极客时间训练营

系统架构设计师

第14章 云原生架构设计理论与实践

授课：王建平

目录

目录

云原生架构定义

◆云原生架构是基于云原生技术的一组架构原则和设计模式的集合，旨在将云应用中的非业务代码部分进行最大化的剥离，从而让云设施接管应用中原有的大量非功能特性(如弹性、韧性、安全、可观测性、灰度等)，使业务不再有非功能性业务中断困扰的同时，具备轻量、敏捷、高度自动化的特点。（★★）

◆云原生的代码通常包括三部分：业务代码、三方软件、处理非功能特性的代码。从业务代码中剥离大量非功能性特性(不会是所有，比如易用性还不能剥离)到IaaS和PaaS中。

◆基于云原生架构的应用特点：（★★★）
(1)代码结构发生巨大变化：不再需要掌握文件及其分布式处理技术，不再需要掌握各种复杂的网络技术，简化让业务开发变得更敏捷、更快速。。
(2)非功能性特性大量委托给云原生架构来解决：比如高可用能力、容灾能力、安全特性、可运维性、易用性、可测试性、灰度发布能力等。
(3)高度自动化的软件交付：基于云原生的自动化软件交付可以把应用自动部署到成千上万的节点上。

云原生架构原则

云原生架构原则（★★★）
◆服务化原则：拆分为微服务架构、小服务架构，分别迭代。
◆弹性原则：系统的部署规模可以随着业务量的变化而自动伸缩。
◆可观测原则：通过日志、链路跟踪和度量等手段。
◆韧性原则：当软件所依赖的软硬件组件出现各种异常时，软件表现出来的抵御能力。
◆所有过程自动化原则：一方面标准化企业内部的软件交付过程，另一方面在标准化的基础上进行自动化，通过配置数据自描述和面向终态的交付过程。
◆零信任原则：默认情况下不应该信任网络内部和外部的任何人/设备/系统，需要基于认证和授权重构访问控制的信任基础，以身份为中心。
◆架构持续演进原则：云原生架构本身也必须是一个具备持续演进能力的架构。

典型真题

下列关于云原生架构原则的选项，有错误的是（ ）。
A．服务化原则、弹性原则、韧性原则
B．可观测原则、所有过程自动化原则
C．零信任原则、接口隔离原则
D．架构持续演进原则

解析：接口隔离原则是面向对象设计原则，其含义是使用多个专门的接口比使用单一的总接口好。它不是云原生架构原则。
参考答案：C

云原生主要架构模式

◆服务化架构模式（★★）
服务化架构要求以应用模块为颗粒度划分一个软件，以接口契约(例如IDL) 定义彼此业务关系，以标准协议(HTTP、gRPC 等)确保彼此的互联通，结合DDD(领域模型驱动)、TDD(测试驱动开发)、容器化部署提升每个接口的代码质量和迭代速度。

服务化架构的典型模式是微服务和小服务模式，其中小服务可以看作是一组关系非常密切的服务的组合，这组服务会共享数据。

小服务模式通常适用于非常大型的软件系统，避免接口的颗粒度太细而导致过多的调用损耗(特别是服务间调用和数据一致性处理)和治理复杂度。通过服务化架构，把代码模块关系和部署关系进行分离，每个接口可以部署不同数量的实例，单独扩缩容，从而使得整体的部署更经济。此外，由于在进程级实现了模块的分离，每个接口都可以单独升级，从而提升了整体的迭代效率。但也需要注意，服务拆分导致要维护的模块数量增多，如果缺乏服务的自动化能力和治理能力，会让模块管理和组织技能不匹配，反而导致开发和运维效率的降低。

云原生主要架构模式

◆Mesh化架构模式（★★）

Mesh 化架构是把中间件框架(如 RPC、缓存、异步消息等)从业务进程中分离，让中间性SDK 与业务代码进一步解耦，从而使得中间件升级对业务进程没有影响，其至迁移到另外一个平台的中间件也对业务透明。分离后在业务进程中只保留很"薄"的 Client 部分，Client 通常很少变化，只负责与 Mesh 进程通信，原来需要在SDK中处理的流量控制、安全等逻辑由Mesh进程完成。

实施Mesh化架构后，大量分布式架构模式(熔断、限流、降级、重试、反压、隔仓)都由Mesh 进程完成，即使在业务代码的制品中并没有使用这些三方软件包；同时获得更好的安全性(比如零信任架构能力)、按流量进行动态环境隔离、基于流量做冒烟/回归测试等。

