文档密级：公开

InCloud OS 8.2.0

技术白皮书

济南浪潮数据技术有限公司

2026 年 2 月

目录

1 前言

1.1 文档用途

本文档用于描述 **InCloud OS** 产品功能、技术原理及适用场景，为公司内部、外部了解 **InCloud OS** 技术实现的参考文档。

1.2 读者对象

本文档提供给以下相关人员使用：

- 客户代表
- 产品经理
- 售前工程师
- 系统维护工程师
- 研发工程师

1.3 安全声明

公司产品不会主动获取或使用用户的个人数据，仅在您同意使用特定功能或服务时，在业务运营或故障定位的过程中可能会获取或使用用户的某些个人数据（如告警邮件接收地址、IP 地址），公司产品在涉及个人数据的收集、存储、使用、传输、删除等全生命周期的处理活动中，已在产品功能上部署了必要的安全保护措施，同时，您也有义务根据所适用国家或地区的法律法规制定必要的用户隐私政策并采取足够的措施以确保用户的个人数据受到充分的保护。

浪潮高度重视产品数据安全，公司产品在涉及系统运行和安全数据的全生命周期处理活动中，已严格按照相关法律法规及监管要求，在产品功能上部署了必要的安全保护措施。作为系统运行和安全数据处理者，您有义务根据所适用国家或地区的法律法规制定必要的数据安全政策并采取足够的措施以确保系统运行和安全数据受到充分的保护。

浪潮将一如既往的严密关注产品与解决方案的安全性，为客户提供更满意的服务。浪潮已全面建立产品安全漏洞应急和处理机制，确保第一时间处理产品安全问题。若您在本产品使用过程中发现任何安全问题，或者寻求有关产品安全漏洞的必要支持，请直接联系浪潮客户服务人员。

1.4 修订记录

修订时间	修订人	修订内容
2021.05	李栋梁、乔霖等	**6.0.0** 版本正式发布。
2021.08	李栋梁、乔霖等	补充信创部分内容。
2021.10	李栋梁、乔霖等	更新至 **V6.0.1** 版本
2022.01	李栋梁、乔霖等	更新至 **V6.0.2** 版本
2022.06	李栋梁、乔霖等	更新至 **V6.5.0** 版本
2022.10	李栋梁、乔霖等	更新至 **V6.5.1** 版本

修订时间	修订人	修订内容
2023.01	李栋梁、乔霖等	更新至 **V6.5.2** 版本
2023.06	李栋梁、乔霖等	更新至 **V6.8.0** 版本
2023.12	李栋梁、乔霖等	更新至 **V6.8.1** 版本
2024.04	李栋梁、乔霖等	更新至 **V6.12** 版本
2024.08	李栋梁、乔霖等	更新至 **V6.15** 版本
2024.11	李栋梁、乔霖等	更新至 **V6.18** 版本
2025.01	李栋梁、乔霖等	更新至 **V6.18.1** 版本
2025.04	李栋梁、乔霖等	更新至 **V8.0.0** 版本
2025.08	李栋梁、雷亚帅等	更新至 **V8.1.0** 版本： 新增 **4.2.9.2**、**4.2.9.3**、**4.4.3.4**、**4.7.7**、**4.8.3** 章节 更新 **4.1.17**、**4.3.1.2.1**、**4.3.3.1**、**4.7.2.1**、**4.7.3**、**4.12**、**5.1** 章节

修订时间	修订人	修订内容
2026.2	李栋梁	更新至 **V8.2.0** 版本： 更新 **4.4.3.3.2**、**4.6.2.1**、**4.6.12**、**4.7.3**、**4.7.7**、**4.13.1**、**5.1** 章节

2 产品概述

浪潮致力于成为中国领先的云计算解决方案供应商，提供面向未来云数据中心的整体解决方案，凭借浪潮高端服务器、海量存储、云操作系统、信息安全技术为客户打造领先的云基础架构，全面支撑企业云、行业云、政务云建设。浪潮云海云操作系统（**InCloud OS**）作为云数据中心的“中枢神经系统”，基于开放架构，兼容多元异构平台，满足云数智不同应用场景需求。**InCloud OS** 是基于浪潮在数据中心领域多年研发积累和大规模实践验证，以及对云业务场景的深入理解， 打造了一款面向政府与企业业务系统上云场景，功能完备、安全可靠、具备智能运维能力的云操作系统。云海云操作系统通过构建可演进的私有云，对数据中心基础设施资源池进行管理和调度、支撑各类政府、企业业务应用高效、安全、可靠和稳定运行。

2.1 产品定位

InCloud OS 采用业界先进的架构设计理念和技术架构，驱动产品在可靠性、稳定性、性能、易用性等方面的全面提升。**InCloud OS** 提供了基于虚拟化技术的云主机、云硬盘等资源类型服务，基于容器的各类应用服务以及基于大模型的人工智能服务，以更好的匹配企业级业务的多样性，在提供相关服务的基础上，为了保障业务的高效稳定运行，也提供配套的智能化监控运维、服务治理能力。通过统一的运营运维门户，保证一致性的用户体验。**InCloud OS** 立足政企 **IT** 基础设施市场，做业内领先的行业深度优化的易用、稳定、安全的企业级智能云操作系统，为用户私有云建设提供一站式交钥匙解决方案。

2.2 产品架构

2.2.1 技术架构

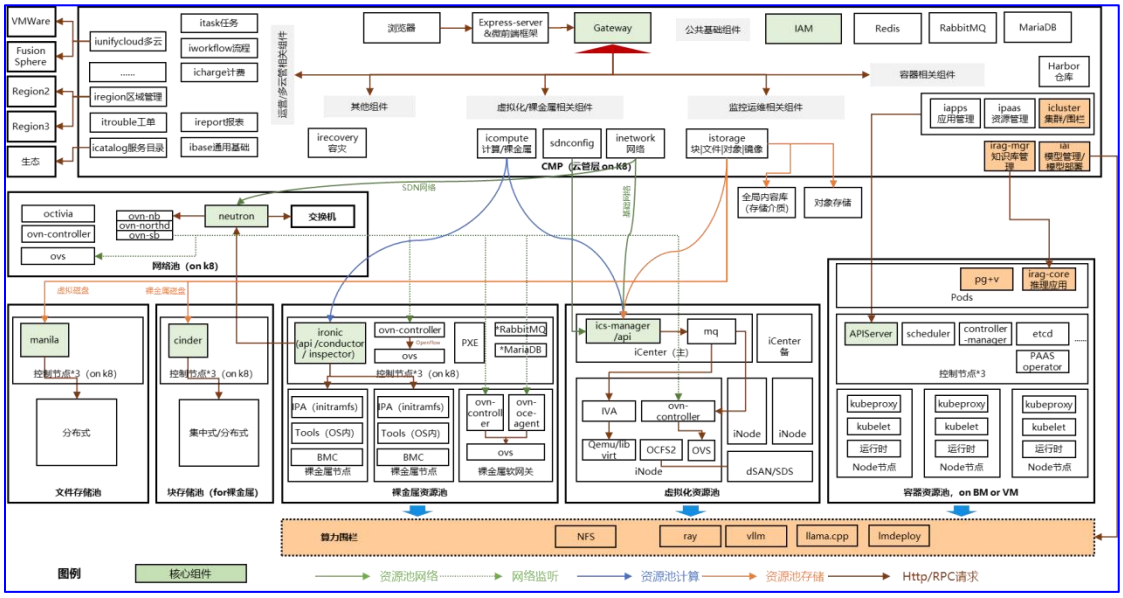


图 2.2.1–1 InCloudOS 技术架构

InCloud OS 采用模块化的架构设计理念，模块的划分遵循“高内聚、低耦合”的设计原则，每个模块均有清晰的职责和功能。模块可以独立运行，支持独立横向扩展及灵活的安装部署方式，模块之间的相互独立降低了故障域，某个模块的故障影响范围可以限定在一定范围之内，减少对于系统整体的影响，并且能够快速恢复。模块之间采用 **REST API** 进行交互。通过模块化的设计，进一步提升了系统的可维护性、可重用性、开发效率和测试效率。

从物理视图角度，系统整体上主要由资源池化层、软件定义层及 **CMP** 云管层组成。

- 资源池化层负责将底层的硬件基础设施（例如服务器、存储设备和网络设备等）进行抽象和池化，供上层服务使用。在 **InCloudOS** 中，虚拟化资源池基于虚拟化技术，为上层提供云主机服务；容器资源池基于内

核轻量虚拟化技术，为上层提供容器服务；裸金属资源池通过对硬件服务器的封装与抽象，提供裸金属计算服务；存储池则为其他资源池或上层应用提供块、文件及对象存储服务。

- 软件定义层包括对上述池化资源通过软件定义的方式进行抽象、池化及调度管理，主要包括软件定义计算）软件定义网络（**Neutron**）、裸金属管理（**Ironic**）、软件定义存储（**Manila**、**Cinder**）等。
- **CMP** 云管层提供多租户云计算资源生命周期管理与资源编排能力，作为整个平台使用入口及门户，提供了统一的资源组织模型、统一的运营运维、统一的服务目录。

从逻辑视图的角度，系统分为资源层、服务层、访问层、用户层模块。

- 资源层：提供了云数据中心各类硬件资源，包括服务器、交换机、存储设备等，也包括运行在其上的基础软件，例如主机操作系统、设备驱动程序等。在此基础上，提供资源的抽象与控制能力。
- 服务层：包含对云操作系统所提供的服务的实现。包含并控制实现服务所需的组件模块，并向上通过访问层为云租户提供云服务。
- 访问层：提供对服务层能力进行访问及管理的通用接口，例如 **Web** 界面或者 **API** 接口，访问层基于 **RBAC** 机制校验用户请求，验证用户是否被授权使用特定的功能，同时保障请求的安全性。
- 用户层：提供了直接的用户接口，通过该接口，云租户与云操作系统及其提供的服务进行交互，执行相关的管理活动。

2.2.1.1 CMP 子系统架构

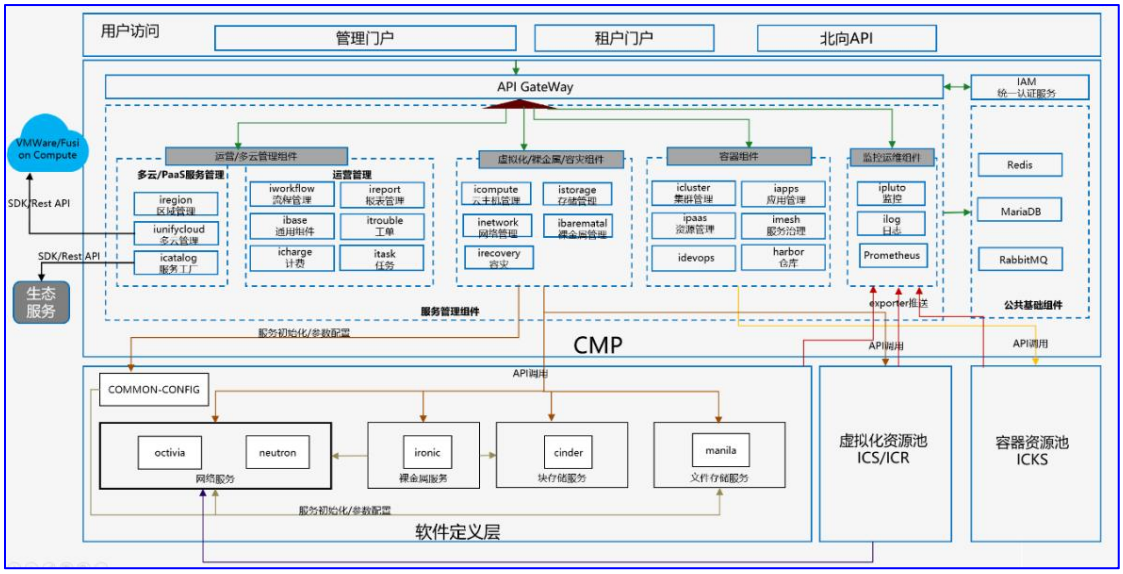


图 **2.2.1.1–1 CMP** 子系统架构

CMP 层采用微服务架构的设计理念，基于 **k8s+springboot** 的架构模式，每个服务节点可分布式独立部署，**UI** 通过 **API** 网关调用功能服务,对外可提供符合 **RESTful** 规范的 **API** 接口供第三方集成。云管平台微服务框架可实现多服务节点间服务的负载均衡、服务路由、集群容错等机制，使系统具有很强的连通性、健壮性、伸缩性和扩展性，**CMP** 层主要包括统一资源管理、运营管理、容灾备份、多云管理、服务工厂等核心功能模块。

云资源管理：定义统一资源管理模型（虚拟数据中心），包括统一的配额管理，资源与组织模型（用户、角色等）的关系。定义统一的资源服务模型，将网络池、裸金属池、文件存储池、块存储池、虚拟化资源池、容器资源池提供的虚拟化和容器等资源抽象成云服务，为用户提供云主机、裸金属、容器、云硬盘、对象存储、文件存储、**VPC** 网络等服务。

