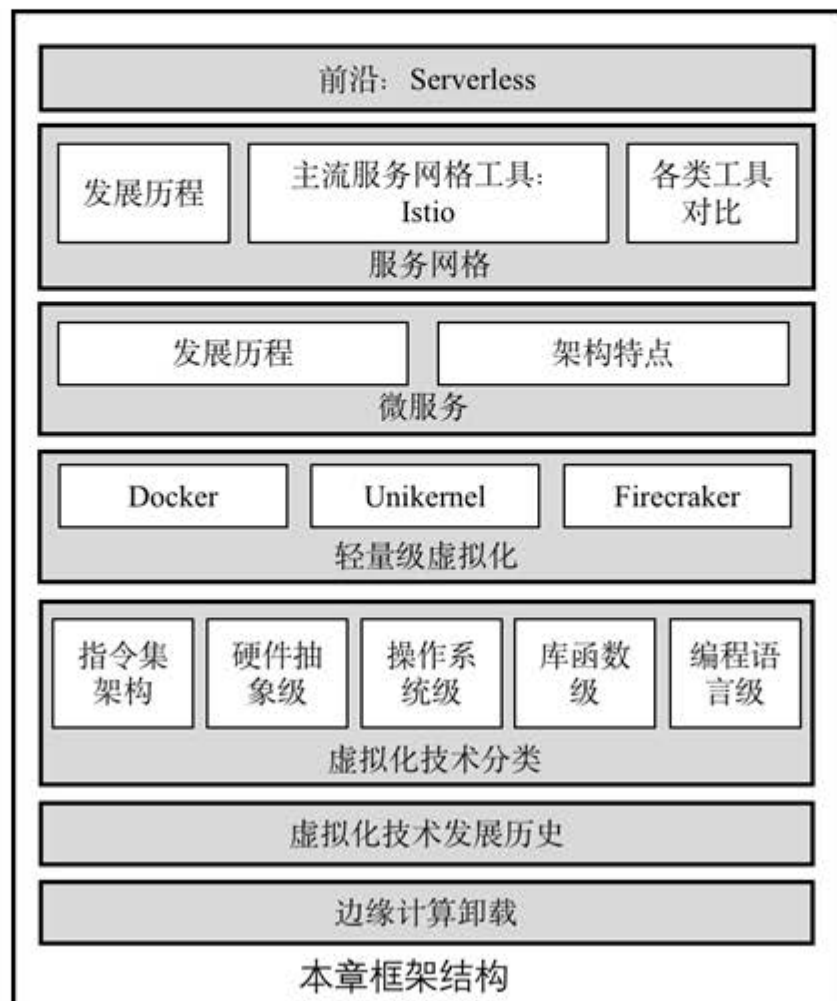


虚拟化技术

► 章节概览:

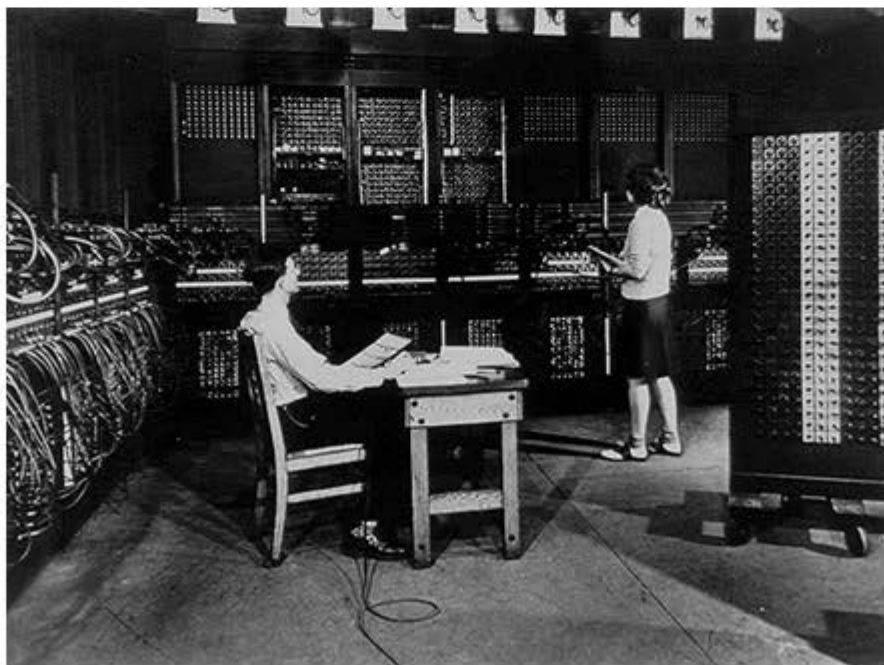


- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 虚拟化技术是一种资源管理技术，为云计算提供了灵活且强大的资源管理能力。
- ▶ 边缘计算也需要使用有限的资源为边缘用户服务，因此虚拟化技术成为了边缘计算核心支撑技术之一。
- ▶ 本章主要讨论服务器虚拟化，即通过虚拟化技术模拟出一个完整的系统或者可执行环境。

- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ **虚拟化的概念最早可追溯到20世纪60年代。**
 - 在计算机刚诞生时，科学家需要提前写好程序，之后输入程序，再串行的执行程序。
 - 1956年6月，克里斯托弗·斯特雷奇首次提出了虚拟化的概念。



虚拟化技术发展历史 (2)

6

20世纪60年代早期，超级计算机Atlas横空出世。



虚拟化技术发展历史 (3)

7

20世纪60年代中期，IBM设计了一个与Atlas相似的架构，首次提及了虚拟机的术语。



- ▶ 1974年，Popek和Goldberg提出了当时虚拟化的三个条件：等价性、可控性和高效性。
- ▶ 1979年，Unix V7增加chroot的系统调用，实现了一种类似于文件系统层的虚拟化和隔离。
- ▶ 1988年，提出SoftPC，它使得用户可以在Unix workstations上运行DOS应用。
- ▶ 随着SoftPC的发布，其他虚拟化产品和技术也相继出现。

- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 指令架构级虚拟化
- ▶ 硬件抽象级虚拟化
- ▶ 操作系统级虚拟化
- ▶ 库函数级虚拟化
- ▶ 编程语言级虚拟化

- ▶ 物理主机称为宿主机，虚拟化技术虚拟出的目标系统一般称为客户机。
- ▶ 指令集体系结构级别的虚拟化通过完全在中间软件中模拟目标指令集体系结构来实现。
- ▶ 指令架构级虚拟化在直观性和健壮性方面表现较好。
- ▶ Bochs：能够模拟一个完整的因特尔x86架构计算机的软件程序。
- ▶ QEMU：一个使用了动态翻译器的快速处理器仿真软件。

- ▶ 指令架构级虚拟化
- ▶ 硬件抽象级虚拟化
- ▶ 操作系统级虚拟化
- ▶ 库函数级虚拟化
- ▶ 编程语言级虚拟化

- ▶ **硬件抽象级虚拟化是指通过虚拟机监视器将宿主机底层硬件资源映射为客户机系统所能识别的资源，为运行在宿主机之上的客户机提供统一抽象的资源访问接口。**
- ▶ **按照实现方式分类：根据其不同的实现方式可将其分为全虚拟化技术和半虚拟化技术。**
 - **全虚拟化：**全虚拟化是指不需要对目标客户机操作系统进行任何修改就可
通过虚拟机监视器在宿主机上虚拟化。
 - **半虚拟化：**半虚拟化需要客户机操作系统协助的虚拟化，运行在虚拟机监
视器上的客户机系统是对内核进行针对性修改的特定操作系统。

► 全虚拟化:

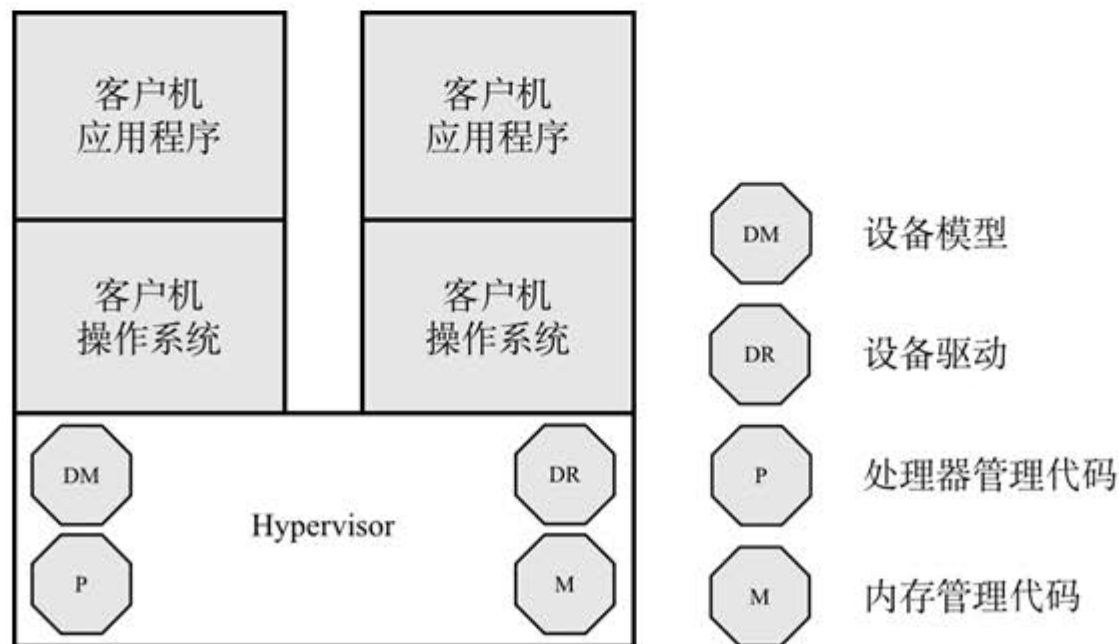
- 软件实现的全虚拟化: 软件实现的全虚拟化主要会应用特权解除和二进制翻译两种机制
- 硬件辅助的全虚拟化: 硬件辅助的全虚拟化需要具备虚拟化功能的处理器的支持, 能够在处理器层面对客户机系统的指令进行虚拟化或模拟。

► 按照VMM实现结构分类

- Hypervisor模型
- 宿主 (Hosted) 模型
- 混合模型

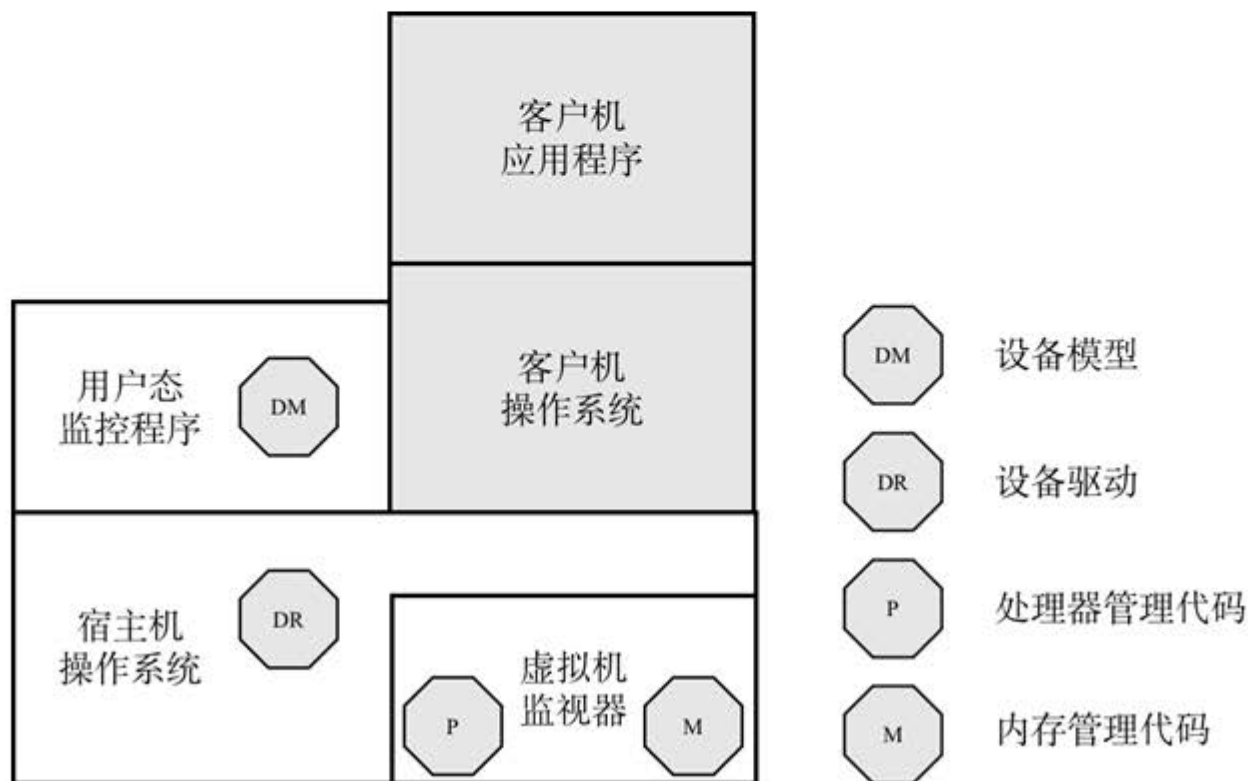
► Hypervisor模型

- 虚拟机监视器能对物理机的底层硬件资源进行分配管理。
- 虚拟机监视器还具备虚拟化的功能，能够创建虚拟环境、为虚拟环境分配资源、管理已创建好的虚拟环境、完成虚拟环境中客户机系统的任务。