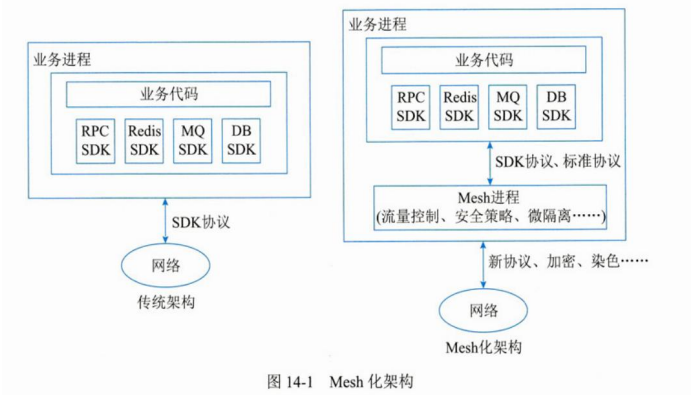
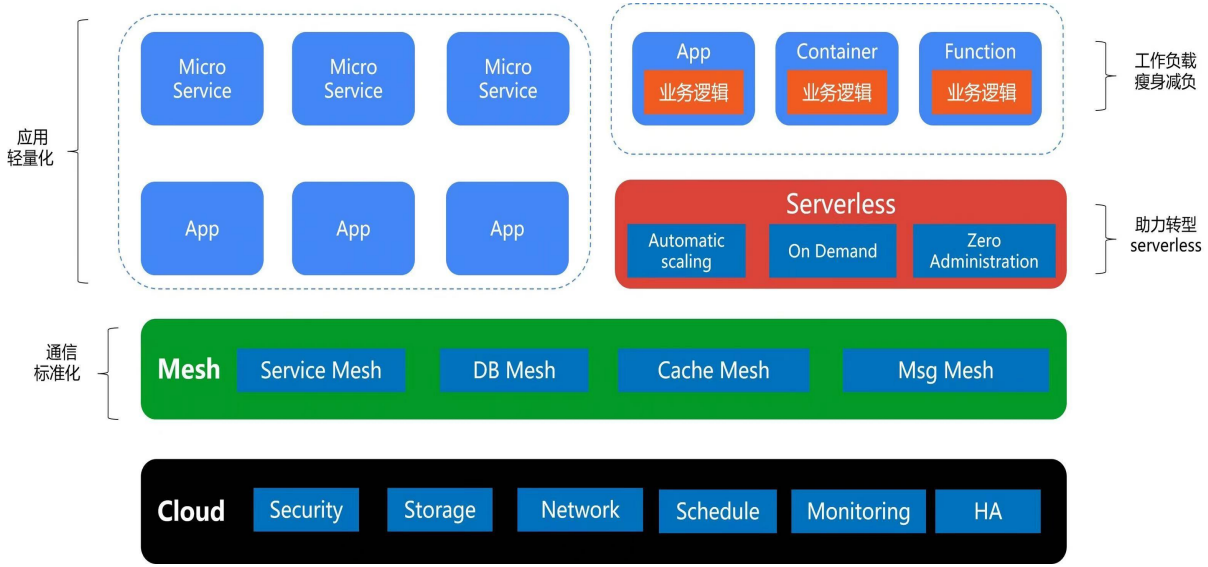


图 14-1 Mesh 化架构



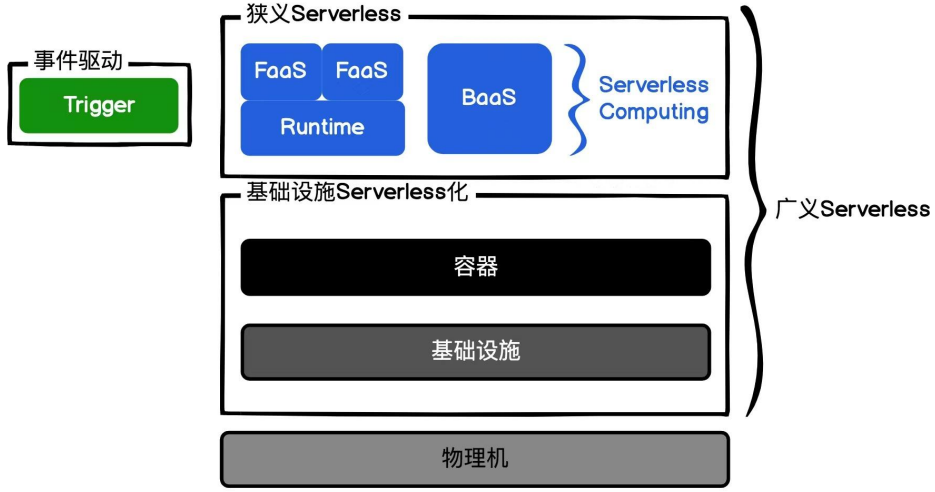
云原生主要架构模式

◆Serverless 模式（★★）

Serverless 将“部署”这个动作从运维中“收走”，使开发者不用关心应用运行地点、操作系统、网络配置、CPU性能等，从架构抽象上看，当业务流量到来/业务事件发生时，云会启动或调度一个已启动的业务进程进行处理，处理完成后云自动会关闭/调度业务进程，等待下一次触发，也就是把应用的整个运行都委托给云。

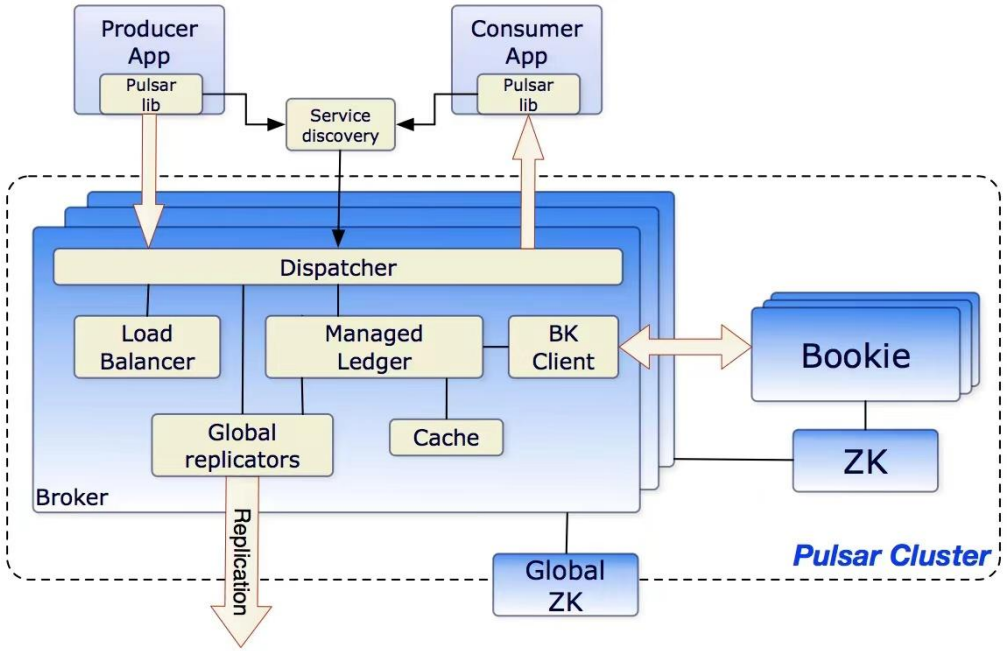
Serverless 并非适用任何类型的应用，因此架构决策者需要关心应用类型是否适合于Serverless运算。如果应用是有状态的，由于Serverless 的调度不会帮助应用做状态同步，因此云在进行调度时可能导致上下文丢失；如果应用是长时间后台运行的密集型计算任务，会无法发挥 Serverless 的优势；如果应用涉及频繁的外部I/O(网络或者存储，以及服务间调用)，也因为繁重的I/O负担、时延大而不适合。