运营管理：定义统一的用户角色级租户模型并基于此实现的多租户管理、申请审批、工单管理、资源计量计费等运营能力，为用户提供系统管理员、租户管

理员和租户用户等多种角色和相应管理界面，满足用户多样化运营模式需求。

多云管理：基于插件化开发、标准 **API** 接口模式，实现对异构云资源池的管理，支持多版本兼容，通过适配器模式，屏蔽云平台不同版本的功能实现差异，统一操作视图。

容灾备份：支持云平台管理组件和业务系统跨数据中心的容灾能力，保证业务连续性。**CMP** 跨数据中心主备运行，基于 **binlog** 解析的数据库同步机制实现 **CMP** 主备数据准实时同步，业务容灾借助集中式/分布式存储远程复制、云主机 **CDP** 技术实现跨数据中心双站点容灾，整体容灾能力满足金融容灾标准要求。

服务工厂：提供基于标准服务包的 **IaaS+**、**PaaS** 服务，同时提供基于微前端+微后端+**IAM** 的三方生态服务接入框架，使三方平台与云平台实现用户体系、界面风格、监控告警、运营操作的统一。以前端微服务的方式内嵌三方服务平台的界面，与生态产品深度融合，提升用户使用体验；支持生态镜像、**OpenServiceBroker** 等方式形成完备的服务接入框架，按需提供外部生态服务。

统一监控告警：采用分布式架构设计，覆盖监控数据、日志数据、告警数据分片及监控等管理模块，满足大规模系统的监控、告警、通知和数据分析的需求。同时提供北向接口供第三方集成，帮助用户打造稳定可靠、准实时的日志分析、监控告警服务。

运维巡检：提供自动化运维能力，支持管理员、租户权限，用于云平台智能化巡检、云资源的状态巡检（包括虚拟化、容器和云管平台）；提供信息收集（如资源规模、平台日志等）、平台调整修复等功能于一体的智能工具集。

网络服务：基于 **Neutron+OVN+OVS** 组件构建软件定义网络技术体系，并封装 **VPC** 网络模型，提供完善的 **L2~L7** 网络功能，支撑云主机、裸金属、云主

机等资源运行。性能方面支持基于 **DPDK**、网卡多队列等技术加速；互联互通方面支持同城跨中心二三层互联互通、跨地域三层互联互通、云内云外互联互通等场景。

裸金属服务：裸金属服务向用户提供独占式的裸金属资源，相对于云主机具有高性能、无虚拟化损耗等特点，适用于性能要求高、业务关键性强、隔离要求高等业务场景。裸金属模基于 **ironic** 组件进行扩展和优化，支持裸金属注册、检查、部署装机、监控、网络配置、存储配置等。裸金属服务以裸金属资源池的形式进行组织管理，同资源池内支持对多种 **CPU** 架构的裸金属进行管理。

文件存储服务：基于 **manila** 组件进行扩展和优化，以驱动的方式对接多套文件存储集群，屏蔽存储差异性，实现对多套文件存储集群的管理。提供统一的文件存储管理接口及文件存储全生命周期管理操作，文件存储实例为云主机和裸金属提供共享的存储空间，保障数据的持久化和稳定性。

块存储服务：以驱动的方式对接多套块存储集群，屏蔽存储差异性，实现对多套块存储集群的管理。提供块存储全生命周期管理操作，为云主机、容器、裸金属提供持久化的数据存储能力，保障数据的性能及稳定性。

2.2.1.2 虚拟化子系统架构

虚拟化子系统属于操作系统级架构，按照标准操作系统进行开发部署，各个组件以系统服务的形式打包运行。系统主要包含两大角色：**iCenter** 和 **iNode**。**iCenter** 是系统的管理节点，主要部署管理组件，涉及到各种代理服务、监控、**ics-manager**、消息中间件、高速缓存、**heartbeat** 和 **drbd** 等，能够监控并管理 **iNode** 主机、云主机和其他物理/虚拟资源。同时，**iCenter** 为系统管理员提供了一套图形化的系统管理界面。**iNode** 是系统的计算节点，主要部署 **agent** 代理用

于支持资源操作和管理，可采用物理机或云主机的部署模式，将硬件层抽象虚拟化，提供云主机运行所需的 **CPU**、内存、存储和网络资源，同时支撑云主机的运行。

虚拟化子系统主要包含如下计算、存储和网络资源管理三大核心功能模块。

计算资源管理：除了将服务器组织到集群中统一管理并通过数据中心、集群、主机进行简单清晰层次划分外，提供了云主机资源的全生命周期管理能力，支持云主机的导入、导出、迁移、克隆、备份、快照、暂停、恢复等功能，提供业内最全的各类云主机操作系统兼容性，针对不用的业务使用场景，可提供云主机分组管理、动态热添加、资源细粒度控制、完善的云主机资源设备管理等功能。用户也可以使用裸金属管理功能直接在物理裸机上部署业务。总之，系统丰富完善的计算能力可以便捷的为用户打造一个高效、可靠、安全的计算环境。

存储资源管理：提供稳定、可靠的存储服务，支持通过 **iSCSI**、**FC**、**NVMe-oF**、**FCoE** 等标准协议对接外部存储，通过自研并经过特殊优化的 **OCFS** 集群文件系统提供稳定、可高的 **CFS** 集群存储。除 **CFS** 数据存储外，还提供本地、**NFS**、裸设备数据存储以满足不同业务场景下对存储要求。镜像库功能为用户提供了简洁易用的镜像存储服务，可以很方便的与云主机或云主机模板配合，完成云主机导入导出、加载光驱软驱、模板批量部署云主机等操作。

网络资源管理：提供功能丰富的虚拟化网络功能，可满足各类应用场景下对网络的要求。大致可分为经典网络管理与 **SDN** 高级网络两部分。经典网络基于 **OVS** 开发设计，提供分布式交换机、网络、交换拓扑、上行链路管理、流量采集、端口镜像、**DHCP** 地址分配等经典网络能力；**SDN** 基于 **OVS/OVN** 开发设计，实现分布式防火墙、分布式路由器、分布式 **NAT** 网关、分布式负载均衡等

高级网络功能，基于分布式技术不存在单点故障问题，保证了网络数据转发层面的高可靠性。

2.2.1.3 容器子系统架构

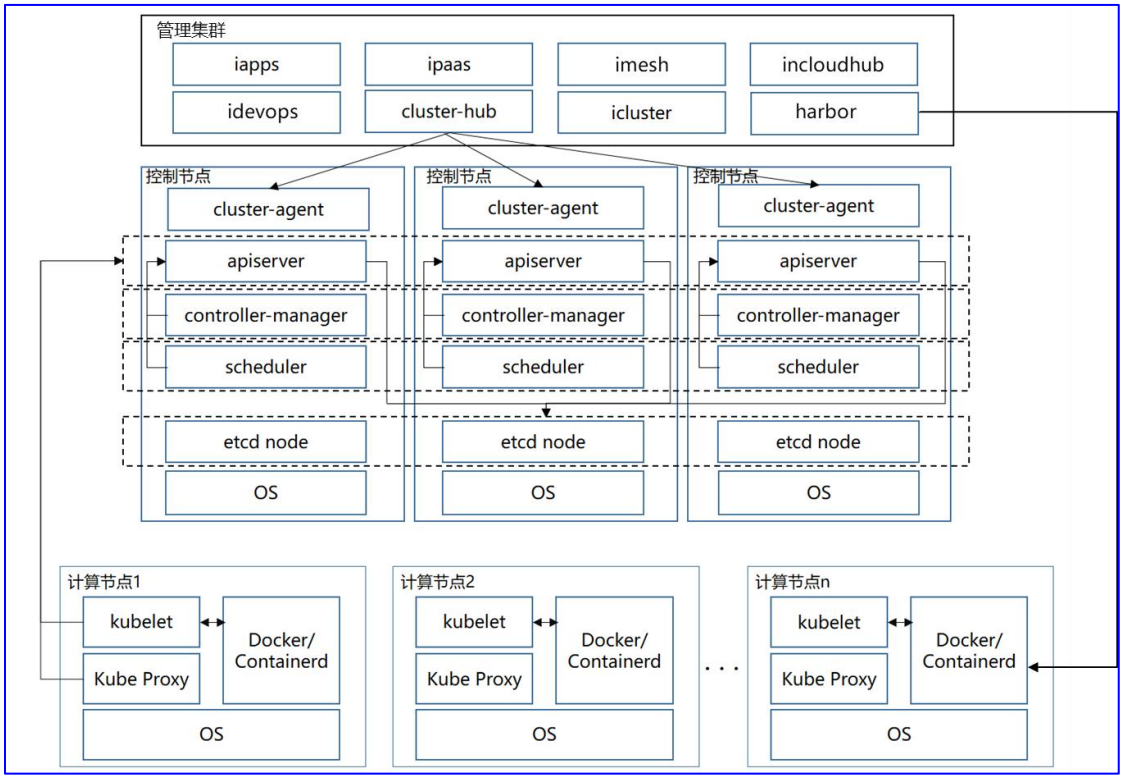


图 **2.2.1.3–1** 容器子系统架构

容器子系统基于开放的容器编排引擎 **Kubernetes** 进行扩展和优化，北向提供符合 **Kubernetes** 规范要求的 **REST API**，以声明式 **API+**控制器为核心设计理念，通过 **CRD** 可以轻松定制和扩展云原生能力。容器子系统由管理集群和工作集群构成，管理集群主要负责工作集群生命周期管理和容器应用生命周期管理，工作集群主要负责资源管理与调度以及应用运行环境。

管理集群为一个独立的 **Kubernetes** 集群，多集群管理 **icluster** 与 **cluster–hub**、容器应用管理 **iapps** 与 **ipaas**、容器镜像仓库 **harbor** 与 **incloudhub**、流水线 **idevops** 以及微服务治理 **imesh** 等组件以无状态/有状态负载的形式运行在该集群上。其

中，**icluster** 负责注册工作集群，**cluster-hub** 运行后等待 **cluster-agent** 主动连接，连接成功后通知 **icluster**，完成工作集群的纳管；**iapps** 负责容器应用编排，编排完成后通过 **ipaas** 将应用调度到指定的工作集群中运行；**idevops** 将接入的代码库进行容器镜像构建，同时自动将镜像部署到指定工作集群；**imesh** 为工作集群中的容器应用开启服务治理、链路追踪、灰度发布等高级能力。

工作集群包括控制节点和计算节点，控制节点负责集群内资源管理与调度，计算节点负责提供容器应用运行环境。控制节点上运行着控制平面组件，其中，**etcd** 负责保存集群状态，**kube-api** 提供资源访问入口，**controller-manager** 负责维护集群资源处于期望的状态，**scheduler** 负责资源根据策略调度到某个计算节点上，**cluster-agent** 运行后主动与 **cluster-hub** 建立安全的 **websocket** 连接，并发起注册请求。计算节点运行着 **kubelet**、**kube-proxy** 和容器运行时，其中 **kubelet** 根据 **scheduler** 调度结果进行容器实例创建，如果本地没有对应的容器镜像时，则向管理集群中 **harbor** 服务拉取容器镜像，容器运行时采用 **docker/containerd** 来运行容器实例，**kube-proxy** 根据 **Service** 资源创建本地的网络规则保证服务可访问。

管理集群与工作集群协同工作共同完成容器应用的管理，在大规模场景下，可以通过构建多个工作集群实现水平扩展，在小规模场景中，管理集群可以与工作集群复用，将平台管理组件部署在工作集群中。

2.2.1.4 人工智能子系统架构

人工智能子系统是在当前 **CMP** 子系统、容器子系统等基础上，新增部分模块，提供人工智能服务，部分模块运行在 **CMP** 层、部分模块运行在容器资源池中，借助已有服务模块实现人工智能业务运行。

