系统架构设计师

云原生架构设计

姜美荣

目录

直播内容：云原生架构

云原生相关案例☆☆☆☆

- 云计算架构
- 云原生的特点
- 云原生的架构原则和模式及其反模式
- 传统>虚拟化>容器技术比较

云原生架构

云计算：

云计算：是集合了大量计算设备和资源，对用户屏蔽底层差异的分布式处理架构，其用户与提供实际服务的计算资源是相分离的。

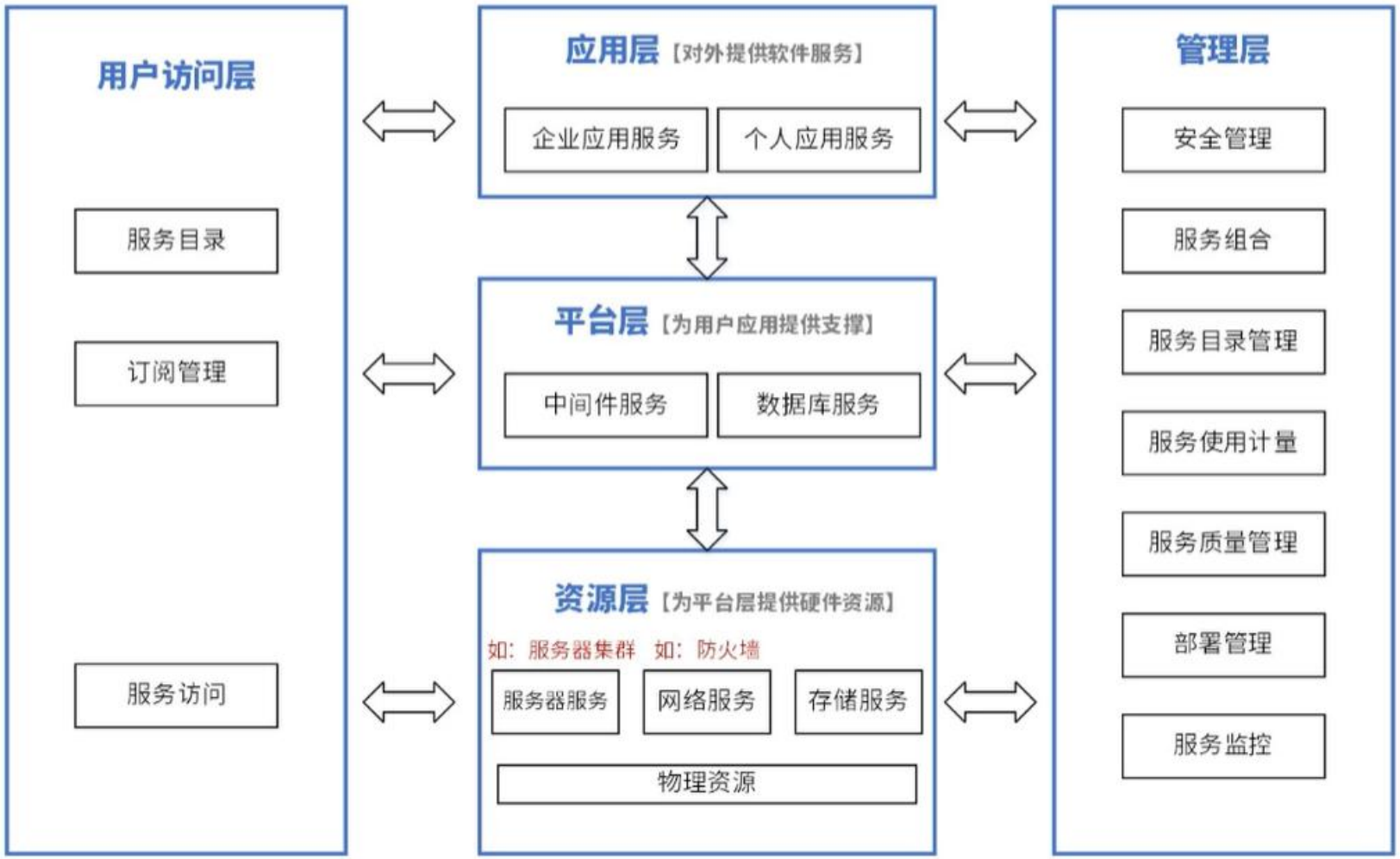
云计算优点:超大规模、虚拟化、高可靠性、高可伸缩性、按需服务、成本低【前期投入低、综合使用成本也低]