Serverless 非常适合于事件驱动的数据计算任务、计算时间短的请求/响应应用、没有复杂相互调用的长周期任务。



云原生主要架构模式

◆存储计算分离模式：在云环境中，推荐把各类暂态数据(如session)、结构化和非结构化持久数据都采用云服务来保存，从而实现存储计算分离。（★★）



云原生主要架构模式

◆分布式事务模式：大颗粒度的业务需要访问多个微服务，必然带来分布式事务问题，否则数据就会出现不一致。架构师需要根据不同的场景选择合适的分布式事务模式。（★）

◆可观测架构：可观测架构包括Logging、Tracing、Metrics 三个方面，其中Logging提供多个级别的详细信息跟踪，由应用开发者主动提供；Tracing提供一个请求从前端到后端的完整调用链路跟踪，对于分布式场景尤其有用；Metrics则提供对系统量化的多维度度量。（★★）

◆事件驱动架构：本质上是一种应用/组件间的集成架构模式。可用于服务解耦、增强服务韧性、数据变化通知等场景中。（★★）

图 14-2　事件驱动架构

云原生主要架构模式

◆典型的云原生架构反模式。（★）
◆架构设计有时候需要根据不同的业务场景选择不同的方式，常见的云原生反模式有：
（1）庞大的单体应用：缺乏依赖隔离，代码耦合，责任和模块边界不清晰，模块间接口缺乏治理，变更影响扩散，不同模块间的开发进度和发布时间要求难以协调，一个子模块不稳定导致整个应用都变慢，扩容时只能整体扩容而不能对达到瓶颈的模块单独扩容等。
（2）单体应用“硬拆”为微服务：强行把耦合度高、代码量少的模块进行服务化拆分；拆分后服务的数据是紧密耦合的；拆分后成为分布式调用，严重影响性能。
（3）缺乏自动化能力的微服务：人均负责模块数上升，人均工作量增大，也增加了软件开发成本。

典型真题

下列不属于云原生架构反模式的是（ ）。
A．庞大的单体应用
B．单体应用“硬拆”为微服务
C．缺乏自动化能力的微服务
D．分布式事务模式
参考答案：D

目录

云原生架构相关技术-容器技术

◆容器技术（★）

容器作为标准化软件单元，它将应用及其所有依赖项打包，在运行的时候就不再依赖宿主机上的文件操作系统类型和配置。帮助开发者跳过设置冗杂的开发环境，不用担心不同环境下的软件运行的环境配置问题（将应用程序的配置和所有依赖打包成一个镜像在容器中）。在不同计算环境间快速、可靠地运行。

