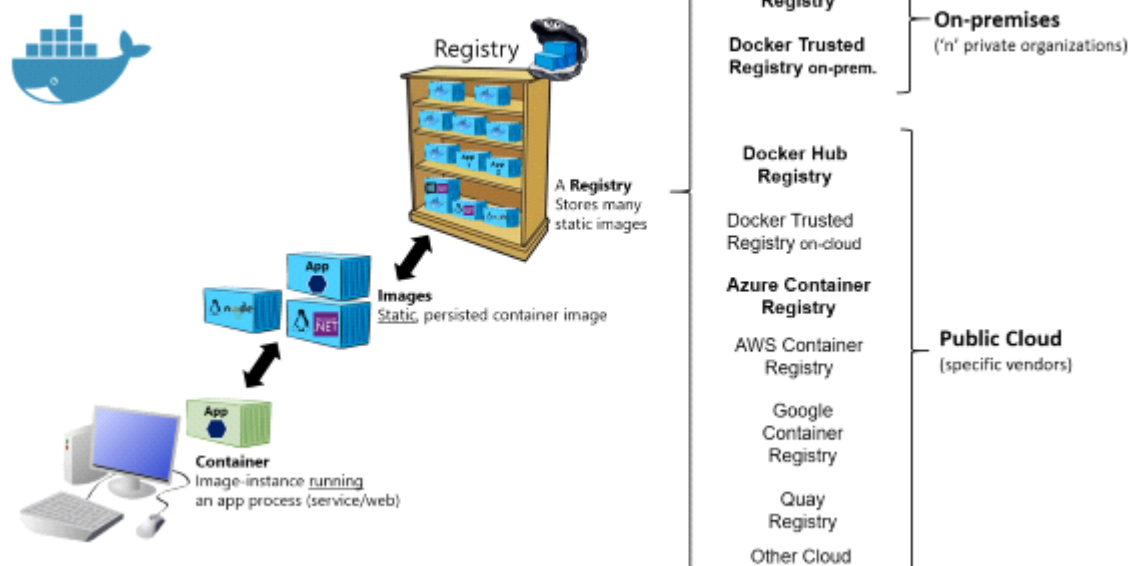


# 第04期-基于微服务的容器化应用程序

2020年1月2日 11:27

## Docker 容器、映像和注册表

### Basic taxonomy in Docker



## 容器设计原则

容器表示单个进程，是进程的边界，生命周期与容器同生共死，可在 Dockerfile 中看到入口点定义，可以是一个长时间运行的 Web 服务，也可以是执行完就结束的批处理程序。

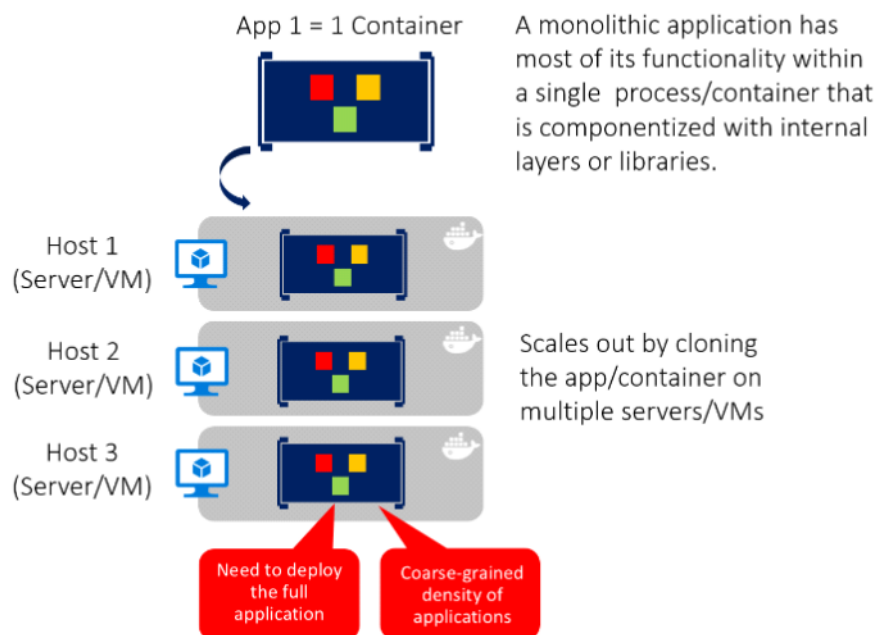
如果进程失败，则容器结束，Orchestrator 接管。如果 Orchestrator 已配置为始终需要5个实例保持运行，而其中一个实例失败，则 Orchestrator 会创建另一个容器实例，来替换失败的进程。在批处理作业中，使用参数启动该进程。进程完成，则工作完成。