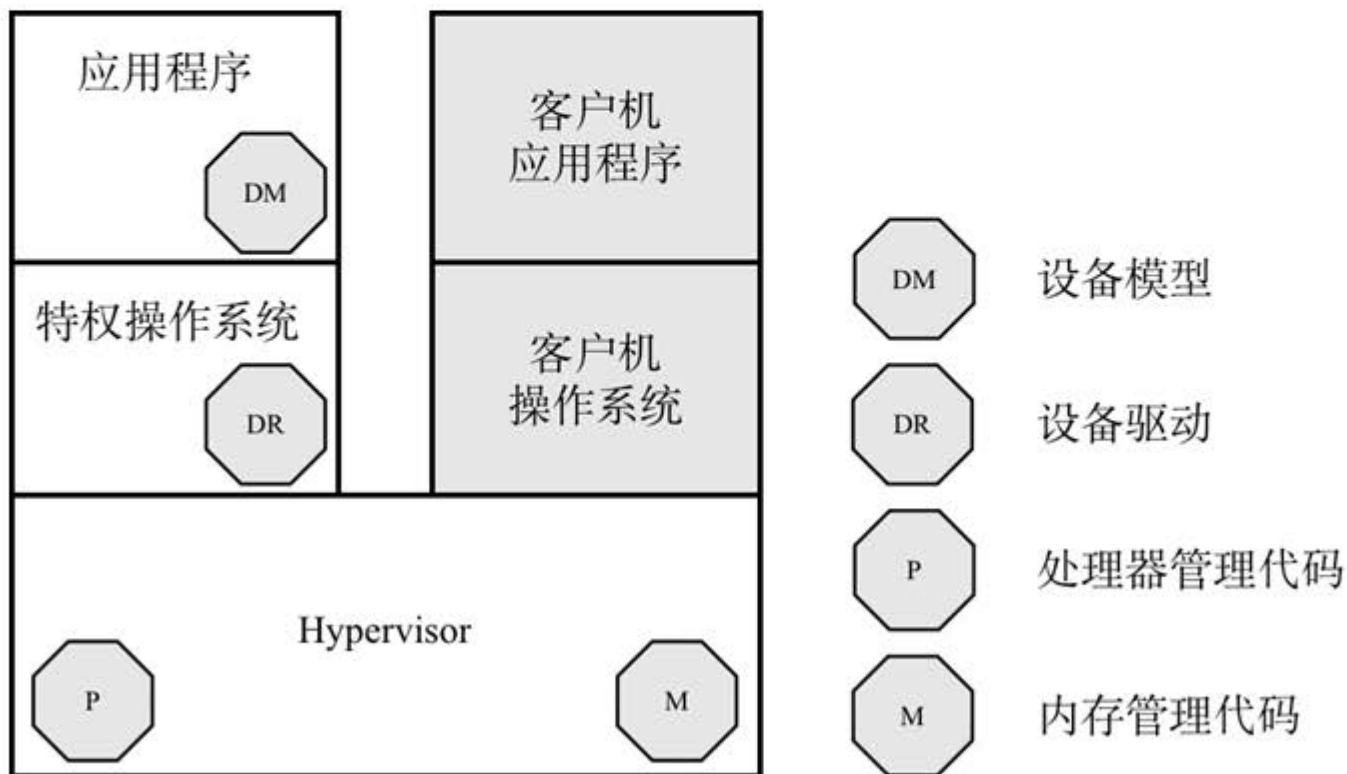
► 宿主 (Hosted) 模型

- 虚拟机监视器会运行在物理机的宿主机系统之上，相互配合实现客户机系统的虚拟化。



► 混合模型

- 混合模型即Hypervisor模型和宿主模型的结合。

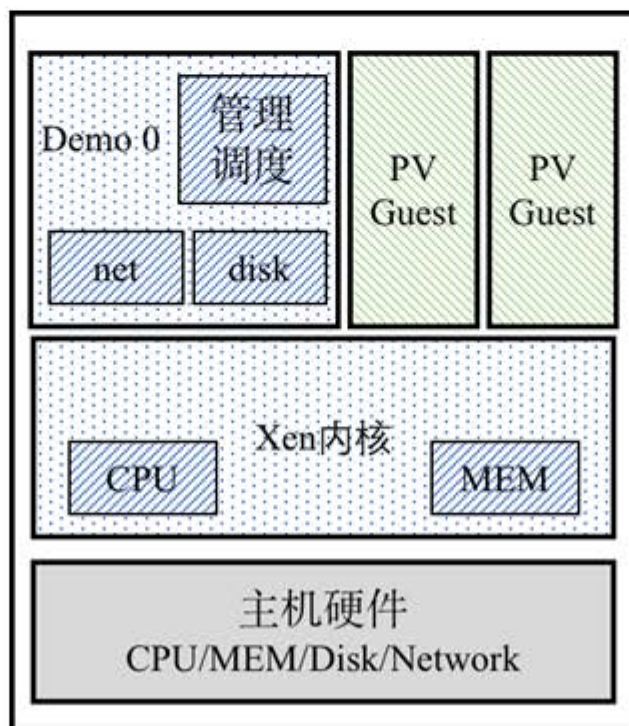


▶ 经典硬件抽象级虚拟化技术

- XEN
- KVM
- VMware Workstation

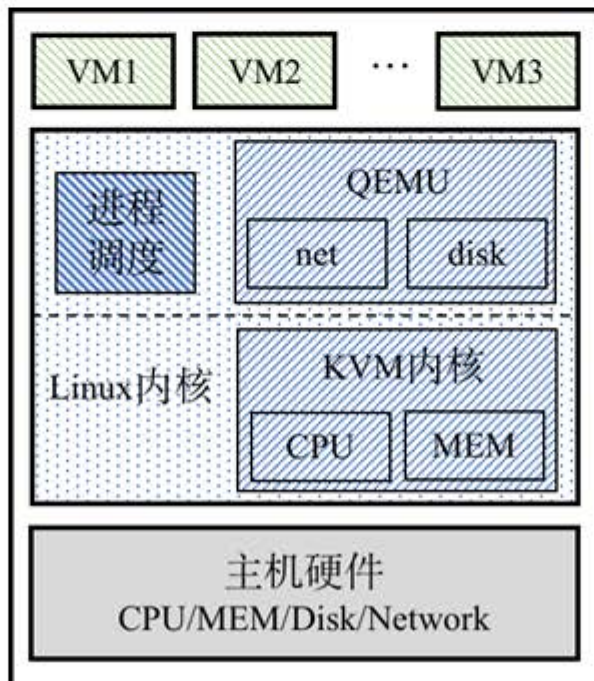
► XEN

- XEN是一个开放源代码虚拟机监视器。
- 主要包含三大部分：Hypervisor、Domain0、DomainU。



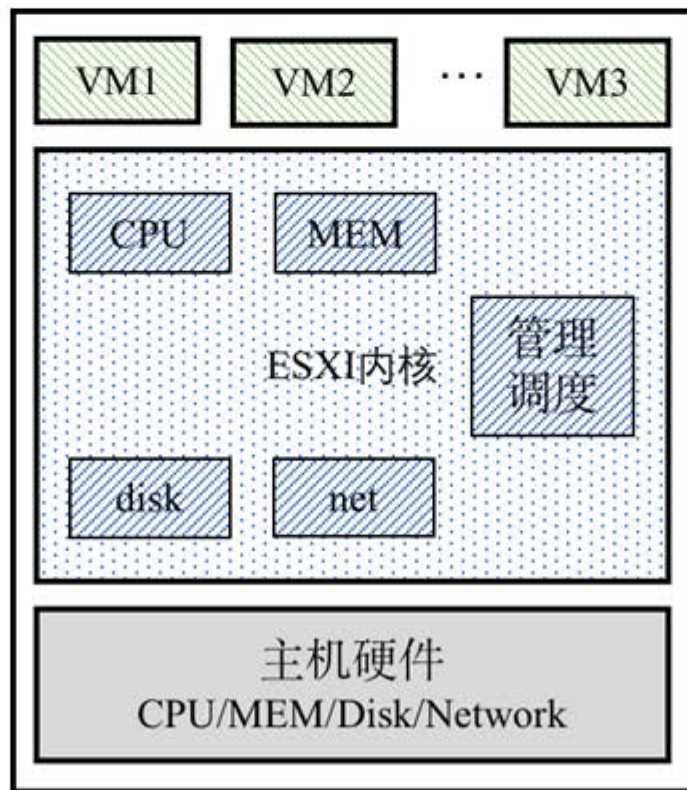
► KVM

- 基于内核的虚拟机，基于Linux环境的开源虚拟化模块。
- 提供了虚拟化功能的内核插件，负责提供CPU和内存资源的虚拟化，以及客户机操作系统I/O请求的拦截。



► VMware Workstation

- 一款桌面虚拟计算机软件，该软件允许用户同时创建运行多个x86虚拟机。



- ▶ 指令架构级虚拟化
- ▶ 硬件抽象级虚拟化
- ▶ 操作系统级虚拟化
- ▶ 库函数级虚拟化
- ▶ 编程语言级虚拟化

- ▶ 操作系统级虚拟化工作在宿主机操作系统上，宿主机的操作系统内核允许多个相互隔离的用户空间实例（容器）共同存在。
- ▶ chroot、Linux Vserver、Open VZ、LXC、Docker都属于操作系统级虚拟化技术。

► chroot

- 它可以将一个进程的根目录改变到另一个指定位置，在该目录下的进程不能够访问此目录之外的文件系统。
- 能够限制进程或用户可访问资源的范围，大大增加系统的安全性。
- 可以在无法启动或登录的系统上进行系统维护以及问题排查。

► Linux Vserver

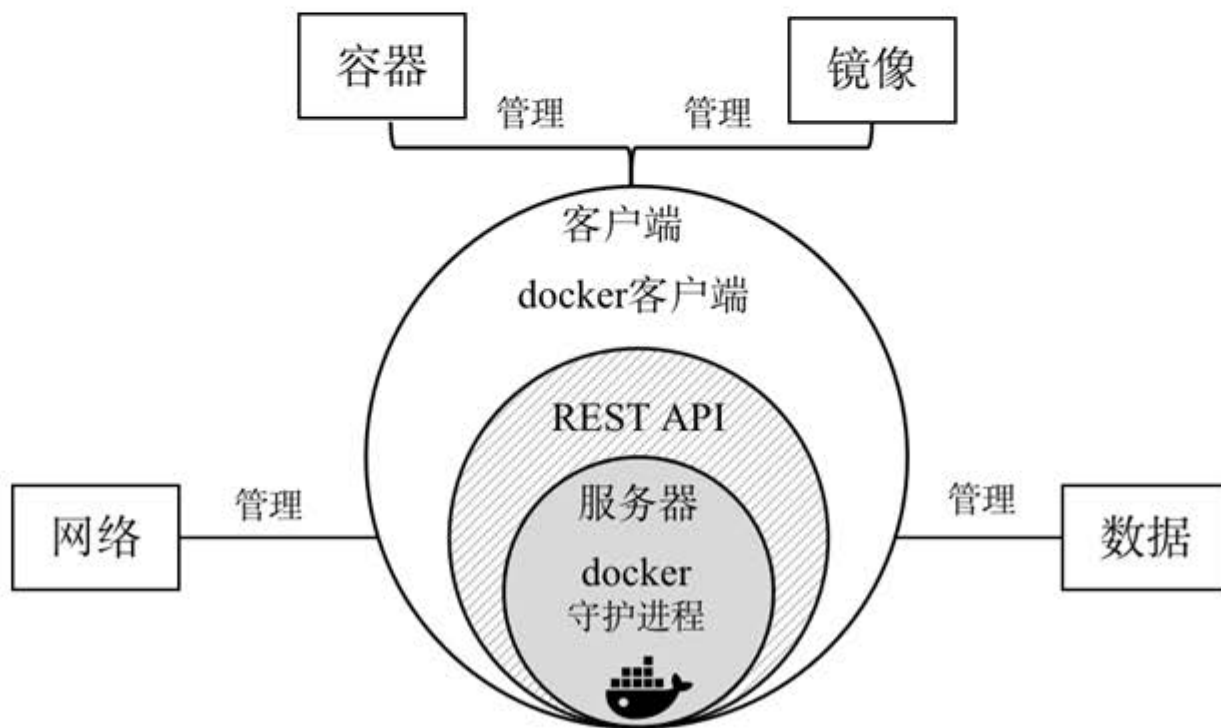
- 利用安全上下文、分段路由、chroot等工具，直接在操作系统内核之上进行虚拟化操作创建出多个相互隔离的系统或执行环境（虚拟专用服务器）。

