一、云计算概念-Concepts

1.什么是云计算

云计算是一种模型，它允许通过无处不在、便捷、按需的网络访问可配置计算资源（例如网络、服务器、存储、应用程序和服务）的共享池，这些资源可以通过最少的管理工作或服务提供商交互快速配置和发布。Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access ta shared pool of configurable computing resources (e.g., networks, servers, storage, applicationsand services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

2.云计算五个基本特征

five essential characteristics

1.On-demand self-service：消费者可以根据需要单方面自动配置计算功能，例如服务器时间和网络存储，而无需与每个服务提供商进行人工交互。consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

2. Broad network access：功能可通过网络使用，并通过标准机制访问，从而促进异构瘦客户端或胖客户端平台的使用。Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms

3. Resource pooling：云提供商的资源被集中在资源池中，为多个用户动态分配，具有位置透明性。There is a sense of location independence in that the customer generally has ncontrol or knowledge over the exact location of the provided resources but may be able tspecify location at a higher level of abstraction

4. Rapid elasticity or expansion：对于消费者来说，可供配置的功能通常看起来是无限的，并且可以随时占用任意数量的资源。When tthe consumer, the capabilities available for provisioning often appear tbe unlimited and can be appropriated in any quantity at any time.

5. Measured service：云系统通过利用与服务类型相适应的某种抽象级别的计量功能，自动控制和优化资源使用。Cloud systems automatically control and optimise resource use by leveraging a metering capability at some level of abstraction appropriate tthe type of service

3.云计算三种服务模型

SaaS, PaaS, IaaS

4.四种部署模型

private, community, public, hybrid

5. 虚拟化服务平台的关键技术是虚拟化，它的好处有哪些

- 服务器系统整合 Consolidation of server systems
- 降低成本 Reduced costs
- 降低复杂性 Reduced complexity
- 简化管理 Simplified administration
- 按使用付费 Pay-per-usage

6. Cloud Virtualised Infrastructures 是什么

一种新兴的计算范式，其中数据和服务驻留在可大规模扩展的数据中心，并且可以通过互联网从任何连接的设备随时随地访问。An emerging computing paradigm in which data and services reside in massively scalable data centers and are accessible anywhere, anytime, from any connected device via the Internet.

7.云计算的好处是什么

Technological benefits

1. 无限计算资源按需提供的幻觉，从而消除了云用户提前规划资源配置的需要。The illusion of infinite computing resources available on demand, thereby eliminating the need for cloud users tplan

ahead for resource provisioning

2. 消除了云用户的前期承诺，使他们能够从小规模开始，并在将来需要时增加硬件资源。The elimination of an up-front commitment by cloud users, thereby allowing them tstart small and increase hardware resources when needed in the future.

Economic benefits

3. 能够根据需要短期支付计算资源成本（例如，按小时支付处理器费用，按天支付存储费用），并在完成后释放这些成本，从而奖励资源节约。The ability tpay the costs of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them after done and thereby rewarding resource conservation.

8.云架构层次

Layer 1: Software as a Service

这是本地运行应用程序的替代方案。软件包可远程运行。用户可以通过互联网连接并使用基于云的应用程序，例如 Google 应用、Microsoft Office 365、电子邮件和日历。This is an alternative tlocally run applications. Software packages run remotely. Users connect tand use cloud-based apps overthe internet, e.g. Google apps, Microsoft Office 365, email, and calendars.

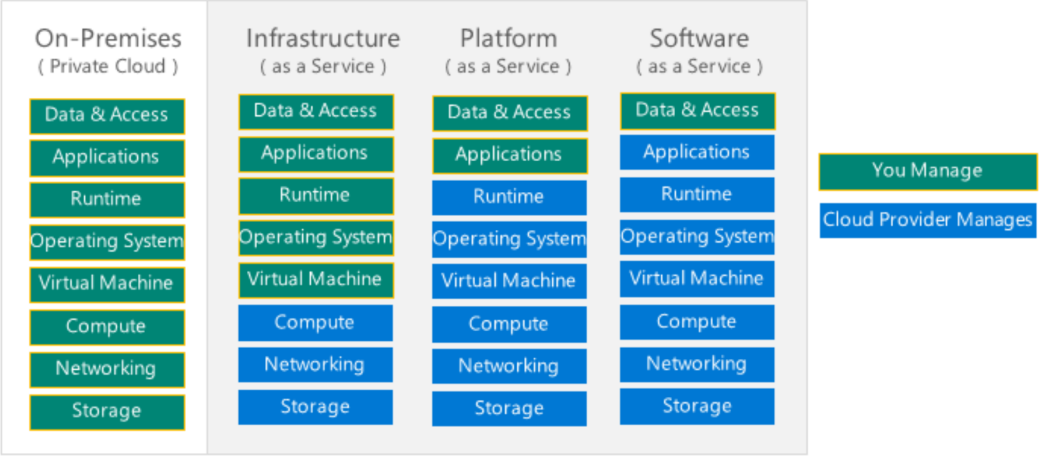
Layer 2: Platform as a Service

这是本地运行应用程序的替代方案。软件堆栈有助于应用程序的部署。提供构建、测试和部署软件应用程序的环境；无需专注于管理底层基础设施。This is an alternative tlocally run applications. Software stack facilitating the deployment of applications. Provides environment for building, testing, and deploying software applications; without focusing on managing underlying infrastructure.

Layer 3: Infrastructure as a Service

最终用户从云提供商访问服务器、虚拟机、存储、网络和操作系统。通过虚拟化，基础设施提供商能够拆分、分配并动态管理最终用户在这些系统上部署的服务。通过从云提供商（例如 Amazon EC2）租用服务器、虚拟机、存储、网络和操作系统，构建按需付费的 IT 基础设施。End-user access servers, virtual machines, storage, networks, and operating systems from a cloud provider. Through virtualization, infrastructure providers are able tsplit, assign and dynamically manage service provision that end-users will deploy on these systems.Build pay-as-you-gIT infrastructure by renting servers, virtual machines, storage, networks, and operating systems from a cloud provider, e.g. Amazon EC2

9.共享责任模型



10.云的类型有哪些

public cloud

- 由云服务或托管服务提供商拥有。Owned by cloud services or hosting provider.
- 为多个组织和用户提供资源和服务。Provides resources and services tmultiple organisations and users.
- 通过安全网络连接（通常通过互联网）访问。Accessed via secure network connection (typically over the internet).

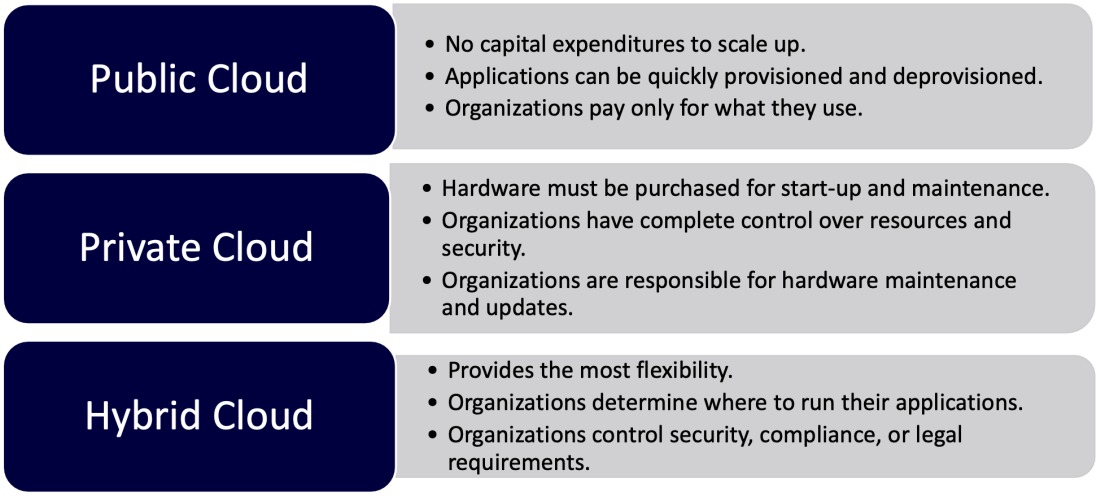
private cloud

- 组织在其数据中心创建云环境。Organizations create a cloud environment in their datacenter.
- 组织负责运营其提供的服务。Organization is responsible for operating the services they provide.
- 不向组织外部的用户提供访问权限。Does not provide access tusers outside of the organization.

hybrid cloud

- 结合公有云和私有云，使应用程序能够在最合适的位置运行。Combines Public and Private clouds tallow applications trun in the most appropriate location.

Cloud Model Comparison



二、云平台架构-Computing and Storage

1. Warehouse Scale Computing 是什么

提供互联网服务：搜索、社交网络、在线地图、视频分享、在线购物、电子邮件、云计算等。Provides Internet services: Search, social networking, online maps, videsharing, online shopping, email, cloud computing, etc.

与高性能计算（HPC）“集群”的区别：集群拥有更高性能的处理器和网络。集群强调线程级并行，而 WSC 强调请求级并行。Differences with HPC “clusters”: Clusters have higher performance processors and network. Clusters emphasize thread-level parallelism, WSCs emphasize request-level parallelism

数据中心：将不同的机器和软件整合到一个位置。强调虚拟机和硬件异构性，以便服务于不同的客户/应用。Datacenters: Consolidate different machines and software intone location. Emphasise virtual machines and hardware heterogeneity in order tserve various customers / applications.

2. WSC 的设计考虑。Design Considerations for Warehouse Scale Computing

- 性价比：积少成多 Cost-performance：Small savings add up
- 能效：影响配电和散热 Energy efficiency：Affects power distribution and cooling
- 通过冗余网络 I/实现可靠性 Dependability via redundancyNetwork I/O
- 交互式和批处理工作负载 Interactive and batch processing workloads
- 运营成本至关重要：功耗是系统设计的主要制约因素 Operational costs count：Power consumption is a primary constraint when designing system
- 规模及其机遇和问题 Scale and its opportunities and problems

3.异构性方面 Heterogeneity Aspects

Heterogeneity has emerged as one of the most profound and challenging characteristics of parallel environments.Levels are identified:

- 宏观层面 宏观层面：由多种节点架构（单核、多核）组成的分布式计算机网络（云），与潜在的异构网络互连 Macrlevel 宏观层面: networks of distributed computers (clouds), composed by diverse node architectures (single, multi-core), are interconnected with potentially heterogeneous networks
- 微观层面 微观层面：更深层的内存层次结构（主存储器、缓存、磁盘存储器、三级存储器）和各种加速器架构（固定式、可编程式，例如 GPU 和可重构式，例如 FPGA）Micrlevel 微观层面: deeper memory hierarchies (main, cache, disk storage, tertiary storage) and various accelerator architectures (fixed, programmable, e.g. GPUs, and reconfigurable, e.g. FPGAs)
- 其他：软件：操作系统、中间件、工具……Other: Software: OS, middleware, tools, …

4.计算机架构根据目的如何区分 Computer architectures – classified regarding purpose.

General Purpose Processors (GPP)

- 通用微处理器 - 通用计算机 General Purpose Microprocessors - general purpose computers
  - – 专为通用计算机（例如 PC、工作站、笔记本电脑、笔记本等）而设计。Designed for general purpose computers such as PCs, workstations, Laptops, notepads etc.
  - – 可执行多个应用程序和多项任务。Execute multiple applications and perform multiple tasks
- 微控制器 - 嵌入式系统 Microcontrollers - Embedded systems
  - – 专为嵌入式系统中的特定任务而设计。Designed for specific tasks in embedded systems
  - – 具有面向控制的外设。Have control oriented peripherals
  - – 片上集成 CPU，RAM、ROM 和 I/端口数量固定。Have on chip CPU, fixed amount of RAM, ROM, I/ports
  - – 低成本、低性能、低功耗，比微处理器更小。Low cost, low performance, low power, smaller than microprocessors
  - – 适用于对成本、功耗和空间要求严格的应用。Appropriate for applications in which cost, power and space are critical.

Application Specific Processors

- 数字信号处理器 Digital Signal Processor (DSPs)
  - Programmable microprocessor for extensive real-time mathematical computations – they support Multiply-accumulate units 用于大量实时数学计算的可编程微处理器 - 支持乘法累加单元

- 专用指令集处理器 Application Specific Instruction Set Processors (ASIPs)
    - Programmable microprocessor where hardware and instruction set are designed together for one special application 可编程微处理器，其硬件和指令集是为某一特定应用而设计的.指令集、微架构和/或内存系统针对特定应用或应用系列定制
        - – Instruction set, micrarchitecture and/or memory system are customised for an application or family of applications
        - – better performance, lower cost, and lower power consumption than GPP
- 专用集成电路 Application Specific Integrated Circuit (ASICs)
    - Algorithm completely implemented in hardware
        - IC designed for a specific line of a company
        - proprietary by nature and not (always) available tthe general public 例如，D1 是专为 Doj 超级计算机设计的 Tesla 定制 ASIC（处理能力达 362 万亿次浮点运算）。用于训练 Tesla 的驾驶辅助系统和自动驾驶 AI 模型

对比：

通用处理器可为所有不同应用提供良好的性能，而专用处理器则更适合特定任务。专用处理器应运而生，旨在实现更高的性能、更低的功耗和更低的成本。General Purpose Processors offer good performance for all different applications but specific purpose processors offer better for a specific task. Application specific processors emerged as a solution for – higher performance – lower power consumption – Lower cost

专用处理器已成为我们生活的一部分，几乎存在于我们日常使用的所有设备中，例如电视、手机和 GPS 等。Application Specific Processors have become a part of our life and can be found almost in every device we use on a daily basis – Devices such as TVs, mobile phones and GPSs

5. 加速器 Accelerators

为什么需要：Accelerators / co-processors are used tperform some functions more efficiently than the CPU – Faster – Lower energy

6. GPU 是什么 Graphics Processing Unit (GPU)

是现代高性能计算 (HPC) 领域加速计算工作负载的硬件之选。该平台支持机器学习 (ML)、深度学习 (DL) 和人工智能 (AI) 等工作负载。Are the choice of hardware taccelerate computational workloads in modern High-Performance Computing (HPC) landscapes. platform serving workloads like Machine Learning (ML), Deep Learning (DL), and Artificial Intelligence (AI).

单指令多线程 (SIMT) 模型 – CUDA / OpenCL 编程 Single Instruction Multiple 高效处理数据并行应用 – 数千个算术逻辑单元 Thread (SIMT) model – CUDA / OpenCL programming

Efficient for Data parallel applications – 1000's of Arithmetic Logic Units

CPU 和 GPU 对比：

CPU：通过使用大量的缓存层实现低延迟内存访问。如果数据不在缓存层 1-3 中，则从全局 DDR-4 内存中获取。CPU:low latency memory access by usingsignificant cache memory layers. If data not residing in caches layers 1-3.fetch from global DDR-4 memory

GPU：由多个处理器集群 (PC) 组成，每个集群包含多个流多处理器 (SM)。每个 SM 使用专用的第一层缓存和共享的第二层缓存，然后再从全局 GDDR-5（或较新的 GPU 型号中的 GDDR-6）内存中提取数据。该架构可容忍内存延迟。GPU：专注于吞吐量优化。比 CPU 拥有更多核心。并行处理任务。GPU: Consists of multiple Processor Clusters(PC) that contain multiple StreamingMultiprocessors (SM)Each SM uses a dedicated layer-1 cacheand a shared layer-2 cache before pullingdata from global GDDR-5 (or GDDR-6 innewer GPU models) memoryArchitecture is tolerant of memory latency.GPU: All about throughput optimisation A lot more cores than a CPU Parallel process a tasks

7.FPGA 是什么

FPGA (Field Programmable Gate Array) is an array of logic gates that can be hardware-programmed tfulfill user-specified tasks. Part of the application can be implemented entirely in HW. 是一种逻辑门阵列，可通过硬件编程来实现用户指定的任务。部分应用可以完全在硬件中实现。

- 结合可编程逻辑器件和门阵列的理念 Combine the idea of programmable logic devices and gate arrays
- 逻辑块阵列实现逻辑功能 Arrays of logic blocks implement logic functions
- 布线通道需要可编程布线 Need programmable wiring in routing channels
- 灵活的 I/接口逻辑与外部世界 Flexible I/interfacing logic toutside world

8.TPU 是什么

Custom Hardware: Google TPU

characteristics

- 仅有少数应用可在定制硬件（ASIC、FPGA、GPU）上运行 Only a few applications could run on custom H/W (ASIC, FPGA, GPUs)
- 另一种计算范式的广泛适用性：神经网络 (NN) – 可使数据中心的计算需求翻倍 Wide applicability of another computational paradigm: Neural Networks (NN)– could double the computation demands on datacenters
- Google Tensor Processing Unit – 性能高达每秒 100 千万亿次浮点运算 Google Tensor Processing Unit - Up t100 petaflops in performance
- 定制设计的机器学习 ASIC，为 Google 产品提供支持 – 翻译、照片、搜索、GmailCustom-designed machine learning ASIC that powers Google products–

Translate, Photos, Search, Gmail

- 非常适合使用机器学习框架（例如 TensorFlow、Pytorch）运行机器学习工作负载 Ideal trun machine learning workloads using machine learning frameworks, e.g. TensorFlow, Pytorch
- Currently a cloud service offering.

9.AWS 计算实例示例

Amazon Elastic Compute Cloud (EC2)

在云端提供安全、可调整大小计算容量的 Web 服务

Amazon EC2 Elastic Graphics：

通过网络将 (GPU) 图形加速附加到各种 EC2 实例

Amazon EC2 F1

配备 FPGA 的计算实例，用户可进行编程，为应用程序创建自定义硬件

10.Storage and Array Switch 是什么

存储和阵列交换机

存储选项

SAN：存储区域网络 (SAN)：使用光纤通道连接多个数据存储设备的本地网络

NAS：网络附加存储 (NAS)：专用硬件设备，

通常通过以太网连接到局域网，并操作数据文件

WSC 通常依赖于服务器中的本地磁盘。Google 文件系统 (GFS) 使用本地磁盘并维护至少三个副本。

三、云平台架构-Network

1. Which components does a data centre network consist of?

Load Balancer：接收外部请求，分配任务并返回结果。 Load Balancer: receives external requests, dispatches workloads, and returns results.

Router： Router:

• Loader Router：初步引导外部流量至数据中心边界。 Loader Router: preliminarily guides external traffic tthe data center boundary.

• Access Router：根据策略进一步分流或过滤。 Access Router: further routes or filters traffic based on policy.

Hierarchical Switches： Hierarchical Switches:

• Tier-1 Switch：核心层连接多个 Tier-2。 Tier-1 Switch: connects multiple Tier-2s at the core layer.

- Tier-2 Switch：聚合层连接多个 TOR。 Tier-2 Switch: connects multiple TORs at the aggregation layer.
- TOR Switch：机架顶层交换机，低延迟、高密度端口。 TOR Switch: Top-of-Rack switch, low latency and high port density.

Server Racks：计算节点通过 TOR 连接，支持横向扩展。 Server Racks: compute nodes connected via TOR, enabling horizontal scaling.

2. What are the advantages of rich interconnection among switches and racks?

- 更高吞吐量：支持多路径传输。 Higher throughput: supports multiple routing paths.
- 更强可靠性：实现冗余。 Greater reliability: enables redundancy.

3. What are the key requirements and trend of Ethernet?

趋势： Trends:

- 数据中心光网络正转向 400GE

Data center optics are transitioning t400GE.

- 接下来将推进至 800GE，甚至 1.6Tbps Moving towards 800GE and even 1.6Tbps.

关键技术需求： Key technology requirements:

- 新型硅片与电路，如宽带放大器与驱动器 New silicon and circuits, e.g., broadband amplifiers and drivers.
- 自适应数字均衡器 Adaptive digital equalizers.
- 标准的制定与实施 Development and ratification of standards.

4. What is InfiniBand?

一种软件定义、硬件加速的网络架构 A software-defined, hardware-accelerated network architecture.

将计算从 CPU 卸载至网络设备（或 DPU） Offloads computation from CPU tnetwork devices (or DPU).

- 优势包括： Advantages include:
- 支持在网络设备中进行计算处理（减轻主机负担） Supports in-network computing toffload host workload.
- 加快消息传递、压缩和自定义算法的处理速度 Accelerates message passing, compression, and user-defined algorithms.
- 适用于超算如 Summit（Top500.org） Used in supercomputers like Summit (Top500.org).

5. What components does an InfiniBand consist of?

Switch ASIC：专用集成电路 Switch ASIC: Application-Specific Integrated Circuit.