某些情况下，可能需要在单个容器中运行多个进程，可以使用 Supervisor 或类似的工具处理在单个容器内启动多个进程的情况。

## 容器化单体式应用程序

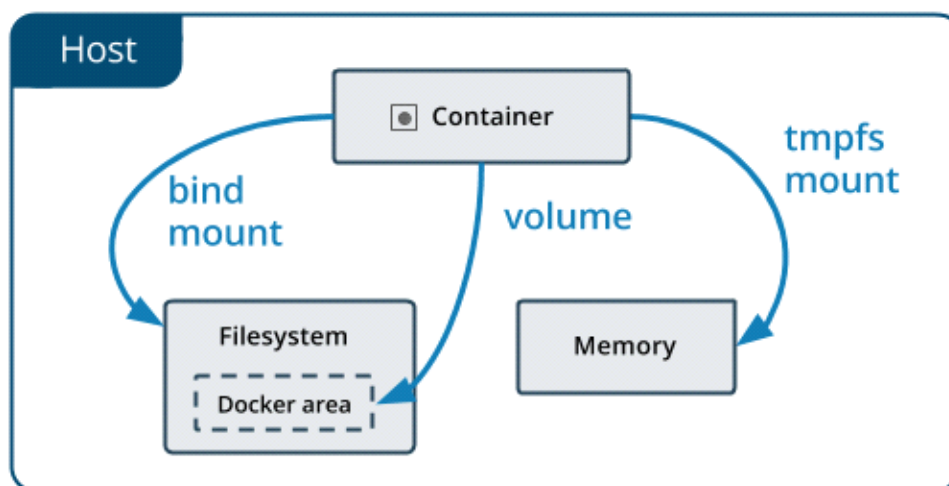
也可用容器部署单体应用，若要增加处理能力，可分布式，可进行横向扩展，即，只需添加更多的容器副本，并将负载均衡器置于容器前端。

# Monolithic Containerized application



## Docker 应用程序中的状态和数据共享

Docker 为容器提供了两个选项来将文件存储在主机中，以便即使容器停止后文件也可以持久存储：卷和 绑定安装。如果您在 Linux 上运行 Docker，则还可以使用 tmpfs 挂载。如果您在 Windows 上运行 Docker，则还可以使用命名管道。



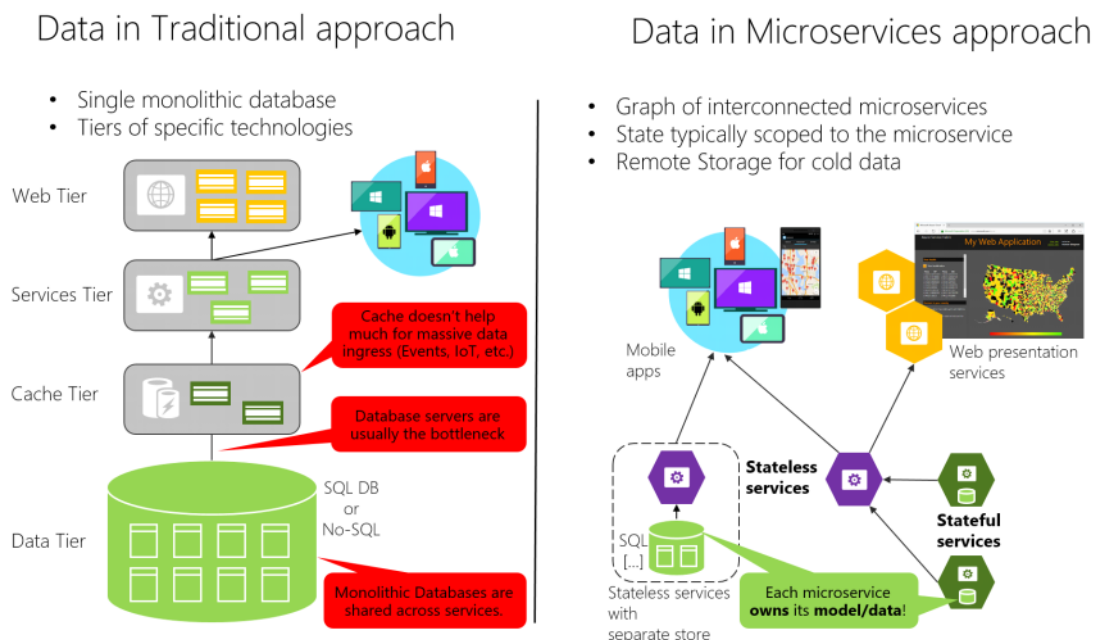
<https://docs.docker.com/storage>

## 面向服务的体系结构

SOA 意味着通过将应用程序分解为多个服务（通常为 HTTP 服务），将其分为不同类型（例如子系统或层），从而来划分应用程序的结构。对于传统的面向服务的体系结构，Docker 容器都是有用的（但不是必需的）。

## 微服务体系结构

每个微服务必须有自己的数据和逻辑，就像一个完整的应用程序一样，必须能独立部署。以下是单体应用与微服务的比较。



单体应用：共享数据库，分层体系结构，ACID事务和SQL连接查询。

微服务：每个服务拥有自己的模型和数据。数据访问较为复杂，（REST、gRPC、SOAP 等）进行同步访问，或通过（AMQP）进行异步最终一致性访问。

## 微服务和DDD界定上下文模式之间的关系

微服务的概念源自域驱动设计 (DDD) 中的界定上下文 (BC) 模式，DDD 将大模型划分为多个 BC 并对其边界十分明确，以此处理大模型。每个 BC 必须具有其自己的模型和数据库；同样，每个微服务拥有其相关的数据。

## 逻辑体系结构与物理体系结构

应用程序的逻辑体系结构和物理体系结构之间存在的差异。系统的逻辑体系结构和逻辑边界不一定要与物理或部署体系结构一对一映射。这种情况存在一定的可能性，但通常不会发生。

微服务的逻辑体系结构并不总是与物理部署体系结构一致，理想状态有时不可达。