▶ LXC

- ▶ 命名空间 (namespace) 是Linux内核用来隔离内核资源的一种机制。
- ▶ 控制组 (cgroups) 功能, 该功能允许对CPU、内存、I/O、网络等资源设置优先级并进行相应的限制。
- ▶ LXC利用Linux内核提供的namespace和cgroups功能实现在宿主机操作系统上隔离出互不干扰的容器。

► Docker

- Docker 底层的核心技术包含（namespaces）、控制组（cgroups）、文件系统（Union file systems）、容器格式（Container format）。



不同虚拟化技术的比较：

	写复制	磁盘配额	CPU配额	文件系统隔离	网络隔离	Root限权隔离	I/O限权	内存限制	热迁移
Chroot	x	x	x	不完全	x	x	x	x	x
Linux Vserver	√	√	√	√	不完全	不完全	√	√	x
openVZ	√	√	√	√	√	√	√	√	√
LXC	√	不完全	√	√	√	√	不完全	√	√
Docker	√	间接	√	√	√	√	√	√	x

- ▶ 指令架构级虚拟化
- ▶ 硬件抽象级虚拟化
- ▶ 操作系统级虚拟化
- ▶ 库函数级虚拟化
- ▶ 编程语言级虚拟化

- ▶ 通过虚拟化操作系统的库函数接口，使得用户的应用程序不需要根据操作系统类型作正对性修改即可直接运行在其他类别的操作系统中，极大提高了应用程序的兼容性以及可移植性。
- ▶ 典型的库函数级虚拟化技术包括Wine和WSL。

► Wine

- Wine是一个能在多种POSIX-compliant操作系统上运行Windows应用的兼容层，是一个在Linux和UNIX之上的Windows 3.x和Windows API的实现。

► WSL

- WSL目的使纯正的Ubuntu 14.04 “Trusty Tahr”映像能下载和解压到用户本地计算机并且映像内的工具和实用工具能在此子系统上原生运行。

- ▶ 指令架构级虚拟化
- ▶ 硬件抽象级虚拟化
- ▶ 操作系统级虚拟化
- ▶ 库函数级虚拟化
- ▶ 编程语言级虚拟化

- ▶ 先将高级语言编译转换成一种中间格式，后将这种中间格式的语言转译为目标平台所能够识别的机器指令序列。
- ▶ 较为经典的是Java虚拟机（JVM）和CLI。

► JVM

- JVM是一种用于计算设备的规范，在实际的计算机上仿真模拟各种计算机功能。
- JVM有自己完整的硬件架构，JVM本质上是一个程序。
- Java虚拟机主要有五大模块：类装载器子系统、运行时数据区、执行引擎、本地方法接口和垃圾收集模块。

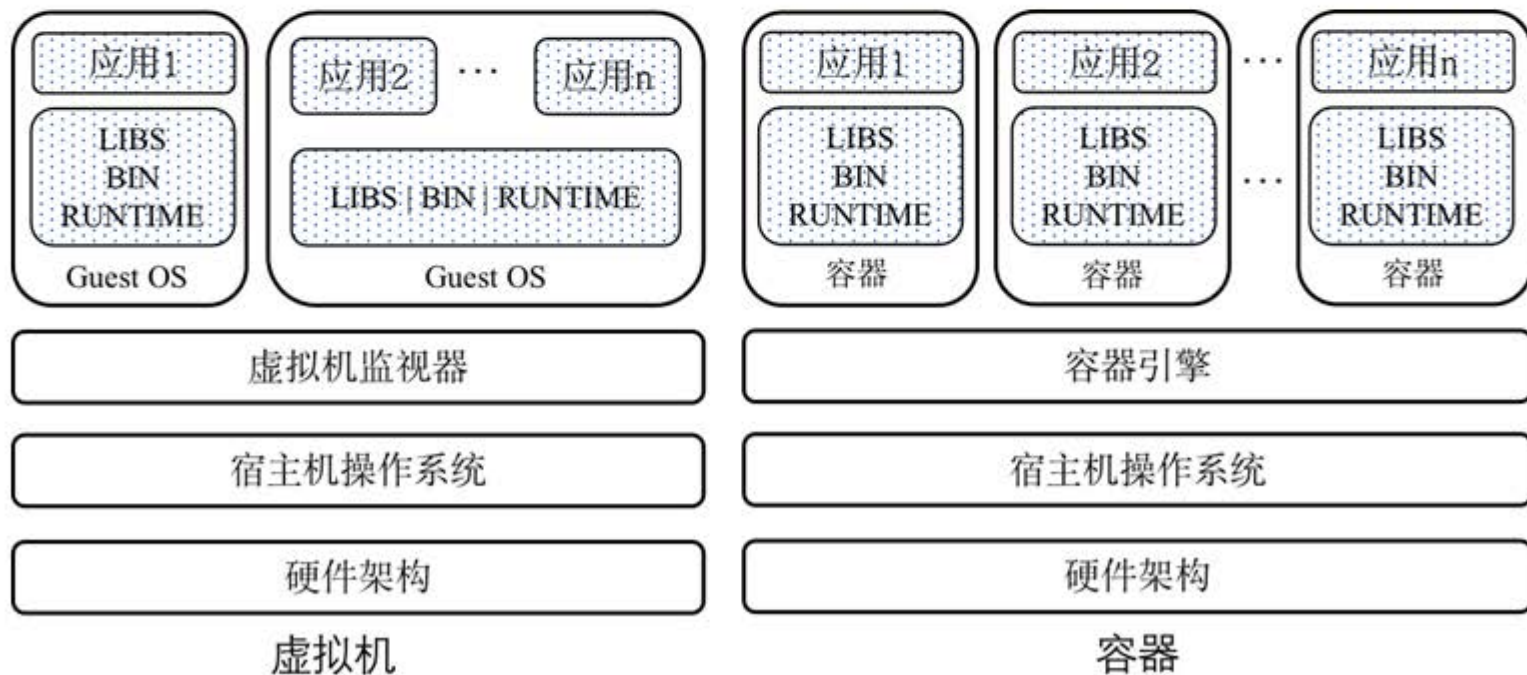
► CLI

- CLI即通用语言基础架构，规定了如何在运行库中声明、使用、管理类型，同时也是运行库支持跨语言互操作的一个重要组成部分。
- 目的是建立一个支持跨语言集成，类型安全和高性能代码执行的框架。

- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 虚拟机与容器
- ▶ Docker
- ▶ Unikernel
- ▶ FireCracker
- ▶ 对比总结

- ▶ 通过硬件抽象级虚拟化技术所虚拟出来的客户机系统一般称为虚拟机，而通过操作系统级虚拟化技术所虚拟出来的客户机系统一般称为容器。
- ▶ 传统虚拟机（宿主模式）与容器的架构。



► 容器相比虚拟机的优势：

- 启动时间：容器启动时间小于虚拟机
- 磁盘和内存资源占用：容器小于虚拟机
- 性能：虚拟机的性能弱于容器
- 系统支持量：容器的支持量较大
- 更加轻量
- 部署（启动）更快
- 更易移植
- 管理更加弹性

▶ 虚拟机相比容器的优势：

- 工作负载的迁移。
- 虚拟机支持跨操作系统的虚拟化。
- 安全性：虚拟机和容器都提供了不同程度的安全机制，而容器属于进程间的隔离。
- 更强大的虚拟能力：相比容器，虚拟机拥有强大的跨平台虚拟能力，甚至能在计算机上运行安卓虚拟机。

- ▶ 虚拟机与容器
- ▶ Docker
- ▶ Unikernel
- ▶ FireCracker
- ▶ 对比总结

► Docker特点:

- 轻量: 能够更快地上传和下载镜像。
- 标准: Docker 容器基于开放式标准。
- 安全: 单个 Docker 运行出现问题, 既不会影响到系统上运行的其他 Docker, 也不会危及系统本身。

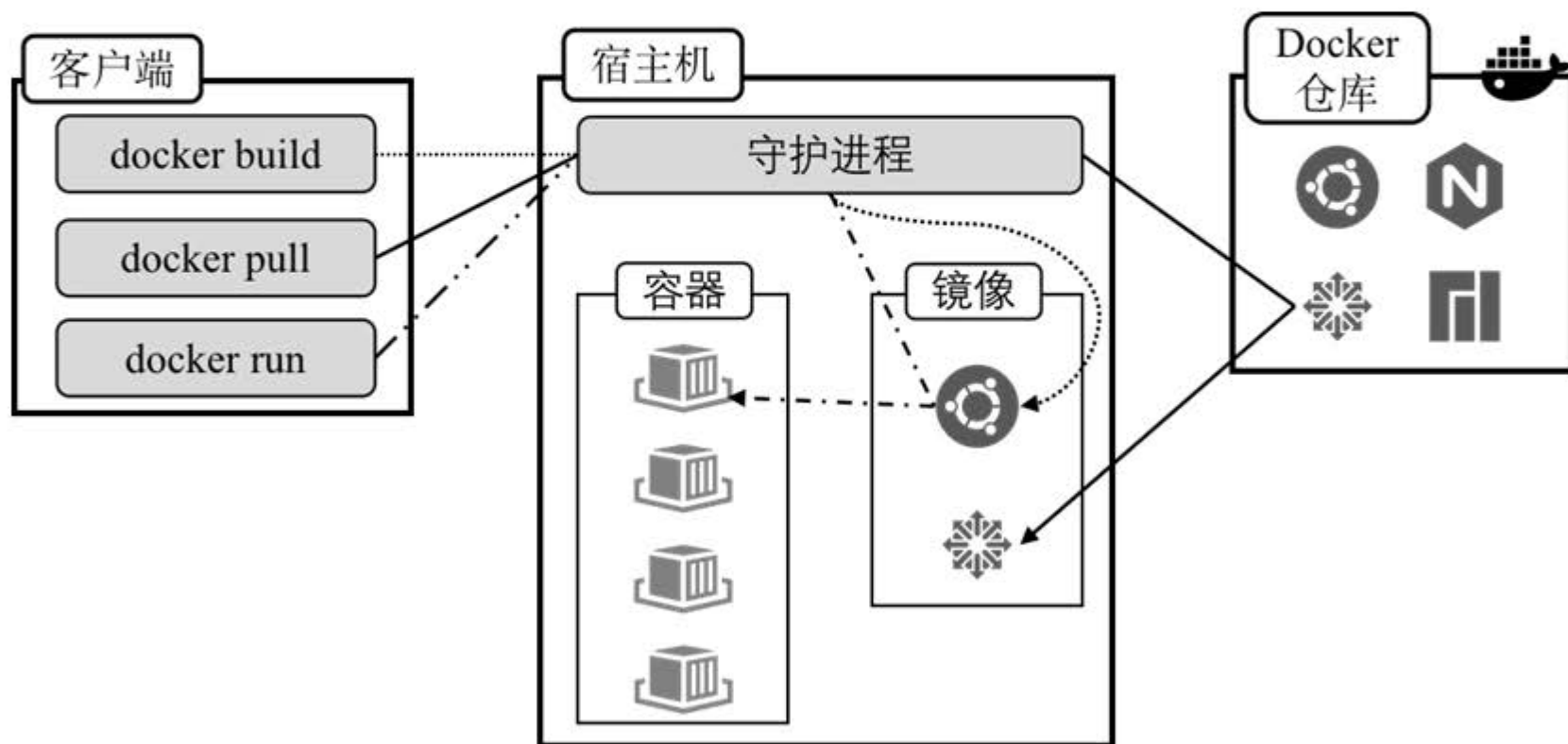
► Docker组成:

- 镜像: Docker 镜像封装了容器运行时所需程序、库、资源、配置参数等数据。
- 容器: 是 Docker 镜像一经启动成为的运行的实体, 其实质是宿主主机的进程。
- 仓库: 仓库主要用于存储、管理和分发 Docker 镜像。

► Docker安全机制:

- 内核命名空间
- 控制组
- 服务端安全
- 内核能力机制
- 其他安全机制

Docker整体构架及组成:



- ▶ 虚拟机与容器
- ▶ Docker
- ▶ Unikernel
- ▶ FireCracker
- ▶ 对比总结

► 操作系统内核种类：

- 宏内核：将所有的系统功能服务全部集中到内核中来
- 微内核：仅将必须在内核态运行的功能服务保留下来，把原本在宏内核中的其他功能服务下放到用户态进行实现。

► LibOS：将传统操作系统所提供的各个内核服务分离开来并分别进行实现。

► Unikernel利用LibOS所构建的系统镜像。Unikernel相较于传统操作系统拥有安全性高、资源开销小、可优化程度高、启动速度快、运行效率高这五个优势。

- ▶ 虚拟机与容器
- ▶ Docker
- ▶ Unikernel
- ▶ FireCracker
- ▶ 对比总结

- ▶ Firecracker是轻量级虚拟机监视器，其专为高效安全的运行无服务器函数和容器而设计的开源虚拟化技术。
- ▶ Firecracker优势：
 - 安全性高
 - 启动速度快
 - 资源开销小

- ▶ 虚拟机与容器
- ▶ Docker
- ▶ Unikernel
- ▶ FireCraker
- ▶ 对比总结

性能对比:

	启动时间	镜像大小	内存开销	编程语言依赖
传统虚拟机	~5/10秒	~1000MBs	~100MBs	弱
容器	~800/1000 微秒	~50MBs	~5MBs	弱
Unikernel	~<50毫秒	~5MBs	~8MBs	弱
Firecracker	~100毫秒	-	~3MBs	-

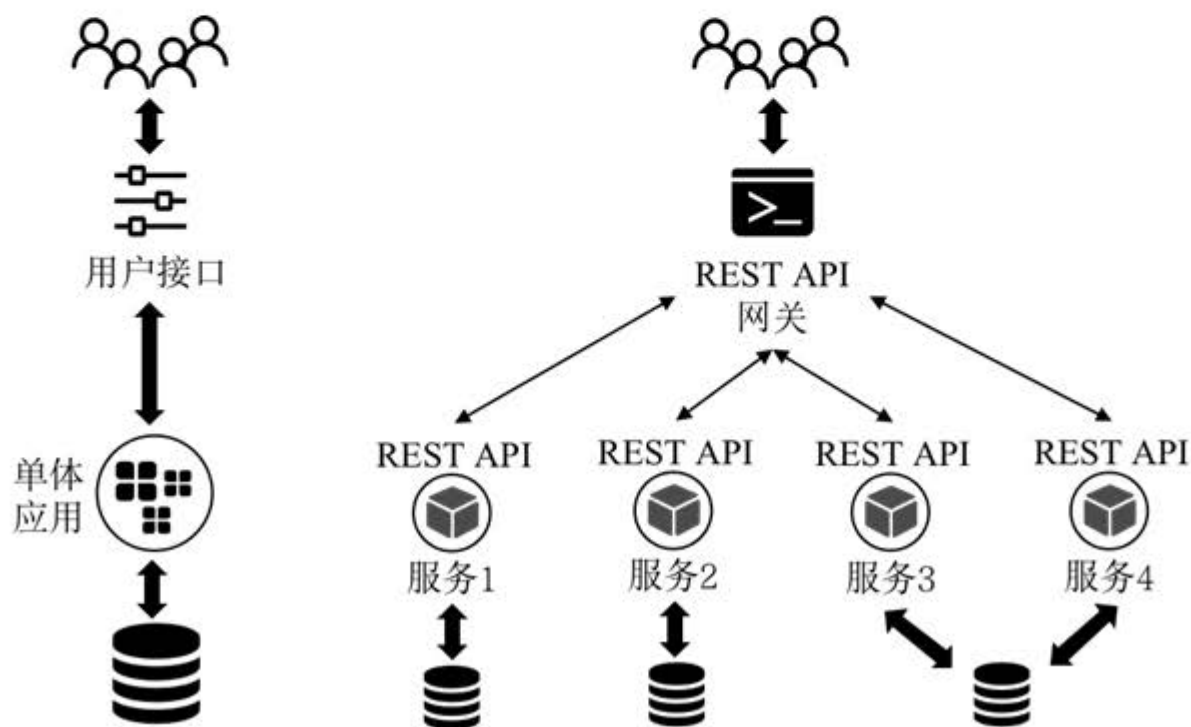
- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 服务架构的发展历程
- ▶ 微服务架构特点
- ▶ 微服务架构面临的挑战

- ▶ 单体架构将所有的功能打包放在一个Web应用中运行。
- ▶ SOA根据业务特点将系统中的不同模块拆分到不同的系统中，将模块进一步拆分成多个。
- ▶ 基于SOA的微服务架构，将业务进行彻底的拆分，使其组件化、服务化。

- ▶ 服务架构的发展历程
- ▶ 微服务架构特点
- ▶ 微服务架构面临的挑战

微服务架构以专注于单一责任和功能的小型功能区块为基础，利用模块化的方式组合出复杂的大型应用程序，各功能区块使用与语言无关的API集相互通信。单体架构与微服务架构对比图：



▶ 微服务架构共同特点:

- 通过服务组件化
- 围绕业务功能组织团队
- 做产品而不是做项目
- 智能端点与傻瓜管道
- 去中心化治理
- 去中心化数据管理
- 基础设施自动化
- 容错设计
- 演进式设计

- ▶ 服务架构的发展历程
- ▶ 微服务架构特点
- ▶ 微服务架构面临的挑战

- ▶ 在微服务规模增加时，在服务治理层面还面临着包括负载均衡、熔断机制、灰度发布、故障恢复、分布式追踪等在内的诸多挑战
- ▶ 主流的微服务框架都对业务代码有具有一定的侵入性，且有限类型的开发语言，导致原有业务替换成本较高。

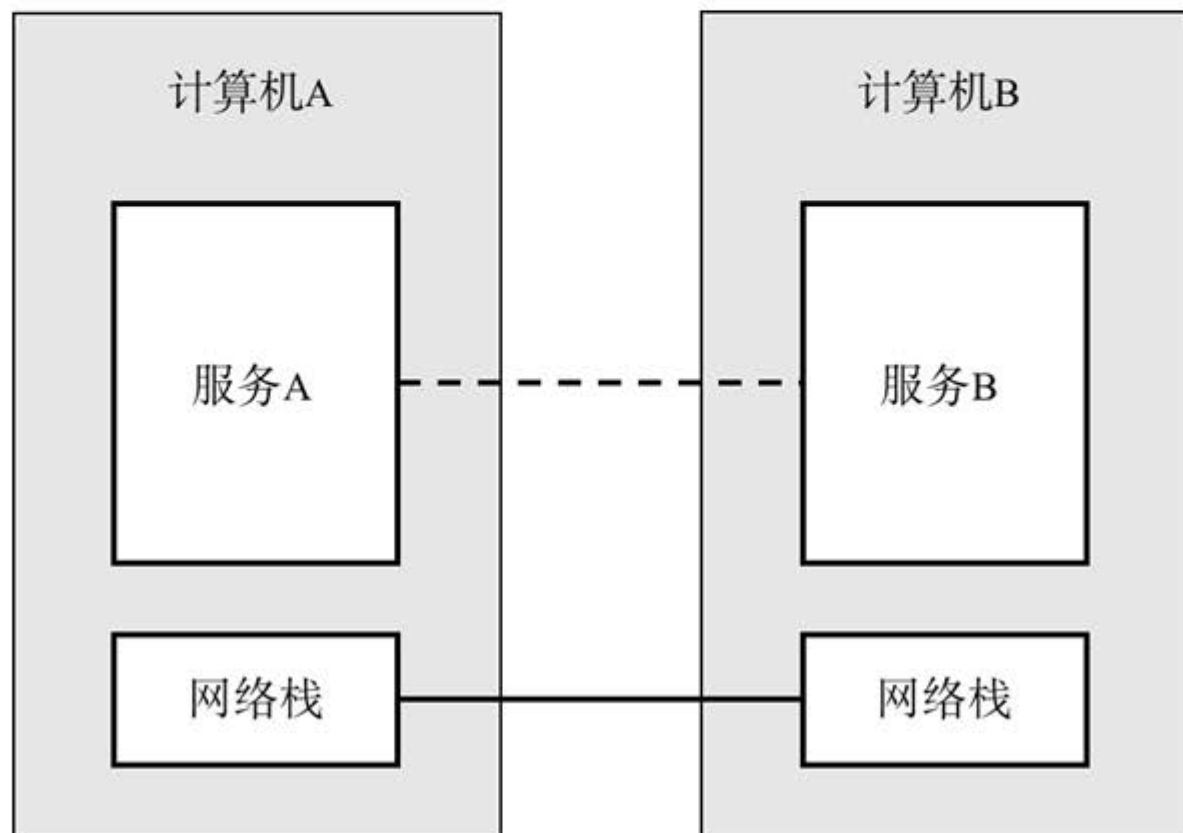
- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 服务网格的发展历程
- ▶ 主流服务网格：Istio
- ▶ 服务网格工具对比

服务网格的发展历程 (1)

61

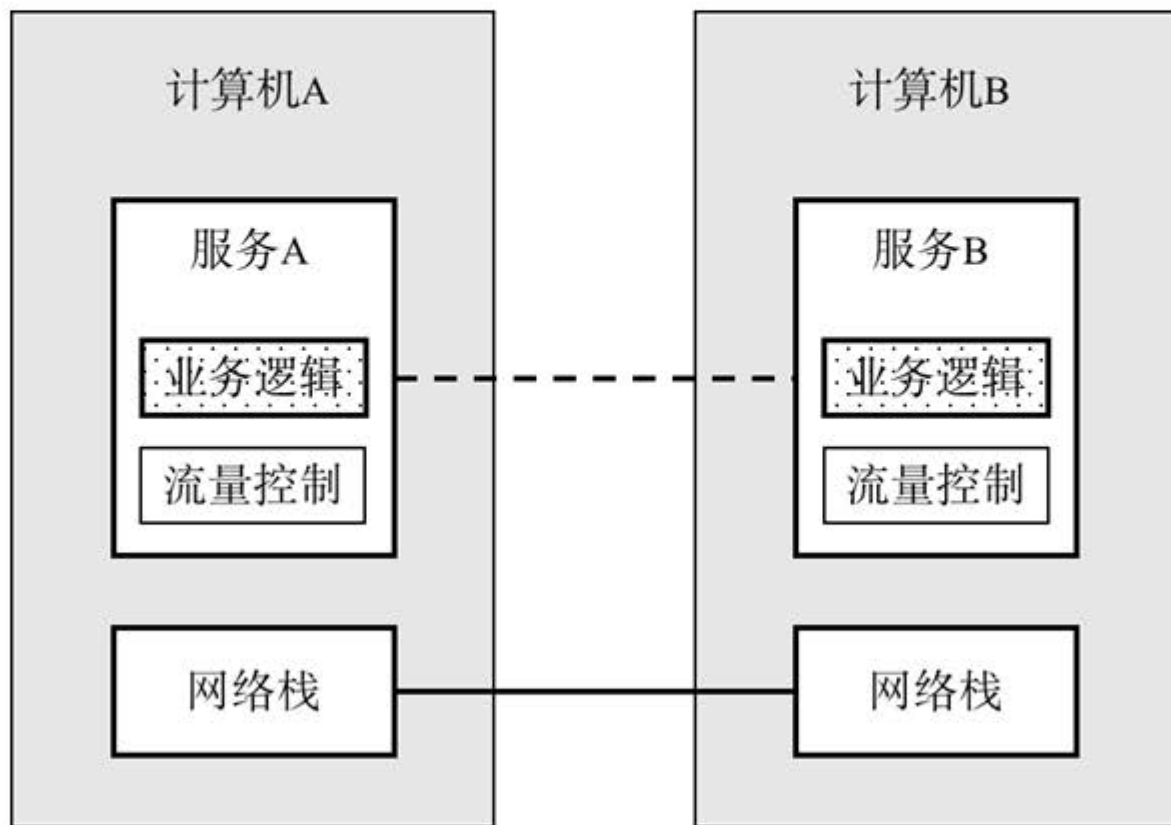
在早期计算机的网络交互中，希望部署在不同计算机上的两个服务能够直接通信：



服务网格的发展历程 (2)