按**服务类型**分类：
SaaS(软件即服务)：基于多租户技术实现，直接提供应用程序
PaaS(平台即服务)：虚拟中间件服务器、运行环境和操作系统
IaaS（基础设计即服务）：包括服务器、存储和网络等服务

按**部署方式**分类：
公有云:面向互联网用户需求，通过开放网络提供云计算服务
私有云:面向企业内部提供云计算服务
混合云:兼顾以上两种情况的云计算服务

云计算架构

云计算：


用户访问层
服务目录
订阅管理
服务访问
应用层【对外提供软件服务】
企业应用服务
个人应用服务
平台层【为用户应用提供支撑】
中间件服务
数据库服务
资源层【为平台层提供硬件资源】
如：服务器集群
如：防火墙
服务器服务
网络服务
存储服务
物理资源
管理层
安全管理
服务组合
服务目录管理
服务使用计量
服务质量管理
部署管理
服务监控

云原生架构

云原生：Cloud+Native（云原生、**微服务、容器**、DevOps、持续交付、声明式API、服务网络Mesh）

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。（★★★）

云原生架构原则（★★★★★）

◆**服务化原则**：使用微服务。

◆**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。

云原生架构

云原生架构模式

1、**服务化架构模式：**典型代表**微服务**，服务拆分使维护压力大增。

2、**Mesh化架构模式：**把中间件框架(RPC、缓存、异步消息)从业务进程中分离，由**Mesh进程**完成。

3、**Serverless模式：**非常适合于**事件驱动**的数据计算任务。

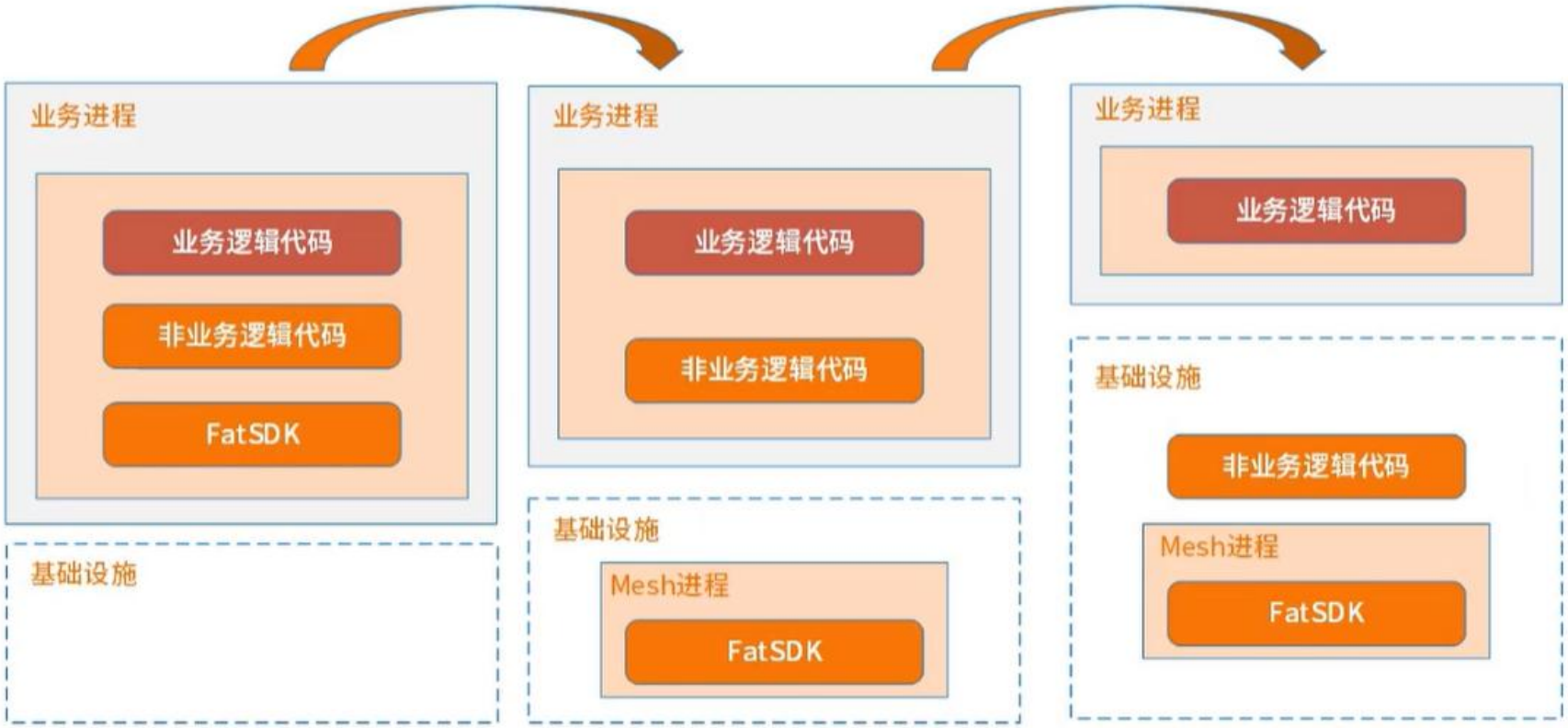
4、**存储计算分离模式：**各类**暂态数据(如session)用云**服务保存。