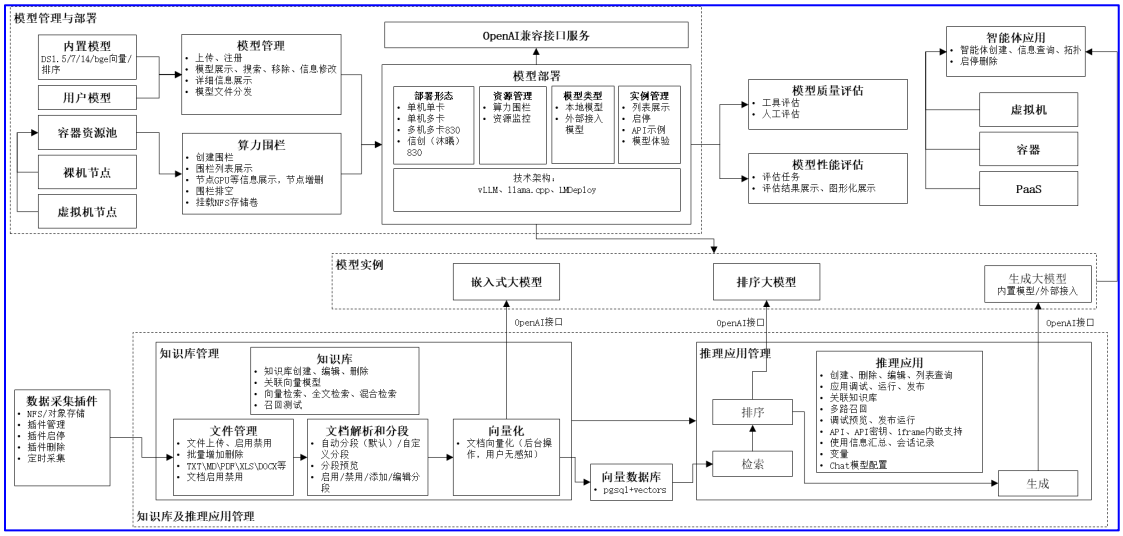


图 **2.2.1.4–1** 人工智能子系统架构

❖ 功能模块

从模块划分上看，人工智能子系统包括模型管理与部署模块、智能体管理模块、知识库管理、推理应用管理模块。其中模型管理与部署包括模块的存储、注册、部署、服务等；智能体管理是使用云资源支撑智能体服务的管理，包括智能体的创建、查询等全生命周期管理；知识库管理是基于向量数据库提供知识库的存储、解析管理，同时支撑数据自动更新；推理应用管理是基于知识库+大模型的 **RAG** 服务。

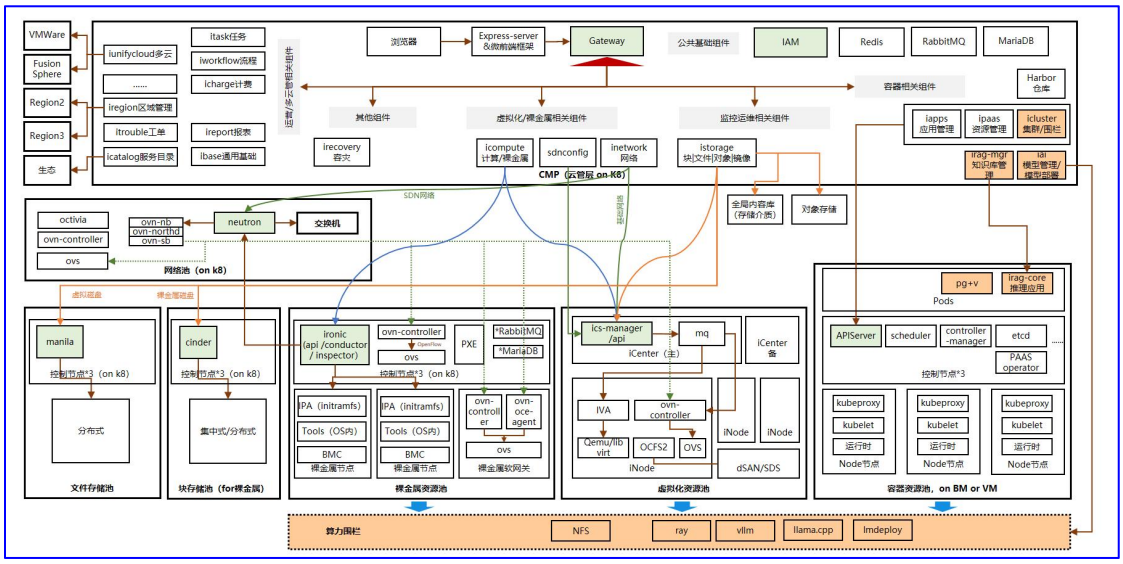


图 **2.2.1.4–2** 功能模块

❖ 技术架构

人工智能子系统通过外接 **NFS** 保存模型，通过围栏的方式将 **AI** 算力进行隔离，通过多种推理引擎将模型服务在围栏内，对外提供大模型服务的 **API** 供 **AI** 应用、智能体应用使用，主要包括模型仓库、模型围栏、模型服务。

模型部署方面，通过容器技术实现大模型的部署；通过模型文件注册机制，组成模型仓库，实现大模型的统一管理，模型文件所在的目录模型部署时将存放模型文件的目录挂载到算力围栏的主机上，直接加载到 **GPU** 显存进行运行。设计推理引擎自适应选择机制，根据模型类型、模型格式、模型系列及模型算力类型，自动选择合适的推理引擎运行大模型服务，根据模型所需的显存大小和围栏内节点的配置，可进行单机单卡、单机多卡、多机多卡等多种方式运行模型。

模型评估方面，包括模型性能评估管理、质量评估管理和评测集管理三个部分，性能评估模块包括性能预测和性能测试部分，预测部分根据输入的模型结构、参数量以及对应的硬件规格，进行模型推理效率的理论预测，性能测试和质量评估通过评估任务进行实测，包括管理和任务两部分，管理模块运行在云管理层，

负责评估任务的创建和结果展示等生命周期管理，评估任务运行在容器资源池内，评测集管理模块负责基准和自定义评测数据集的上传、下载和删除等生命周期管理，评估任务在运行前拉取评测集进行测试。

知识库&推理应用部分，通过 **postgreSQL** 作为数据库载体，支持向量检索、全文检索及混合检索；通过推拉式数据插件 + 定时任务机制实现知识库文件自动更新；结合向量模型、重排模型、大模型，实现推理应用部署发布。

智能体方面，分别调用计算模块、容器模块、裸金属模块、**PaaS** 模块、模型管理模块、知识库&推理应用模块接口实现云主机、容器、裸金属、**PaaS**、大模型、知识库&推理应用的管理、创建、调度，并结合自有状态管理、模板管理，实现资源统一编排、模板及导入导出等。

2.2.2 功能架构

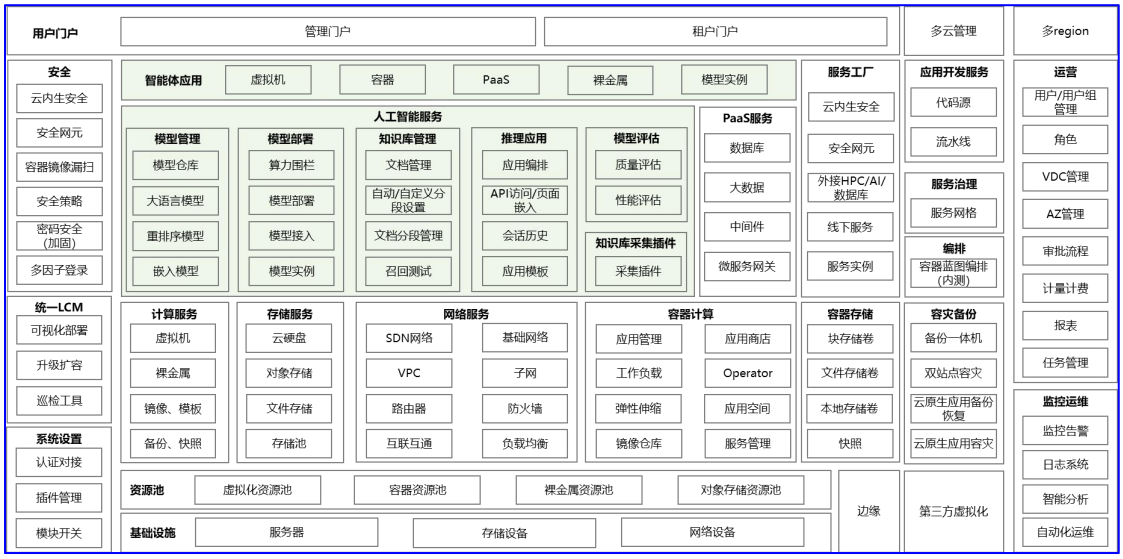


图 **2.2.2–1** 系统功能架构

InCloud OS 采用分层管理架构，基于物理服务器、存储、交换机、路由器等物理设备，通过软件定义和编排技术为用户提供丰富的计算、存储、网络、容器、人工智能、**PaaS**、监控运维、运营管理等云计算服务。

计算服务:平台提供云主机资源的全生命周期管理能力,支持云主机的导入、导出、迁移、克隆、备份、快照、暂停、恢复等功能，提供全面操作系统兼容性。针对不用的业务使用场景，可提供云主机弹性伸缩、资源细粒度控制、完善的云主机资源设备管理、亲和性以及异构设备调度等丰富的调度策略、高可用等功能，支持资源按需弹性，提高资源利用效率，为用户打造一个高效、可靠、安全、强扩展性的计算环境。

存储服务：平台提供块存储、文件存储、对象存储服务，支撑平台资源及用户应用系统对存储的多样化需求。支持云硬盘的创建、修改、删除、分配等全生命周期管理及备份快照能力；文件存储服务支持新增、删除、权限设置、快照、分配回收等，可以通过 **NFS**、**CIFS** 等协议挂载到云主机、裸金属等资源；对象存储服务支持对象桶生命周期管理、账号管理、配额管理等，并提供 **S3** 协议支撑用户业务运行。

网络服务：平台基于软件 **SDN** 架构提供丰富的网络功能，支持用户自主创建隔离的 **VPC** 网络环境,子网内资源二层互通、**VPC** 内子网之间资源三层互通；支持软件负载均衡功能,基于内核参数调优、多进程等性能优化机制提供高性能 **L4**、**L7** 负载均衡服务；支持安全组、虚拟防火墙功能，基于虚拟交换机 **ACL** 策略防护云资源网络安全；支持通过对等连接打通租户 **VPC** 网络，通过云连接打通跨 **Region VPC** 网络；为满足用户多样化南北向流量模型，平台提供 **NAT** 网关、浮动 **IP**、云专线、云连接、**IPSec VPN** 等网络服务，用户可以通过平台界面自主网络网络配置，满足互联网接入、专线接入、**VPN** 接入场景需求。

裸金属服务：提供裸金属服务全生命周期管理功能，包括裸金属注册、部署操作系统、网络配置、监控、控制台等全生命周期管理功能，支持裸金属与云主

机统一 **VPC** 网络、裸金属软网关、裸金属挂载云硬盘等高级特性。

容器服务：提供容器资源、容器应用生命周期管理能力，支持自制应用管理、标准 **Helm** 应用管理和一云多芯应用管理，支持通过镜像、应用包和应用商店创建应用。提供工作负载管理、网络管理、存储管理、配置管理和自定义资源等。支持 **GPU** 等异构设备管理和调度，支持通过界面和 **Yaml** 文件方式创建和编辑资源，支持标准的 **Kubernetes** 资源导入和管理，满足普通用户和高阶用户灵活使用容器服务的诉求。网络方面，提供高度优化的容器网络管理模型，支持租户容器和云主机二层互通，跨租户 **VPC** 网络隔离，支持网络策略、**QoS** 限速、四七层负载均衡以及 **RDMA RoceV2/IB** 等。存储方面，针对部署在云主机上的容器集群，通过 **CSI** 插件对接存储池实现存储卷的相关功能，屏蔽底层存储差异，简化使用成本。

人工智能服务：平台提供全栈 **AI** 推理服务，涵盖大模型部署/评测、知识库构建、推理应用开发及智能体全生命周期管理。兼容 **DeepSeek**、**Qwen** 等主流大模型框架，支持 **671B**、**70B** 等参数版本，支持用户按业务场景与硬件配置弹性选型；知识库引擎支持 **Word/Excel/TXT/PDF** 等文档解析，通过向量数据库实现向量检索与全文检索的融合，并结合大模型提供 **RAG** 服务；基于虚拟机、裸金属、容器、**PaaS** 组件构建智能体，实现智能体全生命周期管理与跨站迁移，赋能业务系统智能化升级。

PAAS 服务：基于容器提供开箱即用的云原生中间件、数据库全生命周期管理功能，支持 **PAAS** 服务的创建、扩容、参数配置、用户管理、对接存储、重启实例、备份恢复、亲和策略、资源和服务监控告警、日志查询等功能，提供业界主流数据库、中间件等 **PaaS** 服务。

容器集群服务：支持容器集群的全生命周期管理，包括纳管、创建、查询、删除集群以及插件管理等。支持直连和代理纳管手动部署的容器集群，一键创建容器集群并自动纳管，支持集群节点伸缩。支持插件管理，可以将插件一键部署到对应的工作集群，支持插件升级、回滚和卸载。

容灾备份：提供全面的容灾备份能力，包括本地备份、同城或者异地容灾，具体支持云管理平台、云主机和容器应用的备份恢复和跨地域容灾，基于存储复制实现业务数据的容灾，云主机同时支持基于 **CDP** 技术、存储解耦的跨地域数据容灾。

应用开发服务：通过 **CI/CD** 技术，用户可以实现从源码到容器镜像快速构建，环境快速部署/更新，支持将已有流水线通过 **Jenkinsfile** 的方式导入到应用开发服务的流水线里，实现流水线的迁移；通过服务网格技术，用户可以实现复杂应用的各微服务之间的流量观测、流量治理、流量追踪，实现应用级智能化管理与运维。