62

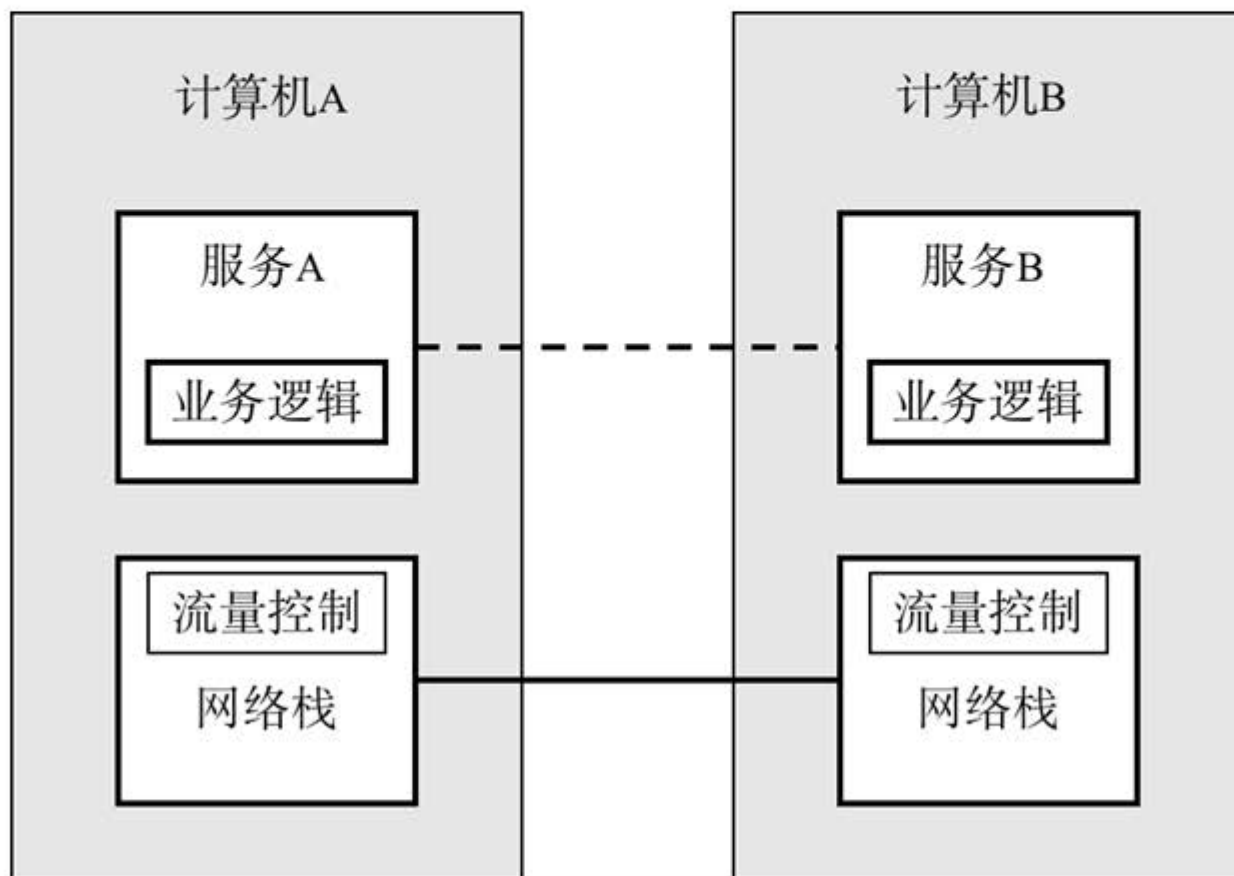
计算机间的通信不是直接建立连接，需要对数据包进行转发路由：



服务网格的发展历程 (3)

63

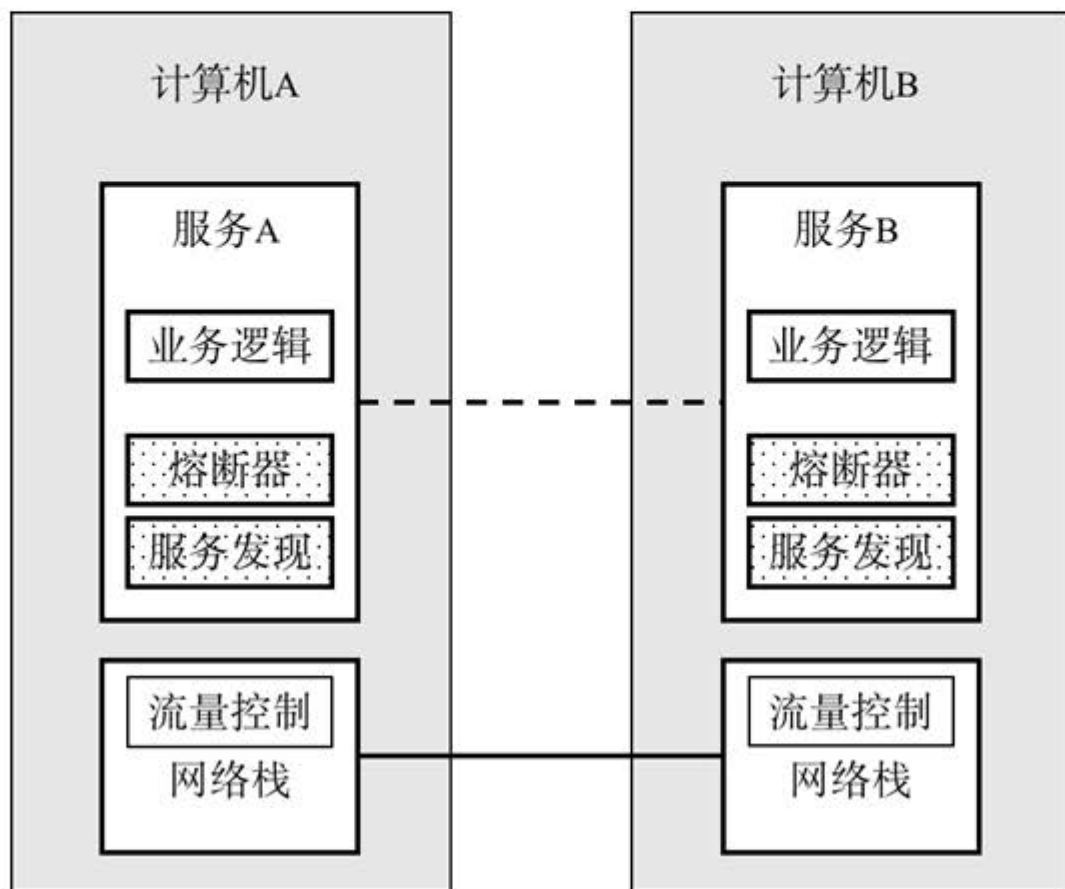
随着TCP/IP等出现，向外提供接口供应用程序使用：



服务网格的发展历程（4）

64

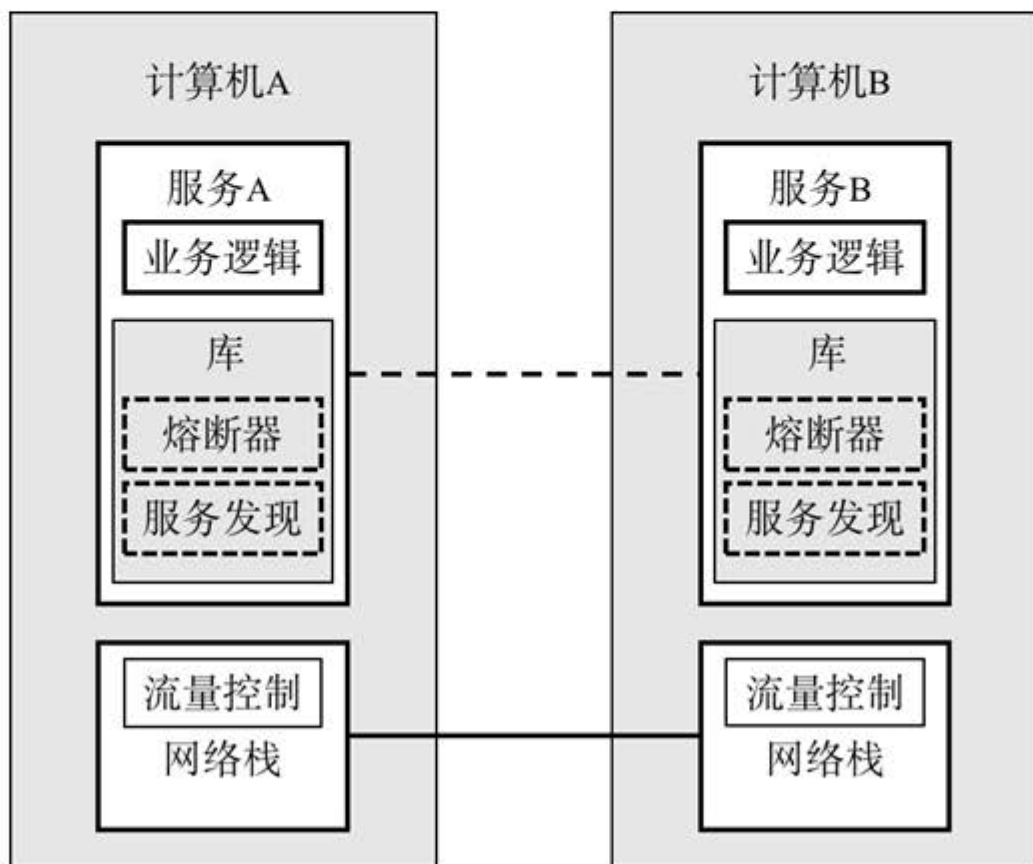
为了更好的应对用户日益增长的应用需求，出现了不同的架构以及设计思想。



服务网格的发展历程 (5)

65

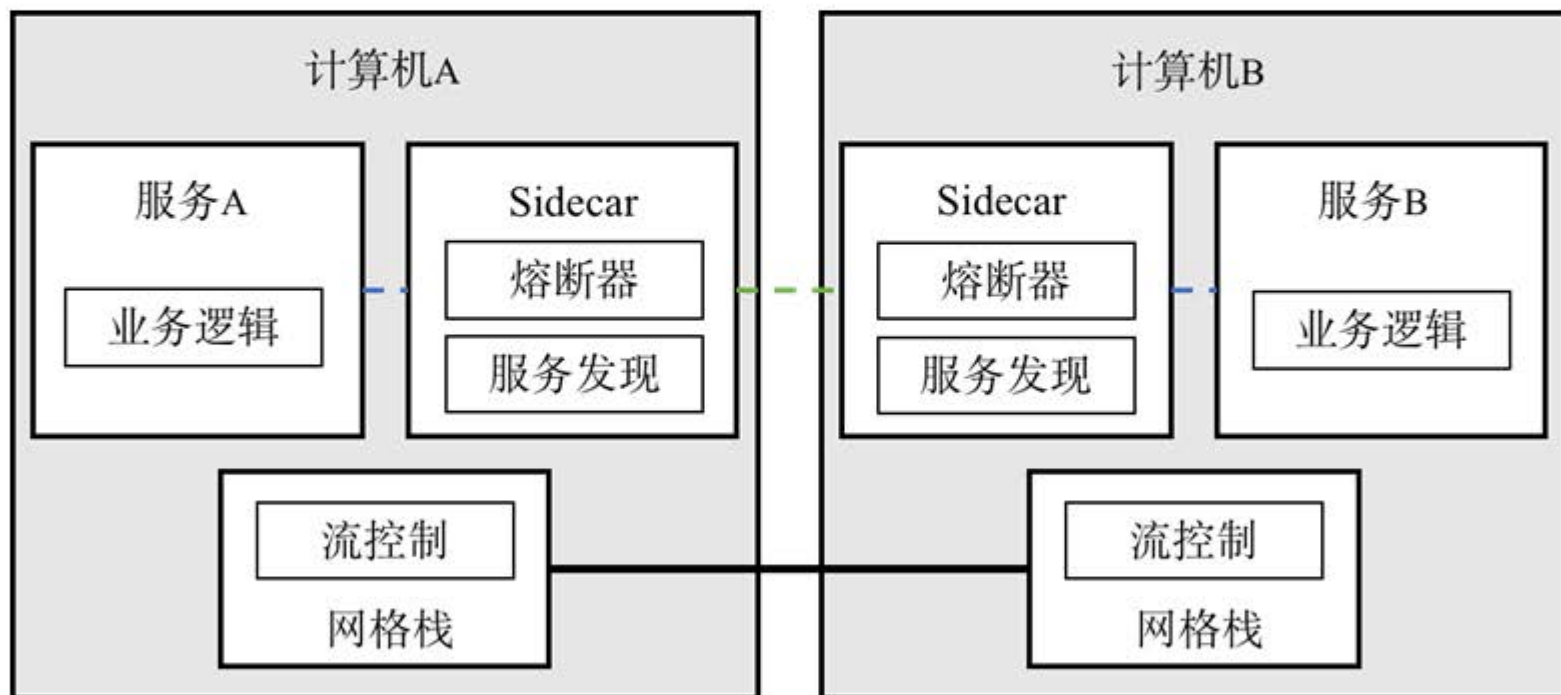
框架内部提供实现了服务发现、熔断器等功能的库，以避免在不同的服务内复写相似的逻辑。