5、**分布式事务模式：**解决微服务模式中多数据源事务问题。

6、**可观测架构：**包括Logging、Tracing、Metrics三个方面。

7、**事件驱动架构：**本质上是一种应用/组件间的集成架构模式。

云原生架构



微服务架构模式　　Mesh架构模式　　Serverless架构模式

云原生架构反模式

云原生架构反模式：

1、**庞大的单体应用**

需要多人开发的业务模块，考虑通过服务化进行拆分，并让组织与架构匹配

2、**单体应用“硬拆”为微服务(服务拆分要适度)**

小规模软件的服务拆分(为拆而拆)、数据依赖(服务间数据依赖)、性能降低

3、**缺乏自动化能力的微服务**

手动维护大量微服务是不现实的

云原生架构

Docker容器基于操作系统虚拟化技术，共享操作系统内核、轻量、没有资源损耗、秒级启动，极大提升了系统的应用部署密度和弹性。

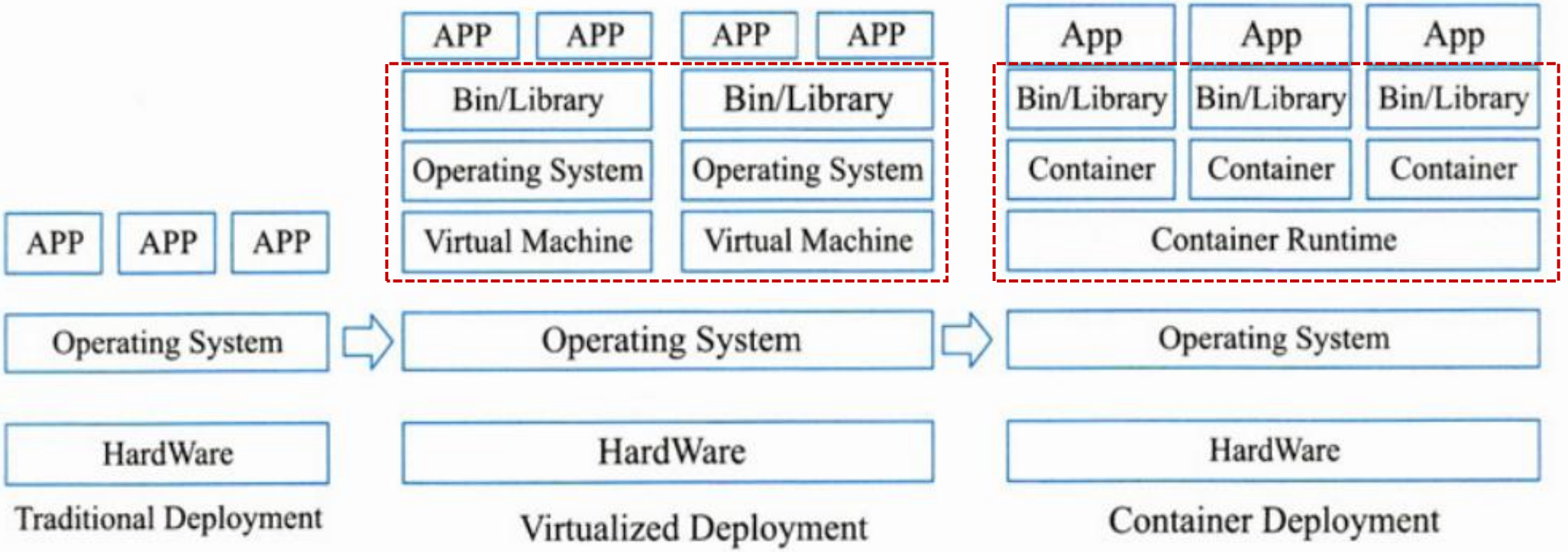


图 14-3　传统、虚拟化、容器部署模式比较

Docker容器基于操作系统虚拟化技术，共享操作系统内核、轻量、没有资源损耗、秒级启动，极大提升了系统的应用部署密度和弹性。

云原生架构

容器技术：

Kubernetes提供了**分布式应用管理的核心**能力。

【资源调度】根据**请求资源量**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。

【自动修复】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**。

【服务发现与负载均衡】结合DNS和负载均衡机制，支持容器化应用之间的相互通信。

【弹性伸缩】可以对这个业务进行自动扩容。

【声明式API】开发者可以关注于应用自身，而非系统执行细节。

【可扩展性架构】所有K8s组件都是基于一致的、开放的API实现和交互。

【可移植性】K8s通过一系列抽象如Load Balance Service(负载均衡服务)、CNI(容器网络接口)、CSI(容器存储接口)，帮助业务应用可以屏蔽底层基础设施的实现差异，实现容器灵活迁移的设计目标。

虚拟机技术VS容器技术

对比项	虚拟机技术	容器技术
镜像大小	包含GuestOS，G量级以上	仅包含运行的Bin/Lib，M量级
资源要求	CPU与内存按核、按G分配	CPU与内存按单核、低于G量级分配
启动时间	分钟级	毫秒级
可持续性	跨物理机迁移	跨操作系统平台迁移
弹性伸缩	VM伸缩，CPU/内存手动伸缩	实例自动伸缩、CPU内存自动在线伸缩
隔离策略	操作系统、系统级别	Cgroups，进程级别

微服务设计约束

1、微服务个体约束

每个微服务都是**独立**的，修改一个微服务**不能影响另一个微服务**

2、微服务与微服务之间的横向关系