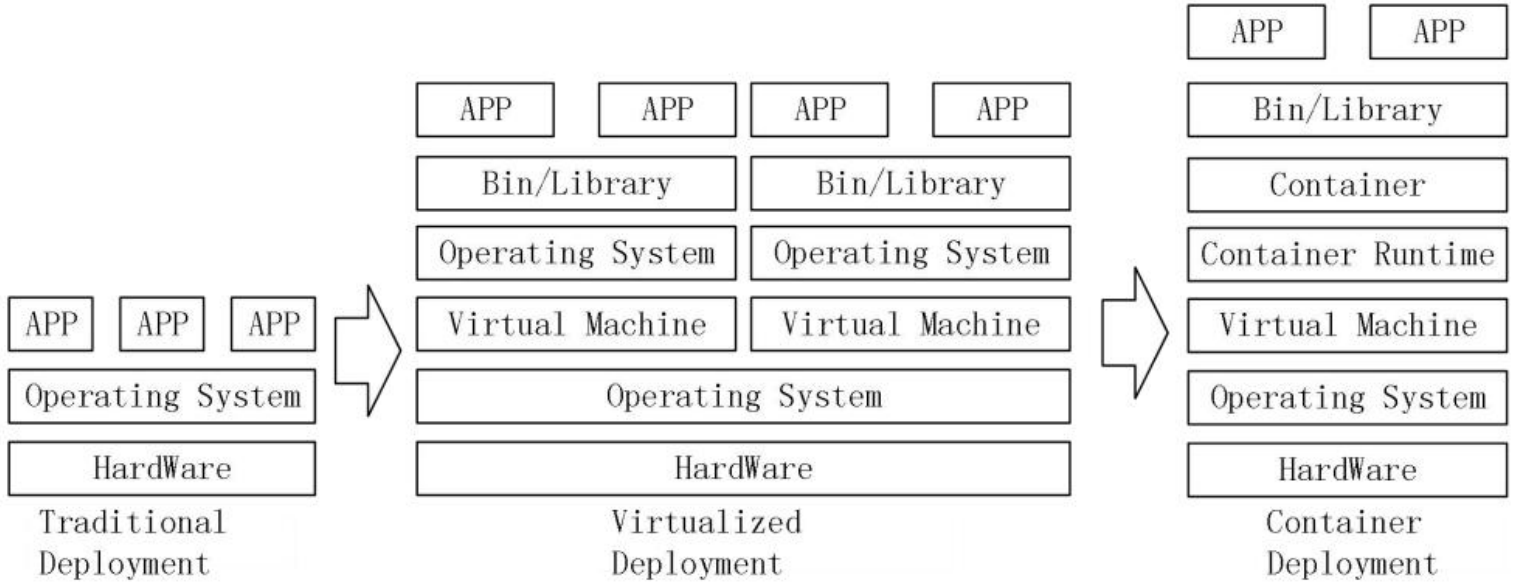


图 17-2 传统、虚拟化、容器部署模式比较

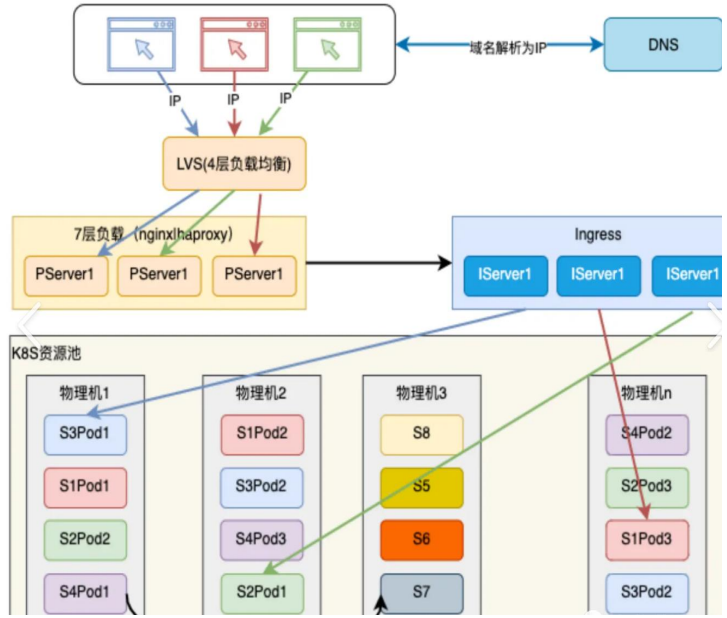
云原生架构相关技术-容器编排

◆容器编排（★）

Kubernetes 已经成为容器编排的事实标准，被广泛用于自动部署，扩展和管理容器化应用，是一个全新的基于容器技术的分布式架构解决方案。

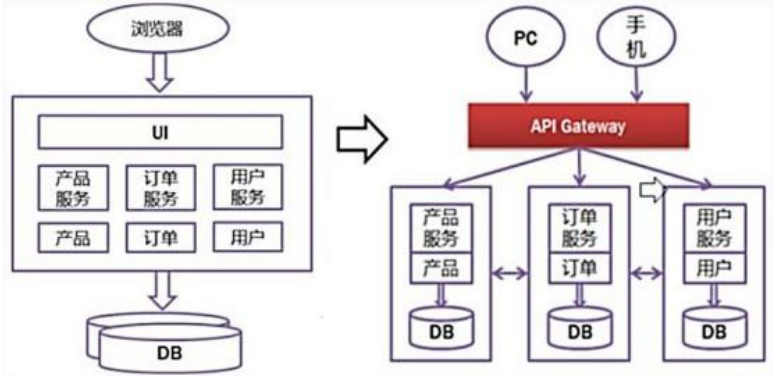
◆Kubernetes 提供了分布式应用管理的核心能力。

- ✓ 资源调度：根据应用请求的CPU、Memory资源量，或者GPU等设备资源，在集群中选择合适的节点来运行应用。
- ✓ 应用部署与管理：支持应用的自动发布与应用的回滚，以及与应用相关的配置的管理；也可以自动化存储卷的编排，让存储卷与容器应用的生命周期相关联。
- ✓ 自动修复：Kubernetes能监测这个集群中所有的宿主机，当宿主机或者OS出现故障，节点健康检查会自动进行应用迁移；K8s也支持应用的自愈，极大简化了运维管理的复杂性。
- ✓ 服务发现与负载均衡：Service是对一组提供相同功能的Pods的抽象，并为它们提供一个统一的入口。借助Service，应用可以方便的实现服务发现与负载均衡，并实现应用的零宕机升级。
- ✓ 弹性伸缩：K8s可以监测业务上所承担的负载，如果这个业务本身的CPU利用率过高，或者响应时间过长，它可以对这个业务进行自动扩容。



云原生架构相关技术-微服务

◆微服务。微服务模式将后端单体应用拆分为松耦合的多个子应用，每个子应用负责一组子功能。这些子应用称为“微服务”，多个“微服务”共同形成了一个物理独立但逻辑完整的分布式微服务体系。这些微服务相对独立，通过解耦研发、测试与部署流程，提高整体迭代效率。（★）
◆微服务设计约束如下：（★）
（1）微服务个体约束：微服务应用的功能在业务领域划分上应是相互独立的。
（2）微服务与微服务之间的横向关系：在合理划分好微服务间的边界后，从可发现性和可交互性处理微服务间的横向关系。可发现性是指当服务 A 发布和扩缩容的时候，依赖服务 A 的服务B 在不重新发布的前提下，能够自动感知到服务 A 的变化。可交互性是指服务 A 采用什么样的方式可以调用服务 B。
（3）微服务与数据层之间的纵向约束：提倡数据存储隔离 (Data Storage Segregation，DSS)原则，对于数据的访问都必须通过相对应的微服务提供的 API 来访问。
（4）全局视角下的微服务分布式约束：高效运维整个系统，从技术上实现全自动化的 CI/CD 流水线满足对开发效率的诉求，并在这个基础上支持蓝绿、金丝雀等不同发布策略，以满足对业务发布稳定性的诉求。