微服务网关：基于 **Gateway API/Ingress** 规范，提供云原生微服务网关，将后端服务接口绑定到网关实例对外暴露服务，实现 **API** 生命周期管理、流量策略配置和执行、流量监控，实现南北向流量的统一管控。

运营：定义统一的用户角色级租户模型并基于此实现统一的用户管理、多租户管理、申请审批、工单管理及资源计量计费等运营能力，为用户提供系统管理员、租户管理员和租户等多种角色，满足用户多样化运营需求。

监控运维：支持对云平台及云资源的监控告警能力，覆盖云管平台、容器平台、虚拟化平台、裸金属、**PaaS** 组件等，并支持短信、邮件、钉钉、企业微信告警；通过集成多种预测算法、检测算法、相似性算法实现动态阈值检测、资源

画像、问答小助手等智能化功能；支持平台自动巡检，可以自动化检测平台服务、计算节点、裸金属、云主机等服务的异常。

边缘服务：将云服务能力从中心扩展到边缘，实现边缘资源统一管理和调度、边缘应用跨域统一分发、边缘自治、边缘轻量化以及监控告警统一管理等功能，实现云边协同，支持边缘节点和边缘集群两种不同规格的边缘算力。

安装部署：基于图形化的部署工具，提供云平台环境的部署、扩容及升级功能。实现云平台控制组件及其配置的分发，云平台规模的扩容以及云平台版本的升级及回滚。

安全：基于 **InCloudOS** 技术架构形成安全网元接入规范对接第三方安全网元，实现基于云主机形态网元的自动化发放、生命周期管理及 **license** 管控；基于共享内存设备适配第三方安全系统，实现不依赖于数据中心网络的虚拟化无代理杀毒功能；通过第三方安全系统适配云平台内核和 **K8S**，实现容器安全防护功能；基于 **IAM+**微前端**+**微后端 生态接入技术方案接入第三方安全系统，实现界面融合、用户权限打通、租户打通，满足云内生安全一体化需求；对接第三方密码机、**ueky**、国密浏览器，实现云平台本身的传输机密性、身份鉴别、重要数据完整性保护、重要数据机密性保护，满足密评要求。

2.2.3 系统部署架构

2.2.3.1 平台网络架构

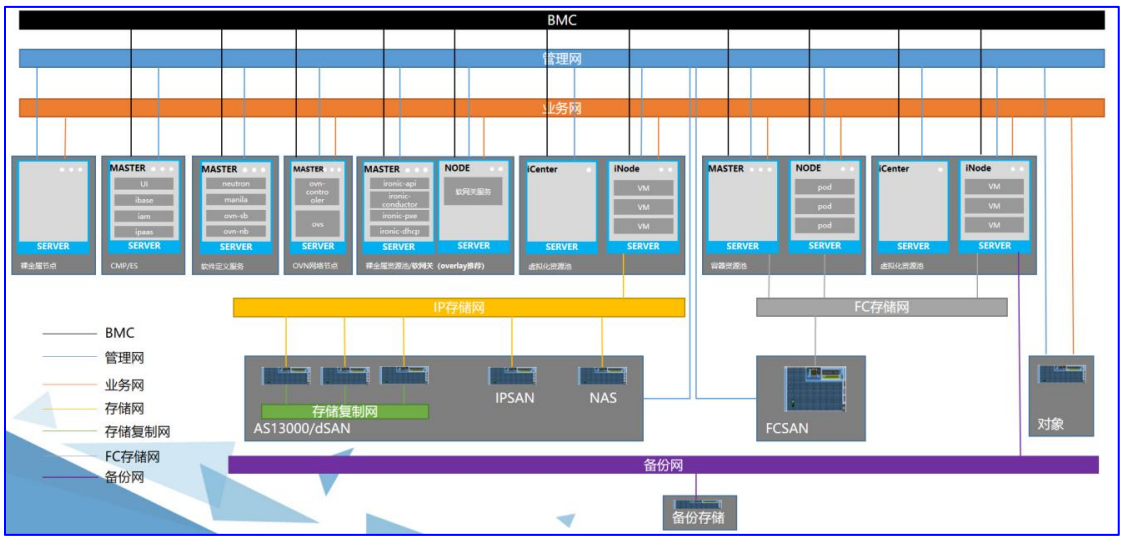


图 **2.2.3.1–1** 平台网络架构

InCloud OS 版本包含统一管理模块(CMP)、容器资源池管理模块、虚拟化资源池管理模块与裸金属资源池管理模块，该版本可以管理虚拟化资源池、容器资源池与裸金属资源池，根据客户的业务场景进行组合部署，由 **CMP** 统一进行资源调度与分配，满足不同客户的业务场景。

在云平台建设时，一般把部署网络划分为 **BMC** 带外管理网络、管理网络、业务网络与存储网络，如上部署网络架构所示，除了 **BMC** 连线外，上图中的连线都代表双端口冗余的。

BMC 带外管理网络一般用于服务器的远程管理；

管理网络是承载页面请求、**API** 调用的网络，通过这个网络，客户可以管理云主机、容器、裸金属、虚拟网络、云硬盘等资源；

业务网络是客户的业务对外提供访问的网络，业务请求都会通过业务网络转发到业务云主机或者业务 **POD** 或者裸金属节点来处理；

存储网络提供云硬盘挂载到云主机的路径或者 pv 资源挂载到 POD 的路径，是业务云主机或者业务 POD 处理完业务请求后需要落盘时所使用的网络，IO 请求将通过存储网络把业务数据持久化到存储后端，存储网络根据后端的不同，对部署又有不同的要求。从虚拟化资源池或者容器资源池的角度来看，资源池对接的存储包括 IP 协议（分布式存储、IPSAN、NAS）与 FC 协议(FCSAN)的后端存储。

当后端是 IP 协议存储时，存储网使用万兆以太网交换机，服务器侧需要配套的以太网口进行连接，而且分布式存储集群还需要增加一个存储复制网（如上图左下角所示），用于副本之间的同步。

当后端是 FCSAN 时，FC 存储网要使用 FC 存储交换机，服务器侧需要配套的 HBA 卡进行连接。当客户现场既有 IP 协议存储又有 FCSAN 时，要根据现场情况进行具体规划。

当对虚拟化资源池有备份需求时，云主机所在的计算节点需要额外规划万兆网卡，与备份存储组成备份网，实现数据备份。

2.2.3.2 系统部署架构

InCloud OS 系统部署模式主要分为虚拟机部署和物理机部署。该部署架构只针对 CMP 组件、软件定义层组件及 ICKS 组件。CMP 组件与软件定义层组件只部署在相同架构的虚拟机或者物理机上，ICKS 组件支持虚拟机或者物理机形式的一云多芯模式的部署。

- 虚拟机部署：使用 ICS 虚拟机环境，进行虚拟机部署云管理平台和软件定义层组件。适用于 50 点以下的小规模集群，可以节约资源。
- 物理机部署：采用物理机部署云管理平台和软件定义层，并使用物理机

独立部署监控和日志节点。适用于大规模部署场景，可灵活扩展及提升性能。

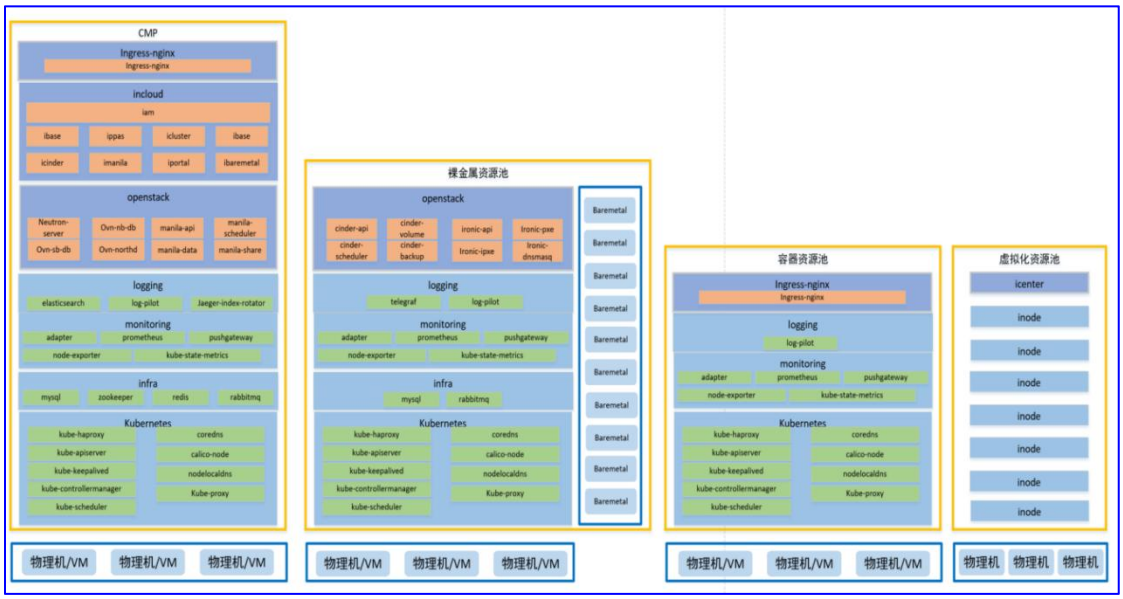


图 **2.2.3.2–1** 部署架构

如上图部署架构所示，部署是面向物理机或者 **VM** 进行操作，在硬件层面上完成 **V8** 版本控制组件的分发。硬件层面及 **OS** 层除外，**InCloud OS V8** 版本软件从大的标签分组包括了 **k8s** 集群组件、**infra** 组件、**monitoring** 组件、**logging** 组件、**openstack** 组件、**incloud** 组件、**ingress** 组件。在构建 **CMP** 控制面时，对承载 **CMP** 组件的物理机或者 **VM** 进行标签组合，而标签关联了模块组合，从而实现 **k8s** 组件、**infra** 组件、**monitoring** 组件、**logging** 组件、**openstack** 组件、**incloud** 组件、**ingress** 组件在物理机或者 **VM** 上的组合部署。在构建裸金属资源池的控制面、容器资源池的控制面时也遵循上述逻辑，虚拟化资源池的部署继承上个版本的部署逻辑。

3 基础技术原理

3.1 虚拟化核心技术与原理

虚拟化 **(Virtualization)** 是资源的逻辑表示，实现虚拟资源与物理硬件解耦。虚拟化技术的实现形式是在系统中加入一个虚拟化层，将下层的资源抽象成另一形式的资源，提供给上层使用。从广义上来说，虚拟化就是一种采用软硬件分区、聚合、部分或完全模拟、分时复用等方法来管理计算资源、构造一个或者多个计算环境的技术。所构建的计算环境还需要有以下三个特点：

- 保真性：应用程序在虚拟机上执行，将表现为与在物理硬件上相同的执行行为。
- 高性能：在虚拟执行环境中，应用程序的绝大多数指令能够在 **VMM** 不干预的情况下，直接在物理硬件上执行。
- 安全性：物理硬件应该由 **VMM** 全权管理，被虚拟出来的执行环境中的程序（包括操作系统）不得直接访问硬件。

在操作系统与硬件之间，虚拟化软件通过空间上的分割、时间上的分时以及模拟，抽象出一个虚拟的硬件接口，向上层操作系统提供一个与它原先期待一致的服务器硬件环境，使得上层操作系统可以直接运行在虚拟环境上，可允许多个操作系统同时运行在单个物理服务器上。

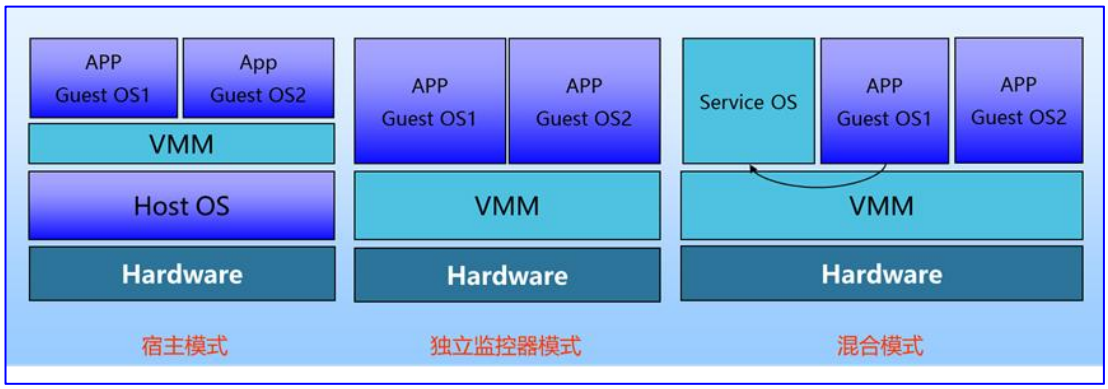


图 **3.1–1** 虚拟化核心技术与原理

服务器虚拟化的虚拟化软件层称为虚拟机监控器 **(Virtual Machine Monitor, VMM)**，也称为 **Hypervisor**，如上图所示，常见虚拟机监控器分为三类：