通过**第三方服务注册中**心来满足服务的可发现性

3、微服务与数据层之间的纵向约束

数据是微服务的“私产”，访问时**需要通过微服务**

4、全局视角下的微服务分布式约束

高效运维整个系统

相关案例（2023年系统分析师）

试题三（25分）

随着嵌入式计算资源快速提升，容器技术（Docker）发挥重要作用，某公司对原有平台升级，公司将平台升级任务交给了张工，张工经过分析、调研，提出在园嵌入式操作系统平台上采用容器技术的升级方案，但该方案引发了争议。

争论焦点是采用容器技术还是虚拟机（VM）技术。李工指出由于容器技术共享主机内核能向虚拟机一样完全隔离，系统存在安全问题；如果采用虚拟机技术除满足需求外，还保证了系统的安全和稳定，会上领导根据系统升级的初衷选择了张工的升级方案。

【问题1】（6分）
请用300字以内的文字说明容器技术和虚拟技术的含义，并简要论述公司领导采纳容器技术的原因。

相关案例（2023年系统分析师）

公司领导采纳容器技术的原因：

（1）资源需求方面：Docker是内核级虚拟化，对资源（内存、硬盘等）额外需求相对于传统虚拟机方式少很多。一台主机可以同时运行上千个Docker容器；传统虚拟机方式需要运行完整操作系统，对资源消耗较大，往往只能运行几十个容器。

（2）跨平台性：Docker跨平台性强，可以在绝大部分平台（物理机、虚拟机、公/私有云等）上运行，可实现跨操作系统平台迁移。

（3）秒级启停：Docker容器可以实现毫秒/秒级启动和停止，比传统虚拟机快很多。

（4）安全性方面。传统虚拟机可以是相对封闭的隔离方式，虚拟化硬件资源隔离操作系统；Dcoker利用Linux多种安全防护技术实现进程级别隔离，改善并加强了容器的安全控制和镜像的安全机制。

（5）更轻松的维护和扩展。Docker使用分层存储以及镜像技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展也变得非常简单。

相关案例（2023年系统分析师）

【问题2】（9分）

表2-3给出了虚拟技术和容器技术的性能对比表，请根据下面的（a）~（h）的8个性能指标；判断这些指标属于哪类对比项，补充完善表3-1的（1）~（8）的空白处。

（a）分钟级　　（b）包含GuestOS，G量级以上　（c）跨操作系统平台迁移

（d）CPU与内存按核、按G分配　（e）毫秒/秒级　　（f）Cgroups，进程级别

（g）VM伸缩，cpu/内存手动伸缩　（h）实例自动伸缩、cpu/内存自动在线伸缩

表 2-3　虚拟技术和容器技术的对比表

对比项	虚拟机技术	容器技术
镜像大小	①	仅包含运行的 Bin/Lib，M 量级
资源要求	②	CPU 与内存按单核、低于 G 量级分配
启动时间	③	④
可持续性	跨物理机迁移	⑤
弹性伸缩	⑥	⑦
隔离策略	操作系统、系统级别	⑧