云原生架构相关技术-微服务

◆主要的微服务技术（★）

- ✓ Apache Dubbo是阿里的一款高性能、轻量级的开源服务框架，提供了六大核心能力：面向接口代理的高性能RPC调用，智能容错和负载均衡，服务自动注册和发现，高度可扩展能力，运行期流量调度，可视化的服务治理与运维。

- ✓ Spring Cloud是一系列框架的有序集合，具有丰富的生态。它利用Spring Boot的开发便利性简化了分布式系统基础设施的开发，主要功能包括服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等。

云原生架构相关技术-无服务器技术

◆无服务器技术。（★）
无服务器技术的特点：
（1）全托管的计算服务——客户只需要编写代码构建应用，无需关注同质化的、负担繁重的基于服务器等基础设施的开发、运维、安全、高可用等工作；
（2）通用性——结合云 BaaS（后端云服务） API 的能力，能够支撑云上所有重要类型的应用；
（3）自动弹性伸缩——让用户无需为资源使用提前进行容量规划；
（4）按量计费——让企业使用成本得有效降低，无需为闲置资源付费。

◆函数计算(FaaS) 是Serverless 中最具代表性的产品形态。通过把应用逻辑拆分多个函数，每个函数都通过事件驱动的方式触发执行。

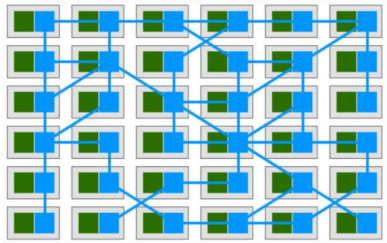
◆无服务器技术关注点：计算资源弹性调度、负载均衡和流控、安全性。

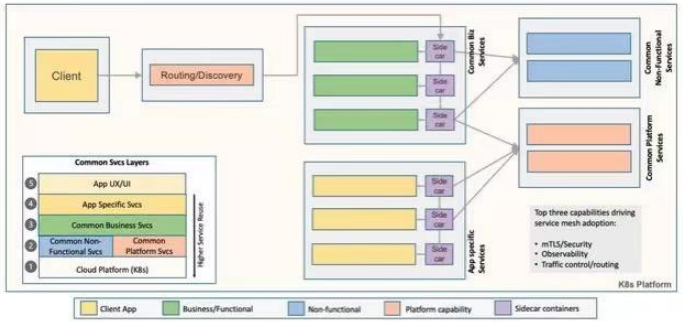
云原生架构相关技术-服务网格

◆服务网格(Service Mesh)是一个专门处理服务通讯的基础设施层。它的职责是在由云原生应用组成服务的复杂拓扑结构下进行可靠的请求传送。在实践中，它是一组和应用服务部署在一起的轻量级的网络代理，并且对应用服务透明。服务网格把 SDK 中的大部分能力从应用中剥离出来，拆解为独立进程，以Sidecar 的模式进行部署。服务网格通过将服务通信及相关管控功能从业务程序中分离并下沉到基础设施层，使其和业务系统完全解耦，使开发人员更加专注于业务本身。服务网格从总体架构上来讲比较简单，不过是一堆紧挨着各项服务的用户代理，外加一组任务管理组件组成。

管理组件被称为控制平面(Control plane)，负责与控制平面中的代理通信，下发策略和配置。代理被称为数据平面(Data plane)，直接处理入站和出站数据包，转发、路由、健康检查、负载均衡、认证、鉴权、产生监控数据等。Istio 是一个由Google和IBM等公司开源的，功能十分丰富的服务网格具体实现，也是目前业界最为流行的实现。Istio 的架构从逻辑上分成数据平面和控制平面。

- ✓ 数据平面：由一组和业务服务成对出现的 Sidecar 代理（Envoy）构成，主要功能是接管服务的进出流量，传递并控制服务和Mixer 组件的所有网络通信。
- ✓ 控制平面：包括了 Pilot、Mixer、Citadel 和Galley 4 个组件，主要功能是通过配置和管理 Sidecar 代理来进行流量控制，并配置Mixer 去执行策略和收集遥测数据。