Network Interface ASIC Network Interface ASIC

DPU (Data Processing Unit)： DPU (Data Processing Unit):

- 智能网卡，含 CPU/GPU Smart NIC with onboard CPU/GPU.
- 支持虚拟网络与存储 Supports virtual networking and storage.
- 可运行安全软件 Can run security software.
- 是 InfiniBand 架构中新加入的关键部分 A key addition tthe InfiniBand stack.

6. What's next for InfiniBand?

当前速率：HDR（200G），EDR（100G），FDR（56G） Current speeds: HDR (200G), EDR (100G), FDR (56G).

新一代： Next generation:

- NDR（400G）：使用 Quantum-2 ASICs，64 端口共 25.6 Tbps NDR (400G): Quantum-2 ASICs, 64 ports totaling 25.6 Tbps.
- XDR（800G） XDR (800G)
- 目标：1.6 Tbps，总吞吐量达 102.4 Tbps Target: 1.6 Tbps, total throughput 102.4 Tbps.

7. What is the plane separation for SDN?

Data Plane：在设备内实现，负责数据包的转发 Data Plane: implemented in the device, responsible for packet forwarding.

Control Plane：集中式控制器实现，负责决策与管理 Control Plane: centralized controller, decision-making and management.

- 数据平面可根据 MAC/IP/VLAN 决定转发路径 The data plane forwards packets based on MAC/IP/VLAN.
- 控制平面运行如 OSPF、BGP 等算法，拥有全局视图 The control plane runs protocols like OSPF and BGP with a global view.

8. What components does SDN operation consist of?

- SDN Switches：如 Open vSwitch，包含 flow tables SDN Switches: e.g., Open vSwitch with flow tables.
- Controller：如 ONOS，提供网络抽象视图 Controller: e.g., ONOS providing network abstraction view.
- Applications：如 OpenFlow，实现路由、安全、负载等逻辑 Applications: e.g., OpenFlow implementing routing, security, load balancing.
- Network Control Protocol：将控制信号发送给交换机 Network Control Protocol: sends control signals tswitches.

9. What characteristics does SDN have?

- 集中式管理：跨数据中心统一控制 Centralized management: unified control

across data centers.

- 网络虚拟化：动态配置、快速部署 Network virtualization: dynamic configuration and fast deployment.
- 流量工程 & 负载均衡：优化资源利用，防止拥堵 Traffic engineering & load balancing: optimize usage, avoid congestion.
- 自动化编排：自动配置与调整网络结构 Automation & orchestration: auto-provision and adjust network structure.
- 安全性强：隔离不同负载，定义精细访问控制 High security: isolate workloads and define strict access policies.

10. List some use cases of SDN

- 数据中心网络优化：提升流量效率与可扩展性 Data center optimization: improves traffic flow and scalability.
- NFV（网络功能虚拟化）：替代硬件防火墙、负载均衡器等 NFV: virtualizes functions like firewalls, load balancers.
- 事件驱动型网络服务：如转码、异常检测等 Event-driven services: e.g., transcoding, anomaly detection.
- 快速调度平台：实现网络服务的低延迟响应 Fast task scheduling: enables low-latency network service response.

11. What is NFT and VNF?

VNF：虚拟化网络功能，如防火墙、IDS 等 VNF: virtualized network functions like firewalls, IDS.

NFV

- 支持按需部署、链式编排网络服务 Supports on-demand deployment and service chaining.
- 可通过容器运行，实现低延迟要求 Can be containerized tmeet low-latency requirements.
- 是推动 5G 应用落地的关键技术 Key enabler for 5G applications.

NFT 在此上下文中应指网络功能相关的“抽象模块”，与 serverless 架构紧密相关 NFT here refers tnetwork-function abstractions, related tserverless architecture.

四、虚拟化-Virtualization

1. 传统计算机和虚拟机有什么区别 What is the difference between Traditional Computer and Virtual Machines?

传统计算机在单一物理主机上运行一个操作系统和多个应用； A traditional computer runs one operating system and multiple applications on a single physical host.

虚拟机可以在同一物理机上通过 Hypervisor 同时运行多个不同操作系统和应用；

Virtual machines can run multiple different operating systems and applications on the same physical machine via a hypervisor.

虚拟机通过虚拟化层获得完整的模拟硬件环境。 Virtual machines get a fully simulated hardware environment through the virtualization layer.

2. 虚拟化有哪些优势 What are the advantages of virtualisation?

更好地利用资源 Better utilisation of resources

降低硬件成本 Reduction in hardware cost

提高故障容错能力 Improved fault tolerance

易于容量扩展 Easier capacity expansion

3. 虚拟化有哪些特点 What are the characteristics of virtual machines?

安全复用（Secure Multiplexing） Secure multiplexing

多个 VM 可在一台主机上运行（Strong Isolation） Run multiple VMs on a single host (strong isolation)

性能隔离（Performance Isolation） Performance isolation

整个 VM 可表示为一个文件（包含操作系统、应用、数据、内存状态等） The entire VM is represented as a file (includes OS, apps, data, memory state, etc.)

支持快照与克隆 Supports snapshots and clones

4. 典型的云架构应该包含哪些组件 What components should a typical cloud architecture contain?

Hypervisor Hypervisor

Virtual Infrastructure Manager Virtual Infrastructure Manager

Virtual Machine Cluster Virtual Machine Cluster

Distributed Applications Distributed Applications

5. Give me some virtual machine manager examples

VMware ESX / ESXi VMware ESX / ESXi

Xen Xen

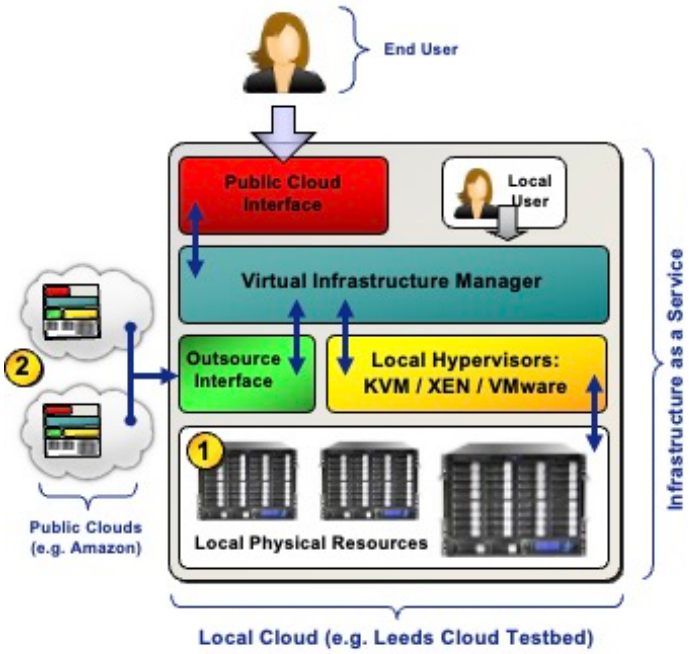
KVM KVM

6. 什么是虚拟基础架构管理器及其主要功能和示例 What is Virtual Infrastructure Manager and its main functions, examples?

功能包括：虚拟资源调度、创建、分配、迁移、终止 Functions include: virtual resource scheduling, creation, allocation, migration, termination

镜像加载、资源监控、用户访问控制 Image staging, resource monitoring, user access control

示例：OpenStack, OpenNebula Examples: OpenStack, OpenNebula



7. 虚拟基础架构管理器、虚拟机集群资源、分布式应用程序如何提供资源 How dVirtual Infrastructure Manager, Virtual Machine Cluster resources, Distributed Application provision resources?

Virtual Infrastructure Manager 从物理资源中通过 Hypervisor 提供虚拟资源; Virtual Infrastructure Manager provisions virtual resources through Hypervisor from physical resources.

Virtual Machine Cluster 由 VIM 管理，为分布式应用服务； Virtual Machine Cluster is managed by VIM and serves distributed applications.

Distributed Applications 部署在 VM 集群上，使用其资源完成计算任务。 Distributed Applications are deployed in VM clusters and use their resources tperform computation tasks.

8. 虚拟化的实现级别有哪些 What are the implementation levels of virtualisation?

应用级（Application） Application-level

进程级（Process API） Process-level (API)

操作系统级（OS Containers） Operating system level (OS containers)

硬件抽象层级（HAL） Hardware Abstraction Layer (HAL)

指令集架构级（ISA） Instruction Set Architecture (ISA)

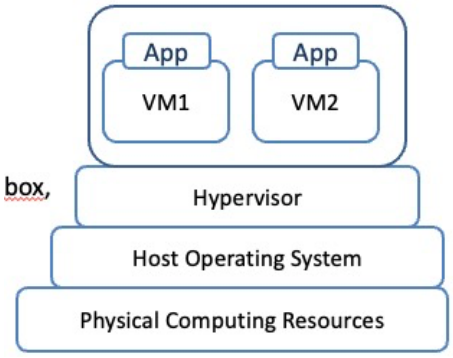
9. 什么是机器级虚拟化 What is Machine level virtualization?

在硬件层创建平台，允许多个虚拟机共享系统资源； Create a platform at the hardware level tallow multiple virtual machines tshare system resources.

取决于使用的 Hypervisor 类型。 The techniques depend on the type of hypervisor

used.

10. 机器级虚拟化的托管方法是什么 优点和缺点是什么 What is Hosted approach of Machine level virtualization? Pros and Cons?



特点：Hypervisor 运行于主机操作系统之上（Type 2） Feature: Hypervisor runs on top of the host operating system (Type 2).

优点： Pros:

易于设置和管理 Easy tset up and manage

兼容性强 Good compatibility

成本较低 Lower cost

灵活性高 High flexibility

缺点： Cons:

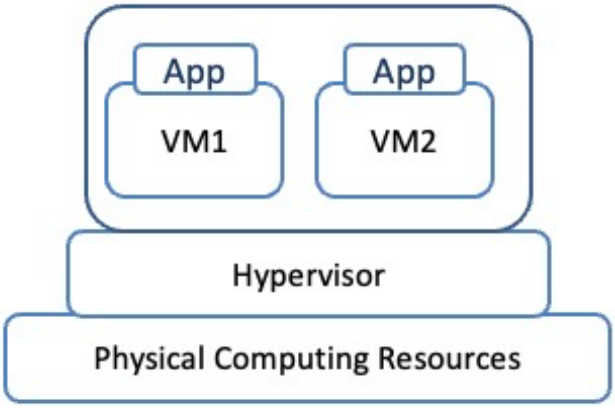
性能开销大 Performance overhead

安全性较低 Lower security

存在资源竞争问题 Resource contention issues

对底层硬件访问受限 Limited access tunderlying hardware

11. 机器级虚拟化的裸机方法是什么 优点和缺点是什么 What is Bare Metal approach of Machine level virtualization? Pros and Cons?



特点：Hypervisor 直接运行在物理硬件上（Type 1） Feature: Hypervisor runs

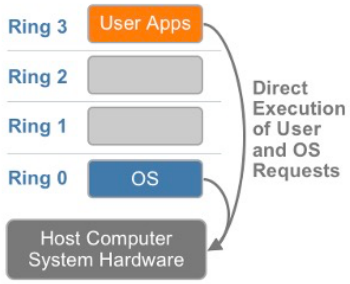
directly on physical hardware (Type 1).

示例：VMware ESX, Xen, KVM, Hyper-V Examples: VMware ESX, Xen, KVM, Hyper-V

优点：性能好、安全性高、硬件资源利用效率高 Pros: High performance, better security, and efficient hardware usage

缺点：部署复杂、成本较高 Cons: Complex deployment and higher cost

12. What is x86 Privileged Architecture?



x86 架构有 4 个特权环（Ring 0-3）： The x86 architecture has four privilege levels (Ring 0 t3):

Ring 0：操作系统内核 Ring 0: Operating system kernel

Ring 3：用户应用程序 Ring 3: User applications

在非虚拟化系统中，OS 在 Ring 0，用户程序在 Ring 3 In non-virtualized systems, the OS runs in Ring 0 and applications in Ring 3

13. 什么是基于虚拟机管理程序的虚拟化 What is Hypervisor-based Virtualisation?

Hypervisor 提供虚拟平台并管理虚拟机的执行； Hypervisor provides a virtual platform and manages virtual machine execution.

将底层硬件模拟呈现给来宾操作系统。 It presents simulated hardware tguest operating systems.

14. 什么是基于虚拟机管理程序的完全虚拟化 优点和缺点是什么 What is Full virtualization of Hypervisor-based Virtualisation? Pros and Cons?

特点： Characteristics:

无需修改 OS 即可运行（如 Windows、Linux） Unmodified OSs like Windows or Linux can run as guest OS.

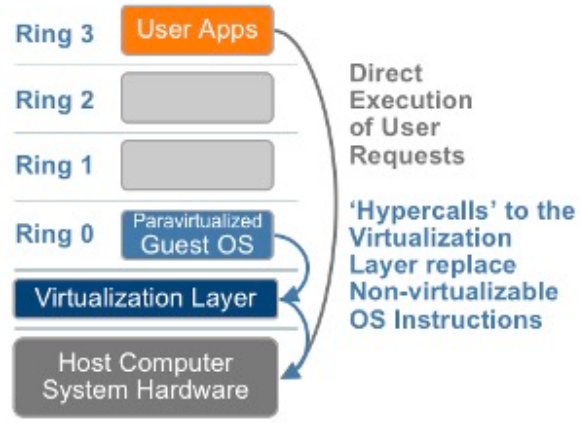
Hypervisor 全面模拟底层硬件 Hypervisor fully emulates underlying hardware.

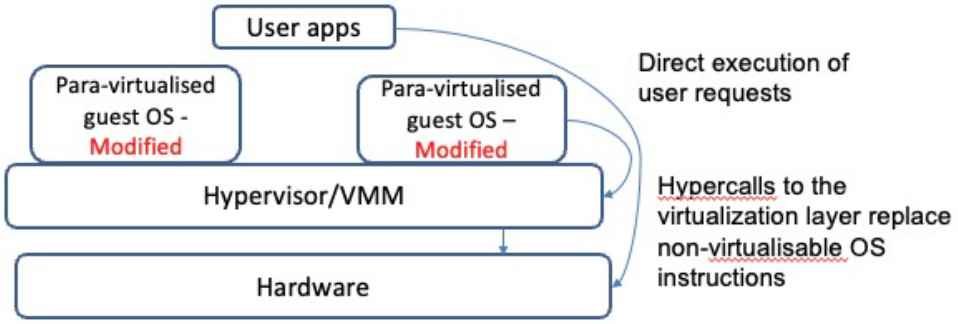
使用二进制翻译处理敏感指令 Uses binary translation for sensitive instructions.

优点：灵活性高，兼容性强 Pros: High flexibility and compatibility

缺点：性能开销大 Cons: Performance overhead

15. 什么是基于虚拟机管理程序的半虚拟化 优点和缺点是什么 What is Para-virtualization of Hypervisor-based Virtualisation? Pros and Cons?





特点：需要修改来宾 OS 使用 hypercall 替代敏感指令 Feature: Guest OS needs modification tuse hypercalls instead of sensitive instructions.

优点： Pros:

性能更高 Higher performance

Hypervisor 更简化 Simplified hypervisor

缺点： Cons:

兼容性差 Lower compatibility

维护成本高 High maintenance cost

16. 基于虚拟机管理程序的硬件辅助虚拟化是什么 优点和缺点是什么 What is Hardware-assisted of Hypervisor-based Virtualisation? Pros and Cons?

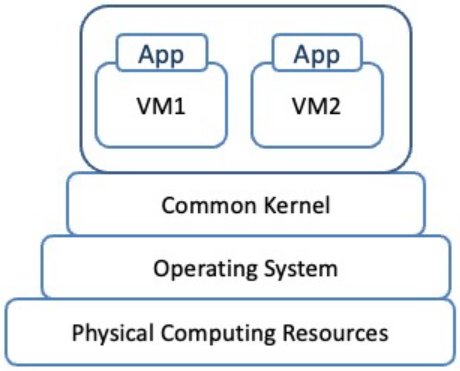
使用 Intel VT-x 或 AMD-V 提供 CPU 层级虚拟化支持 Uses Intel VT-x or AMD-V for CPU-level virtualization support.

特权调用由硬件处理，无需翻译或 hypercall Privileged instructions are handled by hardware, nneed for translation or hypercall.

优点：性能高，开销低 Pros: High performance, low overhead

缺点：依赖硬件支持 Cons: Depends on hardware support

17. 什么是操作系统级虚拟化 What is Operating System Level Virtualisation?



也称为系统级虚拟化 Alsknown as system-level virtualization

通过共享内核创建多个隔离容器 Creates multiple isolated containers using a shared OS kernel

容器具有相同主操作系统 Containers share the same parent operating system

示例：FreeBSD jail, Linux VServer, OpenVZ Examples: FreeBSD jail, Linux VServer, OpenVZ

优点：轻量级、资源消耗小 Advantages: Lightweight and low resource usage

五、容器化-Containerization

1. 虚拟机和容器有什么区别 What is the difference between virtual machine and container

虚拟机在硬件层进行虚拟化（Hypervisor），容器则在操作系统层进行虚拟化（共享内核）。 Virtual machines virtualize hardware (via hypervisor), while containers virtualize the operating system (shared kernel).

虚拟机启动慢（通常几分钟），容器启动快（几秒内）。 Virtual machines are slower tstart (minutes), containers start quickly (within seconds).

虚拟机占用更多资源，容器资源占用较少。 Virtual machines consume more resources, containers use less.

虚拟机镜像以 GB 为单位，容器镜像以 MB 为单位。 VM images are in GBs, while container images are in MBs.

每个虚拟机都需要完整操作系统，管理复杂；容器更易部署和扩展。 Each VM requires a full OS with complex management; containers are easier tdeploy and scale.

容器更适合 DevOps、微服务、批处理等轻量级场景。 Containers are better suited for DevOps, microservices, and batch computing.

2. 什么是容器管理平台 What is Container Management Platforms

也叫容器编排平台，如 Kubernetes、Docker Swarm。 Alsknown as container orchestration platforms, e.g., Kubernetes, Docker Swarm.

用于部署、监控、扩展和安全管理容器。 Used for deploying, monitoring, scaling, and securing containers.

管理员可以像管理虚拟机一样管理容器。 Allows administrators tmanage containers like VMs.

3. 这些术语的含义是什么：镜像、容器、容器运行时、注册表 What is the meaning of these terms: Image, Container, Container Runtime, Registry

Image：只读模板，定义容器如何创建。 Image: a read-only template defining how tcreate a container.

Container：镜像的运行实例。 Container: a running instance of an image.

Container Runtime：运行和管理容器的工具或服务。 Container Runtime: tool or service texecute and manage containers.

Registry：镜像仓库，用于存储和分发镜像。 Registry: a repository tstore and distribute images.

4. 什么是 Docker Engine What is Docker Engine

Docker 的运行时层，用于运行和管理容器。 Docker's runtime layer for running and managing containers.

由 Docker Daemon、Docker Client 和 REST API 构成。 Consists of Docker Daemon, Docker Client, and REST API.

5. Docker Engine 由什么组成 What does Docker Engine consist of

Docker Daemon：主机上运行，执行容器管理命令。 Docker Daemon: runs on the host texecute container management commands.

Docker Client：用户交互接口，发送指令给 Daemon。 Docker Client: user interface tsend instructions tthe Daemon.

REST API：远程访问 Daemon。 REST API: remote access tthe Docker Daemon.

6. What is Dockerfile

用于定义如何构建镜像的指令文件。 A script file defining instructions tbuild an image.

7. What is Docker Images

基于 Dockerfile 构建的只读模板。 Read-only templates built from Dockerfile.

包含应用及其运行所需依赖和启动指令。 Includes application, dependencies, and startup instructions.

8. What is Docker Compose

定义和运行多容器服务的工具。 A tool tdefine and run multi-container services.

使用 YAML 文件配置容器及参数。 Uses YAML files tconfigure containers and their options.

比 Kubernetes 更简单，适合小项目。 Simpler than Kubernetes, suitable for small projects.

9. What is Docker Hub

Docker 的公共镜像仓库。 Public image registry for Docker.

用户可上传（push）镜像供他人使用。 Users can push images for others tuse.

10. Unikernels 的架构是怎样的 What is the architecture of Unikernels

只实现最小的操作系统功能。 Implements only minimal OS functionality.

极轻量级，可高密度部署。 Extremely lightweight, allows high-density deployment.

用于短生命周期服务，几秒或更短即消亡。 Used for short-lived services lasting seconds or less.