服务网格的发展历程 (6)

66

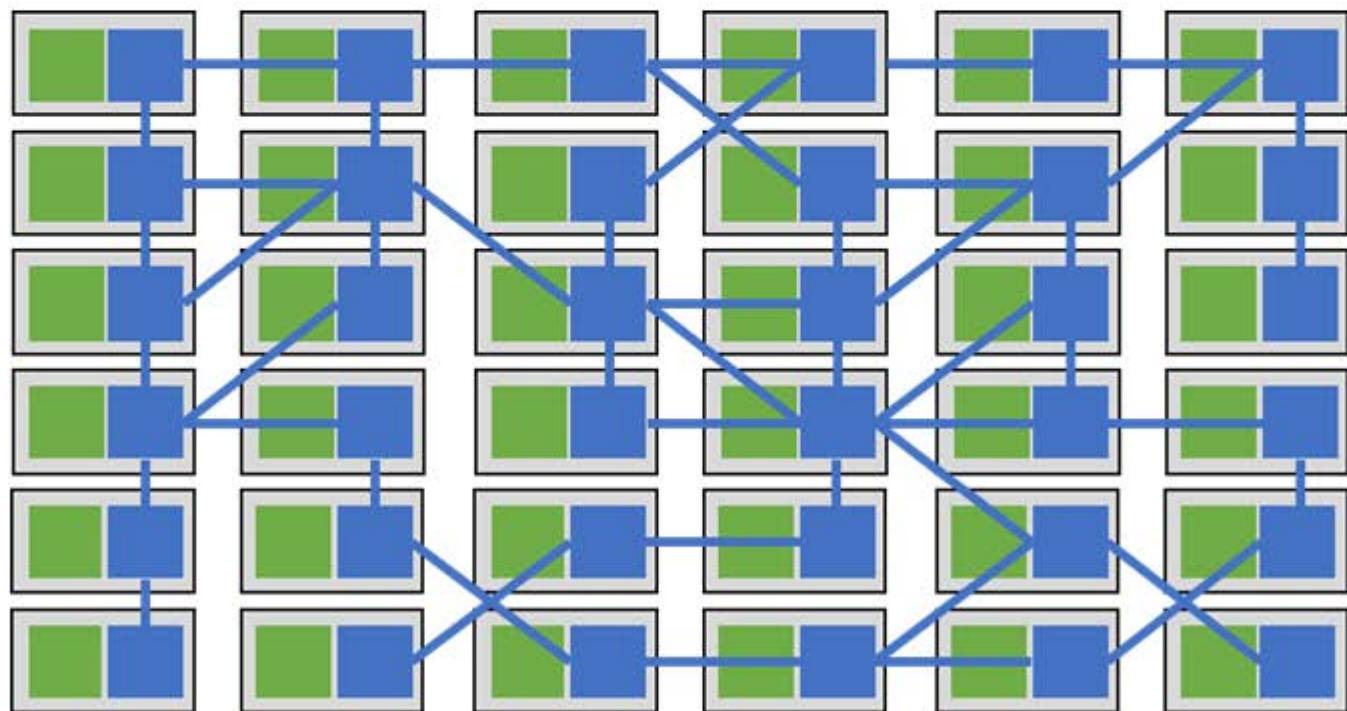
边车模式应运而生：



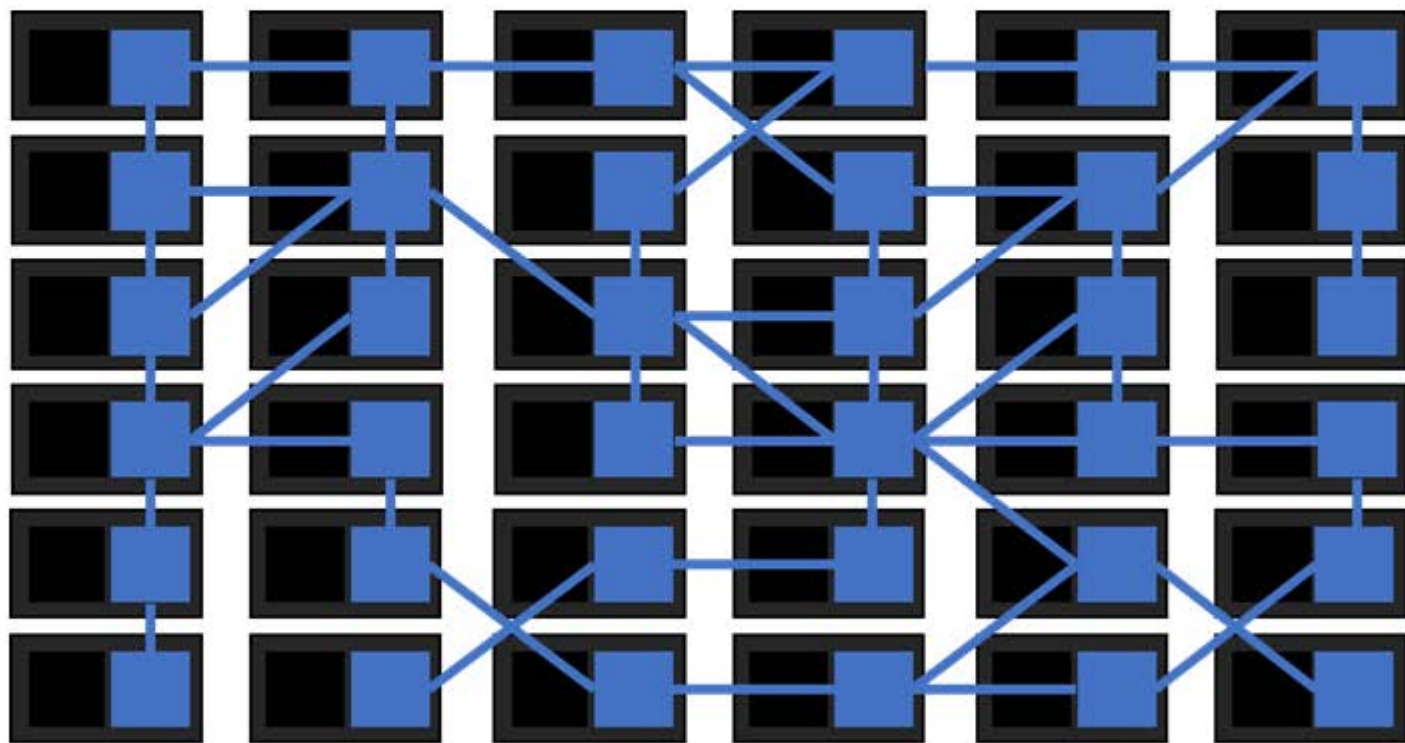
服务网格的发展历程 (7)

67

边车模式计算计算机与服务关系：灰色方框代表一台计算机，里面的绿色方块代表服务本身。



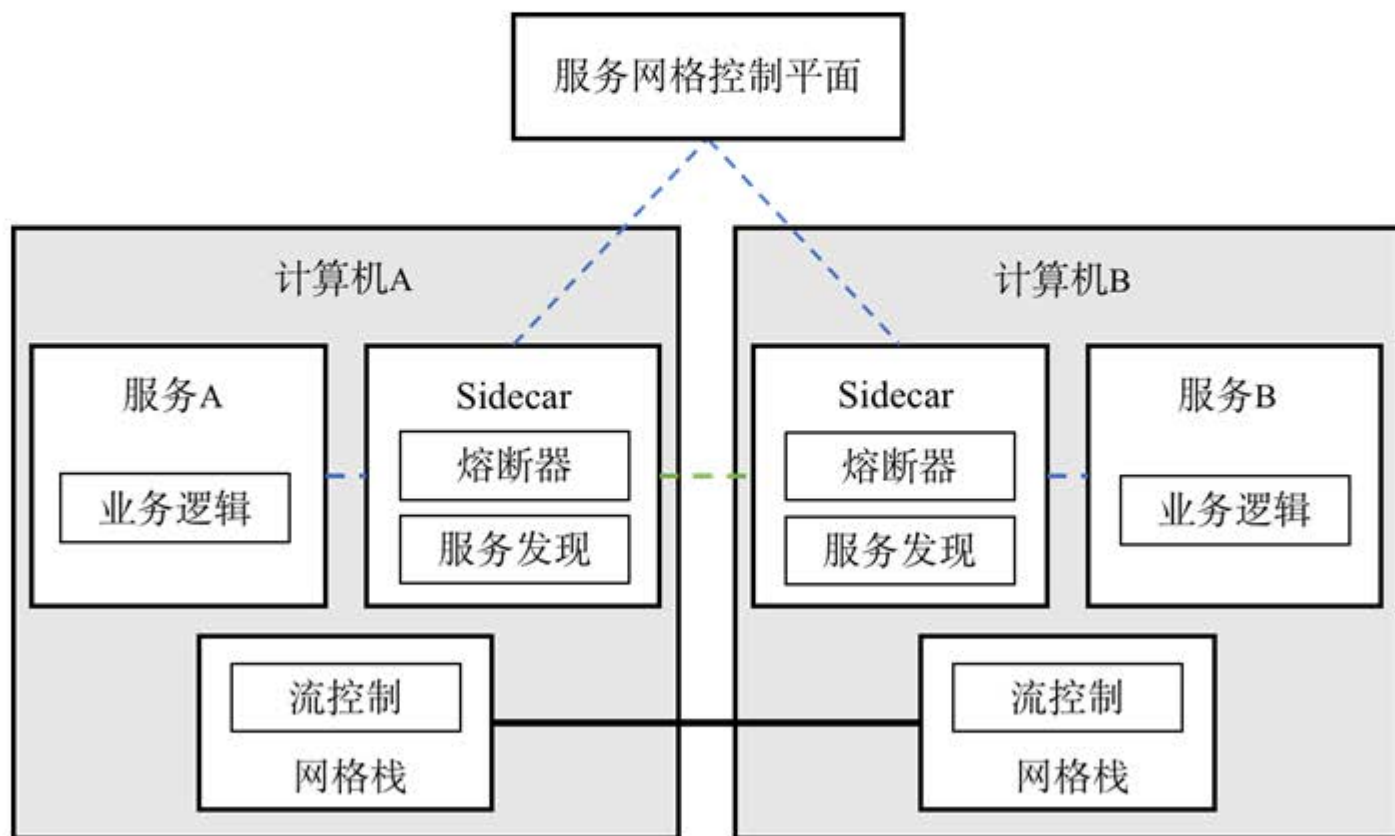
边车模式计算代表拓扑图:



服务网格的发展历程 (9)

