Docker 技术原理：关键概念

- **Image（镜像）**：
  - 是一个可执行的文件，用于启动一个应用，包含
  - 应用代码、依赖的库、运行环境（JRE 等）、环境变量、配置文件。
- **Container（容器）**：
  - 使用 Image 启动的一个进程实例。
- **Service（服务）**：
  - 一组 Container ，提供的对外服务。这些 Container 使用同一个 image 镜像文件。
- **Stack（应用）**：
  - 一组 Service，相互协作，对外提供服务。
  - 可以看做一个完整的 Application
  - 一些复杂业务场景，会拆分为多个 Stack



Docker 技术原理：Dockerfile

- **构建 Image（镜像）**：Dockerfile
  - 基础镜像
  - 环境变量
  - 执行命令
  - 磁盘、网络等的配置

- Dockerfile 文件：
  - 定义「构建镜像」的过程
  - 一个 Dockerfile，只描述一个 Image

```
# Use an official Python runtime as a parent image
FROM python:2.7-slim                                    基础镜像：OS + python 运行环境

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app
                                                        安装其他组件
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World
                                                        执行命令
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Docker 技术原理：Compose

- **构建 Stack：Docker compose**
  - 需要独立安装
  - 描述 Service 的细节：
    - 镜像
    - 容器数量
    - 资源限额
    - 重启策略

- **docker-compose.yml 文件**：
  - 定义 Service 细节
  - 一个 docker-compose文件，可描述多个 Service