只在有任务时启动，用完即停，最大化计算资源利用。 Start just-in-time, terminate immediately, maximize resource usage.

11. 虚拟机、容器和 Unikernels 之间有什么区别 What is the difference between virtual machine, container and Unikernels

虚拟机虚拟化硬件，容器虚拟化操作系统，Unikernel 没有完整 OS，仅运行单一功能。 VMs virtualize hardware, containers virtualize OS, Unikernels lack full OS and run only a single function.

启动速度：VM > Container > Unikernel。 Startup time: VM > Container > Unikernel.

资源占用：VM > Container > Unikernel。 Resource usage: VM > Container > Unikernel.

Unikernel 更适合边缘计算、函数计算等极致性能场景。 Unikernels suit edge computing and high-efficiency workloads.

12. 部署选项的权衡是什么 What are the trade-offs for Deployment Options

专用硬件：隔离强但成本高、弹性差。 Dedicated hardware: strong isolation but high cost and low elasticity.

虚拟机：兼容性好但资源开销大。 VMs: good compatibility but high resource overhead.

容器：效率高、易管理、是主流选择。 Containers: efficient, manageable, and mainstream.

Unikernel：资源利用率最高，但可移植性差、构建复杂。 Unikernel: best resource efficiency, but poor portability and complex tbuild.

六、云虚拟基础设施管理-Cloud Virtual Infrastructure Management

1. Why use a Virtual Infrastructure Manager (VIM)?

VIM 提供资源池的统一视图； VIM provides a unified view of the resource pool.

支持虚拟机的生命周期管理和监控； Supports life-cycle management and monitoring of virtual machines.

整合镜像、网络和虚拟化平台； Integrates image, network, and virtualization platforms.

自动选择放置 VM 的物理节点； Automatically schedules VM placement on physical nodes.

如 MAC 地址分配、桥接等网络配置； Handles networking setup such as MAC address assignment and bridging.

例如克隆和上下文配置实现自动化集群扩展与部署； Automates cluster deployment and scaling through cloning and contextualization.

避免手动操作，提升运维效率。 Avoids manual intervention and enhances operational efficiency.

2. What are the benefits of Virtualisation tClusters?

动态部署与重新调度虚拟机； Dynamic deployment and re-scheduling of virtual machines.

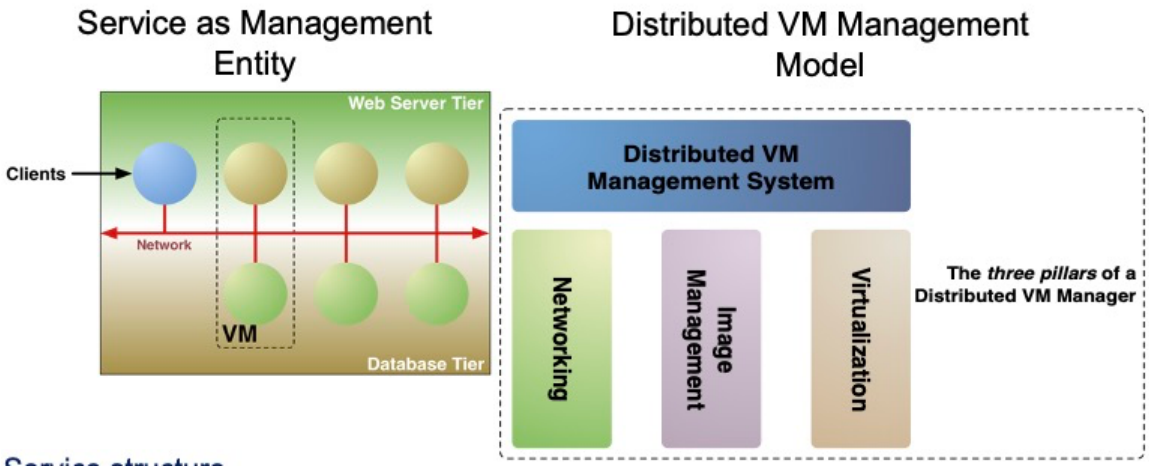
将僵化的物理基础设施转换为灵活、敏捷的虚拟平台； Transforms rigid physical infrastructure intflexible and agile virtual environments.

支持私有云/公有云混合部署； Supports hybrid private/public cloud deployment.

可根据负载变化动态调节资源； Dynamically adjusts resources based on workload fluctuations.

支持跨云部署（Cloud Interoperation）。 Enables cross-cloud deployment (Cloud Interoperation).

3. What does the Virtual Machine Management Model look like?



4. What are the Benefits of VIM for the Infrastructure Manager?

集中管理 VM 与底层资源； Centralized management of VMs and underlying resources.

支持 VM 放置策略，如负载均衡、服务器整合； Supports VM placement policies such as load balancing and server consolidation.

支持基础设施的动态扩展与调整； Enables dynamic resizing and adaptation of infrastructure.

支持集群的动态分区与隔离； Supports dynamic partitioning and isolation of clusters.

可结合本地资源与云资源，降低基础设施成本。 Combines local and remote resources treduce infrastructure expenses.

5. What are the main features of VIM (Example: OpenNebula)?

统一管理资源（统一视图）； Unified resource management (single view).

集成存储/网络/虚拟化技术； Integration of storage, networking, and virtualization technologies.

生命周期管理（VM 创建、迁移、终止）； Lifecycle management (VM creation, migration, termination).

灵活调度、镜像管理、集群隔离； Flexible scheduling, image management, and cluster isolation.

示例平台：OpenNebula。 Example platform: OpenNebula.

6. What are the main considerations for VIM?

存储方面，虚拟机镜像可作为文件或块设备； In terms of storage, VM images can be represented as files or block devices.

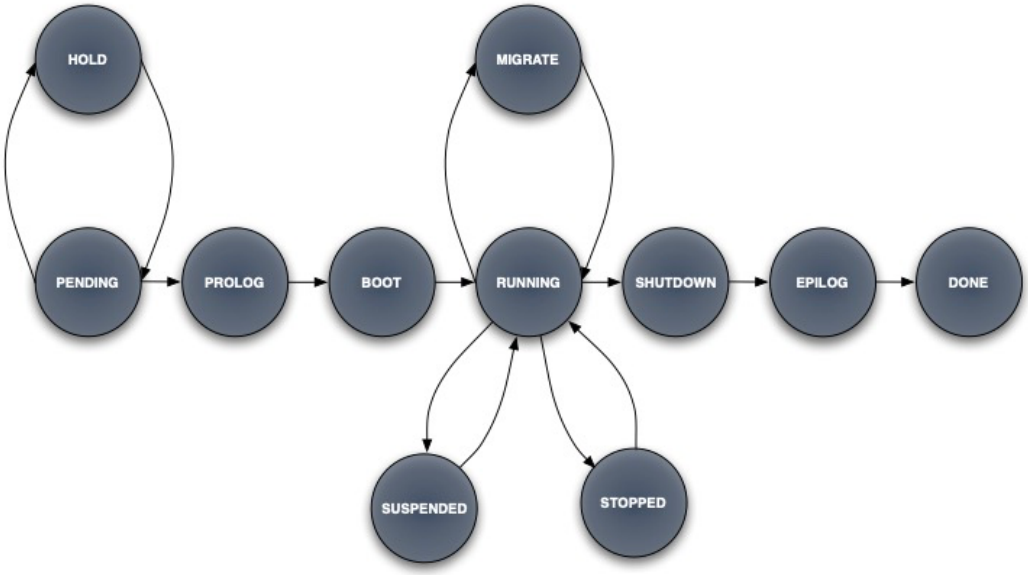
网络方面，管理操作常通过 SSH 进行； On the network side, management operations often use SSH.

VM 镜像可能很大，传输需考虑带宽； VM images can be large, requiring bandwidth consideration during transfer.

成千上万的 VM 对网络提出高要求； Thousands of VMs place high demands on networking.

需配置足够的 NICs 及桥接方式支持 VM 通信。 Adequate NICs and bridging setups are needed tsupport VM traffic.

7. Give me an overview of VM states



8. What is Virtual Infrastructure Manager OpenStack? What are the main capabilities of it?

OpenStack 是全球领先的开源云平台； OpenStack is the world's leading open-source cloud platform.

支持公有云与私有云部署； Supports both public and private cloud deployments.

目标是成为“云计算界的 Linux”； Aims tbecome the 'Linux' of cloud computing.

提供 API 接口及用户访问控制； Provides APIs and user access control.

支持云平台联邦和跨云资源部署； Supports cloud federation and cross-cloud deployments.

具有高度可扩展性和可用性； Highly scalable and highly available.

支持虚拟机热迁移（live migration）； Supports live migration of virtual machines.

支持实例的运行、重启、挂起、扩容、终止等操作； Supports run, reboot, suspend, resize, and terminate operations on instances.

支持基于角色的访问控制（RBAC）； Supports Role-Based Access Control (RBAC).

使用消息队列（如 RabbitMQ）进行通信。 Uses message queues (e.g., RabbitMQ) for communication.

9. Which seven “core” projects that form a complete IaaS solution for OpenStack?

Nova：计算资源管理（虚拟机生命周期）； Nova: compute resource management (VM lifecycle).

Neutron：网络管理（IP、MAC、子网、路由）； Neutron: network management (IP, MAC, subnets, routing).

Glance：镜像服务； Glance: image service.

Keystone：身份验证与访问控制； Keystone: authentication and access control.

Cinder：块存储服务（持久化磁盘）； Cinder: block storage service (persistent volumes).

Swift：对象存储服务； Swift: object storage service.

Horizon：Web 管理仪表盘。 Horizon: web-based management dashboard.

七、云资源管理和调度-Cloud Resource Management and Scheduling

1. 什么是资源管理和调度 What is Resource Management and Scheduling?

是计算系统的关键功能； A critical function of any computer system.

决定如何在用户和任务之间分配资源（如 CPU、内存、磁盘、网络）； Determines how tallocate resources such as CPU, memory, disk, and network among users and tasks.

影响系统的三个核心指标：功能性、性能、成本； Affects the system's functionality, performance, and cost.

包括策略和机制两部分： Consists of policies and mechanisms:

策略：指导资源分配决策的原则； Policy: principles that guide resource allocation decisions.

机制：实现这些策略的技术手段。 Mechanism: the means timplement the policies.

2. 云资源管理存在哪些问题 What is the Problem of Cloud Resource Management?

涉及多目标优化决策，策略复杂； Involves multi-objective optimization with complex policies.

云系统结构复杂，难以准确获取全局状态信息； The complexity of cloud systems makes global state information hard tobtain.

云环境中存在不可预测干扰（如故障、攻击）； Cloud environments are affected by unpredictable events like failures and attacks.

云服务面临负载波动，挑战其弹性能力。 Cloud services face fluctuating loads that challenge their elasticity.

3. 云资源管理策略有哪些 What are the Cloud Resource Management Policies?

Admission Control：拒绝违反高层策略的负载； Admission Control: prevents workload that violates high-level policies.

Capacity Allocation：为服务的每次激活分配资源； Capacity Allocation: allocates resources for each service activation.

Load Balancing：将负载均匀分配到各服务器； Load Balancing: evenly distributes workload across servers.

Energy Optimisation：最小化能耗； Energy Optimisation: minimizes energy consumption.

QoS Guarantees：根据 SLA 保证时延、能耗等性能指标。 QoS Guarantees: ensures timing and energy requirements per SLA.

4. 什么是云调度 What is Cloud Scheduling?

负责多层次资源共享，如主机 → VM → 应用 → 线程； Responsible for multi-level resource sharing: host → VM → application → thread.

不同调度目标： Different scheduling objectives:

批处理系统：最大吞吐、最小响应时间； Batch systems: maximize throughput, minimize turnaround time.

实时系统：满足时限要求； Real-time systems: meet deadlines.

最佳努力系统：分析类任务； Best effort systems: analytics and batch jobs.

常用调度算法：RR、FCFS、SJF、Priority。 Common scheduling algorithms: RR, FCFS, SJF, Priority.

5. OpenNebula 中的虚拟机调度是怎样的 What is VM scheduling like in OpenNebula?

调度器负责匹配等待中的 VM 与可用主机； Scheduler matches pending VMs with available hosts.

使用 Match-Making + Rank Policy： Uses Match-Making and Rank Policy:

优先选择最适合 VM 的主机资源。 Prioritizes resources most suitable for the VM.

6. 虚拟机调度方法有哪些 What are the VM Scheduling Approaches?

Power-aware：以最小能耗为目标，提高云商利润； Power-aware: aims tminimize energy tmaximize provider revenue.

Performance-aware：动态调整 VM 放置，保证服务质量； Performance-aware: maintains service levels through dynamic VM placement.

Network-aware：减少数据传输延迟，适用于数据密集型任务； Network-aware: reduces data transfer delay, suited for data-intensive tasks.

Heuristics（启发式方法）：如 FCFS、Greedy、Round Robin； Heuristics: e.g., FCFS, Greedy, Round Robin.

使用 First Fit 等方式选择第一个能承载 VM 的物理主机。 Uses First Fit tselect the first suitable physical host.

7. 虚拟化、迁移和调度之间的关系是什么 What is the relationship between Virtualisation, Migration and Scheduling?

虚拟化让资源管理和调度更灵活； Virtualization enables flexible resource

management and scheduling.

调度配合迁移策略可避免热点、优化资源使用； Scheduling with migration helps avoid hotspots and optimize usage.

但迁移本身也带来成本和能耗负担。 Migration introduces costs and energy overhead.

8. 什么是虚拟机迁移 What is VM Migration?

指将运行中的虚拟机从一个物理主机迁移到另一个； Means transferring a running VM from one host tanother.

VM 的内存、网络连接和存储状态被一并迁移； Transfers memory, network, and storage state together.

用于计划内维护、负载调整等场景； Used in planned maintenance and load adjustments.

实现不间断服务。 Ensures uninterrupted service.

9. 虚拟机迁移存在哪些问题 What are the problems of VM Migration?

何时迁移 主机过载或空载检测算法； When tmigrate? Host overload/underload detection algorithms.

迁移哪些 VM VM 选择算法； Which VMs tmigrate? VM selection algorithms.

迁移到哪里 VM 目标放置算法； Where tmigrate? VM placement algorithms.

涉及内存、存储和网络三方面的完整状态复制； Involves full state transfer of memory, storage, and network.

要保证迁移后的服务不中断。 Must ensure uninterrupted service post-migration.

10. 虚拟机迁移需要考虑哪些因素（网络、内存、磁盘） What need tbe considered for VM Migration (Network, Memory, Disk)?

Memory： Memory:

Pre-copy：先复制内存，再传 CPU 状态； Pre-copy: copy memory first, then CPU state.

Post-copy：先传 CPU 状态，再按需传内存页。 Post-copy: transfer CPU state first, then memory on-demand.

Network： Network:

局域网内：保留 IP，通过 ARP 广播更新地址； LAN: retain IP, update via ARP broadcast.

广域网：需 VPN、隧道、DNS 等机制支持； WAN: use VPNs, tunneling, DNS, etc.

Storage： Storage:

默认使用 SAN/NAS 跨主机共享磁盘； Defaults tSAN/NAS for cross-host storage.

若是本地磁盘迁移，时间开销最大； Local disk migration takes the most time.

可使用分布式复制策略如 DRBD。 Strategies like DRBD used for distributed replication.

11. 什么是纵向扩展和横向扩展 What is Scale Up and Scale Out?

Scale Up（纵向扩展）：升级现有虚拟机硬件配置； Scale Up: upgrade existing VM hardware.

如换更强的实例、提高内存/CPU； E.g., higher instance type, more memory/CPU.

Scale Out（横向扩展）：添加更多 VM 实例来分摊负载； Scale Out: add more VM instances tbalance load.

更适用于弹性应用与微服务架构。 Best suited for elastic apps and microservices.

八、容器管理-Kubernetes

1. Kubernetes 是什么

是一种容器编排平台，使用声明式语言部署容器； A container orchestration platform using declarative deployment.

起源于 Google 的开源项目，具有 10+ 年的容器管理经验； Originated from Google's open-source project with over 10 years of container management experience.

可用于启动、停止、更新和管理运行容器的集群； Used tstart, stop, update, and manage clusters running containers.

常被称作 Kube 或 k8s（k + 8 个字符 + s）； Often referred tas Kube or k8s (k + 8 characters + s).

支持应用的复制、扩展和升级，是现代 DevOps 与微服务核心技术。 Supports replication, scaling, and upgrading of applications; a core of modern DevOps and microservices.

2. Kubernetes 在云计算中的角色是什么

协调系统中容器的部署： Coordinates container deployment in the system:

哪些容器需要部署 Which containers need tbe deployed?

应部署到哪些节点 Twhich nodes should they be deployed?

不同容器间如何通信 How should containers communicate?

提高系统可靠性：持续监控并管理容器状态； Improves system reliability by continuously monitoring and managing container status.

自动扩缩容，响应负载变化； Supports automatic scaling in response tload changes.

管理基础设施资源，降低资源浪费。 Manages infrastructure resources treduce waste.

3. Kubernetes 中的 Cluster 和 Pod 的定义是什么

Cluster：由主节点与多个工作节点组成的协调单元； Cluster: a coordinated unit with master and multiple worker nodes.

Pod：最小部署单位，包含一个或多个共享网络与存储的容器。 Pod: the smallest deployment unit, containing one or more containers sharing network and storage.

4. Kubernetes 的架构是怎么样的

分为控制面（Control Plane）与节点（Node）； Divided intControl Plane and Nodes.

控制面负责调度和管理集群； Control Plane handles scheduling and management.

节点执行 Pod 的容器工作负载； Nodes run the containerized workloads.

使用 CNI/CRI/OCI 等标准接口； Uses standard interfaces like CNI, CRI, OCI.

所有组件通过 etcd 存储集群状态。 All components store cluster state in etcd.

5. Kubernetes Control Plane Components 有哪些

kube-apiserver：REST API 网关，负责身份验证、请求校验与权限控制； kube-apiserver: REST API gateway handling auth, validation, and RBAC.

etcd：分布式键值数据库，保存集群状态； etcd: distributed key-value store for cluster state.

kube-scheduler：将 Pod 分配到合适的节点； kube-scheduler: assigns Pods tappropriate nodes.

kube-controller-manager：管理所有控制循环。 kube-controller-manager: manages all control loops.

6. Kubernetes Node Components 有哪些

kubelet：节点代理，管理本地 Pod 生命周期； kubelet: node agent that manages local Pod lifecycle.

kube-proxy：配置网络规则，进行 TCP/UDP 转发； kube-proxy: configures network rules for TCP/UDP forwarding.

Container Runtime：容器运行时，负责拉取镜像、启动容器。 Container Runtime: pulls images and runs containers.

7. Kubernetes 中的 Deployment、Service、Namespace 分别是什么

Deployment：定义 Pod 模板和副本数量，实现滚动更新； Deployment: defines Pod templates and replica count, supports rolling updates.

Service：使用标签选择器匹配 Pod，提供稳定通信入口； Service: uses label selectors troute traffic tPods, provides stable endpoints.

Namespace：逻辑隔离机制，方便管理资源与权限。 Namespace: logical isolation mechanism for organizing resources and access.

8. Kubernetes 的 Pods 是什么

是 Kubernetes 中的最小工作单位； The smallest unit of work in Kubernetes.

可包含多个容器，共享网络与存储； Can include multiple containers sharing network and storage.

适用于容器间紧密协作场景。 Suitable for tightly coupled container applications.

9. Kubernetes 的 Node 是什么 Node 和 Pod 的关系是

Node 是运行 Pod 的计算资源，可为物理机或虚拟机； Node is the compute resource (physical or virtual) that runs Pods.

每个 Pod 都运行在某个 Node 上； Each Pod runs on a Node.

Node 可承载多个 Pod，由 Master 调度分配。 A Node can run multiple Pods and is scheduled by the Master.

10. Kubernetes 的 Kubelet 和 Container Runtime 的作用

kubelet：负责 Master 与 Node 的通信，执行 Pod 管理； kubelet: manages communication between Master and Node, and controls Pods.

Container Runtime：负责拉取镜像并启动容器。 Container Runtime: pulls container images and runs containers.

11. Kubernetes 中的 Pod 和 Container 的关系

一个 Pod 中可包含多个容器； A Pod can contain multiple containers.

容器共享 IP、端口和存储卷； Containers share IP, ports, and volumes.

可通过 localhost 相互通信。 Containers communicate with each other via localhost.

12. Pods 的 Scaling and Deployments 是什么

根据负载自动调整 Pod 数量； Automatically adjusts Pod count based on load.