69

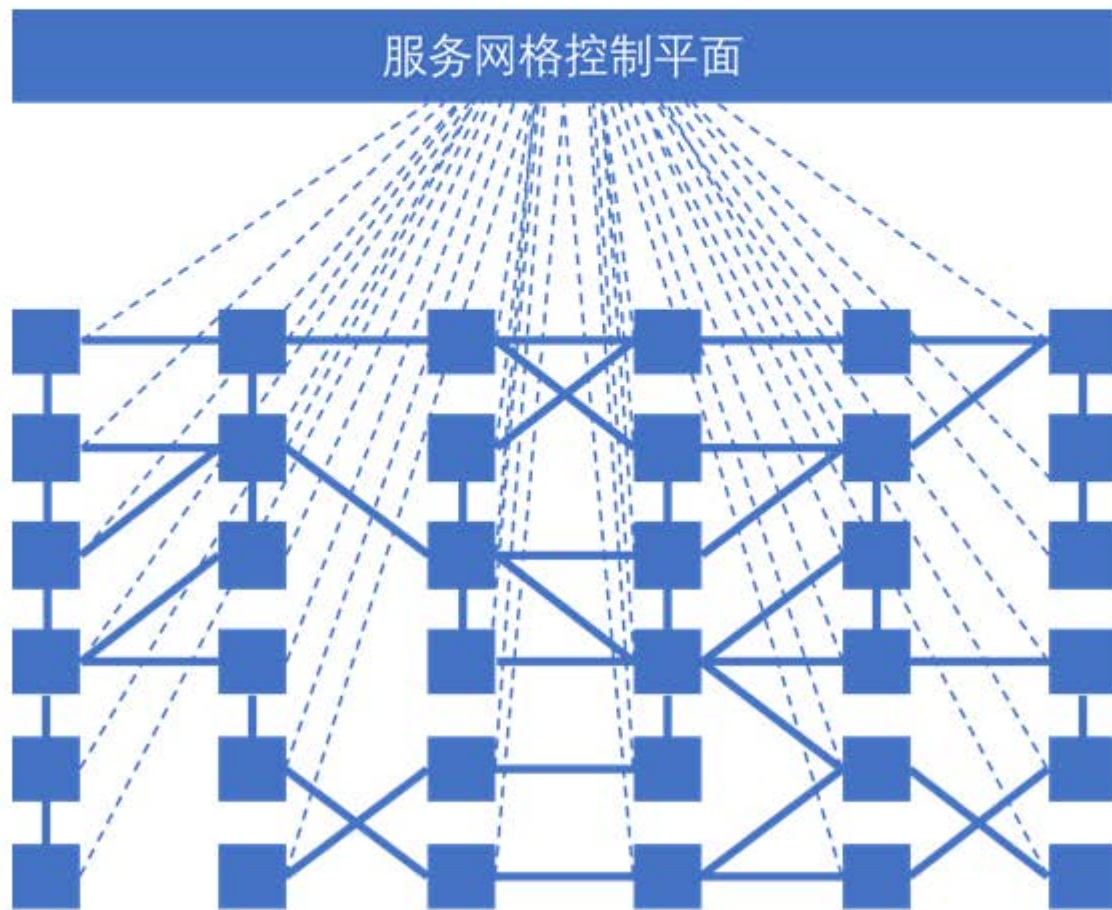
控制平面负责与代理构成的数据层面进行信息交互得到全局代理信息。



服务网格的发展历程 (10)

70

从全局角度观看控制平面负责与代理构成的数据层面：



- ▶ 服务网格的发展历程
- ▶ 主流服务网格：Istio
- ▶ 服务网格工具对比

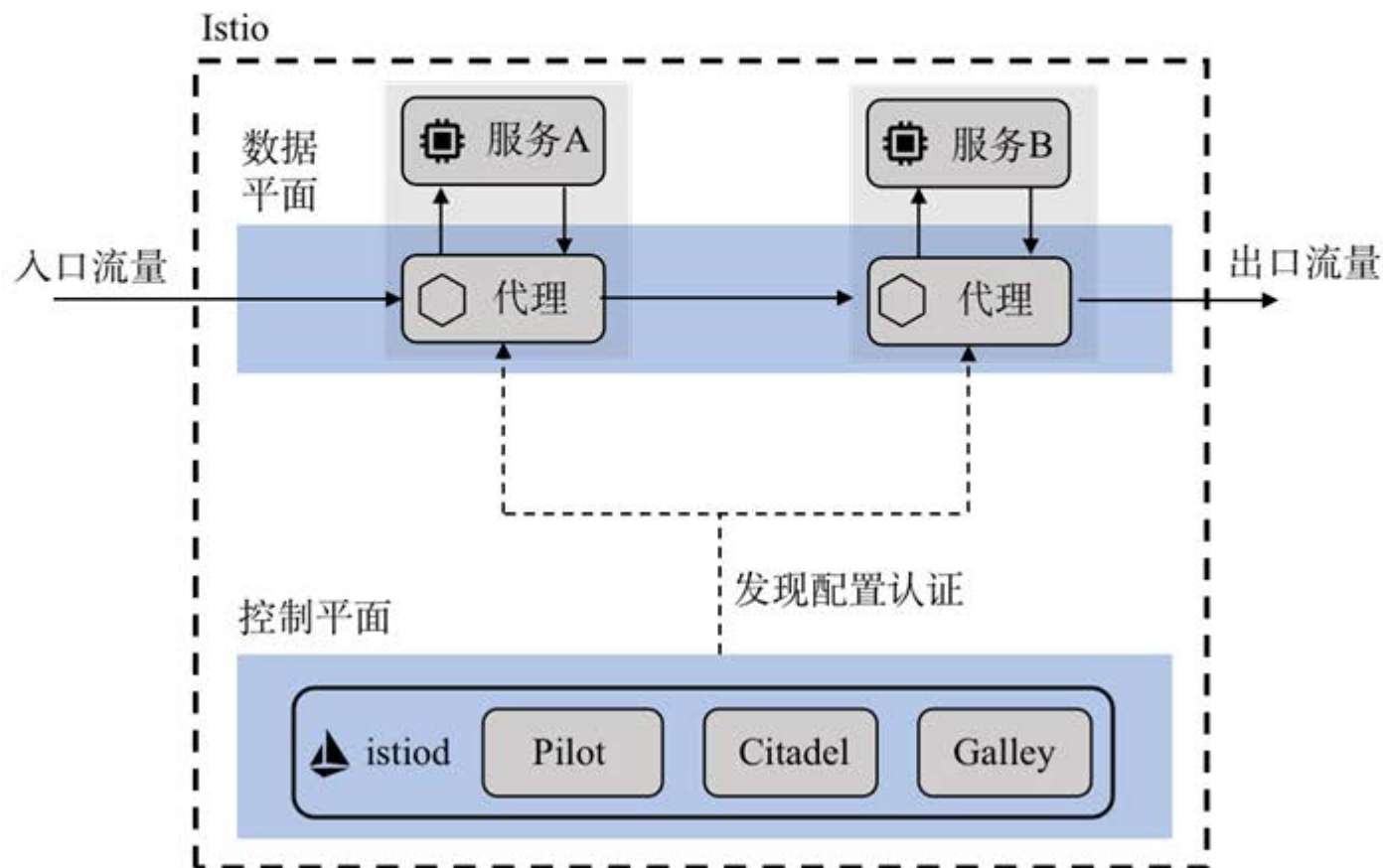
► 关键特征：

- 流量管理：请求路由和流量转移、弹性功能、调试能力。
- 安全：认证、授权。
- 可观察性：指标、访问日志、分布式追踪。
- 多平台支持。

主流服务网格：Istio (2)

73

ISTIO主要组件包括Pilot、Citadel、Galley。

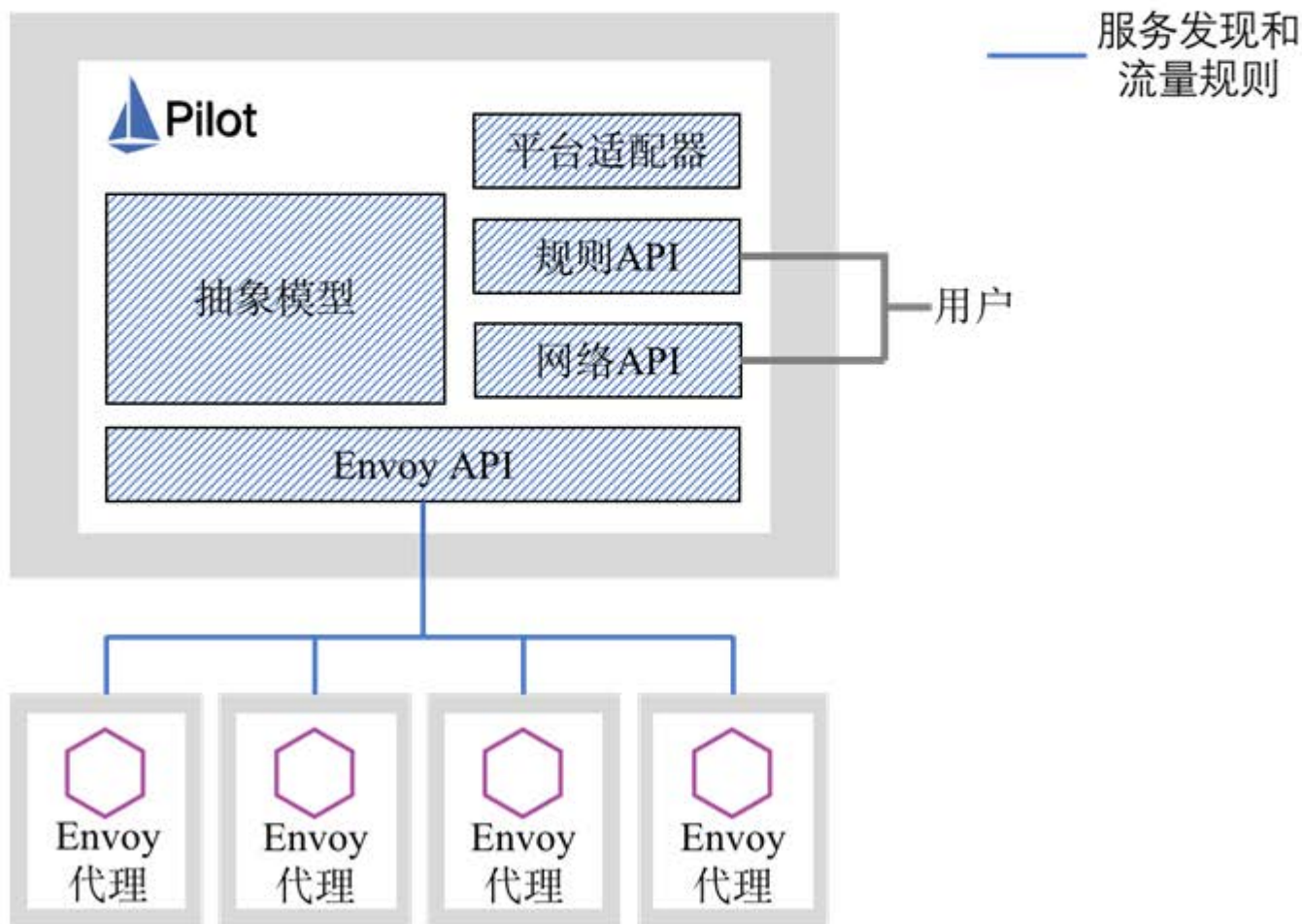


- ▶ **数据平面：** 主要包含以下功能：**服务发现、健康探测、流量路由、负载均衡、验证授权、链路追踪。**
- ▶ **网络代理实现有：** Envoy、MOSN、Linkerd Proxy等。
- ▶ **控制平面主要有Pilot、Citadel、Galley。**
 - Pilot组件负责管理微服务间的流量路由转发规则和故障恢复，并提供服务发现功能。
 - Citadel组件是Istio中的安全核心组件，负责提供身份认证以及证书管理功能。
 - Galley组件是Istio的配置验证、提取、处理和分发组件。

主流服务网格：Istio (4)