相关案例（2023年系统分析师）

【参考答案】

①：b　②：d　③：a　④：e

⑤：c　⑥：g　⑦：h　⑧：f

相关案例（2023年系统分析师）

【问题3】（10分）

张工在方案中指出，容器技术通常包含镜像、镜像仓库和容器等三个基本实体。简述这三个基本实体的特点，并说明是位于远端还是主机。

相关案例（2023年系统分析师）

【参考答案】

Docker包括三个基本实体有镜像、容器和仓库。仓库位于远端，而镜像和容器位于本机。

镜像：镜像就是打包好的环境与应用，相当于是一个root文件系统。

容器：镜像和容器的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。

仓库：仓库可以看成一个代码控制中心，用来保存镜像。一个仓库内可以保存多个镜像。

作业

某公司计划将其在线购物平台迁移到云原生架构，以提高系统的可伸缩性、弹性和运维效率。该平台包括用户管理、商品管理、订单处理、支付处理和物流跟踪等微服务。公司希望在迁移过程中，能够充分利用云服务提供商的资源，实现服务的自动扩展、故障自愈和持续集成/持续部署（CI/CD）。

1.云原生架构的核心原则有哪些?

2.在云原生架构下，如何保证数据的一致性和安全性？请给出一些建议。

作业

参考答案：

【问题1】

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。

作业

参考答案：

【问题2】 数据一致性和安全性建议：

- 使用分布式数据库和存储系统，如Cassandra或Ceph，保证数据的高可用性和一致性。
- 采用数据分片和复制技术，提高数据的读写性能和容灾能力。
- 利用云服务提供商的安全服务，如AWS的IAM和KMS，实现身份认证和数据加密。
- 定期进行安全审计和合规性检查，确保系统的安全性。

作业

论云原生架构及其应用

“云原生”来自于Cloud Native 的直译，拆开来看，Cloud 就是指其应用软件是在云端而非传统的数据中心。Native 代表应用软件从一开始就是基于云环境、专门为云端特性而设计，可充分利用和发挥云平台的弹性＋分布式优势，最大化释放云计算生产力。从技术的角度，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。

请围绕“论云原生架构设计”论题，依次从以下三个方面进行论述。

1、概要叙述你参与开发的软件项目以及你在其中所承担的主要工作。

2、详细说明云原生的主要架构模式有哪些。

3、结合项目实际，详细说明你采用了哪些架构模式进行云原生架构设计的。

作业

摘要

本文以我参与的某公司“酒业上云”项目为例，论述了云原生架构设计方法。该项目的目标是构建以某酒厂生产的白酒产品为主的电子商城，实现该白酒厂商的线下营销转型为在线营销的战略目标，包括线上抢购、支付、线下原厂配送、防伪溯源等一系列电子商务功能。在此项目中，我作为系统架构师及主要管理人员，主导了该项目的系统架构设计等工作。经过实践认为：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，主要的架构模式包括服务化架构模式、Mesh化架构模式、Serverless模式、存储计算分离模式、分布式事务模式、可观测模式等。在本项目中需要根据不同的非功能性需求来选用不同的架构模式，才能保证项目的功能和质量指标的实现，项目因此取得了成功。

作业

正文

近年来，随着互联网科技的发展，中国电子商务发展迅速，变得和我们日常生活息息相关，也受到了越来越多的企业的关注。2021年下半年，某著名酒业公司决定发展电子商城及线上促销业务，发起了“酒业上云项目”，实现线上抢购、支付、线下原厂配送、防伪溯源等电子商务功能。该项目投资3000万，计划6个月完成，并对项目进行了公开招标，我公司成功中标。为此2021年10月，我作为该项目的系统架构师，全面负责酒业上云项目的架构设计工作，并在项目中实践了云原生架构设计方法，得到了项目组成员和公司高层的认可。下面重点阐述我在本项目中关于云原生架构设计方法的实践。

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。云原生架构模式包括：1、服务化架构模式。2、Mesh 化架构模式。3、存储计算分离模式。4、Serverless 模式。5、分布式事务模式。6、可观测架构模式。7、事件驱动架构模式。在具体的实践中，需要根据项目的特点和非功能性需求以及云计算平台的特性，来综合采用不同的架构模式，来实现项目的质量指标。