通过 Deployment 管理版本发布； Uses Deployment tmanage version releases.

使用 Probes 检测健康状态； Uses Probes tmonitor health.

支持流量控制与渐进式发布。 Supports traffic control and progressive rollout.

13. Pods 中的 Horizontal Scaling 是怎么样的

周期性控制循环根据指标自动扩缩容； A periodic control loop automatically scales Pods based on metrics.

常用指标包括 CPU 使用率等； Typical metrics include CPU usage.

使用 ReplicaSet 管理副本数量； ReplicaSet manages the number of replicas.

Deployment 负责版本控制与滚动更新。 Deployment handles version control and rolling updates.

14. Pods 中的 Vertical Scaling 是怎么样的

使用 Vertical Pod Autoscaler (VPA) 动态调整资源； Uses VPA tdynamically adjust resource requests and limits.

包含 Recommender、Updater 和 Admission Controller； Includes Recommender, Updater, and Admission Controller.

实现自动化资源优化配置。 Automatically optimizes resource configuration.

15. Pods 中的 Cluster AutScaler 是怎么样的

自动调整节点数量（增加/减少 Node）； Automatically adjusts node count (adds/removes nodes).

配合 HPA 一起使用； Works together with HPA.

查找不可调度的 Pod 并扩容； Detects unschedulable Pods and scales accordingly.

将 Pod 合并至更少节点以支持缩容。 Consolidates Pods tfewer nodes for scaling down.

16. Pods 中的 ML-Based Scaling Metrics 和预测是怎么样的

使用 Forti 基准测试生成流量负载； Uses Fortifor benchmark traffic load.

Prometheus 收集 CPU、QPS、丢包率等指标； Prometheus collects CPU, QPS, packet loss metrics.

使用 ML 模型如 AR、LSTM、HTM、RL 预测未来负载； Uses ML models like AR, LSTM, HTM, RL for future load prediction.

预测所需 Pod 数量并自动扩缩容。 Predicts required Pod count and scales accordingly.

九、云配置管理-Cloud Configuration Management

1. What is PaaS? What is PaaS used for?

PaaS（Platform as a Service）是一种云中间件； PaaS (Platform as a Service) is a form of cloud middleware.

提供超出基础资源（如 VM、块存储）管理的服务； It provides services beyond basic resources like VMs and block storage.

通常以软件解决方案栈的形式交付，可整合其他 PaaS 和 IaaS 服务； Usually delivered as a software solution stack and integrates with other PaaS and IaaS services.

包括各种工具和/或库（如 API）。 Includes various tools and/or libraries such as APIs.

简化应用开发：通过抽象隐藏底层资源复杂性； Simplifies application development by abstracting underlying resource complexity.

简化应用部署：降低管理复杂度，提升部署效率。 Simplifies deployment by reducing management complexity and improving efficiency.

2. What is Infrastructure as Code (IaC)?

IaC 是一种通过代码方式管理和配置基础设施的方法； IaC is a method of managing and configuring infrastructure through code.

可以通过自动化代码来创建、修改、维护资源； Resources can be created, modified, and maintained through automated code.

典型应用于持续集成（CI）/持续部署（CD）流程中的团队协作； Typically used in CI/CD pipelines for team collaboration.

使基础设施具备版本控制、可追踪、可复现等特点； Enables version control, traceability, and reproducibility for infrastructure.

本质上将“基础设施”转变为软件工程的一部分。 Essentially turns infrastructure inta component of software engineering.

3. Give me some Modern IaC tools examples

现代基础设施即代码（IaC）工具包括： Modern Infrastructure as Code (IaC) tools include:

Terraform Terraform

TOSCA TOSCA

Ansible Ansible

Chef Chef

Puppet Puppet

这些工具可支持跨云资源的配置、部署和管理。 These tools support configuration, deployment, and management across cloud resources.

4. What are the Benefits of IaC?

跨多云平台的自动化； Automation across multi-cloud platforms.

快速高效地部署； Fast and efficient deployment.

可重复和一致性强； Highly repeatable and consistent.

可通过 Git 等工具实现源代码版本控制； Version control using tools like Git.

简化、标准化并易于扩展； Simplifies, standardizes, and enables scalability.

支持应用安全测试。 Supports application security testing.

5. What is the architecture of Terraform?

Terraform 是一种声明式资源编排工具，其架构包括以下核心组件： Terraform is a declarative resource orchestration tool with the following core components:

Core：用户操作接口，是 Terraform 的主要命令入口； Core: the user interaction interface and main command-line entry for Terraform.

Providers：插件模块，用于与各种服务和资源通信（如 AWS、Azure 等）； Providers: plugins that interact with various services and resources (e.g., AWS, Azure).

State File (JSON)：保存资源当前状态、依赖关系及待修改内容； State File (JSON): stores the current state of resources, dependencies, and changes.

Variables (.tfvars)：用于系统化管理参数变量； Variables (.tfvars): used tmanage configuration parameters systematically.

Instances：表示实际资源对象，支持创建、更新和删除。 Instances: represent actual resource objects that can be created, updated, or deleted.

Terraform 使用 HCL（HashiCorp Configuration Language），由 HashiCorp 公司维护。 Terraform uses HCL (HashiCorp Configuration Language) and is maintained by HashiCorp.

采用插件式架构，支持多种云平台。 It uses a plugin-based architecture and supports multiple cloud platforms.

十、无服务计算架构-Serverless Architecture

1. What is Serverless Computing and FaaS?

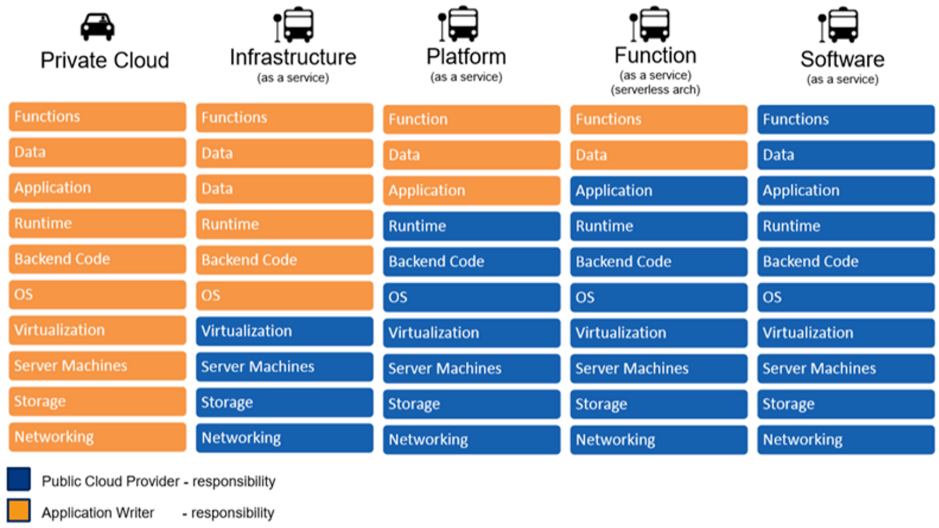
Serverless Computing 是一种让开发者无需管理服务器的计算模型； Serverless computing is a model that allows developers trun code without managing servers.

部署的单位不再是容器或服务器，而是纯粹的代码； Deployment units are not containers or servers, but pure code.

背后的服务器由云平台自动管理（如调度、伸缩、监控）； Servers are automatically managed by the cloud platform (e.g., scheduling, scaling, monitoring).

Function as a Service (FaaS) 是 Serverless 架构的核心实现机制； Function as a Service (FaaS) is the core execution model of serverless architecture.

Serverless 是软件架构概念，FaaS 是其具体的执行模型。 Serverless is an architectural concept; FaaS is its concrete execution model.

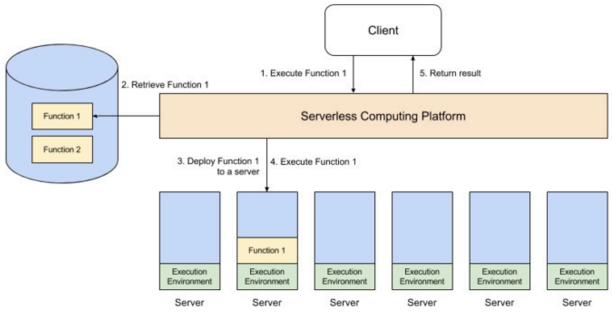


2. What is Serverless Computing Execution Model?

Serverless 平台会先检查函数是否在运行； The serverless platform first checks if the function is already running.

如果未运行，则从存储中加载函数并执行； If not, the function is loaded from storage and executed.

支持事件驱动、自动扩缩容，平台按需激活并调度函数运行。 Supports event-driven, auto-scaling execution; functions are invoked and managed on-demand.



3. What are the Benefits of Serverless Computing?

不需要管理服务器； Nserver management required.

自动、持续地伸缩； Automatic and continuous scaling.

动态资源分配，避免资源浪费； Dynamic resource allocation avoids waste.

按用量计费（pay-per-use）； Pay-per-use billing model.

适合响应式、事件驱动的应用场景。 Ideal for responsive and event-driven use cases.

4. Give me some popular Serverless Applications

单函数类应用： Single-function applications:

Alexa 语音技能、图像压缩等； Alexa voice skills, image compression, etc.

多函数类应用： Multi-function applications:

IoT 处理、图像识别、数据库分析； IoT processing, image recognition, database analysis.

新兴应用： Emerging applications:

并行处理、机器学习推理等。 Parallel processing, ML inference, etc.

5. What is the relation between microservices and serverless function?

微服务：服务被细化为多个小型功能单元； Microservices: services are decomposed intsmall functional units.

Serverless：进一步细化为单一函数（nanoservice）； Serverless: further refined intsingle-function nanoservices.

Serverless 函数常用于事件处理、后端逻辑、数据处理； Serverless functions are often used for event handling, backend logic, and data processing.

强调运行时的灵活组合与解耦。 Emphasizes flexible composition and decoupling at runtime.

6. What is cold start and warm start, what are the benefits of each?

Warm Start：函数所在容器仍在运行，响应迅速； Warm Start: function container is already running, responds quickly.

Cold Start：需要新建容器实例，初始化开销大； Cold Start: new container needs tbe initialized, causing delay.

Cold Start 的缺点是首次请求延迟高； Cold Start suffers from high latency on the first request.

Warm Start 快速响应，但需维持运行资源。 Warm Start offers fast response but requires resource upkeep.

7. Give me some examples of Commercial and Open Source Serverless Architecture Solutions

商用解决方案： Commercial solutions:

AWS Lambda AWS Lambda

Microsoft Azure Functions Microsoft Azure Functions

Google Cloud Functions Google Cloud Functions

IBM OpenWhisk IBM OpenWhisk

开源解决方案： Open-source solutions:Apache OpenWhisk, OpenFaaS, Knative , Fission, Iron.i

8. What is the architecture of OpenFaaS?

基于 Kubernetes 平台构建； Built on top of Kubernetes platform.

每个函数或服务作为 Docker 容器构建与部署； Each function or service is packaged as a Docker container.

使用组件： Uses components:

NATS：异步消息队列； NATS: asynchronous messaging queue.

Prometheus：用于监控与度量； Prometheus: for monitoring and metrics.

Linux：作为容器运行环境； Linux: as container runtime.

支持自动伸缩与事件驱动； Supports auto-scaling and event-driven invocation.

开发友好，支持 REST 接口与模板部署。 Developer-friendly, supports REST APIs and templated deployments.

9. What is the architecture of Knative?

Knative 架构由三大核心组件组成： Knative has three core components:

Knative Serving：使用 CRDs 管理服务，提供自动伸缩、路由与发布控制； Knative Serving: manages services via CRDs, provides auto-scaling, routing, and rollout.

Knative Eventing：提供事件源、Broker 与 Trigger 支持事件处理； Knative Eventing: enables event-driven capabilities via sources, brokers, and triggers.

Knative Build：从源码构建容器镜像。 Knative Build: builds container images from source code.

10. What are the features of serverless workflow?

可视化或编程方式定义工作流（DSL、YAML、JSON）； Workflows can be defined visually or via code (DSL, YAML, JSON).

支持并行执行任务； Supports parallel task execution.

支持异常处理与任务重试策略； Supports error handling and retry strategies.

提供自动伸缩能力； Provides auto-scaling capability.

支持通过 Web 界面、终端命令或事件触发执行； Supports triggering via web UI, terminal commands, or events.

示例平台：AWS Step Functions、Azure Durable Functions、Google Composer、Airflow、CNCF Workflow。 Example platforms: AWS Step Functions, Azure Durable Functions, Google Composer, Airflow, CNCF Workflow.

11. What are the serverless key metrics?

通信性能与函数组合模型：如 Sequence Chain、Nested Chain； Communication performance and function composition models: e.g., Sequence Chain, Nested Chain.

启动延迟（Cold vs Warm Start）； Startup latency (Cold vs Warm Start).

状态管理开销：显式状态需存储服务支持，隐式状态如缓存丢失影响性能； State management overhead: explicit states need storage support, implicit ones like cache may degrade.

资源效率： Resource efficiency:

平台层面：函数与任务协同调度提升资源利用； Platform level: cooperative scheduling improves utilization.

用户层面：资源配置与成本性能平衡能力。 User level: balance between resource provisioning and performance/cost.

十一、云编程前景-Cloud programming landscape

1. What is Cloud Service Lifecycle?

云服务生命周期包含以下阶段： The cloud service lifecycle consists of the following phases:

Cloud Application Design & Programming：使用编程模型开发服务逻辑； Cloud Application Design & Programming: develop service logic using programming models.

Service Construction：构建 VM 镜像和打包应用； Service Construction: build VM images and package applications.

Service Deployment：将服务部署到云平台； Service Deployment: deploy services tcloud platforms.

Service Operation：监控运行状况，自动扩缩容、维护等。 Service Operation: monitor, scale, and maintain service performance.

每个阶段都有配套的组件协同完成服务的交付与运行。 Each phase includes supporting components tenable service delivery and operation.

2. Framework Examples for Programming in the Cloud

以下框架和平台支持云端应用的开发与部署： These frameworks and platforms support cloud-based application development and deployment:

MPI（Message Passing Interface）：并行程序通信接口，用于高性能计算； MPI (Message Passing Interface): used for communication in parallel programs, common in HPC.

OpenMP（Open Multi-Processing）：基于线程的共享内存并行； OpenMP (Open Multi-Processing): thread-based shared memory parallelism.

MapReduce：适用于大数据处理的编程模型； MapReduce: a programming model for large-scale data processing.

TensorFlow：用于机器学习开发与训练的开源库； TensorFlow: an open-source library for machine learning development and training.

云平台 API：如 Azure、Google App Engine，提供构建云原生应用的 SDK/API。 Cloud platform APIs like Azure and Google App Engine provide SDKs for building cloud-native apps.

3. What is Cloud Service Construction?

云服务构建包括： Cloud service construction includes:

开发应用逻辑：使用 MPI、MapReduce 等模型实现； Developing application logic using models such as MPI and MapReduce.

构建支持服务的 VM 镜像（包括依赖环境）； Building VM images including dependencies.

指定并配置服务的运行要求（如容量、地理位置、能源效率）； Specifying and configuring runtime requirements such as capacity, location, energy efficiency.

与编程语言无关，可用 Java、Python、G 等； Language-agnostic, supporting Java, Python, Go, etc.

G 语言因并发能力和云原生兼容性被广泛应用。 Gis widely used for its concurrency and cloud-native compatibility.

4. Example 1: High Performance Computing and the Message-Passing Model

在多计算节点间通信需使用消息传递模型； Communication between compute nodes requires a message-passing model.

MPI（Message Passing Interface）是高性能计算标准； MPI is the standard for high-performance computing.

适用于在多个进程间发送数据，如 MPI_Send、MPI_Recv； Used for inter-process communication such as MPI_Send and MPI_Recv.

要求通信双方明确协调； Requires explicit coordination between communicating parties.

提供一对一与集体通信模式； Supports point-to-point and collective communication modes.

常用于超算、科学模拟。 Commonly used in supercomputing and scientific simulations.

5. What is OpenMP?

OpenMP 是用于共享内存并行编程的 API； OpenMP is an API for shared-memory parallel programming.

支持线程级并行（multi-threading），共享内存空间； Supports multi-threading with shared memory space.

通过编译指令、库函数、环境变量来控制并行； Uses compiler directives, library functions, and environment variables tcontrol parallelism.

支持并行 for 循环、并发任务创建； Supports parallel for-loops and concurrent task creation.

适用于单节点多核场景，作为 MPI 的替代方案。 Best suited for single-node multi-core environments and as an alternative tMPI.

6. Example 2: MapReduce - Scalable Data Processing on Large Clusters

适用于大规模集群数据处理的编程模型； A programming model for large-scale data processing on clusters.

Map 函数：将输入数据映射为中间的 key/value 对； Map function: transforms input intintermediate key/value pairs.

Reduce 函数：聚合所有相同 key 的值； Reduce function: aggregates values for the same key.

广泛用于 Web 搜索、日志分析等场景； Widely used in web search and log analysis.

提供数据并行性与容错机制。 Provides data parallelism and fault tolerance.

7. What is Hadoop and its main features?

Hadoop 是 Yaho 发起的开源大数据平台； Hadoop is an open-source big data platform initiated by Yahoo.

支持在大规模集群上运行分布式应用。 Supports distributed applications across large clusters.

可扩展（Scalable）：可扩展到 PB 级数据； Scalable: scales up tpetabyte-level data.

经济（Economical）：开源平台，降低大规模计算成本； Economical: open-source platform reducing large-scale computing cost.

高效（Efficient）：任务并行调度，数据本地处理； Efficient: parallel task scheduling and data locality.

可靠（Reliable）：支持副本冗余，节点故障自动恢复。 Reliable: supports replication and automatic node recovery.

8. What is TensorFlow?

TensorFlow 是 Google Brain 团队开发的开源机器学习库； TensorFlow is an open-source ML library developed by Google Brain.

通过数据流图表达计算过程； Represents computations as dataflow graphs.

图的节点为运算（operations），边为张量（tensor）； Nodes represent operations, edges represent tensors.

支持神经网络、回归、分类等任务； Supports tasks such as neural networks, regression, classification.

图模型便于在分布式环境下运行。 Graph-based model facilitates distributed execution.

10. What is AWS EC2, S3, EBS, AMI?

EC2（Elastic Compute Cloud）：提供弹性可扩展的云计算资源（虚拟机）； EC2 (Elastic Compute Cloud): provides scalable virtual compute instances.

S3（Simple Storage Service）：对象存储服务，支持海量数据访问； S3 (Simple Storage Service): object storage for massive data access.

EBS（Elastic Block Store）：为 EC2 提供持久块存储，自动镜像冗余； EBS (Elastic Block Store): persistent block storage with redundancy for EC2.

AMI（Amazon Machine Image）：EC2 虚拟机模板，包含操作系统与用户软件。
AMI (Amazon Machine Image): VM image including OS and pre-installed software.

十二、大数据概念-Concepts

1. What are the different types of data?

数据的类型包括： Types of data include:

结构化数据（Relational Data）：如数据库中的表、事务、传统系统； Structured data (Relational Data): such as tables, transactions, and legacy systems.

文本数据（Text Data）：如网页中的文本信息； Text data: such as text from websites.

半结构化数据（Semi-structured Data）：如 XML 格式； Semi-structured data: such as XML.

图数据（Graph Data）：如社交网络、语义网络； Graph data: such as social networks and semantic webs.

流数据（Streaming Data）：只能被扫描一次的数据，如实时传感器数据。
Streaming data: data that can only be scanned once, e.g., real-time sensor feeds.

这些数据往往具备以下特点： These data typically have the following characteristics:

数据量大（Large）； Large volume;

结构复杂（Complex）； Complex structure;

多为半结构化（Semi-structured）。 Mostly semi-structured.

2. What is Business Intelligence?

商业智能（Business Intelligence，BI）是指从大量数据中提取可操作的业务洞察，
Business Intelligence (BI) refers textracting actionable insights from large volumes of data,

常用于： Commonly used for:

分析用户行为模式； Analyzing user behavior patterns;

评估系统中各组件的经济效率； Assessing economic efficiency of system components;

发现资源浪费的环节； Identifying areas of resource waste;

预测业务增长对系统的影响； Predicting impact of business growth on systems;

识别流行应用及其使用时段； Identifying popular applications and their usage periods;

支持环境监测、科学探索、财务决策等。 Supporting environmental monitoring, scientific research, and financial decisions.