- 宿主模式：在 **VMM** 之下还有一层宿主操作系统，并不直接运作在裸机上。由于 **Guest OS** 对硬件的访问必须经过宿主操作系统，因而带来了额外的性能开销，但可充分利用宿主操作系统提供的设备驱动和底层服务来进行内存管理、进程调度和资源管理等。
- 独立监控器模式：**VMM** 直接运作在裸机上，管理和使用底层硬件资源，**Guest OS** 对硬件资源的访问都要通过 **VMM** 来完成，作为底层硬件的直接操作者，**VMM** 拥有硬件的驱动程序。
- 混合模式：为宿主模式和独立监控器模式的综合，**VMM** 直接运行在裸机上，但是驱动程序需要由 **Service OS** 提供，**Service OS** 为运行在 **VMM** 上的一台特殊虚拟机。

虚拟化技术的实质：将底层资源进行分区，并向上层提供特定的和多样化的执行环境。

虚拟机的定义：是指在一个硬件平台上模拟多个高效的、独立的和实际硬件相同的虚拟硬件系统，在每个虚拟硬件系统上都可以运行不同的客户操作系统

(Guest OS)。**Guest OS** 通过虚拟机监控器访问实际的物理资源。因此操作系统和应用程序在虚拟机中的运行方式与它们在物理服务器上的运行方式没有区别。

虚拟机和物理机的本质区别：与物理服务器相比，虚拟机不是由真实的电子元件组成，而是由一组虚拟组件（文件）组成，这些虚拟组件与物理服务器的硬件配置无关，与物理服务器相比，虚拟机具有以下优势：

- 抽象解耦：可以在任何架构的服务器上运行。虚拟机的运行环境不再受限于具体硬件配置，可以进行灵活的配置和迁移。
- 分区隔离：多个虚拟机可以同时运行，同时虚拟机之间还实现数据处理、网络连接和数据存储的安全隔离。
- 封装移动：可以将整个虚拟机系统（包括虚拟硬件、操作系统和配置好的应用程序）封装于一个或多个文件之中，通过简单的文件复制实现快速部署、备份及还原，同时还可以在不同的物理服务器之间进行迁移，甚至可以在虚拟机正在运行的情况下进行迁移。
- 弹性扩展：可以对单个物理服务器上的虚拟资源（虚拟 **CPU**、虚拟网卡、虚拟磁盘等）在不关闭虚拟机的情况下，进行按需动态扩展。

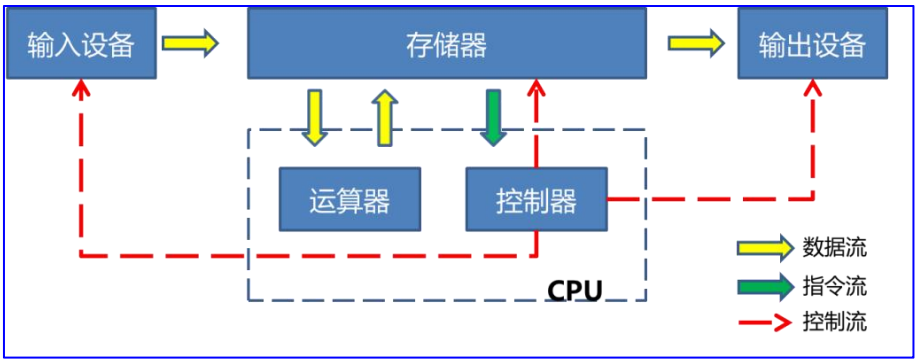


图 **3.1–2** 冯•诺依曼体系结构

如上图所示，冯•诺依曼体系结构中计算机被划分为 **CPU**（运算器和控制器）、

存储器和输入输出设备三个模块,相应的 VMM 对物理资源的虚拟可以划分为三个部分：CPU 虚拟化、内存虚拟化和 I/O 设备虚拟化，其中以 CPU 虚拟化最为关键。

3.1.1 CPU 虚拟化

VMM (Virtual Machine Monitor)对物理资源的虚拟可以划分为三个部分：CPU 虚拟化、内存虚拟化和 I/O 设备虚拟化。

3.1.1.1 经典 CPU 虚拟化方法

现代计算机体系结构一般至少有两个特权级（即用户态和核心态，如 X86 架构的 CPU 有四个特权级 Ring0~ Ring3）用来分隔系统软件和应用软件。那些只能在处理器的最高特权级（内核态）执行的指令称之为特权指令，一般可读写系统关键资源的指令（即敏感指令）绝大多数都是特权指令。如果执行特权指令时处理器的状态不在内核态,通常会引发一个异常而交由系统软件来处理这个非法访问（陷入）。经典的虚拟化方法就是使用“特权解除”和“陷入–模拟”的方式，即将 Guest OS 运行在非特权级，而将 VMM 运行于最高特权级（完全控制系统资源）。解除了 Guest OS 的特权级后，Guest OS 的大部分指令仍可以在硬件上直接运行，只有执行到特权指令时，才会陷入到 VMM 模拟执行（陷入–模拟）。“陷入–模拟”的本质是保证可能影响 VMM 正确运行的指令由 VMM 模拟执行，大部分的非敏感指令还是照常运行。

3.1.1.2 X86 架构虚拟化方法

X86 体系结构拥有四个特权级以分隔系统软件和应用软件，如下图所示。其中，内核在 Ring 0 级，应用程序在 Ring 3 级。依照经典的 CPU 虚拟化技术，

在引入虚拟化之后，在全虚拟化和半虚拟化场景中，**VMM** 接管系统最高权限，由此 **VMM** 在 **Ring 0** 级，内核迁移至 **Ring 1** 级，如下图所示。

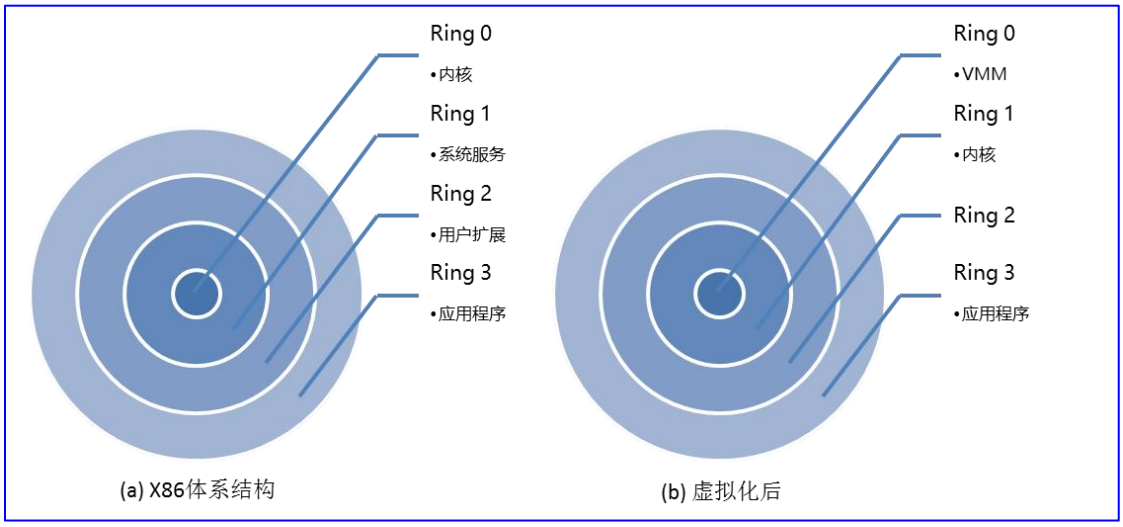


图 **3.1.1.2–1 X86** 体系结构

❖ 特权级

在虚拟化技术中，具有以下功能的指令被认为是敏感指令：操作特权资源、修改虚拟机的运行模式、修改物理机的运行状态、读写敏感的寄存器和内存、访问存储保护系统、内存系统或地址重定位系统以及所有 **I/O** 指令。但是 **X86** 架构中存在若干敏感指令是非特权指令的情况，即敏感指令中只有一部分属于特权指令，如下图所示在传统的计算机体系架构中，只有特权指令才能在最高特权级运行。由此，虚拟机调用了某些会影响正常运行的敏感指令便不会陷入到 **Ring 0** 级，而是由 **VMM** 模拟执行，从而产生“虚拟化漏洞”。

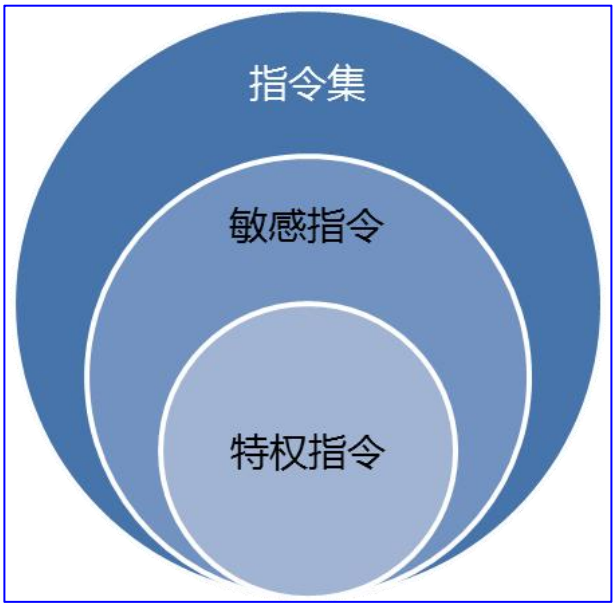


图 **3.1.1.2–2** 特权级

为解决以上“虚拟化漏洞”问题，全虚拟化技术在执行时将 **VM** 上执行的 **Guest OS** 指令翻译成 **X86** 指令集，其某一子集中的敏感指令被替换成陷入指令，陷入到 **Ring 0** 模式执行；半虚拟化技术通过修改 **Guest OS** 的代码，将含有敏感指令的操作，替换为对 **VMM** 的超调用 **Hypercall**，将控制权转移到 **VMM**；硬件辅助虚拟化技术将非根模式下所有敏感指令重新定义，使它们能不经虚拟化直接运行或通过“陷入再模拟”的方式处理，在模式切换过程中，上下文的保存恢复由硬件来完成，极大提高了切换效率。

而在硬件辅助虚拟化场景中，其基本思想就是引入新的处理器运行模式和新的指令，使得 **VMM** 和 **Guest OS** 运行于不同的模式下，**Guest OS** 运行于受控模式，原来的一些敏感指令在受控模式下全部会陷入 **VMM**，这样就解决了部分非特权的敏感指令的“陷入–模拟”难题，而且模式切换时上下文的保存恢复由硬件来完成，这样就大大提高了“陷入–模拟”时上下文切换的效率。以 **Intel VT–x** 技术为例，该技术增加了在虚拟状态下的两种处理器工作模式，即根模式和非根

模式。**Guest OS** 在非根模式的 **Ring 0** 级，**VMM** 在根模式下。通过硬件辅助的方式，尽可能少地在客户机和 **VMM** 之间切换，减少上下文切换的开销，并减少 **VMM** 的模拟任务。

图 **3.1.1.2–3** 根模式和非根模式

3.1.2 内存虚拟化

操作系统对内存有两点认识，首先，内存都是从物理地址 **0** 开始的；其次，内存是连续的，或者至少在一些大的粒度（如 **256M**）上连续。但是在虚拟化环境下，物理内存被多个客户机操作系统同时使用，物理地址 **0** 只有一个，可以保证为每台虚拟机分配连续内存，却牺牲了使用效率。而 **VMM** 需要“欺骗”客户机操作系统，使客户机操作系统认为其对内存的两点要求都还满足。这种欺骗的过程，即内存虚拟化。