作业

在项目之初，我充分认识到该项目的特点和挑战：由于业主方是一个广受欢迎的白酒品牌，可以预见到电子商城APP一旦发布，会引来海量用户的广泛关注，业主方要求软件功能必须满足线上下单、支付、物流、溯源、防伪等多种完整业务链功能，也针对高可用性、高扩展性、高性能、高安全、高并发等非功能性需求做了极高的要求。由于我公司自建了云计算平台，以及在以往的项目中对于云计算架构的应用沉淀了丰富的经验，因此在此项目中我采取了面向云原生的架构设计方法，以期更好满足业主方对于非功能特性的需求。下文我将以常用的三个架构模式为例，详述项目组云原生架构设计的详细过程。

一、利用容器技术，来实现服务化架构。服务化架构是云时代构建云原生应用的标准架构模式，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)确保彼此的互联互通，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，通常适用于非常大型的软件系统。

在该项目中，公司的云计算平台维护了一套kubernates作为微服务的管理中间件，因此在该系统的应用层，项目组采用已经成为工业标准的docker容器技术来构建微服务。通过前期的功能模块设计，我们根据项目需求划分了用户、商户、商铺、购物车、订单、支付、物流、溯源等核心业务并分别封装成微服务，在统一的REST/HTTP协议之上定义了各自的API，输出了每个应用的docker镜像，做了自动化部署平台，结合kubernates针对微服务的自动化调度，节省了该项目中的大部分运维团队的成本。由于微服务本身具有解耦性，因此也给开发团队带来了技术选型、迭代开发、自动化测试的高度灵活性。

作业

二、封装独立的中间件和Paas平台，实现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，让中间件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，甚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。

在该项目中，针对数据库、缓存、消息队列三大中间件，项目组封装出了云计算Paas平台，并发布常用编程语言的SDK组件作为代理访问Paas的client部分。为了保护核心业务进程和中间件进程在秒杀场景下免受大流量冲击带来的风险，我们自研了风控服务来实现项目所要求的流量控制、安全等需求，通过风控服务后台配置页面，对各个业务模块和微服务设置访问上线和安全校验规则，client端代理访问或转发请求到其他微服务或中间件时，必须通过风控服务来访问，此时风控机制将会发生作用，阻止可能有风险的请求和流量，避免雪崩效应对整个平台带来的风险。

三、设计无状态服务，实现存储计算分离模式。分布式环境中的CAP困难主要是针对有状态应用，因为无状态应用不存在C(一致性)这个维度，因此可以获得很好的A(可用性)和P(分区容错性)，因而获得更好的弹性。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。

作业

在该项目的架构设计时，需求中需要持久化存储的实体数据，例如用户信息、订单、支付信息、商户、商品等，我们都一律落地存放在mysql中。对于一些临时的状态数据，例如用户登录状态、临时的短信校验码、限流的token等，我们也尽量避免存放在微服务本地，例如我们可以用redis作为session服务器存储用户的登录状态数据，用JWT构件来生成token和校验token，无需再存储token等。将各类状态数据从服务中剥离，使得每个微服务是完全相同的“计算节点”，可以自由地横向扩展，而redis、mysql即成为持久化或临时的“存储节点”。存储计算分离的模式，使得项目中应用层的微服务满足了A(可用性)和P(分区容错性)，解放了应用层的开发同事的工作，使得他们只专注于业务功能的实现。

得益于因地制宜采用了多个架构模式，酒业上云项目质量指标顺利达成。经过全体成员历时6个月的艰苦奋战，我们终于按期完成了项目目标，并于2022年4月1日试运行及验收测试，于5月1日正式全面运营，第一次秒杀活动系统运行平稳，顺利通过大考。当面对比较苛刻的非功能性需求时，适时采用云原生的架构设计方法是架构师的一项基本素质。我会继续学习架构技术，使得架构能力更上一层楼。

202505真题

事件驱动架构

事件驱动架构的概念和特点

分析、设计、开发的全过程

事件驱动架构（Event-Driven Architecture，EDA）是一种软件设计范式，通过事件的产生、发布和消费来实现组件间的解耦与协作。其核心思想是将系统分解为生产者（发布者）和消费者（订阅者），通过事件流（Event Stream）连接，使各组件能够独立演化。

特点：松耦合与高内聚、异步通信、可扩展性、高可用性、灵活性与自治、事件溯源、最终一致性。

THANKS