云原生架构相关技术-服务网格

◆服务网格的主要技术：Istio、Linkerd、Consul。（★）

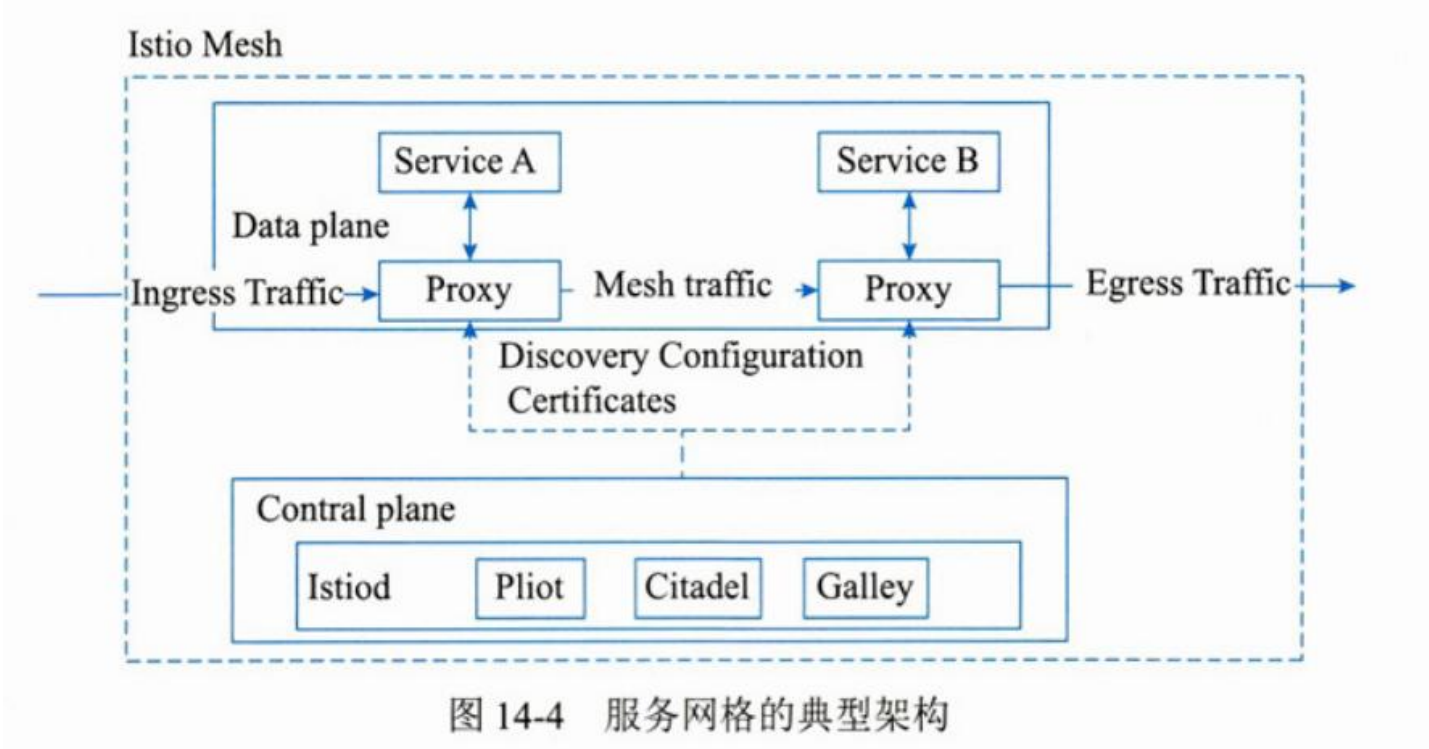


图 14-4　服务网格的典型架构

目录

云原生架构案例分析

◆某旅行公司云原生改造
面临两个问题：公司主体由两个公司合并，技术体系不同，需要整合为一体；节假日高并发流量。
改造第一阶段，某旅行技术团队为了提升集群资源利用率，降低资源使用成本。利用云原生思维重构部分技术体系，将多套旧有系统合并、收拢到一套以云原生应用为核心的私有云平台上，同时将IDC、物理网络、虚拟网络、计算资源、存储资源等通过laaS、PaaS等，实现虚拟化封装、切割、再投产的自动化流程。随着服务器集群规模的扩大，部分机器开始频繁出现故障。此时，保障服务稳定性成了第二阶段改造的首要任务。
第二阶段基于公有云、私有云和离线专属云集群等新型动态计算环境，某旅行公司的技术团队帮助业务构建和运行具有弹性的云原生应用，促进业务团队开始使用声明式API,同时通过不可变基础设施、服务网格和容器服务，来构建容错性好、易于管理和观察的应用系统，并结合平台可靠的自动化恢复、弹性计算来完成整个服务稳定性的提升。
第三阶段通过基础组件、服务的云原生改造、服务依赖梳理和定义等方式，使应用不再需要考虑底层资源、机房、运行时间和供应商等因素。此外，还利用标准的云原生应用模型，实现了服务的跨地域、跨云自动化灾备、自动部署，并向云原生场景下的Dev0ps演进。