BI 也包含数据的： BI alsincludes data:

实时可视化； Real-time visualization;

模式识别与建模； Pattern recognition and modeling;

深度学习与统计分析。 Deep learning and statistical analysis.

3. What is Big Data?

大数据是指能够从收集来的数据中及时地、可扩展地管理、分析、可视化和发现知识的能力。 Big Data refers tthe ability tmanage, analyze, visualize, and discover knowledge from data in a timely and scalable manner.

不仅仅是数据体量大，它也是一种新兴趋势，强调从数据中“做出有益之事（doing something beneficial）”。 It's not just about data volume; it's a new trend focused on doing something beneficial with the data.

大数据的实际应用示例包括： Examples of Big Data in practice include:

Facebook 每日上传数据； Daily uploads tFacebook;

美国国会图书馆的数据总量； Data stored by the US Library of Congress;

全球每日人类数据生产。 The total amount of data generated by humanity daily.

4. What are the five V of big data?

五个 V 特征（The Five V's）是描述大数据本质的五个维度： The Five V's describe the essential dimensions of Big Data:

Volume（体量）：数据规模巨大； Volume: massive scale of data;

Velocity（速度）：数据生成与处理速度快； Velocity: high speed of data generation and processing;

Variety（多样性）：数据类型繁杂（结构化/非结构化/半结构化）； Variety: data comes in many forms (structured, unstructured, semi-structured);

Veracity（真实性）：数据准确性和可信度有待确认； Veracity: data accuracy and trustworthiness are variable;

Value（价值）：是否能从数据中提取有用的信息。 Value: whether the data yields useful insights.

5. What are the popular Big Data Tools & Techniques?

流行的大数据技术与方法包括： Popular Big Data tools and techniques include:

数据操作与分析（Data Manipulation & Analysis）：执行计算并提出明确的问题；Data manipulation and analysis: performing computation and formulating precise questions;

数据挖掘（Data Mining）：从数据中发现模式与规律； Data mining: discovering patterns and relationships in data;

机器学习（Machine Learning）：推断缺失值、预测未来数据； Machine learning: inferring missing data and predicting future trends;

数据可视化（Data Visualization）：用图表展现数据； Data visualization: representing data visually;

推荐使用 Tableau 学生免费版。 Tableau Student Edition is recommended.

数据收集与预处理（Data Collection & Preparation）：补全缺失值、移除异常值、格式化数据等。 Data collection and preparation: filling missing values, removing outliers, formatting data, etc.

十三、大数据架构

1. What are the techniques towards big data?

大数据技术的核心在于利用分布式资源，以应对其 Volume、Velocity、Variety、Veracity、Value 五大特征。 The core of big data technology lies in leveraging distributed resources thandle the five V's: Volume, Velocity, Variety, Veracity, and Value.

主要技术手段包括： Main technical approaches include:

大规模并行计算（Massive Parallelism） Massive parallel computation

数据存储与分发（Huge volumes, distributed storage） Data storage and distribution (huge volumes, distributed storage)

高速网络（High-speed networking） High-speed networking

高性能计算（High-performance computing） High-performance computing

任务/线程管理、数据挖掘与分析 Task/thread management, data mining and analysis

机器学习与数据可视化 Machine learning and data visualization

涉及硬件基础设施： Involves hardware infrastructure:

多节点 CPU/GPU、FPGA、ASIC、内存、存储与网络资源。 Multi-node CPU/GPU, FPGA, ASIC, memory, storage, and network resources.

方法包括： Methods include:

任务并行化（Task Parallelisation） Task parallelisation

数据并行化（Data Parallelisation） Data parallelisation

2. What is the layered architecture overview - technical?

技术架构层次包括以下四层： The technical architecture includes the following four

layers:

1. Data Sources：所有可分析的数据源，来自多通道、格式、速度、内部/外部来源；
1. Data Sources: All analyzable data sources from various channels, formats, speeds, internal/external origins;

2. Data Acquisition & Storage：采集并转换数据，适配分析，存储至 digest 或其他指定位置； 2. Data Acquisition & Storage: Acquiring and transforming data for analysis, storing tdigest or other targets;

3. Analytics Layer：使用算法与工具分析数据，产生洞察； 3. Analytics Layer: Analyze data using tools and algorithms tgenerate insights;

4. Consumption Layer：消费分析结果，如可视化、人类用户、服务、业务流程等。
4. Consumption Layer: Consumes the insights, e.g., via visualization, human users, services, workflows.

这些层是逻辑层，不一定对应物理设备。 These layers are logical and dnot necessarily map tphysical infrastructure.

3. What is the layered architecture overview - business?

业务架构层关注企业管理与运营，主要包括： The business architecture layer focuses on enterprise management and operations, including:

Information Integration：数据源连接与整合，使用通信协议、API、Web 服务等；
Information Integration: Connecting and integrating data sources using protocols, APIs, web services, etc.;

Big Data Governance：数据法务、质量管理，满足 GDPR 法规，包含合法性、公平性、透明性等； Big Data Governance: Legal compliance and quality management aligned with GDPR—legality, fairness, transparency, etc.;

Systems Management：性能、安全性、可依赖性等非功能需求； Systems Management: Non-functional aspects like performance, security, reliability;

Quality of Service：服务的稳定性与可靠性保障。 Quality of Service: Ensuring service stability and reliability.

4. What are the Big Data Challenges?

大数据面临以下挑战： Big data faces the following challenges:

数据体量巨大，增长速度快，数据类型多样； Enormous data volume, rapid growth rate, and diverse data types;

元数据与语义描述不足，难以搜索与识别； Lack of metadata and semantic tags makes it hard tsearch and identify;

存储与保留问题：数据重建比存储更可行； Storage and retention issues: recomputing data is sometimes more feasible than storing it;

信噪比低：如何识别有用信息； Low signal-to-noise ratio: distinguishing valuable information is difficult;

监管与采集：数据收集策略与权限问题； Governance and collection: strategies and permissions around data gathering;

质量控制：需验证数据的可信度与完整性。 Quality control: requires validation of data credibility and integrity.

5. What is parallelization and challenges for parallelization?

并行化是大数据计算的重要解决思路。 Parallelization is a key solution for big data computing.

基本流程：分区任务 → 分配给多个 Worker 并行执行 → 组合结果。 Basic steps: partition tasks → assign tworkers in parallel → combine results.

挑战包括： Challenges include:

调度：任务如何分配 Scheduling: how tassign tasks?

资源可用性：Worker 数量不足怎么办 Resource availability: what if there aren't enough workers?

依赖性：Worker 之间如何共享结果 Dependencies: how dworkers share results?

容错：Worker 中断后如何恢复 Fault tolerance: how trecover from worker failures?

通信与同步：共享状态、数据访问如何协调 Communication and synchronization: how tcoordinate shared state and data access?

难点在于并发控制、分布式服务交互、故障管理等。 The difficulty lies in concurrency control, distributed service interaction, and fault management.

十四、MapReduce

MapReduce and DFS - Bilingual Summary

1. What is MapReduce?

MapReduce 是一种编程模型，用于在大规模系统中表达分布式计算； MapReduce is a programming model used texpress distributed computation in large-scale systems.

它也是一个执行框架，可以在普通硬件上可靠、容错地运行； It is alsan execution framework that runs reliably and with fault tolerance on commodity hardware.

最初由 Google 开发，后来开源版本为 Hadoop MapReduce； Originally developed by Google, it was later open-sourced as Hadoop MapReduce.

MapReduce 的思想核心是：将任务分解成 map 和 reduce 两部分，分别并行处理和聚合结果； The core idea of MapReduce is tdivide tasks inta map and a reduce phase tbe processed and aggregated in parallel.

它支持可扩展、大数据量的处理任务。 It supports scalable processing of large-scale data workloads.

2. What is MapReduce process?

MapReduce 过程包括以下步骤： The MapReduce process includes the following steps:

Map 阶段：遍历大量记录，提取感兴趣的信息（生成中间键值对 <k', v'>）； Map Phase: Scans large records and extracts interesting inftgenerate intermediate key-value pairs <k', v'>;

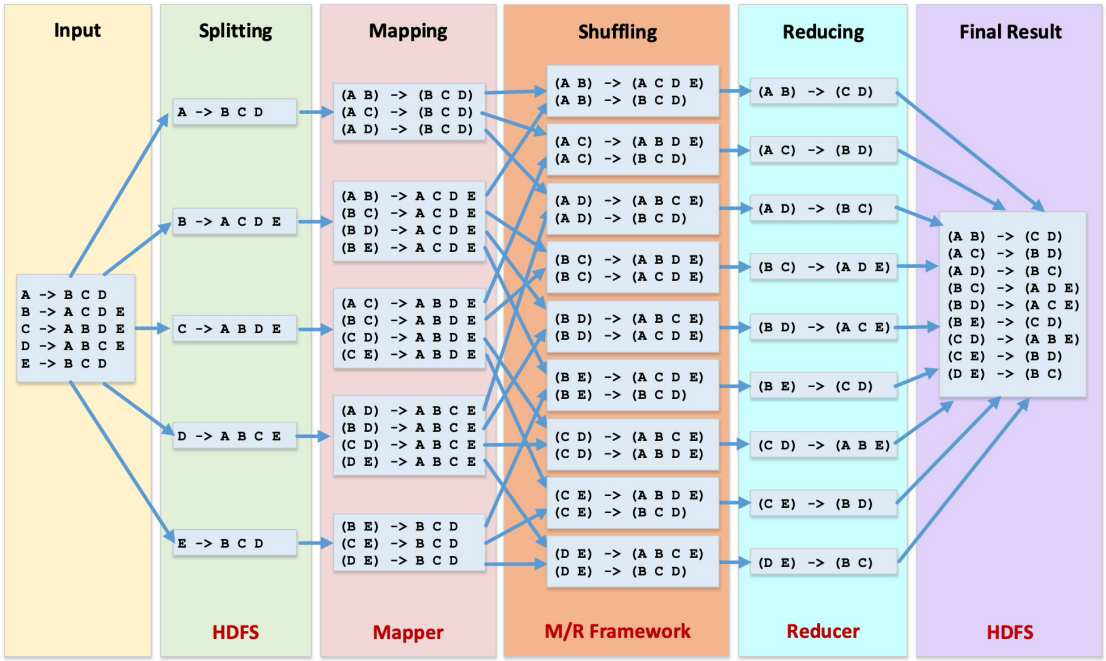
Shuffle & Sort 阶段：对所有中间结果按 key 聚合、排序，将同一个 key 的值发送到相同的 reduce worker； Shuffle & Sort Phase: Aggregates and sorts intermediate results by key and sends all values of the same key tone reduce worker;

Reduce 阶段：汇总、聚合中间结果，生成最终输出。 Reduce Phase: Aggregates the intermediate results tproduce final output.

示例：统计一批文档中每个单词出现次数： Example: Counting word occurrences in a collection of documents:

map(w) => (w, 1) map(w) => (w, 1)

reduce(w, [1,1,1,...]) => (w, count) reduce(w, [1,1,1,...]) => (w, count)



（1） MapReduce 的最终输出是由 reducer 决定的，即为 r 个 reducer 的输出。所以最终输出的 job 数应该是 r，不是 m（m 是 mapper 数量）。

（2） 在 MapReduce 中，Reducer 的 reduce 方法不会立即启动。它会等所有 mapper 完成并完成 shuffle + sort 阶段后，才开始执行 reduce()。

（3） 同一个 key 的所有 values 被送到同一个 reducer 进行处理。

（4） 在 shuffle 阶段，每个 mapper 会将输出分区为 r 份，分别发送给 r 个 reducer;

（5） 所以会产生 m × r 个 copy 操作。

（6） 每个 mapper 处理的输入块大小可以不同；并且输出的 key/value 数量取决于映射逻辑，不一定与输入数相同。

3. What will MapReduce Runtime do?

MapReduce 的运行时系统主要负责以下任务： The MapReduce runtime system is responsible for the following tasks:

调度任务：为 map 和 reduce 分配 worker； Task scheduling: Assigning workers tmap and reduce tasks;

数据分布：将计算移动到数据位置（数据本地化）； Data distribution: Moving computation twhere the data resides (data locality);

同步控制：收集、排序、shuffle 中间数据； Synchronization: Collecting, sorting, and shuffling intermediate data;

错误与容错处理：检测失败 worker 并重启； Fault detection and recovery: Detecting failed workers and restarting them;

分区与组合器支持： Partitioning and combiner support:

partition(k', n) => 分区号：决定哪组数据送往哪个 reducer； partition(k', n) => partition number: determines which reducer receives which key group;

combine(k', v')：在 map 阶段后进行本地聚合（减少网络开销）。 combine(k', v'): performs local aggregation after the map phase treduce network overhead.

MapReduce 框架大大简化了编程人员的工作，只需实现 map() 和 reduce() 函数，其余由系统处理。 The MapReduce framework simplifies developer work—only map() and reduce() functions need tbe implemented; the rest is handled by the system.

4. What is DFS and the importance of the underlying distributed file system?

DFS（分布式文件系统）是 MapReduce 能高效工作的关键基础。 DFS (Distributed File System) is the key foundation for efficient MapReduce operation.

主要作用与优势包括： Main roles and advantages include:

将数据保存在集群每个节点的本地磁盘上； Storing data on the local disks of each node in the cluster;

数据不动，任务移动：把任务分派到存有数据的节点上（“将计算移至数据”）； Moving tasks tthe data rather than moving data—'bringing computation tthe data';

解决内存限制问题：因为所有数据无法完全装入内存； Solving memory limitation issues—data can't all fit in memory;

避免带宽瓶颈：本地读取比跨网络传输更高效； Avoiding bandwidth bottlenecks—local reads are more efficient than remote transfers;

支持高吞吐的磁盘访问。 Supports high-throughput disk access.

常见实现包括： Common implementations include:

Google 的 GFS（Google File System）； GFS (Google File System) by Google;

Hadoop 的 HDFS（Hadoop Distributed File System）。 HDFS (Hadoop Distributed File System) in Hadoop.

DFS 使得 MapReduce 可以处理 PB 级别的海量数据。 DFS enables MapReduce thandle petabyte-scale data.

十五、Hadoop and Spark

1. What are the characteristics of MapReduce?

适合子任务间无依赖的计算问题； Suitable for computation problems with ndependency between subtasks;

系统自动负责任务调度、监控和失败重启； System automatically handles scheduling, monitoring, and fault recovery;

reduce 阶段在所有 mapper 完成后才开始； The reduce phase begins only after all mappers complete;

map 输出和 reduce 输入输出都是 <key, value> 形式； Map output and reduce input/output are all in <key, value> format;

所有数据元素不可变（immutable）； All data elements are immutable;

Hadoop 中 MapReduce 对机架和 HDFS 结构感知。 MapReduce in Hadoop is aware of rack and HDFS structure.

2. What is the main components of Hadoop?

HDFS（Hadoop Distributed File System）：分布式文件系统； HDFS (Hadoop Distributed File System): a distributed file system;

MapReduce 引擎：分布式计算框架； MapReduce engine: distributed computing framework;

YARN（Yet Another Resource Negotiator）：资源调度与任务协调； YARN (Yet Another Resource Negotiator): resource scheduling and task coordination;

Hadoop Common：支撑工具和 API。 Hadoop Common: supporting tools and APIs.

3. What is the MapReduce in Hadoop?

客户端提交 MapReduce 任务至 JobTracker； Client submits MapReduce job tJobTracker;

JobTracker 与 NameNode 通信，确定数据所在 DataNode； JobTracker communicates with NameNode tlocate data on DataNode;

JobTracker 向 DataNode 上的 TaskTracker 分发 Map 代码； JobTracker dispatches Map code tTaskTrackers on DataNodes;

TaskTracker 执行 Map 任务并上传心跳与进度； TaskTracker executes Map tasks and reports heartbeat and progress;

Map 任务完成后生成中间数据，传输给 Reduce； Map tasks produce intermediate data tbe sent tReduce tasks;

Reduce 任务从所有 Map 获取数据，最终输出写入 HDFS； Reduce tasks fetch data from all mappers and write final output tHDFS;

使用 Rack Awareness 优化本地化调度与容错处理。 Uses Rack Awareness toptimize locality and fault tolerance.

4. What are the limitations of the original Hadoop?

资源管理与 MapReduce 强耦合； Resource management is tightly coupled with MapReduce;

JobTracker 是单点瓶颈，限制可扩展性； JobTracker is a single point of bottleneck and limits scalability;

Map/Reduce Slot 固定，资源利用率不高； Fixed Map/Reduce slots result in poor resource utilization;

NameNode 内存中保存全部元数据，支持文件数有限； NameNode stores all metadata in memory, limiting file support;

整体扩展性差，如 Yaho 实测最大 5000 个节点、4 万任务并发。 Poor scalability, e.g., Yahobenchmarked at max 5000 nodes and 40,000 concurrent tasks.

5. What are the differences in Hadoop 2.0?

引入 HDFS Federation 支持多个 NameNode； Introduced HDFS Federation tsupport multiple NameNodes;

引入 YARN：将资源管理和作业调度解耦； Introduced YARN: decoupling resource management and job scheduling;

支持高可用 NameNode； Supports high-availability NameNode;

支持更广泛的处理模型（不仅限于批处理）； Supports broader processing models (not just batch processing);

支持多类型应用运行（支持 DAG 和交互式任务）； Supports multiple application types including DAG and interactive tasks;

支持容器机制（Containers）提高并发能力。 Supports container mechanism tenhance concurrency.

6. What is the high-availability NameNode in Hadoop 2?

Hadoop 1 的 NameNode 是单点故障（SPOF）； Hadoop 1's NameNode is a single point of failure (SPOF);

Hadoop 2 引入 Active-Passive 双 NameNode 配置； Hadoop 2 introduces Active-Passive dual NameNode configuration;

一主一备 NameNode 协同工作，提升系统可用性； One active and one standby NameNode enhance availability;

若主 NameNode 宕机，自动切换至备用节点。 If active NameNode fails, the system switches tthe standby node automatically.

7. What is HDFS Federation in Hadoop 2?

每个 NameNode 独立运行，无需彼此协调； Each NameNode runs independently without coordination;

DataNode 向所有 NameNode 注册，共用底层块存储； DataNodes register with all NameNodes and share common block storage;

解决 NameNode 扩展性问题，提升横向扩展能力； Solves NameNode scalability issue and enhances horizontal scalability;

Heartbeat、Block Report 等支持多 NameNode。 Heartbeat and Block Reports support multiple NameNodes.

8. What is Resource Negotiator YARN?

ResourceManager：资源统一协调器，替代 JobTracker； ResourceManager: global resource coordinator replacing JobTracker;

ApplicationMaster：为每个应用分配资源，与 NodeManager 协调任务；
ApplicationMaster: allocates resources per application and coordinates with NodeManager;

NodeManager：每台物理机负责容器创建与资源监控； NodeManager: manages container creation and resource monitoring per machine;

将资源管理与调度解耦，提高并发性与可扩展性； Decouples resource management and scheduling for higher concurrency and scalability;

支持多种类型的应用，如 MapReduce、DAG、Spark。 Supports multiple types of applications like MapReduce, DAG, and Spark.

9. What is Application submission in YARN?

客户端将应用提交至 ResourceManager； Client submits application tResourceManager;

ResourceManager 为应用启动 ApplicationMaster（在容器中）；
ResourceManager launches ApplicationMaster in a container for the application;

ApplicationMaster 请求资源容器，监控任务、处理失败任务； ApplicationMaster requests containers, monitors tasks, and handles failures;

ResourceManager 不监控任务细节，只负责 ApplicationMaster 生命周期；
ResourceManager does not monitor tasks, only the lifecycle of ApplicationMaster;

支持多种 QoS 和调度策略（优先级、公平性等）。 Supports various QoS and scheduling strategies (priority, fairness, etc.).

10. What is Apache Spark and features of Spark?

Apache Spark 是通用的大数据处理引擎； Apache Spark is a general-purpose big data processing engine;

支持批处理、流处理、交互式查询等统一框架； Supports batch, streaming, and interactive queries in a unified framework;

高级 API：支持 Java、Scala、Python； High-level APIs: supports Java, Scala, Python;

支持交互式开发（REPL）； Supports interactive development (REPL);

执行图优化，支持结构化数据处理； Execution graph optimization and support for structured data;

与 HDFS 等外部系统集成良好； Good integration with external systems like HDFS;

快速，支持内存计算。 Fast and supports in-memory computation.