内存虚拟化的核心，即引入一层新的地址空间：客户机物理地址空间，如图所示：

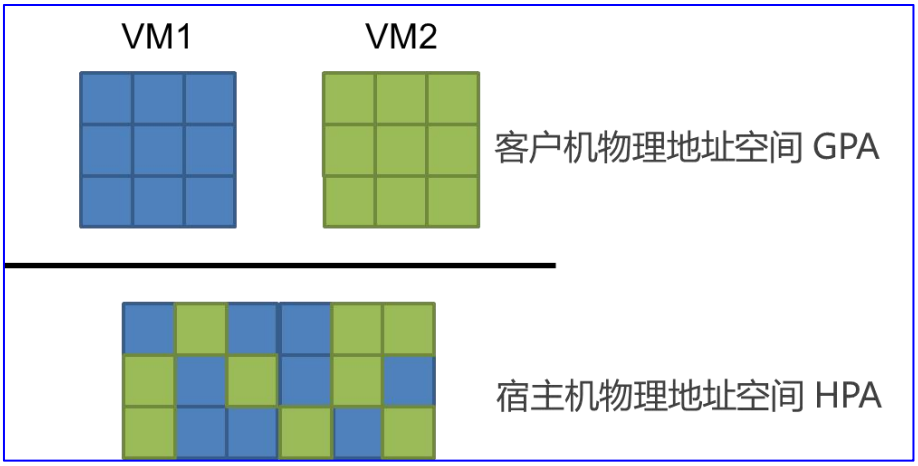


图 **3.1.2–1** 客户机物理地址空间

由此，内存虚拟化前两层地址映射关系扩展为三层地址映射关系，如下图所示：

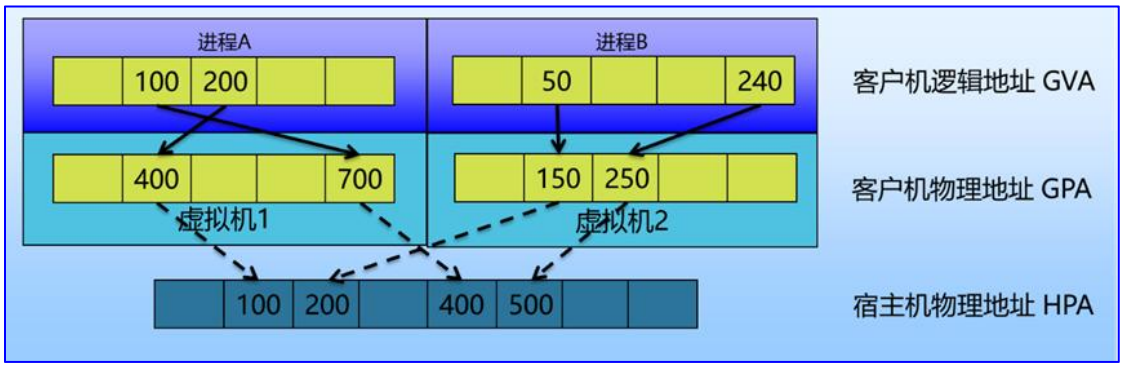


图 **3.1.2–2** 三层地址映射关系

内存虚拟化要处理好以下两个问题：首先，给定一个虚拟机，维护客户机物理地址到宿主机物理地址之间的映射关系；其次，截获虚拟机对客户机物理地址的访问，并根据所记录的映射关系，将其转换至宿主机物理地址。

在全虚拟化技术中，**VMM** 负责了客户机物理地址到宿主机物理地址间的映射；半虚拟化技术中，**KVM** 将客户机物理地址到机器地址的翻译表（**P2M** 表）暴露给客户操作系统，从 **VMM** 中获得属于本虚拟机的内存空间信息；在硬件辅助虚拟化技术中，以 **Intel–EPT** 技术为例，通过硬件实现的扩展页表（**Extended Page Table**）技术，加速实现了客户机逻辑地址到宿主机物理地址的转换，如下

图所示。

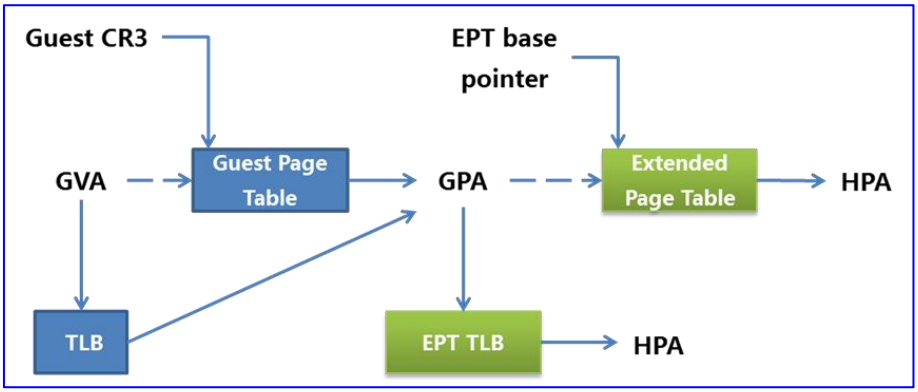


图 3.1.2–3 扩展页表

3.1.3 I/O 设备虚拟化

在虚拟环境中，I/O 设备面临着有限的外设资源与多个客户操作系统对外设访问需求的矛盾。而 VMM 则需要截获客户机操作系统对外设的访问请求，以软件或硬件辅助的方式模拟真实物理设备的效果。

设备模拟的方式上，全虚拟化技术使用软件精确模拟与物理设备完全一样的接口，客户操作系统驱动无须修改就能驱动这个虚拟设备；半虚拟化技术通过修改客户操作系统，使前后端相互协作，提供更加高效的 I/O 虚拟化；而硬件辅助虚拟化技术直接将物理设备分配给某个客户操作系统，由客户操作系统直接访问 I/O 设备不经过 VMM。

VMM 通过 I/O 虚拟化来复用有限的外设资源，其通过截获 Guest OS 对 I/O 设备的访问请求，然后通过软件模拟真实的硬件，目前 I/O 设备的虚拟化方式主要有三种：

1、设备接口完全模拟：

即软件精确模拟与物理设备完全一样的接口，Guest OS 驱动无须修改就能驱动这个虚拟设备，VMware 即使用该方法。

优点：没有额外的硬件开销，可重用现有驱动程序。

缺点：为完成一次操作要涉及到多个寄存器的操作，使得 VMM 要截获每个寄存器访问并进行相应的模拟，这就导致多次上下文切换；由于是软件模拟，性能较低。

2、前端／后端模拟：

VMM 提供一个简化的驱动程序（后端，Back-End），Guest OS 中的驱动程序为前端(Front-End，FE),前端驱动将来自其他模块的请求通过与 Guest OS 间的特殊通信机制直接发送给 Guest OS 的后端驱动，后端驱动在处理完请求后再发回通知给前端，KVM 即采用该方法。

优点：基于事务的通信机制，能在很大程度上减少上下文切换开销，没有额外的硬件开销。

缺点：需要 VMM 实现前端驱动，后端驱动可能成为瓶颈。

3、直接映射：

即直接将物理设备分配给某个 Guest OS,由 Guest OS 直接访问 I/O 设备(不经 VMM），目前与此相关的技术有 IOMMU（Intel VT-d、SR-IOV 等），旨在建立高效的 I/O 虚拟化直通道。

优点：可重用已有驱动，直接访问减少了虚拟化开销。

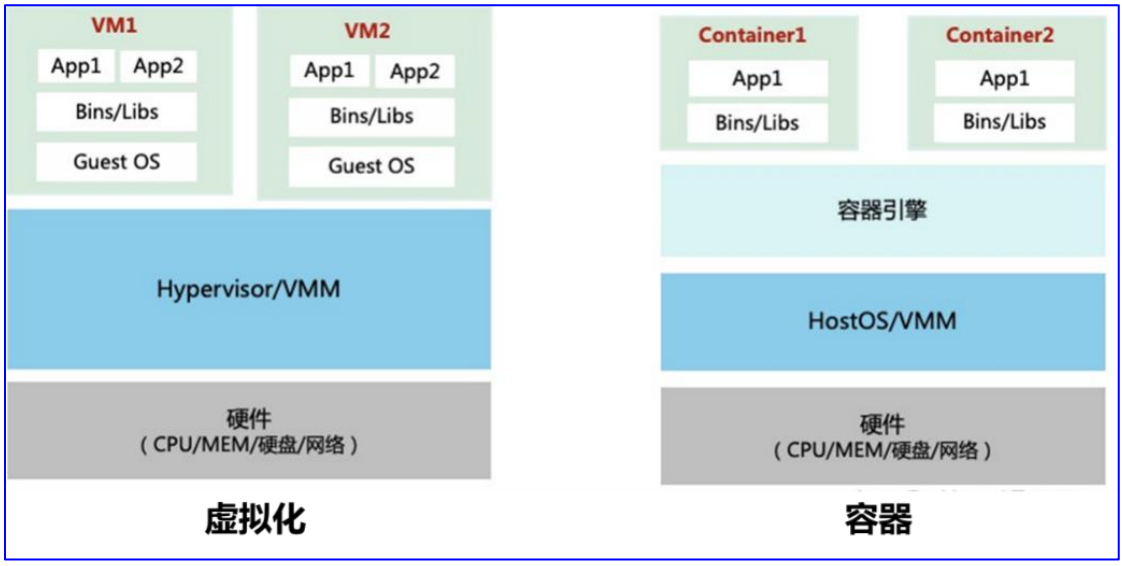
缺点：需要购买较多额外的硬件。

3.2 容器核心技术与原理

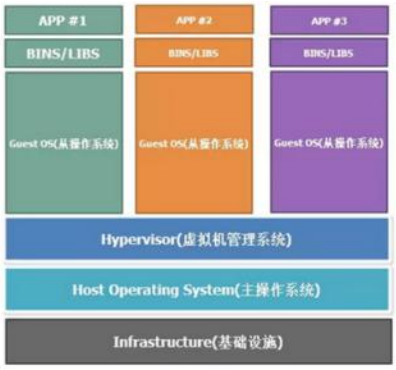
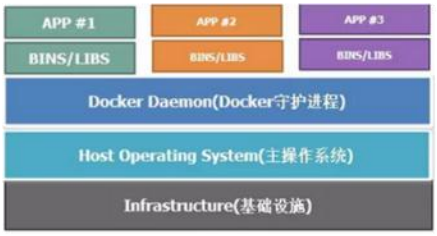
容器主要依托两种技术：Namespace 和 CGroups、Namespace 提供了不同类型资源的隔离，CGroups 则针对相应的资源进行了限制分配和统计。

3.2.1 容器与虚拟化

虚拟化技术通过 **Hypervisor** 实现虚拟机与底层硬件的解耦，容器（**container**）技术是一种更加轻量级的操作系统虚拟化技术，将应用程序及其运行依赖环境打包封装到标准化、强移植的镜像中，通过容器引擎提供进程隔离、资源可限制的运行环境，实现应用与 **OS** 平台及底层硬件的解耦，一次打包，随处运行。容器基于镜像运行，可部署在物理机或虚拟机上，通过容器引擎与容器编排调度平台实现容器化应用的生命周期管理。

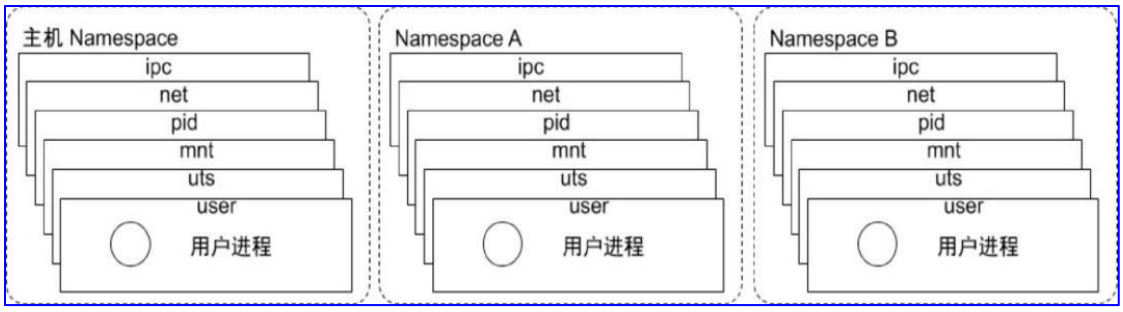


虚拟机中包含 **GuestOS**，调度与资源占用都比较重。而容器仅包含应用运行所需的文件，管理容器就是管理应用本身。容器具有极其轻量、秒级部署、易于移植、敏捷弹性伸缩等优势；但虚拟机是 **OS** 系统级隔离，而容器是进程级隔离，容器的安全性相对更弱一些，需要一些额外的安全技术或安全容器方案来弥补。作为云原生的核心技术，容器、微服务与 **DevOps/ CICD**）等技术已成为应用架构转型或实现技术中台不可或缺组件。

架构	虚拟机	容器
		
启动	分钟级	秒级
资源占用量	大	小（没有臃肿的从操作系统）
内存占用	一般为GB	一般为MB
运行密度	几个	单机支持上百个容器
拥有独立操作系统	是	否，共用宿主机内核
隔离性	更强（虚拟硬件级全隔离）	文件系统、进程及网络隔离
监控粒度	资源级	应用级

3.2.2 Namespace

Linux 内核从 **2.4.19** 版本开始引入 **Namespace** 技术，译为“命名空间”，它提供了一种内核级别的系统资源的隔离方式。系统可以为进程分配不同的 **Namespace**，并保证不同的 **Namespace** 资源独立分配、进程彼此隔离，即不同的 **Namespace** 下的进程互不干扰。



如上图所示，**Linux** 内核支持 **6** 种 **Namespace**。主机有其默认的 **Namespace**，

行为与用户的 **Namespace** 完全一致，不同进程可以通过不同的 **Namespace** 进行隔离。当系统启动一个容器时，会为该容器创建相应的不同类型的 **Namespace**，为运行在容器内的进程提供相应的资源隔离。