架构图如下：

云原生架构案例分析

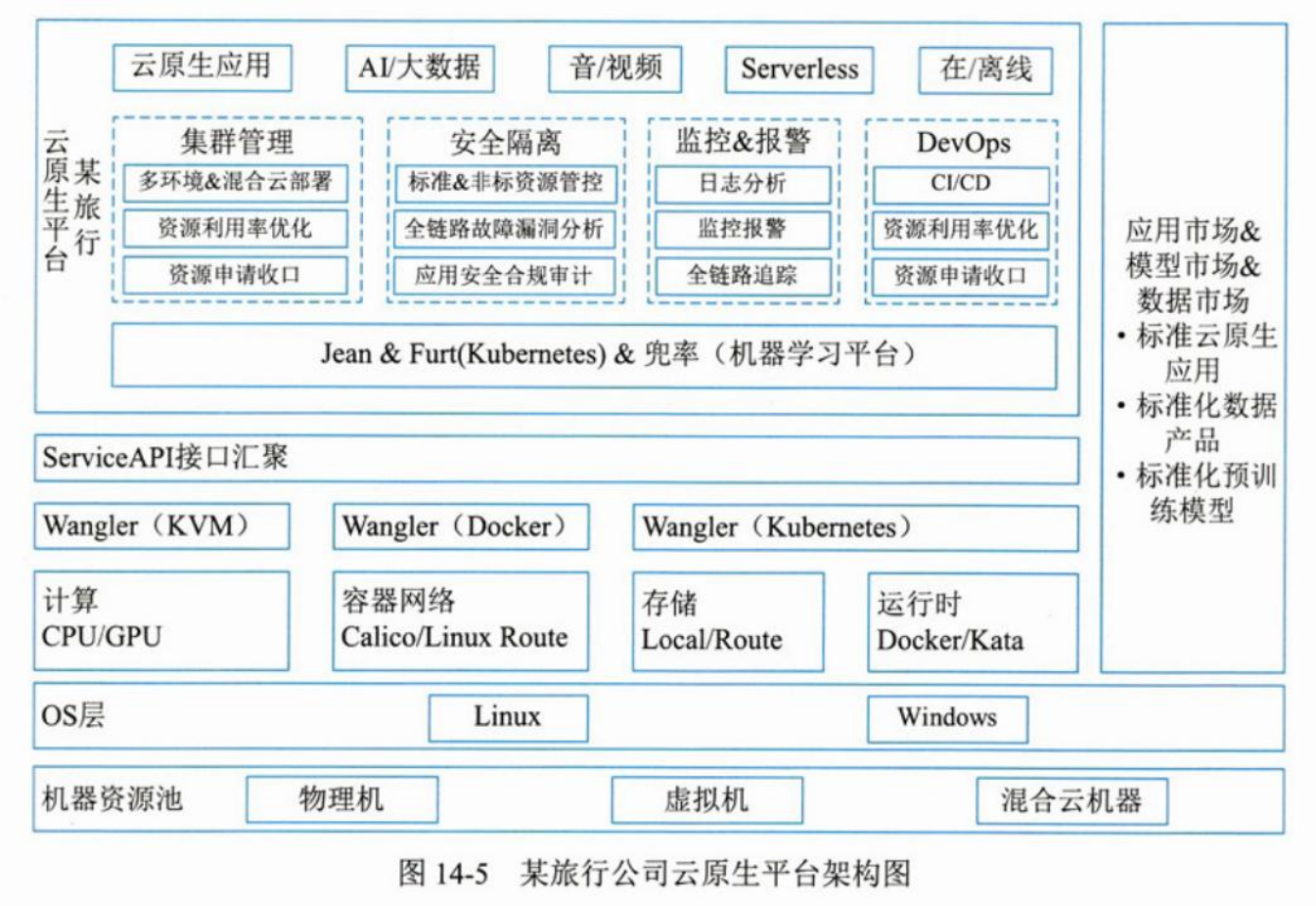


图 14-5　某旅行公司云原生平台架构图

云原生架构案例分析

◆云原生技术助力某汽车公司数字化转型实践

战略性构建容器云平台。通过平台实现对某云行App、二手车、在线支付、优惠券等核心互联网应用承载。以多租户的形式提供弹性计算、数据持久化、应用发布等面向敏捷业务服务，并实现高水平资源隔离。标准化交付部署，快速实现业务扩展，满足弹性要求。利用平台健康检查、智能日志分析和监控告警等手段及时洞察风险，保障云平台和业务应用稳定运行。

数字混合云交付。采用私有云+公有云的混合交付模式，按照服务的敏态/稳态特性和管控要求划分部署，灵活调度公有云资源来满足临时突发或短期高TPS业务支撑的需求。利用PaaS平台标准化的环境和架构能力，实现私有云和公有云一致交付体验。

深度融合微服务治理体系，实现架构的革新和能力的沉淀，逐步形成支撑数字化应用的业务中台。通过领域设计、系统设计等关键步骤，对原来庞大的某云体系应用进行微服务拆分，形成能量、社群、用户、车辆、订单等多共享业务服务，同步制定了设计与开发规范、实施路径和配套设施，形成一整套基于微服务的分布式应用架构规划、设计方法论。