11. What is Spark Architecture?

Spark 自身不提供存储或资源调度功能； Spark does not provide its own storage or resource scheduling;

框架由三层组成： The framework consists of three layers:

语言 API 层（Java, Scala, Python）； Language API layer (Java, Scala, Python);

执行引擎（支持通用执行图）； Execution engine (supports general execution graph);

高层库与工具（Spark SQL, MLlib 等）； High-level libraries and tools (e.g., Spark SQL, MLlib);

与外部资源管理系统（如 YARN）协同工作。 Works with external resource managers like YARN.

12. What is Spark Core?

Spark Core 是 Spark 的核心组件； Spark Core is the core component of Spark;

实现了关键概念：RDD（Resilient Distributed Dataset）； Implements key concept: RDD (Resilient Distributed Dataset);

RDD 是分布式、不可变、容错的数据集合，可并行处理； RDD is a distributed, immutable, fault-tolerant collection that can be processed in parallel;

管理任务调度、容错、内存处理； Manages task scheduling, fault tolerance, and memory operations;

与存储系统（如 HDFS）交互； Interacts with storage systems like HDFS;

提供对其他模块如 Spark SQL、Streaming 的支持基础。 Provides foundation for other components like Spark SQL and Streaming.

十七、云中的能源效率-Energy Efficiency in Clouds

Data Center Energy Efficiency - Bilingual Summary

1. Where Does the Power Gin a Data Center?

数据中心中的主要能耗包括： The main power consumption in data centers includes:

服务器（Compute resources）：占据最核心位置； Servers (Compute resources): occupy the central role;

冷却系统：如风扇和空调； Cooling systems: such as fans and air conditioning;

电源转换与 UPS 系统：电力传输和备用； Power conversion and UPS systems: for power delivery and backup;

网络设备：如交换机、路由器等； Networking devices: such as switches and routers;

此外，还包括照明、安全设备、储存系统等。 Additionally, lighting, security, and storage systems alsconsume power.

2. What is Power, Energy, and Energy Efficiency and how tcalculate them?

功率：瞬时能耗率，单位是瓦特（W）； Power: the instantaneous rate of energy use, measured in watts (W);

能量：一段时间内消耗的能量，单位是瓦时（Wh）； Energy: the total energy consumed over time, measured in watt-hours (Wh);

能效：单位能耗下完成的有效工作量。 Energy efficiency: the amount of useful work per unit of energy consumed.

能量 = 功率 × 时间 Energy = Power × Time

能效 = 有效工作量 ÷ 能量 或 = 性能 ÷ 功率 Efficiency = Useful work ÷ Energy or = Performance ÷ Power

3. What is Joule?

焦耳是能量单位： Joule is a unit of energy:

1 焦耳 = 1 瓦秒（Watt × Second）； 1 Joule = 1 Watt × 1 Second;

表示在 1 秒内产生 1 瓦功率所需的能量。 It represents the energy required tproduce 1 watt of power for 1 second.

4. What are the ways tMeasure Energy Consumption in Computing Systems?

测量方式包括： Measurement methods include:

电表和服务器连接的能耗监控； Power meters connected tservers;

智能管理接口（IPMI）； Intelligent Platform Management Interface (IPMI);

RAPL（Intel 的能耗追踪）； RAPL (Intel's energy tracking interface);

软件工具如 powerstat、powertop、kepler 等。 Software tools like powerstat, powertop, kepler, etc.

5. What are Ways tReduce Energy Consumption in Computing Systems?

主要措施包括： Main strategies include:

硬件方向：使用定制芯片、多线程与并行处理； Hardware approaches: use of custom chips, multi-threading and parallelism;

软硬件协同设计； Co-design of hardware and software;

软件优化：低抽象层更高效但更难编程； Software optimization: lower-level abstractions are more efficient but harder tprogram;

结合离线与在线优化策略； Combining offline and online optimization strategies;

实现性能与功耗之间的权衡。 Balancing performance and power consumption.

6. Whis tblame? Hardware or Software?

两者都有责任： Both are responsible:

硬件通常优化为功耗/成本/性能权衡； Hardware is often optimized for power/cost/performance tradeoffs;

软件控制硬件行为，对能效影响更大； Software governs hardware behavior and has a larger impact on efficiency;

软件调度、线程数、频率设置都会影响最终功耗与性能。 Software scheduling, thread count, and frequency settings all affect final power and performance.

7. What is Software Operation?

软件运行的能耗优化包括： Energy optimization in software operations includes:

配置优化（线程数、频率、核心种类）； Configuration tuning (threads, frequencies, core types);

实时监控硬件指标如 CPU 使用率、磁盘读写速率等； Real-time monitoring of hardware metrics like CPU utilization, disk I/O;

探索不同配置下的能耗与性能权衡。 Exploring tradeoffs between energy and performance under various settings.

8. What are solutions for Energy Efficiency in Clouds?

可行方案包括： Solutions include:

虚拟化与调度策略； Virtualization and scheduling policies;

动态电压频率调整（DVFS）； Dynamic Voltage and Frequency Scaling (DVFS);

虚拟机合并与空闲服务器关机； VM consolidation and idle server shutdown;

能耗感知的软件设计； Energy-aware software design;

优化网络协议与中间件层。 Optimizing network protocols and middleware layers.

9. What are solutions of Data Centre Design Considerations?

设计层面的措施包括： Design-level considerations include:

服务器布局与冷却系统设计； Server placement and cooling system design;

冗余机制提升可靠性； Redundancy for increased reliability;

结合交互式与批处理工作负载； Hybrid workloads of interactive and batch processing;

设备生命周期管理，如 18 个月更新周期； Lifecycle management such as an 18-month refresh cycle;

靠近清洁能源区域部署数据中心。 Placing data centers near clean energy sources.

10. What is PUE?

PUE（Power Usage Effectiveness）是衡量数据中心能效的指标： PUE is a metric tmeasure data center energy efficiency:

PUE = 总能耗 / IT 设备能耗 PUE = Total Energy / IT Equipment Energy

理想值为 1.0，越低越好； Ideal value is 1.0, and the lower the better;

2022 年中位数为 1.55； The median value in 2022 was 1.55;

低 PUE 并不代表真正高效，可能存在虚报。 A low PUE doesn't always imply true efficiency, as values may be misreported.

11. What is DVS?

DVS 是一种通过降低电源电压来减少动态能耗的技术： DVS is a technique that reduces dynamic energy consumption by lowering the supply voltage:

减小供电电压会减小功耗，但也会降低性能； Reducing the supply voltage lowers power consumption but alsreduces performance;

现代处理器支持动态调节电压（以及频率）； Modern processors support dynamic adjustment of voltage (and frequency);

可关闭不活跃的晶体管，降低泄露功耗； Inactive transistors can be shut off treduce leakage power;

与 DFS（动态频率调整）相比，DVS 更能减少能量消耗。 Compared tDFS (Dynamic Frequency Scaling), DVS can reduce more energy.

12. What is DVFS-based Power Aware Scheduling?

DVFS 是一种降低能耗的调度技术，其核心思想： DVFS is a power-aware scheduling technique with the core idea:

降低频率，从而可以降低电压； Reduce frequency tallow voltage reduction;

减慢程序运行，但降低动态功耗； Slow down program execution while lowering dynamic power;

构建资源管理与调度算法以减少能耗、满足任务截止时间。 Design resource management and scheduling algorithms treduce energy while meeting deadlines;

示例：降低电压和频率 10%，程序执行变慢 8%，但总功率减少 23%、能量减少 17%。 Example: Reduce voltage and frequency by 10%, execution slows by 8%, but power reduces by 23%, and energy by 17%.

13. What are the solutions of Server Consolidation?

服务器整合是节能的关键技术： Server consolidation is a key technique for energy saving:

将多个工作负载迁移至少量服务器上； Migrate multiple workloads ta minimal number of servers;

允许空闲服务器进入休眠或关闭状态； Allow idle servers tsleep or shut down;

减少运行服务器数量来降低整体能耗； Reduce the number of running servers tlower total energy;

利用虚拟化技术实现高效调度和资源分配。 Use virtualization tenable efficient scheduling and resource allocation.

14. What are the problems of Server Consolidation?

尽管整合带来节能，但也面临挑战： Despite saving energy, consolidation has challenges:

大多数服务器利用率低于 5%； Most servers have utilization under 5%;

空闲服务器仍可消耗最多 70% 能源； Idle servers may still consume up t70% power;

关闭服务器会破坏冷热通道设计，可能无法及时唤醒； Shutting down servers may disrupt cooling designs and delay wake-up;

管理员难以确定何时关闭或如何有效利用服务器。 Admins find it hard tdecide when tshut down or efficiently use servers.

15. What is VM Power Modelling?

VM 功耗建模用于估算每个虚拟机的能耗： VM power modelling estimates the energy usage of each VM:

基于 CPU 使用率； Based on CPU utilization;

假设宿主机总能耗为 Host_P； Assume total host power as Host_P;

分配能耗的比例依据每个 VM 的 CPU 利用率； Distribute power proportionally teach VM's CPU usage;

常用于数据中心能耗不可直接测量的场景。 Common in cases where energy can't be measured directly.

16. What is Physical Machine Power Modelling?

物理机功耗建模方法： Physical machine power modelling methods:

建立 CPU 使用率与功耗 的回归模型； Build regression models of CPU usage and power consumption;

支持线性回归与多项式回归建模； Supports linear and polynomial regression modelling;

用于更精确地估算服务器在不同工作负载下的功耗。 Used for accurate estimation under different workloads.

17. What is solution of Energy Efficiency and Software Design?

软件设计阶段就需考虑能效： Energy efficiency should be considered in software design phase:

软件是否能“意识”到其能耗； Whether the software is aware of its energy use;

在需求与设计阶段引入能效约束； Introduce energy constraints in requirements and design;

通过编程模型 + 插件 + 优化器检测能耗热点； Use models + plugins + optimizers tdetect energy hotspots;

支持自适应的开发模型（Monitor–Analyse–Plan–Execute）； Support adaptive models like Monitor–Analyse–Plan–Execute;

目标是在共享服务环境中开发尽可能节能的软件。 Goal: develop energy-efficient software in shared environments.

18. What is Self Adaptation?

自适应软件设计用于动态调整能耗行为： Self-adaptive software dynamically adjusts energy behavior:

何时自适应：反应式或前瞻式； When tadapt: reactive or proactive;

为何自适应：上下文变化、资源波动； Why tadapt: context changes, resource fluctuations;

自适应层次：应用级、软件级、资源级； Adaptation levels: application, software, or resource;

自适应方式：参数调整、结构调整； Adaptation forms: parameter or structural tuning;

控制方法：依据策略进行决策和执行。 Control methods: decisions and actions based on policies;

强调在整个云栈中实现统一的能效管理。 Emphasizes unified energy management across the cloud stack.

19. What are Metrics Context for Measuring Energy Consumption?

衡量软件/系统能耗的指标需考虑上下文： Energy metrics should consider contextual factors:

应用运行时上下文； Application runtime context;

程序代码结构与热点识别； Program structure and hotspot detection;

运行平台与硬件状态； Execution platform and hardware state;

用户行为与服务访问模式。 User behavior and service access patterns;

这些因素决定了能耗评估的准确性与适应性。 These factors determine the accuracy and adaptability of energy measurement.

十八、物联网边缘-IoT Edge

IoT and Edge Computing - Bilingual Summary

1. What is IoT?

IoT（Internet of Things，物联网）是指通过传感器、智能设备等物理对象将互联网连接到现实世界，使其具备可测量、可控制、可编程的能力，以提供智能服务。 IoT (Internet of Things) connects physical objects like sensors and smart devices tthe Internet tmake the real world measurable, controllable, and programmable for smart services.

2. What are TwDifferent Views of IoT?

单域视角：管理上较为统一，包括传感器/执行器、聚合器、网关、服务控制等，通信协议可能统一或专用。 Single-domain view: unified management with components like sensors/actuators, aggregators, gateways, and service control; protocols may be unified or custom.

多域视角：管理松散、系统异构，不同技术、接口与管理策略，实现自配置与自适应，更复杂但更全面。 Multi-domain view: loosely managed, heterogeneous systems with diverse technologies, interfaces, and policies; more complex but comprehensive with self-configuring and adaptive abilities.

3. What is the Architecture of the Internet of Things?

IoT 架构包含多个层次： IoT architecture consists of multiple layers:

感知层（传感器、RFID） Perception layer (sensors, RFID)

网络层（互联网或无线通信技术） Network layer (Internet or wireless communication)

中间件层（数据聚合与分析） Middleware layer (data aggregation and analytics)

应用层（为人提供服务） Application layer (service tusers)

预计到 2025 年将部署 416 亿个设备，生成 79.4 ZB 数据。 By 2025, 41.6 billion devices will be deployed, generating 79.4 ZB of data.

4. What are the Challenges of IoT?

IoT 的主要挑战包括： The main challenges of IoT include:

海量数据处理与管理； Massive data processing and management;

实时通信和设备互操作； Real-time communication and device interoperability;

隐私保护和访问控制； Privacy protection and access control;

网络带宽限制与延迟问题； Network bandwidth and latency limitations;

安全性威胁。 Security threats.

5. What is the IoT using Distributed Clouds?

IoT 与分布式云结合，支持以下功能： IoT combined with distributed clouds supports:

本地资源管理 Local resource management

公有云交互与服务暴露 Interaction with public cloud and service exposure

质量服务保障（QoS） Quality of Service assurance (QoS)

身份管理与计费 Identity management and billing

动态资源请求与响应机制。 Dynamic resource request and response mechanisms.

6. What is 5G Support for IoT?

5G 网络架构包括：用户设备、无线接入网、核心网。 5G architecture includes: User Equipment (UE), RadiAccess Network (RAN), and Core Network (CN).

5G 通过网络切片实现不同功能的隔离，使同一基础设施支持不同 IoT 用例。 5G enables functional separation via network slicing, supporting different IoT use cases on the same infrastructure.

7. What is examples of IoT: Smart Everything?

IoT 已应用于“智慧一切”： IoT has been applied in 'Smart Everything':

智慧农业：葡萄园监测 Smart agriculture: vineyard monitoring

天文空间：太阳风辐射监测 Astrophysics: solar wind radiation monitoring

核能监测：辐射水平感知 Nuclear monitoring: radiation level detection

城市管理：停车位、空气污染 Urban management: parking and air pollution

环境保护：水质监测 Environmental protection: water quality monitoring

8. What are New Requirements of Disruptive Applications of IoT?

颠覆性应用带来的新需求： New requirements from disruptive IoT applications:

响应时间需小于 1 秒； Response time must be less than 1 second;

云分析会造成延迟与数据瓶颈； Cloud analytics cause delay and bottlenecks;

网络拥塞加剧； Increased network congestion;

安全风险上升； Rising security risks;

需要边缘计算辅助处理； Edge computing is needed for assistance;

商业模型需实时数据分析。 Business models require real-time data analytics.

9. What is the relationship between Fog / Edge Computing and IoT?

边缘计算与雾计算是 IoT 的重要支持架构： Edge and Fog computing are key tIoT infrastructure:

都把计算“下沉”至数据源附近； Both push computation closer tdata sources;

雾计算：在局域网或网关级别处理； Fog computing: processing at LAN or gateway level;

边缘计算：计算能力嵌入终端设备。 Edge computing: computation embedded in end devices;

减少延迟，提升本地实时处理能力。 Reduce latency and improve local real-time processing.

10. What is Edge Computing - Drivers?

推动边缘计算发展的因素包括： Factors driving edge computing include:

延迟：靠近源头，响应更快； Latency: faster response near the source;

带宽：减少云通信； Bandwidth: reduces communication with cloud;

隐私/安全：数据不出本地； Privacy/Security: data stays local;

连接性：离线时也可运行； Connectivity: can operate offline;

本地依赖：设备之间需实时互动。 Local dependency: real-time interaction between devices.

十九、手机网络和边缘计算-Mobile Network and Edge Computing

1. What is Open RadiAccess Network (O-RAN)?

O-RAN 是开放的无线接入网络架构，它将传统基站功能虚拟化，并将其划分为中央单元（CU）、分布式单元（DU）和无线单元（RU）。 O-RAN is an open radiaccess network architecture that virtualizes traditional base station functions, splitting them intCentral Unit (CU), Distributed Unit (DU), and RadiUnit (RU).

2. What is Immutability and Smart Contracts?

不可变性指的是数据或程序一旦登记就无法更改的能力。 Immutability is the ability tstore data or run programs in a way that they cannot be changed once registered.

智能合约是一个自执行的数字协议，协议条款被直接写入代码中。 A smart contract is a self-executing digital agreement with terms directly written intcode.

3. What is Service Delivery Models (SDMs)?

服务交付模型定义了如何向用户和第三方提供网络服务。 Service Delivery Models define how network services are provided tend users and third-party operators.

传统模型包括 MNO（移动网络运营商）和 VMNO（虚拟运营商）。 Traditional models include MNOs (Mobile Network Operators) and VMNOs (Virtual Mobile Network Operators).

新兴模型如 DeWi（去中心化无线网络）和 DePIN（去中心化物理基础设施网络）。 Emerging models include DeWi (Decentralised Wireless) and DePIN (Decentralised Physical Infrastructure Nets).

4. What is Digital Twin (DT)?

数字孪生是对物理资产的近实时虚拟表示，允许通过服务级 API 进行间接可观测与可编程操作。 A Digital Twin is a near-real-time virtual representation of a physical asset, enabling indirect observability and programmability via service-level APIs.

5. What is Key Features and Limitations of 5G?

5G 的关键特性包括云原生虚拟网络功能、O-RAN 支持、IoT 和卫星连接、低延迟与高吞吐率。 5G key features include cloud-native VNFs, O-RAN support, IoT and satellite connectivity, low latency and high throughput.

限制包括高频率下信号衰减导致的基础设施成本、缺乏端到端切片、体验不佳以及市场竞争力不足。 Limitations include high infrastructure cost due tsignal attenuation, lack of slicing and poor experience, and limited competitiveness.

6. What is Disruptive 6G (D6G) Architecture?

D6G 是一种从零开始设计的 6G 架构，强调原生云/边智能、智能合约、去中心化 AI 和数字孪生。 D6G is a from-scratch 6G architecture emphasizing native cloud/edge intelligence, smart contracts, decentralized AI, and digital twins.

7. What is D6G Enabler Families?

D6G 的关键使能技术家族包括：能源、感知与执行、通信、软件化、不可变性、智能、安全、量子技术。 D6G enabler families include: Energy, Sensing and Actuating, Communication, Softwarization, Immutability, Intelligence, Security, Quantum.

8. What is D6G Vision?

D6G 的愿景是创建灵活的开放市场，利用智能合约自动化协议，实现安全可信的资源共享。 The D6G vision is tcreate flexible open digital markets using smart contracts tautomate agreements and enable secure resource sharing.

9. What is D6G On-Chain/Off-Chain Interoperability?

D6G 架构允许区块链上的智能合约与链外系统互操作，借助 Oracle 桥接外部数据和内部协议。 D6G architecture supports interoperability between on-chain smart contracts and off-chain systems using oracles as bridges.

10. What is AIPyCraft?

AIPyCraft 是一个使用 LLM（如 Gemini Pro）自动调试并执行区块链任务脚本（TOML）的智能代理框架。 AIPyCraft is an LLM-powered framework (e.g., Gemini Pro) tautomatically debug and run blockchain task scripts (TOML).

11. What is Edge Intelligence (EI) and role in 6G?

边缘智能结合边缘计算与 AI，减少延迟、增强隐私并提升上下文感知能力。 Edge Intelligence combines Edge Computing with AI treduce latency, enhance privacy, and improve context awareness.

在 6G 中，EI 可用于训练与部署模型，并支持 XR、实时分析等高要求应用。 In 6G, EI supports training/deployment of models and enables demanding apps like XR and real-time analytics.

12. What is LLM-Based Edge Intelligence?

基于大语言模型的边缘智能使设备更具语义理解能力，提高自动化与隐私保护。 LLM-based Edge Intelligence enhances semantic understanding and autonomy of edge devices while preserving privacy.

13. What is AI-RAN Alliance?

AI-RAN 联盟推动 AI 与无线接入网络的深度融合，支持 AI-for-RAN、AI-on-RAN 等多种形态。 AI-RAN Alliance promotes integration of AI and RAN via models like AI-for-RAN, AI-on-RAN, and shared infrastructures.

14. What is Supporting AI/ML with Serverless Computing in 6G?

6G 使用轻量级容器与 Serverless 框架（如 k3s, microk8s, Knative）来支持边缘上的 AI/ML 服务。 6G uses lightweight containers and serverless frameworks (e.g., k3s, microk8s, Knative) tsupport AI/ML at the edge.