```yaml
version: "3"
services:
  web:                                  # 定义 service
    # replace username/repo:tag with your name and image details
    image: username/repo:tag            # 对应的 Image
    deploy:
      replicas: 5                       # 容器数量
      restart_policy:
        condition: on-failure
      resources:
        limits:                         # 资源限额：CPU 和 内存
          cpus: "0.5"
          memory: 50M
        reservations:
          cpus: "0.25"
          memory: 20M
    ports:
      - "80:80"
    networks:
      - webnet
  visualizer:                           # 这是另一个 Service
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - webnet
networks:
  webnet:
```

Docker 技术原理：底层原理

- **资源隔离 & 资源限额**
  - 系统资源，从 OS 角度，分为：
    - CPU：进程（涵盖 CPU + 内存）
    - 存储：外部磁盘
    - 网络
- **资源的隔离**：namespaces 技术
- **资源的限额**：control groups 技术（cgroups），CPU 数量 和 内存大小

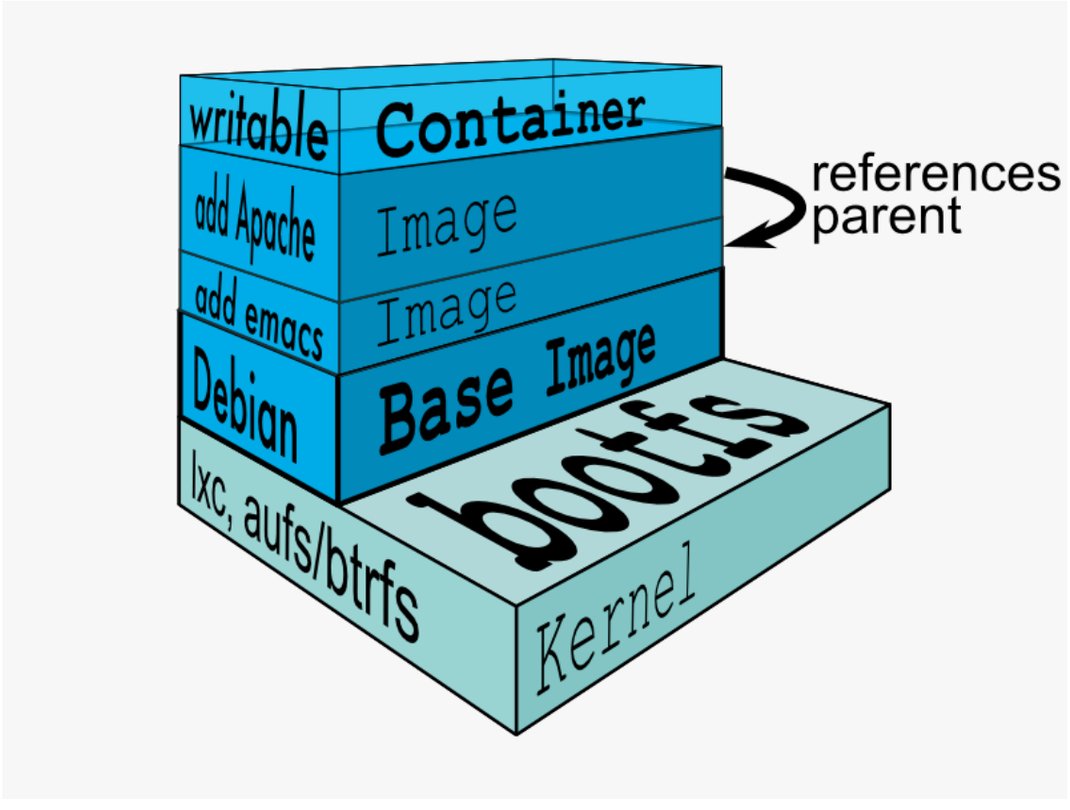
资源类别	资源隔离：namespaces	资源限额：cgroups	备注
CPU：进程（涵盖 CPU + 内存）	• pid namespace：进程隔离 • ipc namespace：进程间通信隔离	内存 + CPU	
存储：外部磁盘	• mnt namespace：磁盘挂载点隔离	--	
网络	• net namespace：网络接口隔离	--	
用户	• uts namespace：用户隔离	--	

Docker 技术原理：底层原理

- **联合文件系统（分层文件系统）**：Union file systems, or UnionFS，通过分层方式，标识**差量部分**。

- 提供几点便利：
  - **构造镜像**：只构造**差量部分**，高效
  - **分发镜像**：只分发**差量部分**，高效

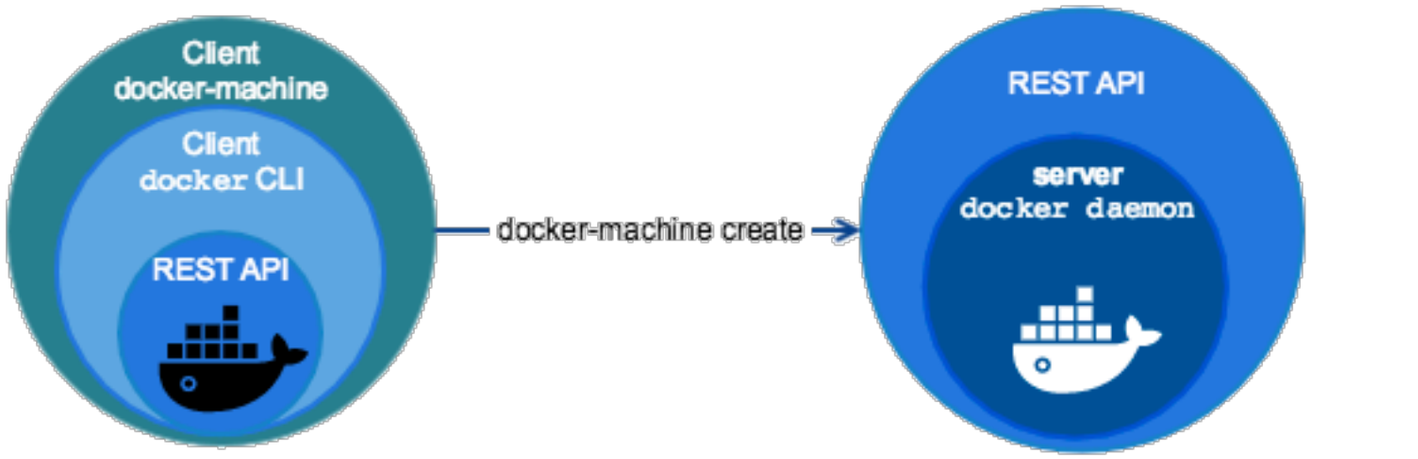
- 现在有多种联合文件系统的实现：
  - AUFS
  - btrfs
  - vfs
  - DeviceMapper
- 使用命令： docker info 可以查看当前系统的文件格式



目录

Docker 集群：Docker Machine

- **Docker Machine**，目标：
  - 在**各种平台**上，快速安装 **Docker 环境**，即，快速创建 **Docker 服务器**（**Docker Node**）。



Docker 集群：Docker Machine

- docker-machine 官方支持的驱动，-d 选项可以指定：
  - amazonec2
  - azure
  - digitalocean
  - exoscale
  - generic
  - google
  - hyperv
  - none
  - openstack
  - rackspace
  - softlayer
  - virtualbox
  - vmwarevcloudair
  - vmwarefusion
  - vmwarevsphere

基本操作：

1. Docker Machine 安装：是独立的组件，需要单独安装
2. Docker Machine 使用

在本地物理机上，使用 `virtualbox` 驱动，创建 Docker 运行环境：

```
# 使用 virtualbox 驱动，  创建 Docker 主机，命名为 test
docker-machine create -d virtualbox test
```

创建 Docker Node