图 14-6 某容器云平台架构示意图

云原生架构案例分析

◆某快递公司核心业务系统云原生改造

某快递公司核心业务系统原架构基于Vmware+0racle数据库进行搭建。随着搬迁上阿里云，架构全面转型为基于Kubernetes的云原生架构体系。

通过引入云原生数据库、应用容器化、微服务改造技术，得到了如下的上云架构：

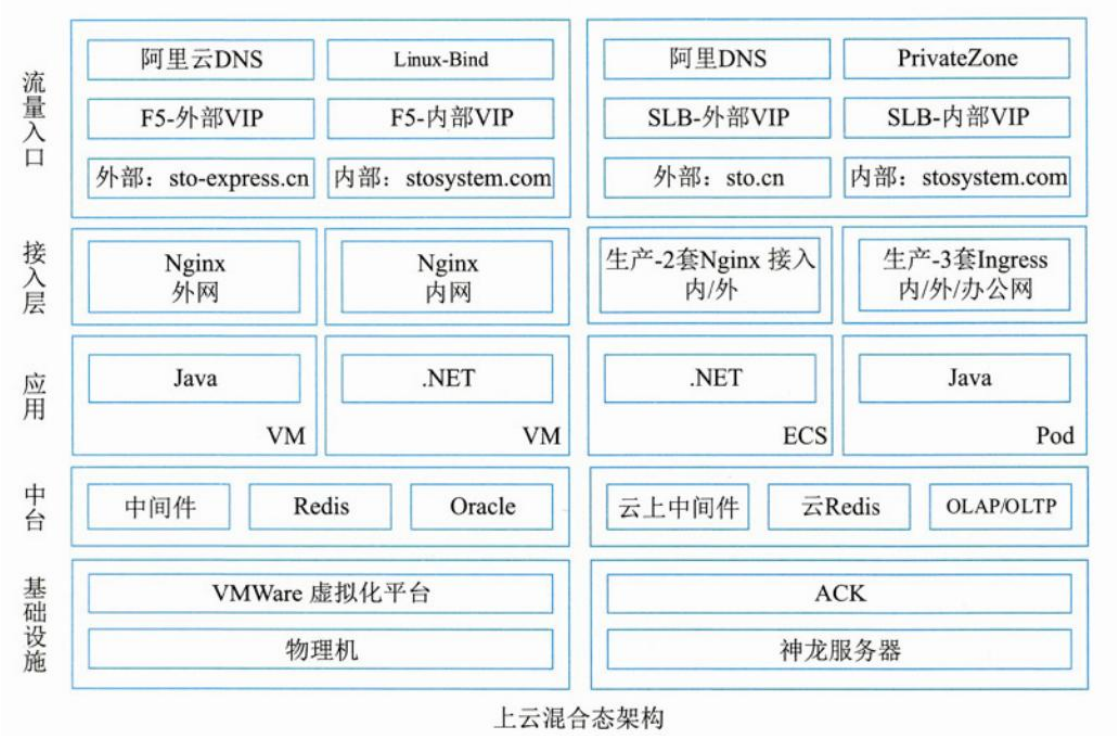


图 14-7　某快递公司核心业务上云架构示意图

基础设施，全部计算资源取自阿里云的神龙服务器。

流量接入，阿里云提供两套流量接入，一套是面向公网请求，另外一套是服务内部调用。

基于Kubernetes打造的云原生PaaS平台优势明显突出。每个应用都在Kubernetes上面创建单独的一个Namespace,应用和应用之间实现资源隔离。

线上Kubernetes集群采用阿里云托管版容器服务，免去了运维Master结点的工作，只需要制定Worker结点上线及下线流程即可。

云原生架构案例分析

◆某电商业务云原生改造

方案的关键点是：

- 通过容器化部署，利用阿里云容器服务的快速弹性应对大促时的资源快速扩容。
- 提前接入链路追踪产品，用于对分布式环境下复杂的服务调用进行跟踪，对异常服务进行定位，帮助客户在测试和生产中快速定位问题并修复，降低对业务的影响。
- 使用阿里云性能测试服务(PTS)进行压测，利用秒级流量拉起、真实地理位置流量等功能，以最真实的互联网流量进行压测，确保业务上线后的稳定运营。
- 采集压测数据，解析系统强弱依赖关系、关键瓶颈点，对关键业务接口、关键第三方调用、数据库慢调用、系统整体负载等进行限流保护。
- 配合阿里云服务团队，在大促前进行ECS/RDS/安全等产品扩容、链路梳理、缓存/连接池预热、监控大屏制作、后端资源保障演练等，帮助大促平稳进行。

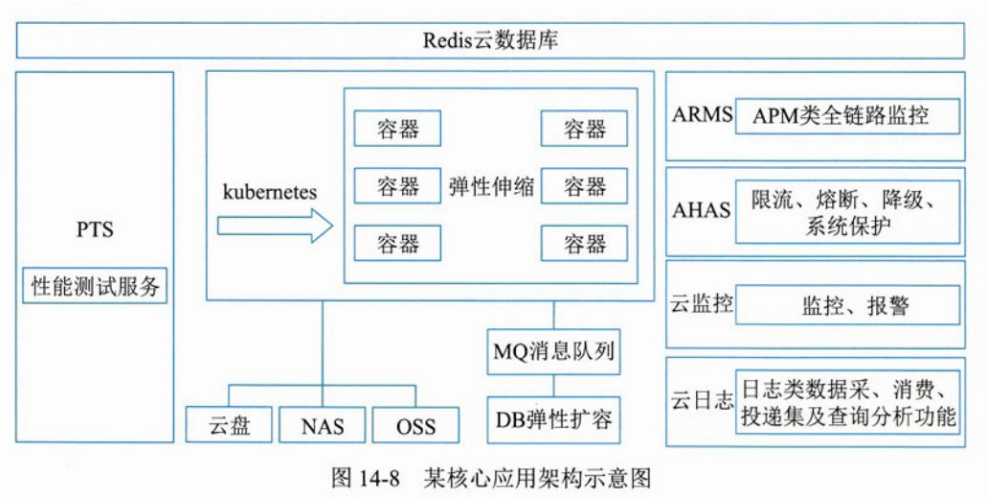


图 14-8　某核心应用架构示意图

典型真题

无服务器技术的技术特点之一是全托管的计算服务：客户只需要编写代码构建应用，无需关注同质化的、负担繁重的基于服务器等基础设施的 （）等工作；。
A．开发、测试、发布、交付
B．开发、运维、安全、高可用.
C．机房建设、服务器装机、操作系统安装、软件安装
D．资源调度、性能压测、负载均衡、数据统计

解析：无服务器技术的特点如下：全托管的计算服务——客户只需要编写代码构建应用，无需关注同质化的、负担繁重的基于服务器等基础设施的开发、运维、安全、高可用等工作；通用性——结合云 BaaSAPI 的能力，能够支撑云上所有重要类型的应用；自动弹性伸缩——让用户无需为资源使用提前进行容量规划；按量计费——让企业使用成本得有效降低，无需为闲置资源付费。因此 B 选项正确。
答案：B

典型真题

容器作为标准化软件单元，它将应用及其所有依赖项打包，使应用不再受（ ）限制，在不同计算环境间快速、可靠地运行。
A．环境.　　B．操作系统　　C．硬件　D．网络

解析：在容器的帮助下，应用程序无需关注操作系统及更加低层的硬件、网络、存储的限制，因此选项 B、C、D 的说法有局限性，选项 A 更贴切。
答案：A

本章重点回顾

1、云原生架构原则
2、主要架构模式
3、关键技术

THANKS