3.2.2.1 IPC

进程间通信（**Interprocess Communication**）是 **Linux** 系统中常见的、进程与进程直接通信的手段。

IPC Namespace 用于隔离 **IPC** 资源，包含 **System V IPC** 对象和 **POSIX** 消息队列。其中 **System V IPC** 对象包含信号量、共享内存和消息队列，用于进程间的通信。**System V IPC** 对象具有全局唯一的标识，对在该 **IPC Namespace** 内的进程可见，而对其外的进程不可见。当 **IPC Namespace** 被销毁后，所有的 **IPC** 对象也会被自动销毁。

Kubernetes 允许用户在 **Pod** 中使用 **hostIPC** 进行定义，通过该属性使授权用户容器共享主机 **IPC Namespace**，达到进程间通信的目的。

3.2.2.2 Network

Network Namespace 提供了关于系统上网络资源的隔离，例如网络设备、**IPv4** 和 **IPv6** 协议栈、**IP** 路由表、防火墙规则、**/proc/net** 目录（**/proc/pid/net** 目录的符号链接）、**/sys/class/net** 目录、**/proc/sys/net** 目录下的很多文件、端口号（**socket**）等。一个物理的网络设备通常会被放到主机的 **Network Namespace**（就是系统初始的 **Network Namespace**）中。

不同网络的 **Namespace** 由网络虚拟设备（**Virtual Ethernet Device**，即 **VETH**）连通，再基于网桥或者路由实现与物理网络设备的连通。当网络 **Namespace** 被

释放后，对应的 **VETH Pair** 设备也会被自动释放。

在 **Kubernetes** 中，同一 **Pod** 的不同容器共享同一网络的 **Namespace**，没有例外。这使得 **Kubernetes** 能将网络挂载在更轻量、更稳定的 **sandbox** 容器上，而用户定义的容器只需复用已配置好的网络即可。另外，同一 **Pod** 的不同容器中运行的进程可以基于 **localhost** 彼此通信，这在多容器进程、彼此需要通信的场景下是非常有效的。

3.2.2.3 PID

PID Namespace 用于进程号隔离，不同 **PID Namespace** 中的进程 **PID** 可以相同。容器启动后，**Entrypoint** 进程会作为 **PID** 为 **1** 的进程存在，因此是该 **PID Namespace** 的 **init** 进程。它是当前 **Namespace** 所有进程的父进程，如果该进程退出，内核会对该 **PID Namespace** 的所有进程发送 **SIGKILL** 信号，以便同时结束它们。**init** 进程默认屏蔽系统信号，即除非该进程对系统信号做特殊处理，否则发往该进程的系统信号默认都会被忽略。不过 **SIGKILL** 和 **SIGSTOP** 信号比较特殊，**init** 进程无法捕获这两个信号。

Kubernetes 默认对同一 **Pod** 的不同容器构建独立的 **PID Namespace**，以便将不同容器的进程彼此隔离，同时允许通过 **ShareProcessNamespace** 属性设置不同容器的进程共享 **PID Namespace**。

Kubernetes 支持多重容器进程的重启策略，默认行为是用户进程退出后立即重启。**Kubernetes** 用户只需中止其容器中的 **Entrypoint** 进程，即可实现容器重启。

3.2.2.4 Mount

Mount Namespace 提供了进程能看到的挂载点的隔离。在主机上，通过 **/proc/[pid]/mounts**、**/proc/[pid]/mountinfo**、**/proc/[pid]/mountstats** 等文件来查看挂载点。在容器内，可以通过 **mount** 或 **lsmnt** 命令查看 **MountNamespace** 中的有效挂载点。

不同于其他类型的 **Namespace** 的严格隔离，**Linux** 内核针对 **MountNamespace** 隔离性开发了共享子树（**Shared Subtree**）功能，用于在不同 **Mount Namespace** 之间自动可控地传播 **mount** 和 **umount** 事件。共享子树引入了对等组的概念，对等组是一组挂载点，其成员之间互相传播 **mount** 和 **umount** 事件。此特性使得当主机磁盘发生变更（比如系统导入新磁盘）时，只需在一个 **Mount Namespace** 中进行挂载，该磁盘即可在所有 **Namespace** 中可见。

挂载点可以设置的传播类型如下：

MS_SHARED：此挂载点与同一对等组里其他挂载点共享 **mount** 和 **umount** 事件。在此挂载点下添加或删除挂载点时，事件会传播到对等组内的其他 **Namespace** 中，事件在 **Namespace** 中也会自动进行相同的 **mount** 或 **umount** 操作。同样地，对等组中其他挂载点上的挂载和卸载事件也会传播到此挂载点。

MS_PRIVATE：此挂载点是私有点，没有对等组，**mount** 和 **umount** 事件不会与其他的 **Mount Namespace** 共享。

MS_SLAVE：该挂载点可以从主对等组接收 **mount** 和 **umount** 事件，但是在本挂载点下的 **mount** 和 **umount** 事件不会传播到任何其他的 **MountNamespace** 下。

MS_UNBINDABLE：与 **MS_PRIVATE** 不同的是，该挂载点不可以执行

bindmount 操作。

在目录**/proc/[pid]/mountinfo** 下，可以看到该 **PID** 所属的 **mountNamespace** 下的挂载点的传播类型及所属的对等组。

在 **Kubernetes** 中，挂载点通常是 **private** 类型。如果需要设置挂载点的类型，那么可以在 **Pod** 的 **spec** 中填写相应的挂载配置。

3.2.2.5 UTS

UTS（“**UNIX Time-Sharing System**”）**Namespace** 允许不同容器拥有独立的 **hostname** 和 **domain name**。**UTS Namespace** 中的一个进程可以看作一个在网络上独立存在的节点。也就是说，除 **IP** 外，还能通过主机名进行访问。

3.2.2.6 USR

User Namespace 主要隔离了安全相关的标识符和属性，比如用户 **ID**、用户组 **ID**、**root** 目录、密钥等。一个进程的用户 **ID** 和组 **ID** 在 **User Namespace** 内外可以有所不同。在该 **User Namespace** 外，它是一个非特权的用户 **ID**；而在 **User Namespace** 内，进程可以使用 **0**（**root**）作为用户 **ID**，且其具有完全的特权权限。

User Namespace 允许不同容器有独立的 **user** 和 **group ID**，它主要提供两种职能：权限隔离和用户身份标识隔离。我们可以通过在容器镜像中创建和切换用户，来为文件目录设置不同的用户权限，从而实现容器内的权限管理，而无须影响主机配置。

3.2.2.7 CGroup

Linux 内核从 **4.6** 版本开始支持 **CGroup Namespace**。如果容器启动时没有

开启 **CGroup Namespace**，那么在容器内部查询 **CGroup** 时，返回整个系统的信息；而开启 **CGroup Namespace** 后，可以看到当前容器以根形式展示的单独的 **CGroup** 信息：

```
root@test# cat /proc/1/cgroup
12:pids:/
10:blkio:/
9:cpuset:/
7:freezer:/
5:hugetlb:/
4:net_cls,net_prio:/
3:devices:/
2:cpu,cpuacct:/
1:name=systemd:/
```

CGroup 视图的改变使容器更加安全，而且在容器内也可以有自己的 **CGroup** 结构。

3.2.3 Cgroups

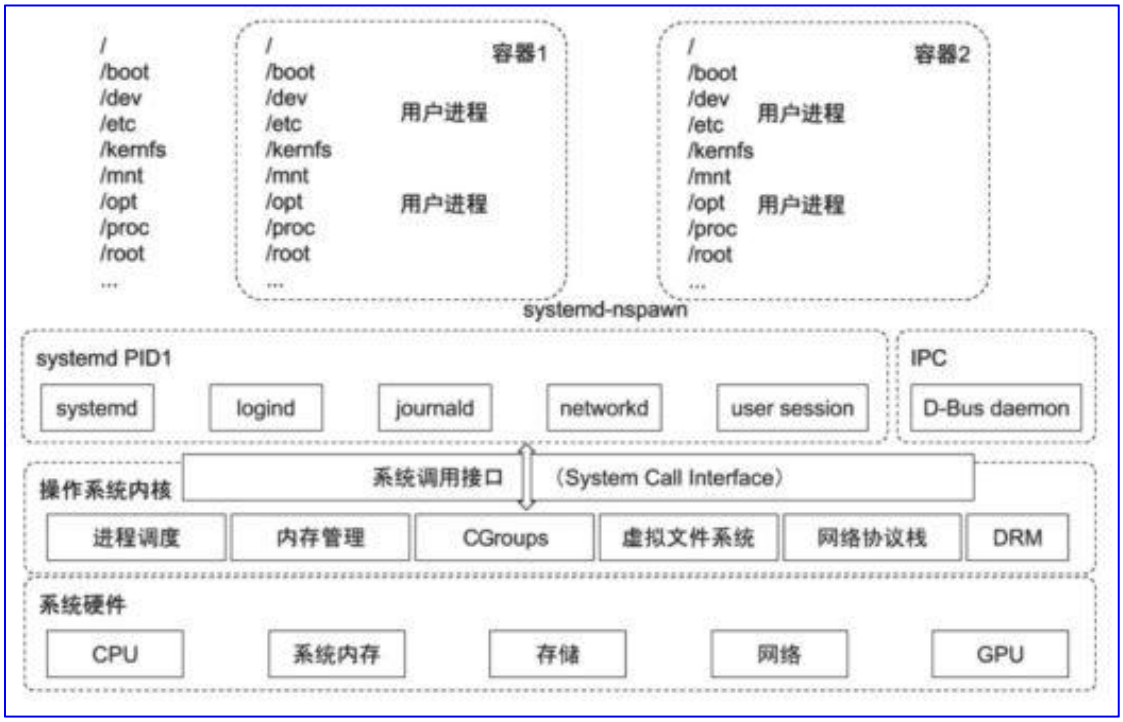
CGroups（**Control Groups**）是 **Linux** 下用于对一个或一组进程进行资源控制和监控的机制。利用 **CGroups** 可以对诸如 **CPU** 使用时间、内存、磁盘 **I/O** 等进程所需的资源进行限制。**Kubernetes** 允许用户为 **Pod** 的容器申请资源，当容器在计算节点上运行起来时，可以通过 **CGroups** 来完成资源的分配和限制。

在 **CGroups** 中，对资源的控制都是以 **CGroup** 为单位的。目前 **CGroups** 可以控制多种资源，不同资源的具体管理工作由相应的 **CGroup** 子系统（**Subsystem**）来实现。因此，针对不同类型的资源限制，只要将限制策略在不同的的子系统上进行关联即可。**CGroups** 被引入 **2.6.24** 版本的内核中，最初只对 **CPU** 进行了资源限制，然而随着对其他资源控制需求的增多，它们的 **CGroup** 子系统不断地被引入内核。在容器时代，特别是 **Kubernetes** 中，为了提高对节点资源的利用率，一个节点上会运行尽可能多的容器，这就对资源隔离的多样性和精确性提出了越

来越高的要求，因此 **CGroups** 也发挥着越来越重要的作用。

对于 **CGroups** 的组织管理，用户可以通过文件操作来实现，对资源的控制可以细化到线程级别。**CGroups** 在不同的系统资源管理子系统中以层级树（**Hierarchy**）的方式来组织管理：每个 **CGroup** 都可以包含其他的子 **CGroup**，因此子 **CGroup** 能使用的资源，除了受本 **CGroup** 配置的资源参数限制，还受到父 **CGroup** 设置的资源限制。

Kubernetes 1.18 之前的版本主要使用了 **CPU**、**cpuset**、**memory** 子系统，而 **blkio**、**PID** 子系统可以根据需求选择性地开启使用。



3.2.3.1 CPU

CPU 子系统用于限制进程的 **CPU** 使用时间。在 **CPU** 子系统中，对于每个 **CGroup** 下的非实时任务，**CPU** 使用时间可以通过 **cpu.shares**、**cpu.cfs_period_us** 和 **cpu.cfs_quota_us** 参数来进行控制，而系统的 **CFS**（**Completely Fair Scheduler**）调度器则根据 **CGroup** 下进程的优先级、权重和 **cpu.shares** 等配置来给该进程分

配相应的 **CPU** 时间。

3.2.3.2 cpuacct

cpuacct 用于统计 **CGroup** 及其子 **CGroup** 下进程的 **CPU** 的使用情况。

1、cpuacct.usage 包含该 **CGroup** 及其子 **CGroup** 下进程使用 **CPU** 的时间，单位是 **ns**（纳秒）。

2、cpuacct.stat 包含该 **CGroup** 及其子 **CGroup** 下进程使用的 **CPU** 时间，以及用户态和内核态的时间。

3.2.3.3 Cpuset

cpuset 为 **CGroups** 的进程分配单独的 **CPU** 和内存节点，将进程固定在某个 **CPU** 或内存节点上，以达到提高性能的目的。

3.2.3.4 Memory

memory 用于限制 **CGroup** 下进程的内存使用量，亦可获取到内存的详细使用信息。

1、memory.stat

该文件中包含该 **CGroup** 下进程的详细的内存使用信息

CGroup 下进程的内存使用信息

内存类型	说明
cache	页缓存（Page Cache），包含 tmpfs（shmem），单位是 Byte（字节）
rss	匿名页缓存（Anonymous Page Cache）和可交换页缓存（Swap Page Cache），不包含 tmpfs(shmem)
shmem	tmpfs
mapped_file	内存映射文件的大小，包含 tmpfs（shmem）
dirty	脏页个数统计
writeback	writeback 次数