15. What is Wasm & Trusted Execution Environments (TEEs)?

WebAssembly（Wasm）提供便携、安全的沙箱执行环境。 WebAssembly (Wasm) offers portable, safe sandboxed execution.

Intel SGX 是一种硬件隔离环境，用于保护代码和数据的机密性。 Intel SGX provides hardware-isolated environments tprotect code and data confidentiality.

例如 TWINE Runtime 可在 SGX 中安全运行 Wasm，并进行双层沙箱隔离与远程验证。 TWINE Runtime can securely run Wasm in SGX with double sandboxing and remote attestation.

二十、边缘计算-Edge Computing

1. What is Industrial Revolutions?

工业革命是指人类历史上生产方式发生根本变革的几个阶段： The Industrial Revolutions refer ttransformative stages of production:

第一次工业革命：机械化生产与蒸汽动力； First: mechanization with steam power;

第二次工业革命：电气化与大规模生产； Second: electrification and mass production;

第三次革命（互联网革命）：自动化、电子与 IT； Third: automation with electronics and IT;

第四次工业革命（Industry 4.0）：大数据、AI、(I)IoT 的整合。 Fourth (Industry 4.0): integration of big data, AI, and (I)IoT.

2. What is Industry 4.0 / 5.0?

Industry 4.0：智能工厂，具备感知、互联、自主决策能力。 Industry 4.0 introduces smart factories with perception, connectivity, and autonomous decision-making.

Cyber-Physical Systems（CPS）监控生产，实时与人和机器交互。 CPS monitors production and interacts with humans and machines in real time.

Industry 5.0：强调人机协作，通过智能机器人和大数据让人工作得更快更精准。 Industry 5.0 focuses on human-machine collaboration with smart robots and big data for speed and accuracy.

3. What are Industry 4.0 Six Design Principles?

互操作性（Interoperability） Interoperability

虚拟化（Virtualisation） Virtualisation

去中心化（Decentralization） Decentralization

实时性（Real-Time Capability） Real-Time Capability

服务导向（Service Orientation） Service Orientation

模块化（Modularity） Modularity

4. What is Content Delivery Network?

CDN（内容分发网络）是分布式服务器系统，依据用户地理位置将网页/内容快速推送给用户。 A Content Delivery Network (CDN) delivers web content tusers based on their location via distributed servers.

5. What is Driverless Cars?

无人驾驶汽车需大量传感器与本地计算，同时连接云端分析系统，实现大数据和实时计算任务的分布式架构。 Driverless cars rely on sensors, local edge computing, and cloud analytics in a distributed architecture.

6. What is Multi-Access Edge Computing?

多接入边缘计算（MEC）是在基站等边缘部署微型数据中心，提升 QoS，满足低延迟与移动性需求。 MEC deploys micrdata centers at the edge (e.g., base stations) tenhance QoS and meet low latency and mobility requirements.

7. What is Edge Computing – The Appliance View?

设备视角的边缘计算：将“微型数据中心”封装成硬件设备部署到现场，结合网关和智能路由器等功能。 Edge computing as appliances offers data centers in a box, combined with gateways and routers for field deployment.

8. What is Network-edge computing?

网络边缘计算在靠近用户的网络设备上监控、分析、缓存数据，实现低延迟与快速响应。 Network-edge computing monitors, analyzes, reduces, and caches data near users for low-latency responses.

9. What is Data Pre-Processing, Filtering, and Pre-Fetching (Cloud tEdge)?

边缘设备在数据送往用户前进行预处理、预取与过滤，仅传输必要信息，降低延迟与带宽消耗。 Edge devices pre-process, pre-fetch, and filter data before sending tusers, reducing latency and bandwidth usage.

10. What is Kubernetes-Connected Cloud-Edge System Architecture?

IoT 设备访问边缘节点上的应用服务，边缘节点资源异构，通过 Kube-Proxy 将请求分配至各 pods，实现负载均衡。 IoT devices access services on heterogeneous edge nodes with Kube-Proxy handling request routing and load balancing.

11. What are Challenges?

边缘计算面临的挑战包括： Edge computing challenges include:

资源发现与调度 Resource discovery and scheduling

数据与计算分配 Data and computation allocation

隐私与安全 Privacy and security

移动性 Mobility

能源效率与碳中和 Energy efficiency and carbon neutrality

边缘 AI 所需硬件 Hardware for edge AI

12. What is Artificial Intelligence at the Edge?

边缘 AI 指将已在云端训练好的 AI 模型迁移至边缘执行，应对设备激增、延迟、隐私与带宽受限问题。 Edge AI moves cloud-trained models tthe edge thandle device growth, latency, privacy, and bandwidth limits.

13. What is Edge-Driven Future for Computing - Example?

Waggle 与 SAGE 是基于 AI 的边缘平台，使用 GPU 实时分析环境数据，部署在科学传感网络中。 Waggle and SAGE are AI-driven edge platforms using GPUs for real-time scientific data analysis in sensor networks.

二十一、分解- Disaggregation

1. What is the Future Direction of Computing?

未来计算的发展方向是： The future of computing is:

由于摩尔定律所依赖的技术推动已经放缓，传统 CPU 性能增长受限； As Moore's law slows down, traditional CPU performance scaling is limited;

因此转向"极端异构计算"（Extreme Heterogeneity）时代； Hence, we are moving toward an era of extreme heterogeneity;

利用多种加速器（不只是 GPU）并针对特定问题设计专用架构，提升晶体管利用率；Using various accelerators (not just GPUs) and designing specialized architectures for specific problems increases transistor efficiency;

即"通过架构专业化适应具体科学问题或应用"。 This means using architectural specialization taddress specific scientific or application needs.

2. What is Disaggregated Data Centres?

分解式数据中心是一种新兴的颠覆性方法： Disaggregated data centres are a novel and disruptive approach:

将数据中心的计算、存储、网络等硬件组件解耦，变为"资源模块"； They decouple compute, storage, and networking components int'resource modules';

有助于灵活配置资源，应对计算密集型任务； This enables flexible resource allocation for compute-intensive tasks;

软件与硬件分离，便于频繁更新和运维优化； Separating software from hardware facilitates frequent upgrades and maintenance;

存储与计算解耦还可以降低整体投资成本； Decoupling storage and compute helps reduce overall investment costs;

提供资源调度的灵活性与效率提升。 It improves flexibility and efficiency of resource scheduling.

3. What is the difference between Conventional and Disaggregated Computing?

传统计算与分解计算的差异： Differences between conventional and disaggregated computing:

传统计算：单一服务器节点集成 CPU、内存、存储等组件； Conventional: server nodes integrate CPU, memory, and storage;

分解计算：每种资源分布在独立硬件模块中； Disaggregated: resources are spread across independent hardware modules;

传统计算资源固化、扩展性差； Conventional systems are fixed and hard tscale;

分解计算模块可按需动态组合； Disaggregated systems allow dynamic resource combinations;

传统计算易造成资源浪费（如 CPU 忙但内存闲）； Conventional systems often waste resources (e.g., busy CPU, idle memory);

分解计算提高资源利用率，支持异构任务。 Disaggregated systems increase resource utilization and support heterogeneous tasks.

4. What is Disaggregated Rack-Scale Computing?

机架级分解计算是将硬件资源（如 CPU、内存、存储）集中于一个大型机架中：Rack-scale disaggregated computing consolidates hardware resources intone large rack:

每个机架被视为一台“超大计算机”； Each rack acts as a super-sized computer;

可动态划分为多个 VM，支持多种资源组合； It can be partitioned intmultiple VMs with diverse resource configurations;

减少资源碎片化，提升数据中心运行效率。 It reduces resource fragmentation and enhances data center efficiency.

5. What is Resource Centric Architecture?

以资源为中心的架构是对数据中心架构的革新： Resource-centric architecture is a data center architecture innovation:

一个机架由多个专用资源板构成（如仅 CPU、仅内存）； A rack is composed of dedicated resource boards (e.g., CPU-only, memory-only);

板间通过超高速网络（如硅光网络）连接； Boards are connected via ultra-fast networks (e.g., silicon photonics);

网络分为两类：机架内部使用专用高速网络；不同机架之间通过标准网络互联。Networks are divided inttwo: intra-rack high-speed networks and inter-rack standard networks.

1. Consider an application deployment scenarion a cloud infrastructure where the aspect of migration is key. Which type of virtualisation mechanism would you choose, Virtual Machines or Containers? Explain your answer.

我会选择容器，因为它们更轻量级、迁移速度更快，并且在云环境中更易于移植。与虚拟机不同，容器不包含完整的操作系统，这显著降低了迁移的开销。当频繁或快速迁移是关键需求时，容器是理想之选。I would choose Containers because they are more lightweight, faster tmigrate, and more portable across cloud environments. Unlike VMs, containers dnot include a full OS, which significantly reduces the overhead of migration. This makes them ideal when frequent or fast migration is a key requirement.

2. 设想一个在云虚拟化基础设施上部署云原生应用的场景。目前，大多数云原生应用都在虚拟机和容器中运行，您可以选择使用 Openstack 和 Kubernetes 来部署虚拟化服务。请描述一个 Openstack 和 Kubernetes 相互补充的部署场景 Consider a cloud-native application deployment scenarion a cloud virtualised infrastructure. A majority chunk of cloud-native applications now runs in Virtual Machines and containers, and you have the choice of using Openstack and Kubernetes tdeploy virtualised services. Describe a deployment scenariwhere Openstack and Kubernetes complement each other.

典型的部署将涉及：

1. OpenStack 提供虚拟基础架构 (IaaS)：

• 管理虚拟机、网络、块存储和身份。

• 允许用户动态创建和管理虚拟机、网络和卷。

• 示例：OpenStack Nova（计算）、Neutron（网络）和 Cinder（块存储）。

2. Kubernetes 部署在 OpenStack 提供的虚拟机之上：

• Kubernetes 集群在 OpenStack 管理的虚拟机内运行。

• 它提供容器编排功能——管理 Pod、扩展工作负载、服务发现等。

• 适用于部署微服务和基于无状态/有状态容器的应用。

3. 它们如何相互补充：

• OpenStack 处理基础架构配置和多租户。

• Kubernetes 处理应用程序级部署和扩展。

• 它们共同构成了一个全栈云原生环境：可扩展、安全且自动化。

A typical deployment would involve:

1. OpenStack provides the virtual infrastructure (IaaS):

- Manages virtual machines, networking, block storage, and identity.
- Allows users tcreate and manage VMs, networks, and volumes dynamically.

- Example: OpenStack Nova (compute), Neutron (network), and Cinder (block storage).

2. Kubernetes is deployed on top of the VMs provided by OpenStack:

- Kubernetes clusters run inside OpenStack-managed VMs.
- It provides container orchestration—managing pods, scaling workloads, service discovery, etc.
- Suitable for deploying microservices and stateless/stateful container-based apps.

3. How they complement each other:

- OpenStack handles infrastructure provisioning and multi-tenancy.
- Kubernetes handles application-level deployment and scaling.
- Together they enable a full-stack cloud-native environment: scalable, secure, and automated.

3. 无服务器计算如何适应分布式 Web 范式 请解释。How does serverless computing fit intthe distributed Web paradigm? Explain.

无服务器计算通过在分布式云基础设施上提供抽象层自然地融入分布式 Web 范式，无需管理服务器即可实现事件驱动、可扩展且经济高效的代码执行。Serverless computing fits naturally intthe distributed Web paradigm by providing an abstraction layer over distributed cloud infrastructure, enabling event-driven, scalable, and cost-efficient execution of code without the need tmanage servers.

1. 解耦的事件驱动架构

在无服务器计算（例如 AWS Lambda、Azure Functions）中，小单元代码（函数）由事件（HTTP 请求、文件上传、数据库更改等）触发。

这与分布式 Web 架构非常契合，因为分布式 Web 架构本质上是异步且松散耦合的。

2. 可扩展性和弹性

无服务器平台可根据需求自动扩展和缩减。

这支持分布式 Web 处理全球分布客户端的不同负载的需求。

3. 无需服务器管理 = 专注于逻辑

开发人员专注于应用程序逻辑，而不是基础设施。

这种抽象使得快速可靠地部署分布式 Web 服务变得更加容易。

4. 细粒度的成本效率

按使用付费模式（例如按请求或执行时间）是对分布式系统中服务可能偶尔或不可预测地触发的补充。

5. 无状态性和互操作性

无服务器函数是无状态的，这与 RESTful Web API、微服务和多云分布式架构非常契合。

1. Decoupled, event-driven architecture
    - In serverless computing (e.g. AWS Lambda, Azure Functions), small units of code (functions) are triggered by events (HTTP requests, file uploads, database changes, etc.).
    - This aligns well with distributed Web architectures, which are inherently asynchronous and loosely coupled.
2. Scalability and elasticity
    - Serverless platforms automatically scale up and down based on demand.
    - This supports the distributed Web's requirement for handling varying loads across globally distributed clients.
3. Nserver management = focus on logic
    - Developers focus on application logic rather than infrastructure.
    - This abstraction makes it easier tdeploy distributed web services quickly and reliably.
4. Fine-grained cost efficiency
    - Pay-per-use billing models (e.g., per request or execution time) complement distributed systems where services may be triggered sporadically or unpredictably.
5. Statelessness and interoperability
    - Serverless functions are stateless, which fits well with RESTful Web APIs, microservices, and multi-cloud distributed architectures.

4. 通过一个例子解释数据局部性如何促进云的能源效率。Explain through an example how data locality contributes tenergy efficiency in clouds.

数据局部性是指在尽可能靠近数据所在位置的地方执行计算。这减少了通过网络传输大量数据的需要，从而节省能源并提高性能。Data locality refers texecuting computation as close as possible twhere the data resides. This reduces the need ttransfer large amounts of data across the network, thereby saving energy and improving performance.

以一个 MapReduce 作业（例如，在 OpenStack 或 AWS 中运行的 Hadoop）为例：

1. 降低网络能耗

由于数据传输最小化，交换机、路由器和网卡的能耗更低。

避免数据中心之间高能耗的长距离传输。

2. 更快的数据访问 = 更短的计算任务 = 更低的 CPU/I 功耗

本地访问可提高 I/吞吐量。

作业完成速度更快 → CPU 和存储磁盘的活跃时间更短 → 动态功耗更低。

3. 降低网络基础设施负载

避免网络拥塞和二次开销，从而节省其他网络服务的能耗。

Consider a MapReduce job (e.g., Hadoop running in OpenStack or AWS):

1. Reduced network energy usage
    - Less energy consumed by switches, routers, and NICs since data transfer is minimized.
    - Avoids energy-expensive long-distance transfers between data centers.
2. Faster data access = shorter compute job = lower CPU/Ipower usage
    - Local access improves I/throughput.
    - Job completes faster → CPUs and storage disks are active for shorter periods → less dynamic power consumed.
3. Less load on network infrastructure
    - Avoids network congestion and secondary overhead, which alssaves energy on other networked services.

5. You are asked tschedule 3 Virtual Machines (VM) on 3 physical servers S1, S2 and S3. The Idle Power (in Watt) of the individual servers is 115, 100 and 85, and the Power Consumption per VM (in Watt) is 15, 10 and 20, respectively. Assuming all three servers are in idle state, what allocation mechanism would minimise the total power consumption? What is the total power consumed in this case?

Tminimise power consumption, all three VMs should be allocated tthe physical server with the lowest idle power consumption, which is S3 (85 W). By doing so, only one server is powered on, and the total power consumed is:

Total Power = Idle Power of S3 + Sum of VM Powers = 85 + (15 + 10 + 20) = 130 Watts

This allocation minimises the number of active physical servers, hence reducing overall power usage.

6. 分解在云计算中被广泛应用，它涉及将数据中心的内存、计算和存储资源解耦，以便每个资源可以独立扩展和配置。以分解存储为例，解释一下 Kubernetes 等用于管理容器化工作负载和服务的平台如何实现这种分解。Disaggregation is widely used in cloud computing and involves decoupling the data center resources of memory, compute and storage, sthat each resource can be scaled and

provisioned independently. Considering disaggregated storage, explain how a platform for managing containerised workloads and services such as Kubernetes can enable such disaggregation.

Kubernetes 作为容器化工作负载管理平台，通过持久化存储编排和动态配置机制，为分解式存储提供强大的支持。

1. 持久卷 (PV) 和持久卷声明 (PVC)

Kubernetes 使用 PV/PVC 模型抽象存储，应用程序（通过 PVC）请求存储，平台将其映射到实际的分解式存储后端（例如 AWS EBS、Ceph、NFS 或 CSI 插件）。

存储无需位于节点本地，支持计算和存储分离。

2. 容器存储接口 (CSI)

CSI 允许 Kubernetes 与各种外部存储系统（块存储、文件存储、对象存储）交互。

支持动态配置，这意味着存储卷可按需创建，无需人工干预。

3. StorageClass 和动态配置策略

管理员定义代表不同分解式存储后端（例如，高 IOPS SSD 与标准 HDD）的 StorageClass，从而实现对配置策略的细粒度控制。

4. 工作负载可移植性和无状态计算

由于存储位于外部且不绑定到特定节点，因此容器（例如 Pod）可以在集群内自由地重新调度。

这增强了弹性和韧性，因为工作负载可以在保持其数据完整的同时进行迁移。

5. 多租户和云原生设计

通过 Kubernetes 实现分解式存储简化了多租户环境中的数据管理，实现了隔离、配额执行和可扩展性。

Kubernetes, as a platform for managing containerised workloads, provides strong support for disaggregated storage through persistent storage orchestration and dynamic provisioning mechanisms.

1. Persistent Volumes (PV) and Persistent Volume Claims (PVC)
    - Kubernetes abstracts storage using the PV/PVC model, where the application (via PVC) requests storage and the platform maps it tan actual disaggregated storage backend (e.g. AWS EBS, Ceph, NFS, or CSI plugins).
    - The storage does not need tbe local tthe node, supporting separation of compute and storage.
2. Container Storage Interface (CSI)
    - CSI allows Kubernetes tinteract with a wide range of external storage systems (block, file, object storage).

    - Supports dynamic provisioning, meaning storage volumes are created on-demand without manual intervention.
3. StorageClass and dynamic provisioning policies
    - Admins define StorageClasses that represent different disaggregated storage backends (e.g., high IOPS SSD vs standard HDD), allowing fine-grained control over provisioning policies.
4. Workload portability and stateless compute
    - Because storage is external and not tied tspecific nodes, containers (e.g., Pods) can be freely rescheduled across the cluster.
    - This enhances elasticity and resilience, as workloads can be moved while keeping their data intact.
5. Multi-tenant and cloud-native design
    - Disaggregated storage via Kubernetes simplifies data management in multi-tenant environments, enabling isolation, quota enforcement, and scalability.

7. 温室草莓生长易受两个主要因素影响：气温（从温室覆盖到收获期间影响显著）和灌溉量（在种植后的最初几个月和收获前都非常重要）。请提出一个基于云/边缘计算的解决方案，该方案能够即时检查环境和生产参数，并在需要紧急修正时生成通知。请讨论如何部署您的解决方案，并提供您认为相关的支持信息。Strawberries growing in greenhouses are very susceptible ttwmain factors: air temperature (significant from the coverage of the greenhouse up tharvest) and water irrigation amount (very important both during the first few months after planting and before harvest). Propose a cloud/edge computing-based solution that allows tinstantly check environmental and production parameters and generate notifications when urgent corrections are needed. Discuss how tdeploy your solution and include any information that you feel is relevant tback it up.