```
# 查看列表
docker-machine ls

# 通过 ssh 登录机器
docker-machine ssh test
```

登录 Docker Node

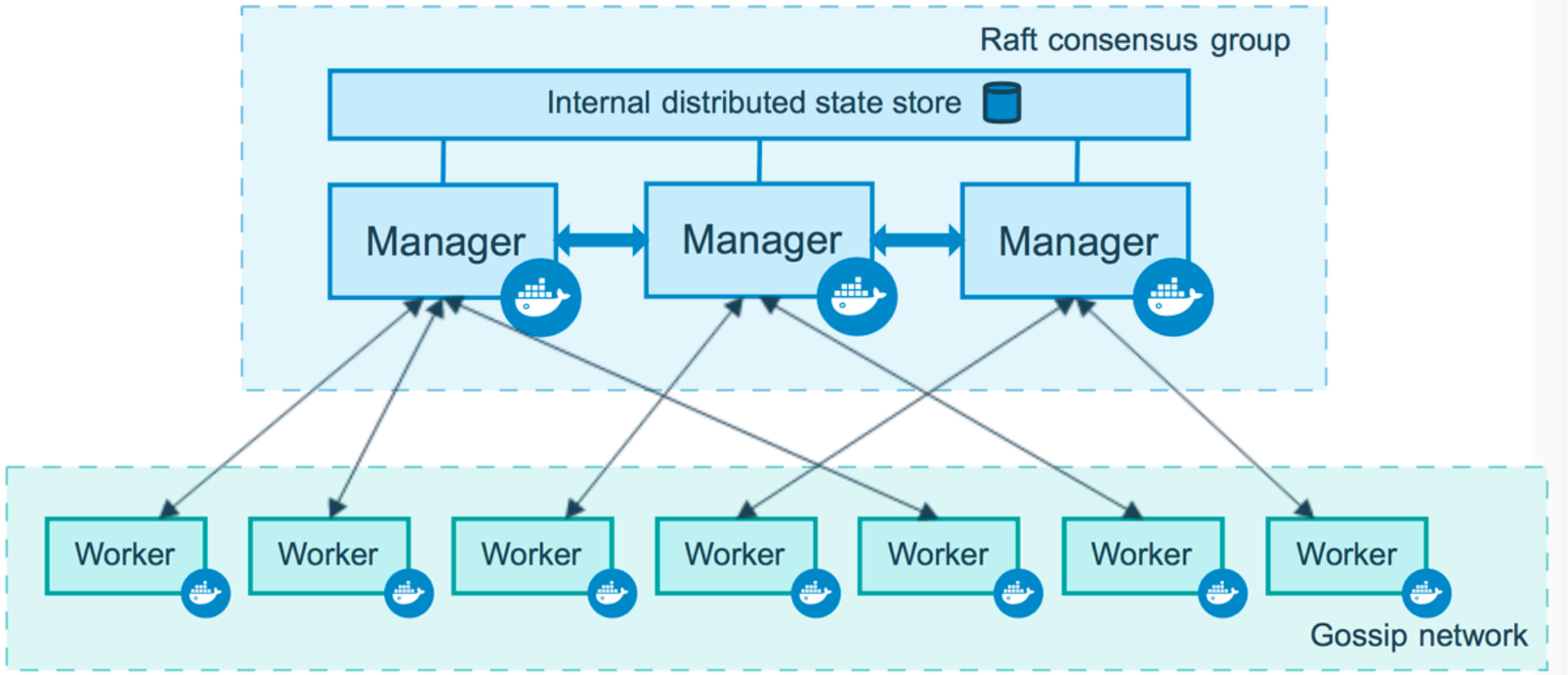
Docker 集群：Docker Swarm

- **swarm 的作用：**
    - Docker Engine 内置（原始）的**「集群管理」**工具。

- 关于 swarm 的**历史演进**：
    - **Docker Swarm**：
        - 在 Docker 1.12 版本之前，是独立的组件
        - 独立于 Docker Engine 之外，需要**独立安装**；
    - **swarm mode**：
        - 在 **Docker 1.12+** (涵盖1.12)，**内置**到了 Docker Engine 之中，无需独立安装；

Docker 集群：Swarm mode

- 几个核心概念：
    - **Swarm（集群）**：一堆机器，构造成 docker 集群，叫做 Swarm。
    - **Node（节点）**：Swarm 集群中，安装了 Docker 运行环境的机器，物理机 or 虚拟机
        - 角色：
            - **Manager**：负责管理集群
                - Manager 中，只能有一个 Leader
                - 多个 Manager 之间，依赖 Raft 协议，选举出 Leader
            - **Worker**：负责运行具体的 Container
                - Manager 节点，默认，也做为 Worker 节点，承担 Container 运行任务



Docker 集群：Kubernetes

- 涉及几个**核心概念** & 逻辑：
    - Node
    - Pod
    - Deployment
    - Label

- TODO：
    - 今后单独再说

目录

讨论环节

- Q & A

参考资料

- https://www.docker.com/ 官网
- https://docs.docker.com/ 官网文档
- https://docs.docker.com/engine/docker-overview/ 概览

附录：Docker 版本编号

- Docker 的所有版本，都可以在 GitHub 上查看：
  - https://github.com/moby/moby/labels

- Docker 在 2017.01.18 发布了 **version/1.13** 版本后，
  - 就**不再使用 1.x** 的版本编排方式了
  - 改为使用 **YY.MM** 的**日期格式**

- 当前讨论时，最新的版本为 18.09