75

Pilot组件：



- ▶ 服务网格的发展历程
- ▶ 主流服务网格：Istio
- ▶ 服务网格工具对比

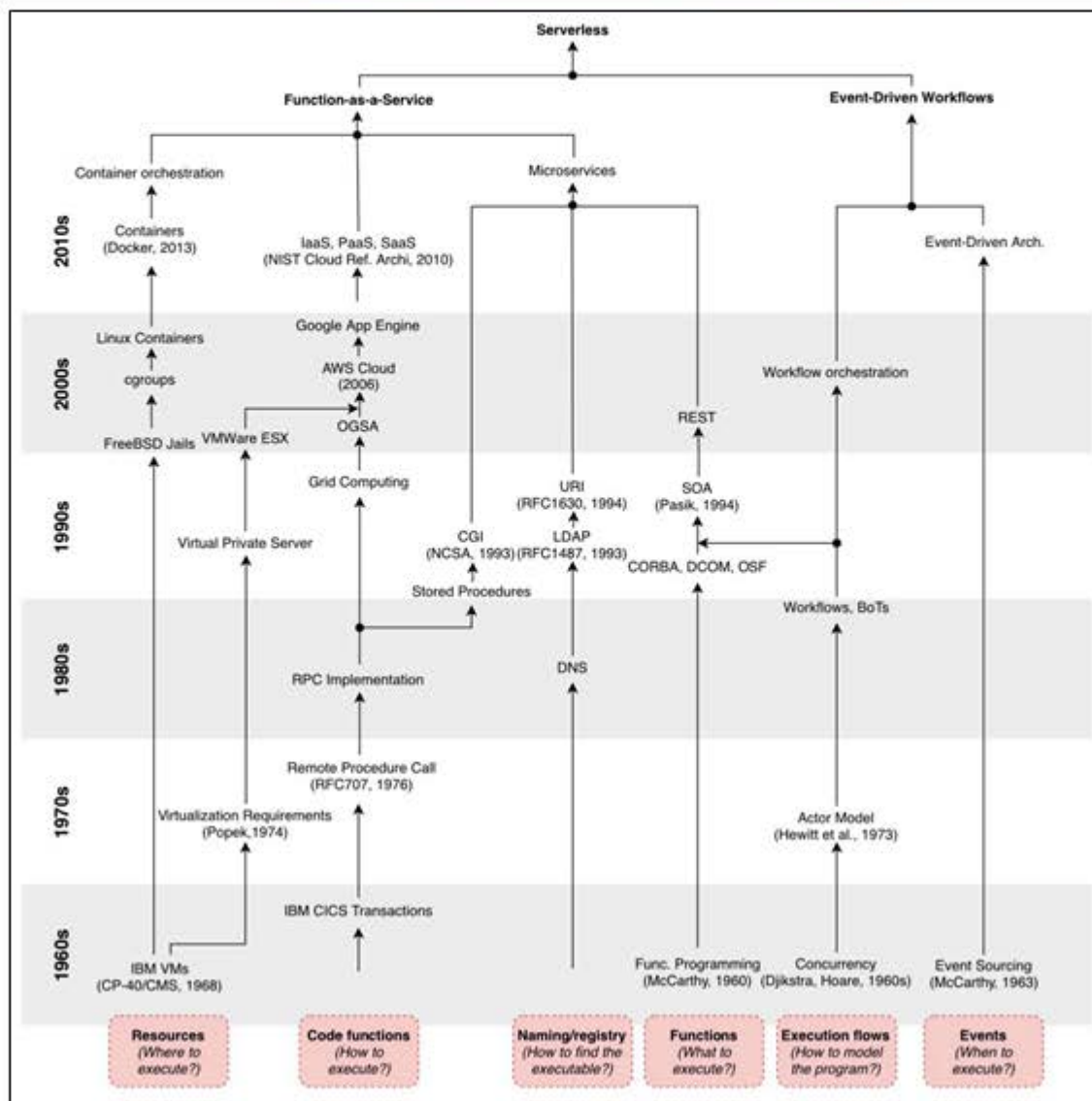
- ▶ 从基本概况、支持的协议、数据平面、支持平台和扩展性、接口兼容性、监视特性、路由特性、弹性、安全特性这几个方面对上述的服务网格工具进行对比。

- ▶ 概述
- ▶ 虚拟化技术发展历史
- ▶ 虚拟化技术分类
- ▶ 轻量级虚拟化
- ▶ 微服务
- ▶ 服务网格
- ▶ 无服务架构

- ▶ 发展历史
- ▶ 开源工具

发展历史 (1)

80



- ▶ 发展历史
- ▶ 开源工具


相关产品概览:

CNCF Serverless Landscape
2020-12-29T00:38:07Z 4c0c3af9


See the serverless interactive display at
s.cncf.io

Greyed logos are not open source


Tools




Security




Framework




Hosted Platform



Installable Platform





s.cncf.io


Serverless computing refers to a new model of cloud native computing, enabled by architectures that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment

CLOUD NATIVE Landscape

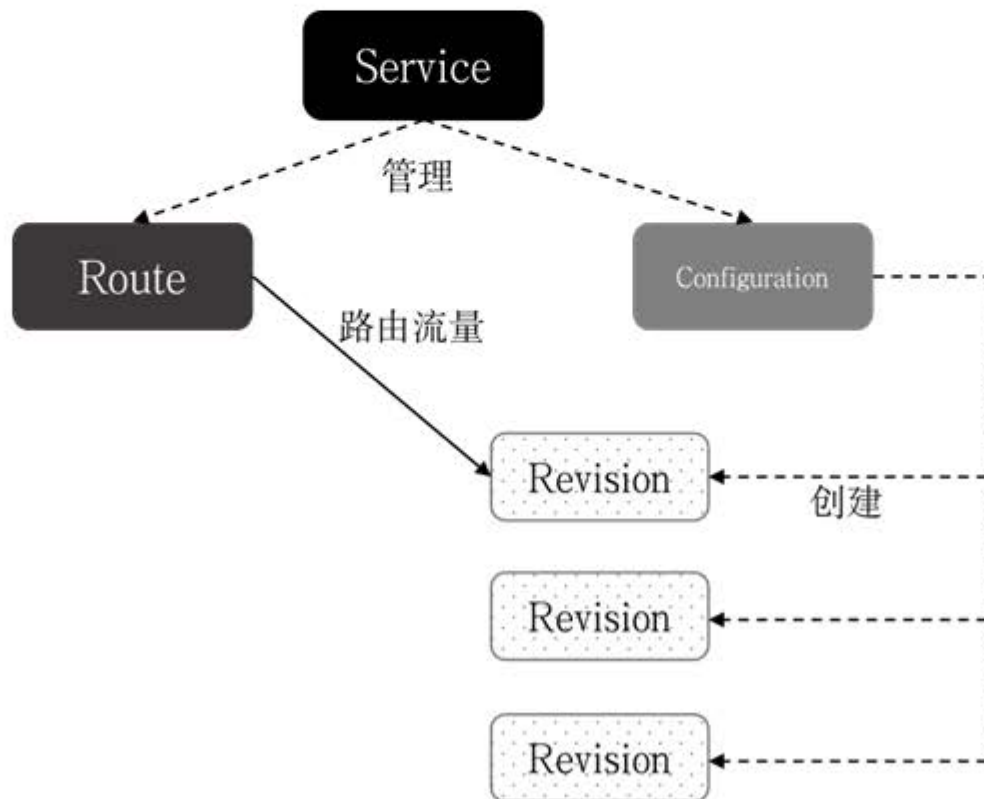
CLOUD NATIVE COMPUTING FOUNDATION

Redpoint

Cloud Native Landscape

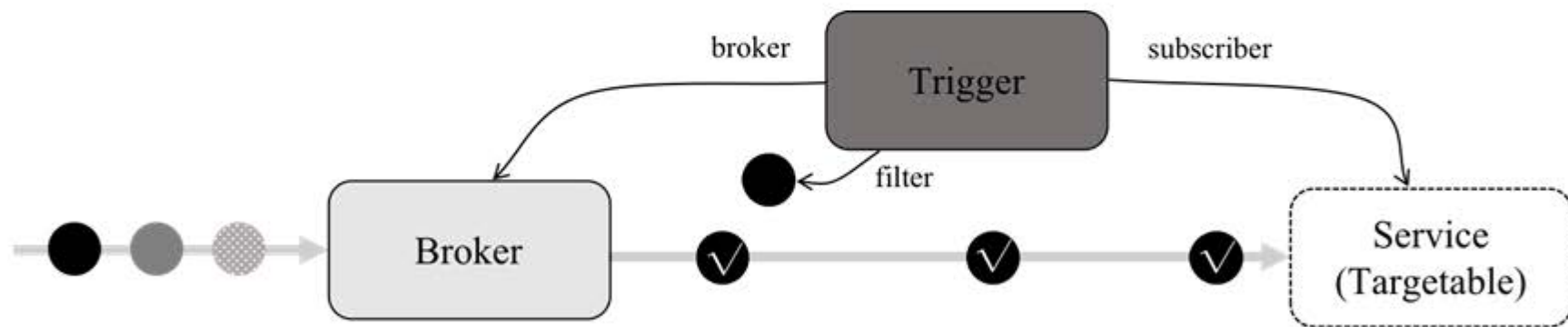


Knative: 旨在为开发者提供一台简单易用的无服务器架构部署方案。包含Serving和Eventing。

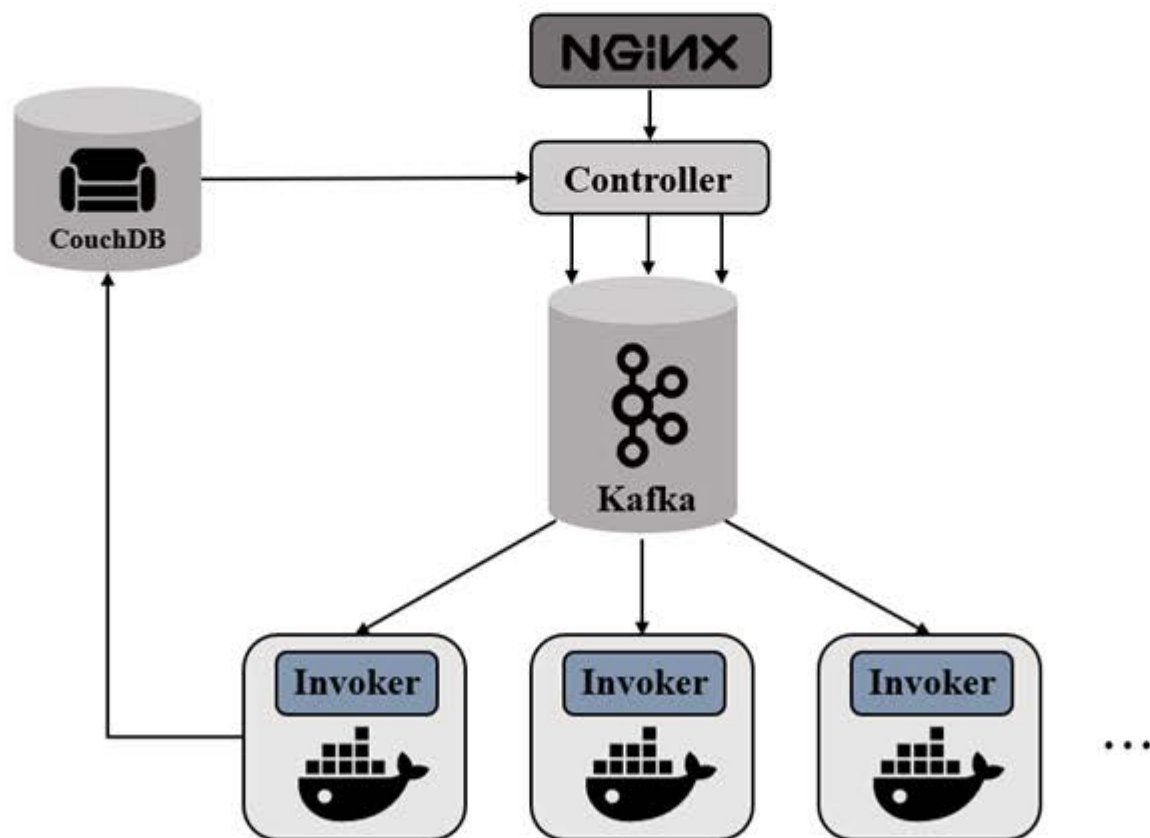


► Knative流程图:

- Trigger描述了基于事件属性的过滤器。
- Broker提供了一个事件桶，桶内的事件能够通过事件属性进行筛选。



Apache OpenWhisk高层次结构:



- ▶ 边缘计算为何需要虚拟化技术？
- ▶ 虚拟化技术大致可分为几类？
- ▶ 请简述全虚拟化和半虚拟化。
- ▶ 简要对比传统虚拟机和容器，概述其各自的优点。