Proposed Architecture:

建议架构：

1. 传感器和边缘设备（部署在温室内）

• 物联网传感器用于测量：

空气温度

湿度

土壤湿度（用于推断灌溉水平）

• 边缘网关（例如 Raspberry Pi、Jetson Nano）：

本地收集传感器数据

执行初始处理和基于阈值的过滤

即使连接中断也能继续运行

2. 边缘计算功能

• 本地规则（例如，如果温度 > 30°C 或土壤湿度 < 阈值）：

立即触发警报

发送控制信号（例如，打开窗户、启动灌溉泵）

• 最大限度地减少延迟并避免云端往返，以便做出紧急决策

3. 云平台（例如，AWS、Azure、GCP）

• 长期存储传感器数据以进行趋势分析

• 集中式仪表板用于可视化（通过 Grafana 或 AWS IoT Core）

• 通知服务（例如，AWS SNS / Twili/ 电子邮件 / 应用推送）

• 可选的机器学习模型预测最佳灌溉需求

4. 通知与控制

• 如果边缘检测到阈值突破：

立即采取本地措施（例如，触发灌溉）

云端通知，进行远程警报传递

• 通过以下方式向农民发送警报：

手机应用程序

短信或 WhatsApp

电子邮件

1. Sensors and Edge Devices (Deployed in Greenhouse)

- IoT sensors tmeasure:
  - Air temperature
  - Humidity
  - Soil moisture (tinfer irrigation levels)
- Edge gateway (e.g., Raspberry Pi, Jetson Nano):
  - Collects sensor data locally
  - Performs initial processing and threshold-based filtering
  - Can continue operating even with intermittent connectivity

2. Edge Computing Functions

- Local rules (e.g., if temp > 30°C or soil moisture < threshold):
  - Trigger alarms instantly
  - Send control signals (e.g., open windows, start irrigation pump)
- Minimize latency and avoid cloud round trips for urgent decisions

3. Cloud Platform (e.g., AWS, Azure, GCP)

- Long-term storage of sensor data for trend analysis
- Centralized dashboard for visualization (via Grafana or AWS IoT Core)
- Notification service (e.g., AWS SNS / Twili/ Email / App push)
- Optional ML models tforecast optimal irrigation needs

4. Notification & Control

- If edge detects threshold breach:
    - Immediate local action (e.g., trigger irrigation)
    - Cloud notified for remote alert delivery
- Alerts sent tfarmer via:
    - Mobile app
    - SMS or WhatsApp
    - Email

Deployment Plan:

Sensor Layer	Temperature, Humidity, Soil Moisture Sensors	Installed throughout greenhouse
Edge Layer	IoT Gateway (e.g., Raspberry Pi with MQTT agent)	One per greenhouse zone
Network	Wi-Fi / LoRa / 4G	Low-bandwidth protocols supported
Cloud Layer	AWS IoT Core + SNS + S3	Central data and alert management
Frontend	Mobile/Web App Dashboard	Deployed with real-time data stream

Advantages of This Design:

- Real-time reaction through edge computing
- Scalable and flexible cloud backend for analytics
- Energy-efficient and reliable even with poor connectivity
- Customizable thresholds and alerting logic

8. 一家企业需要高度控制其数据库的存储和访问，以及管理 Web 前端和其他应用程序基础设施的能力。他们拥有庞大的现有 IT 基础设施，并且正在不断扩展其功能。哪种云计算模式能够满足他们当前的所有需求并帮助他们降低成本 请解释你的答案。
An enterprise needs highly controlled storage and access to their databases as well as the ability to manage the infrastructure for Web front ends and other applications. They have a large existing IT infrastructure and are continually expanding its capability. Which cloud computing model will satisfy all their current needs and enable them to reduce cost? Justify your answer.

最适合此场景的云计算模型是混合云模型。

理由：

混合云允许企业继续利用其现有的本地 IT 基础架构，同时集成公有云资源以实现可扩展性和成本效益。

私有云组件确保高度受控的存储和数据库访问，满足合规性和安全性要求。

公有云支持 Web 前端服务和其他应用程序的动态扩展，而无需产生大量的前期资本成本。

混合云通过优化工作负载来帮助降低成本——敏感工作负载可以私有运行，而需求变化的工作负载可以公开运行。

它还支持灵活性和未来增长，这对于不断扩展 IT 规模的企业来说是理想的选择。

The most appropriate cloud computing model for this scenario is the Hybrid Cloud model.

Justification:

A hybrid cloud allows the enterprise to continue leveraging its existing on-premise IT infrastructure, while integrating public cloud resources for scalability and cost efficiency.

The private cloud component ensures highly controlled storage and database access, meeting compliance and security requirements.

The public cloud enables dynamic expansion of web front-end services and other applications without incurring large upfront capital costs.

Hybrid cloud helps in cost reduction by enabling workload optimization — sensitive workloads can run privately, while variable-demand workloads can run publicly.

It also supports flexibility and future growth, which is ideal for enterprises continually expanding their IT footprint.

9. 举例说明支持虚拟机实时迁移、情境化和重新情境化的场景 Give an example of a scenario where live migration of Virtual Machines, contextualisation, and re-contextualisation are supported

一所大学运行一个托管在私有云（例如基于 OpenStack 或 VMware）上的大型在线学习平台。该平台包括：

• 运行 Moodle/LMS 服务的虚拟机 (VM)

• Web 服务器、数据库和分析工具

1. 实时迁移

• 在考试高峰期，流量会激增。

• 为了保持性能，托管学习服务的虚拟机会从超负荷的物理服务器实时迁移到利用率较低的服务器，而无需关闭服务。

• 这确保了零停机时间和学生访问的不间断性。

2. 情境化

• 虚拟机首次启动时，情境化会注入配置数据，例如：

o 主机名、IP 配置

o 网络存储路径

o 应用程序启动脚本（例如，自动启动 Apache 和 MySQL）

• 这使得通用虚拟机映像能够在启动时根据用户/项目情境进行自定义。

3. 重新上下文化

• 在新区域迁移或重启后（例如，从工程数据中心迁移到科学数据中心），虚拟机需要：

o 更新其上下文参数（例如，新的 DNS、新的 IP 池、本地存储路径）

o 应用新的网络策略或用户访问配置

• 这可以通过使用 cloud-init 或 config-drive 重新上下文化来完成，通常在迁移或管理员命令后自动触发。

A university runs a large-scale online learning platform hosted on a private cloud (e.g., based on OpenStack or VMware). The platform includes:

- Virtual Machines (VMs) running Moodle/LMS services
- Web servers, databases, and analytics tools

1. Live Migration

- During high exam seasons, traffic spikes.
- To maintain performance, VMs hosting learning services are live-migrated from overloaded physical servers to underutilized servers without shutting down the services.
- This ensures zero downtime and uninterrupted student access.

2. Contextualisation

- When a VM is initially launched, contextualisation injects configuration data such as:
    - Hostname, IP configuration
    - Network storage paths
    - Application startup scripts (e.g., auto-start Apache and MySQL)
- This enables a generic VM image to be customized at boot time based on user/project context.

3. Re-contextualisation

- After migration or reboot in a new zone (e.g., from the engineering data center to the science data center), the VM needs to:
    - Update its contextual parameters (e.g., new DNS, new IP pool, local storage paths)
    - Apply new network policies or user-access configurations
- This is done using cloud-init or config-drive re-contextualisation, often triggered automatically after migration or administrator commands.

10. 从建立和运营一家大型云计算服务提供商的角度，列举所涉及的一种主要资本支出类型、一种主要的运营支出项目以及主要收入来源 From the perspective of setting up and operating as a major cloud-computing service provider, name one major type of capital expenditure involved, one major line of operating expenditure, and the primary source of revenue

资本支出（CapEx）：

数据中心基础设施的投资，包括服务器、存储系统和网络设备。

运营支出（OpEx）：

运行和维护数据中心的电力和冷却成本。

主要收入来源：

基于订阅的按需付费云服务，例如计算（例如 AWS EC2）、存储（例如 S3）和托管服务（例如数据库、分析）。

Capital Expenditure (CapEx):

Investment in data center infrastructure, including servers, storage systems, and network equipment.

Operating Expenditure (OpEx):

Electricity and cooling costs for running and maintaining data centers.

Primary Source of Revenue:

Subscription-based pay-as-you-go cloud services, such as compute (e.g., AWS EC2), storage (e.g., S3), and managed services (e.g., databases, analytics).

11. 云提供商通过与最终用户签署服务水平协议 (SLA) 来提供云服务。讨论提供商资源配置不足和过度配置的问题，以及它们对 SLA 履行的影响。Cloud providers supply cloud services by signing Service Level Agreements (SLA) with end-users. Discuss the issues of under-provisioning and over-provisioning of resources by providers and their impact in the context of SLAs fulfilment

配置不足：

定义：分配的资源少于处理实际工作负载所需的资源。

影响：

• 违反 SLA：导致性能下降、延迟或中断，违反正常运行时间/响应保证。

• 处罚：提供商可能因违反 SLA 而面临经济处罚或声誉受损。

• 客户不满意：如果未满足 SLA，客户可能会转向其他提供商。

• 被动扩展开销：即使自动扩展触发得太晚，延迟仍然可能违反 SLA 指标。

示例：一个电商应用在促销期间流量激增，但云提供商未能及时配置足够的计算实例 → 响应缓慢，未达到 SLA 要求。

过度配置：

定义：分配的资源超过实际需要，超出工作负载需求。

影响：

• 资源浪费：导致运营成本增加（例如，虚拟机闲置、存储未使用）。

• 利润率降低：为了弥补超额配置，提供商必须提高收费或接受更低的利润率。

• 始终满足 SLA，但效率低下：虽然超额配置可以确保 SLA 合规，但这与云计算的按使用付费效率模式相矛盾。

示例：一个静态内部报告系统被分配了 10 个虚拟机，全天候运行，但实际上它每天只需要 2 个虚拟机，运行 6 小时。

Under-Provisioning:

Definition**: Allocating** fewer resources than required **to handle actual workloads.**

Impact:

- SLA Violations**: Leads to degraded performance, latency, or outages, violating uptime/response guarantees.**
- Penalties**: Providers may face financial penalties or reputation damage for SLA breaches.**
- Customer dissatisfaction**: Clients may move to other providers if SLAs are not met.**

- Reactive scaling overhead**: If auto-scaling is triggered too late, the delay can still violate SLA metrics.**

Example**: An e-commerce app experiencing** traffic **spikes during a sale, but the cloud provider doesn't provision enough compute instances in time → slow response, missed SLA.**

Over-Provisioning:

Definition**: Allocating** more resources than necessary**, exceeding workload demand.**

Impact:

- Wasted resources**: Leads to** increased operational costs **(e.g., idle VMs, unused storage).**
- Reduced profit margins**: Providers must charge more or accept reduced margins to cover excess provisioning.**
- SLA always met, but inefficient**: While over-provisioning ensures SLA compliance, it contradicts cloud's pay-per-use efficiency model.**

Example**: A static internal reporting system being allocated 10 VMs 24/7, even though it only needs 2 VMs for 6 hours per day.**

12. 从云提供商的角度来看，基于服务的模型的一个负面后果是，它可能很快导致难以管理整个云系统并将服务质量 (QoS) 保持在可接受的水平。讨论当前云自管理解决方案面临的挑战，涉及以下方面：**From the cloud provider perspective, a negative consequence of the service-based model is that it may quickly lead to a situation where it becomes difficult to manage the whole cloud system and keep Quality of Service (QoS) at an acceptable level. Discuss the challenges faced by current cloud self-managed solutions in relation to:**

• Cloud heterogeneity, Cloud automation, Cloud evolution

1. 云异构性

挑战：云基础设施构建于不同的硬件、虚拟化平台、操作系统发行版、API 和服务之上。

由于组件之间不兼容，管理异构环境非常复杂。

不同的供应商拥有专有解决方案（例如 AWS、Azure 和 OpenStack），这使得可移植性和互操作性变得困难。

当工作负载优化或容错机制无法跨平台无缝协作时，QoS 就会受到影响。

2. 云自动化

挑战：自动化需要可靠的编排工具和策略来管理配置、扩展、恢复和更新。

许多云系统缺乏统一的自动化标准，导致工具和脚本碎片化。

自动化可能会在意外的工作负载模式或故障下中断，尤其是在动态的多租户环境中。

如果没有实时、自适应的智能，通过自动化决策（例如虚拟机迁移、自动扩展和修复）来确保 QoS 将非常困难。

3. 云演进

挑战：云服务快速发展，不断引入新的功能、API、定价模式和硬件选项。

持续的变化使得自主管理解决方案难以保持最新状态。

旧版应用程序或基础设施可能与最新产品不兼容。

在升级、迁移或集成过程中确保一致的服务质量 (QoS) 成为一项重大挑战，需要持续验证和测试。

1. Cloud Heterogeneity

Challenge: Cloud infrastructures are built using diverse hardware, virtualization platforms, OS distributions, APIs, and services.

Managing a heterogeneous environment is complex due to incompatibility among components.

Different vendors have proprietary solutions (e.g., AWS vs Azure vs OpenStack), making portability and interoperability difficult.

QoS suffers when workload optimization or fault tolerance mechanisms cannot work seamlessly across platforms.

2. Cloud Automation

Challenge: Automation requires reliable orchestration tools and policies to manage provisioning, scaling, recovery, and updates.

Many cloud systems lack unified automation standards, leading to fragmented tools and scripts.

Automation can break under unexpected workload patterns or failures, especially in dynamic, multi-tenant environments.

Ensuring QoS through automated decisions (e.g., VM migration, auto-scaling, healing) is difficult without real-time, adaptive intelligence.

3. Cloud Evolution

Challenge: Cloud services evolve rapidly, introducing new features, APIs, pricing models, and hardware options.

Continuous change makes it hard for self-managed solutions to remain up to date.

Legacy applications or infrastructure might not be compatible with the latest offerings.

Ensuring consistent QoS during upgrade, migration, or integration becomes a significant challenge, requiring continuous validation and testing.

13. Consider a large MapReduce job with m mappers and r reducers.

(i) If you run the word count MapReduce program introduced in the lecture, how many output files will you get at the end of the job?

(ii) How many key-value pairs will there be in each file? Assume k is the number of unique words in the input files.

(iii) How many distinct copy operations will there be in the sort/shuffle phase?

最终输出文件数量由 reducer 数量 r 决定 → r 个输出文件；

每个输出文件大约有 k/r 个 key-value 对（假设分布均匀）；

shuffle 阶段有 m × r 条数据路径，每个 mapper 对每个 reducer 分发一份数据。

14. 您需要设计一个云安全解决方案。云作为多用户分布式环境，带来了独特的安全挑战，具体取决于用户的操作级别：应用程序、虚拟或物理。针对每个级别，请提出一项安全需求和一个相关威胁。**You have been asked to design a cloud security solution. The cloud as a multi-user distributed environment brings unique security challenges, dependent on the level at which the user operates: application, virtual or physical. For each level propose one security requirement and one associated threat.**

1. 应用层

安全要求：

确保数据机密性和应用程序用户的访问控制（例如，通过加密、身份验证、RBAC）。

注入攻击（例如，SQL 注入、跨站点脚本攻击），攻击者利用未经严格审查的输入来访问或操纵应用程序数据。

2. 虚拟化层（例如，虚拟机/虚拟机管理程序）

安全要求：

在租户之间提供隔离（例如，虚拟机隔离、虚拟机管理程序安全强化）。

虚拟机逃逸攻击，恶意虚拟机利用漏洞访问主机或同一物理机器上的其他虚拟机。

3. 物理层（硬件、数据中心）

安全要求：

确保物理访问控制和安全的硬件生命周期管理（例如，限制访问、监控、篡改检测）。

内部威胁或硬件篡改，具有物理访问权限的人员可以窃取数据或破坏硬件组件（例如，通过冷启动攻击或物理提取）。

1.Application Level

Security Requirement:

Ensure data confidentiality and access control for application users (e.g., via encryption, authentication, RBAC).

Injection attacks (e.g., SQL injection, cross-site scripting), where attackers exploit

poorly sanitized inputs to access or manipulate application data.

2. Virtualisation Level (e.g., VM/Hypervisor)

Security Requirement:

Provide isolation between tenants (e.g., VM isolation, hypervisor security hardening).

VM escape attacks, where a malicious VM exploits vulnerabilities to access the host or other VMs on the same physical machine.

3. Physical Level (Hardware, Data Centers)

Security Requirement:

Ensure physical access control and secure hardware lifecycle management (e.g., restricted access, surveillance, tamper detection).

Insider threats or hardware tampering, where a person with physical access can steal data or compromise hardware components (e.g., via cold boot attacks or physical extraction).

15. 大气污染和道路拥堵降低了生活质量，导致驾驶员浪费时间和燃油。您受邀领导一个智慧城市项目，该项目旨在设计、部署并验证一个由传感器、执行器、摄像头和屏幕组成的平台，并将其应用于利兹及其周边地区，为市民提供实用信息。该平台可以监测噪音、温度、亮度、一氧化碳和空闲停车位等各种参数。请您讨论如何部署您的解决方案，并附上您认为相关的支持信息。**Atmospheric pollution and congested roads depreciate the quality of life, resulting in lost time for drivers and wasted fuel. You have been contacted to lead a smart city project which aims at designing, deploying and validating in Leeds and its environment a platform composed of sensors, actuators, cameras and screens to offer useful information to citizens. Different parameters such as noise, temperature, luminosity, carbon monoxide and free parking slots can be monitored. Discuss how to deploy your solution and include any information that you feel is relevant to back it up.**

系统架构概述

1. 感知层（边缘设备）

部署在路灯、交通信号灯和公交车站的传感器用于监测：

一氧化碳 (CO) → 城市空气质量

噪音水平 → 检测污染、拥挤程度或夜生活对环境的影响

温度和亮度 → 天气和照明条件

停车传感器 → 检测空闲/占用停车位

用于交通拥堵监测和可视化分析的摄像头

2. 通信层

使用 5G、LoRaWAN 和 Wi-Fi 网状网络实现安全、低延迟、节能的数据传输。

边缘设备将数据传输到边缘网关，由边缘网关进行初步预处理。

3. 数据平台（云+边混合）

部署基于云的分析平台（例如 AWS IoT、Azure IoT Hub 或基于 Kafka + Spark + InfluxDB 的开源技术栈）：

聚合、存储和分析实时传感器数据

使用 AI/ML 预测交通拥堵、污染峰值或停车位可用性

边缘计算确保对时间关键操作（例如即时更改标识）进行低延迟决策。

4. 市民互动层

在关键位置（例如市中心、公共交通枢纽）安装智能屏幕：

显示当前污染水平、附近的免费停车位和拥堵警报

移动应用程序或网页仪表板：

允许个性化通知、路线建议和停车警报

与公共交通 API 集成，提供全面的出行洞察。

部署策略

试点阶段：

在交通繁忙区域部署（例如，利兹市中心、海丁利、主要停车场）

使用有限的传感器进行测试并收集基线数据

运行模拟（例如，重新规划交通路线对二氧化碳水平的影响）

扩展阶段：

推广至住宅区、学校区域和公共建筑

利用本地历史数据完善预测模型

实现机构间数据共享（例如，与交通部门、卫生服务部门）

安全与隐私

在传感器和云端之间使用加密通信 (TLS)

确保视觉/视频数据的匿名化

为市政工作人员和市民实施访问控制策略

优势/理由

通过更智能的路线规划和停车位分配减少交通拥堵

通过提高污染意识改善公众健康

更有效地利用公共基础设施（例如，停车场、照明）

数据驱动的城市规划和环境政策支持

System Architecture Overview

1. Sensing Layer (Edge devices)

Sensors deployed on lamp posts, traffic lights, and bus stops to monitor:

Carbon monoxide (CO) → urban air quality

Noise levels → detect pollution, crowding, or nightlife impact

Temperature and luminosity → weather and lighting conditions

Parking sensors → detect free/occupied parking slots

Cameras for traffic congestion and visual analytics

2. Communication Layer

Use 5G, LoRaWAN, and Wi-Fi mesh networks for secure, low-latency, energy-efficient data transmission.

Edge devices transmit data to edge gateways, which do initial preprocessing.

3. Data Platform (Cloud + Edge Hybrid)

Deploy a cloud-based analytics platform (e.g., AWS IoT, Azure IoT Hub, or open-source stack with Kafka + Spark + InfluxDB):

Aggregates, stores, and analyzes real-time sensor data

Uses AI/ML to predict traffic congestion, pollution peaks, or parking availability

Edge computing ensures low-latency decision-making for time-critical actions (e.g., changing signage instantly).

4. Citizen Interaction Layer

Smart screens installed at key locations (e.g., city center, public transport hubs):

Show current pollution levels, free parking spots nearby, congestion alerts

Mobile app or web dashboard:

Allows personalized notifications, route suggestions, parking alerts

Integration with public transit APIs for full mobility insight.

Deployment Strategy

Pilot Phase:

Deploy in high-traffic zones (e.g., Leeds city centre, Headingley, major parking areas)

Test with limited sensors and gather baseline data

Run simulations (e.g., impact of rerouting traffic on CO levels)

Expansion Phase:

Roll out to residential areas, integrate school zones, public buildings

Refine predictive models with local historical data

Enable inter-agency data sharing (e.g., with transport authority, health services)

Security and Privacy

Use encrypted communication (TLS) between sensors and cloud

Ensure anonymization of visual/video data

Implement access control policies for municipal staff and citizens

Benefits / Justification

Reduced traffic congestion via smarter routing and parking allocation

Improved public health through pollution awareness

More efficient use of public infrastructure (e.g., parking, lighting)

Data-driven urban planning and environmental policy support