内存类型	说明
pgpgin	缓存进内存的页个数
pgpgout	从内存中移出的页个数
pgfault	产生缺页的个数，包含 major 缺页和 minor 缺页
pgmajfault	产生 major 缺页的个数
inactive_anon	不活跃的匿名页和可交换页
active_annon	活跃的匿名页和可交换页
inactive_file	不活跃的文件映射页
active_file	活跃的文件映射页
unevictable	不可回收的内存
hierarchical_memory_limit	memory CGroup 下的内存限制

2、memory.usage_in_bytes

CGroup 下进程使用的内存，包含 **CGroup** 及其子 **CGroup** 下的进程使用的内存。

3、memory.max_usage_in_bytes

CGroup 下进程使用内存的最大值，包含子 **CGroup** 的内存使用量。

4、memory.limit_in_bytes

设置 **CGroup** 下进程最多能使用的内存。如果设置为 **–1**，则表示对该 **CGroup** 的内存使用不做限制。

5、memory.failcnt

CGroup 下的进程达到内存最大使用限制的次数。

6、memory.force_empty

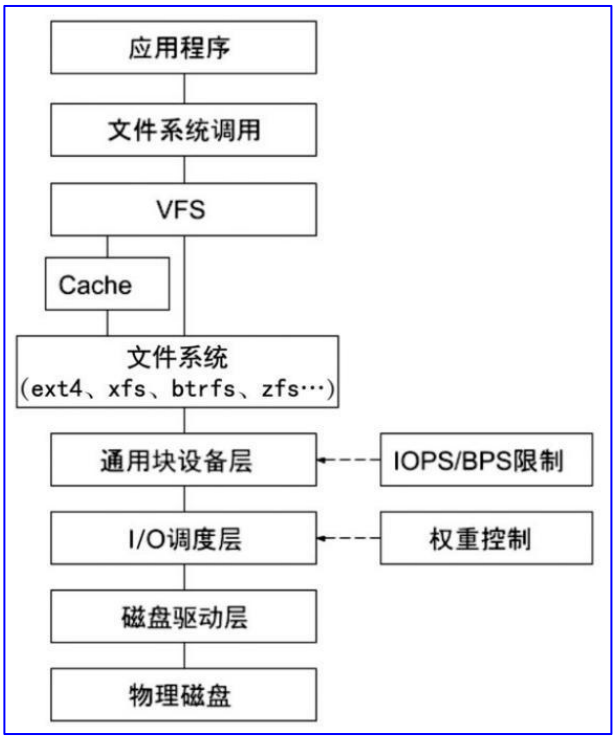
当没有进程属于该 **CGroup** 后，将该值设置为 **0**，系统会尽可能地将该 **CGroup** 使用的内存释放掉。对于不能释放的内存，则会将其移动到父 **CGroup** 上。在 **CGroup** 销毁之前，将能释放的内存释放掉，可以尽量避免将已经不使用的内存移动到父 **CGroup** 上，从而避免对运行在父 **CGroup** 中的进程造成内存压力。

7、memory.oom_control

设置是否在 **CGroup** 中使用 **OOM**（**Out of Memory**）**Killer**，默认为使用。当属于该 **CGroup** 的进程使用的内存超过最大的限定值时，会立刻被 **OOMKiller** 处理。

3.2.3.5 blkio

blkio 子系统用来实现对块设备访问的 **I/O** 控制，按权重分配目前有两种限制方式：一是限制每秒写入的字节数（**Bytes Per Second**，即 **BPS**），二是限制每秒的读写次数（**I/O Per Second**，即 **IOPS**）。**blkio** 子系统按权重分配模式工作于 **I/O** 调度层，依赖于磁盘的 **CFQ**（**Completely Fair Queuing**，完全公平算法）调度，如果磁盘调度使用 **deadline** 或者 **none** 的算法则无法支持。**BPS**、**IOPS** 工作于通用设备层，不依赖于磁盘的调度算法，因此有更多的适用场景。



3.2.3.6 PID

PID 子系统用来限制 **CGroup** 能够创建的进程数。

●**pids.max**：允许创建的最大进程数量。

●**pids.current**：当前的进程数量。

3.2.3.7 其他

CGroup 还支持如下子系统：

● **devices** 子系统，控制进程访问某些设备。

● **perf_event** 子系统，控制 **perf** 监控 **CGroup** 下的进程。

● **net_cls** 子系统，标记 **CGroups** 中进程的网络数据包，通过 **TC** 模块（**Traffic Control**）对数据包进行控制。

● **net_prio** 子系统，针对每个网络设备设置特定的优先级。

● **hugetlb** 子系统，对 **hugepage** 的使用进行限制。

●**freezer** 子系统，挂起或者恢复 **CGroups** 中的进程。

● **ns** 子系统，使不同 **CGroups** 下面的进程使用不同的 **Namespace**。

● **rdma** 子系统，对 **RDMA/IB−spe−cific** 资源进行限制。

4 重点功能和实现原理

4.1 云主机服务

4.1.1 云主机全生命周期管理

4.1.1.1 功能描述

(1) 创建云主机

- 支持基于 OVA、 ISO、QCOW2、RAW 格式镜像创建云主机。
- 支持使用镜像、备份、模板来创建云主机。
- 云主机继承镜像文件所属的 CPU 架构、操作系统、启动引导固件属性。
- 系统盘配置支持设置系统盘的大小、是否加锁、磁盘接口类型。
- 网络配置支持经典网络和 VPC 网络，支持设置网卡连接配置
- 支持在创建云主机时添加 GPU 设备
- 支持在创建云主机时对选择的 VPC 的网络配置安全组，也可以不选择安全组。
- 支持选择镜像库和全局内容库中的 ISO、OVA、QCOW2、RAW 格式镜像文件创建云主机。
- 支持在创建时根据镜像文件的 CPU 架构、操作系统配置云主机的 CPU 架构、操作系统；虚拟数据中心、用户、资源池、主机、存储池、到期时间、到期处理策略、崩溃处理策略、开启防病毒、是否加入保护组、CPU 模式、软驱、描述信息等。

(2) 批量创建云主机

- 支持导入 **Excel** 批量创建云主机。
- 支持下载 **Excel** 模版。
- 支持手动选择、添加云主机属性(名称、资源池、虚拟数据中心、用户、镜像文件、目标镜像库、**CPU** 架构、 主机、存储池、操作系统、**CPU**、内存、网卡、磁盘)创建云主机。
- 镜像文件支持从全局内容库中选择。
- 支持选择 **OVA**、**ISO**、**QCOW2**、**RAW** 格式的镜像文件。
- 网络支持经典网络和 **VPC** 网络。
- **ISO**、**QCOW2**、**RAW** 创建云主机支持 **GPU** 设备绑定
- 磁盘支持 **IDE**、**SCSI**、**VIRTIO** 磁盘接口。
- 批量创建云主机列表中支持云主机数据的复制、删除。
- 批量创建云主机只支持系统管理员进行操作。

(3) 纳管云主机

- 云主机纳管功能可以将底层虚拟化资源池正在使用的云主机纳管到云平台中来进行管理

(4) 云主机开机、关闭、强制关闭、重启、强制重启

(5) 分配、回收云主机

- 系统管理员分配时，若不分配 **VDC**，则必须分配用户，用户列表为所有的系统管理员，默认为当前系统管理员；若分配 **VDC**，可以不分配用户，分配时可以选择所选 **VDC** 内的 **VDC** 管理员或 **VDC** 用户。
- **VDC** 管理员仅支持分配用户，可选用户为当前所在 **VDC** 内的 **VDC** 管

理员和 **VDC** 用户。

- 系统管理员回收时，不允许回收自己的云主机；回收后，所属 **VDC** 变更为空，所属用户变更为当前操作用户。
- **VDC** 管理员回收时，不允许回收自己的云主机；回收后，所属 **VDC** 变更为 **VDC** 管理员所在 **VDC**，所属用户变更为当前操作用户。

(6) 移除、删除云主机

- 移除云主机,仅删除当前云管平台层面的数据,并不会删除底层云主机。
- 选择从存储池删除云硬盘和彻底清除数据时,将物理删除该云主机的底层磁盘文件，该操作不可恢复，请谨慎操作!

(7) 清理云主机

- 实际已在底层已被删除的云主机，云管平台中会将其更新为“丢失”状态，可通过此操作一键清理所有“丢失”状态的云主机。

(8) 打开控制台

- 在新窗口中打开控制台页面。通过 **web** 控制台界面，可以直接对云主机进行控制，无需再使用其他工具与云主机连接。

(9) 重置密码

- 在云主机使用过程中,用户可能需要在不影响正常业务的情况下修改云主机密码,并且一台云主机可能有多个用户角色权限,在修改某个用户密码时,不能影响其他用户正常访问,为此云平台开发了在线修改云主机用户密码功能。

(10) 云主机 VMTools

- 在云主机的生命周期管理中，虚拟化层需要与云主机 **Guest OS** 交互以

实现个性化定制、高级管理功能。

(11) 导出镜像

- 云主机处于“关闭”状态时，支持导出 **OVA** 镜像。创建云主机使用导出镜像后的 **ova** 模板，默认将磁盘中的引导盘设置为加锁。

(12) 迁移云主机

- 支持更改主机：即云主机计算迁移，将云主机迁移到另一台主机上；
- 支持更改存储：即云主机存储迁移，将云主机的存储移动到另一数据存储；
- 支持更改主机和存储：即云主机整机迁移，将云主机迁移到另外一台主机上，并将其存储器移动到另一数据存储。

(13) 克隆云主机

- 使客户能够以现有云主机为模板，快速复制出相同的虚拟机，也可以先创建一台云主机进行软件和系统配置，然后以该云主机为模板批量的创建出相同规格、属性和状态的云主机，实现了对虚拟机数据、属性以及状态的快速复制

(14) 开启/关闭防病毒

- 云主机防病毒主要是指宿主机集成了生态相关防病毒软件，创建虚拟机时可以选择开启防病毒，这样创建出来的虚拟机也是具备防病毒功能的。也可以对现有的云主机开启、关闭防病毒。

(15) 云主机快照、恢复

- 通过云主机快照、内存快照与还原功能可以实现云主机中硬盘信息的安全快照。虚拟机生命周期中，通过快照和还原操作，可以对云主机数据

安全保存和恢复。

(16) 云主机暂停

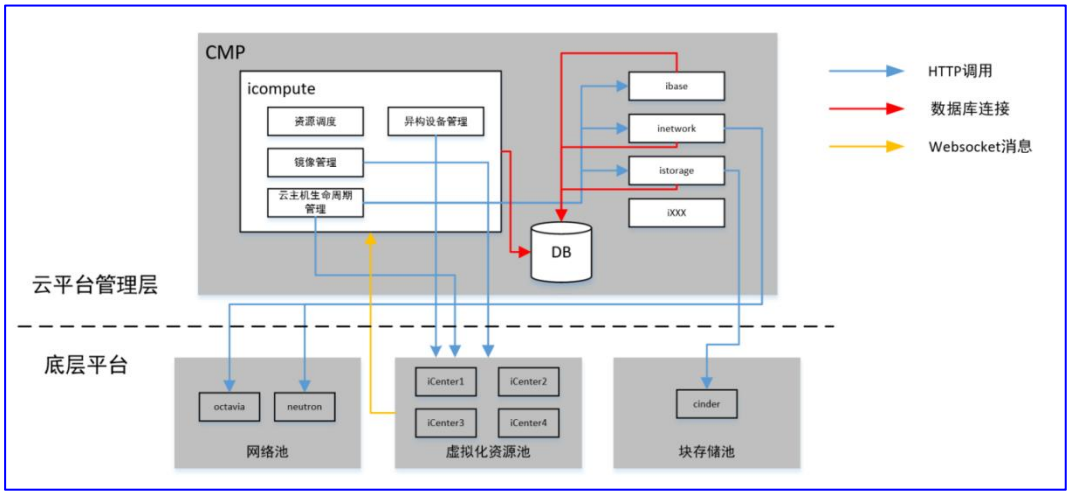
- 开机状态的云主机(容灾云主机和服务实例云主机除外)支持暂停操作，暂停后的云主机只支持恢复、分配、回收、移除操作，并且无法打开控制台，支持批量操作。

(17) 云主机恢复

- 暂停状态的云主机支持恢复操作，支持批量操作。
- 备份恢复为原云主机或备份恢复为新云主机
- 关机状态的云主机支持使用备份恢复为原云主机;使用备份恢复为新云主机时不受原源主机状态的影响。

4.1.1.2 技术原理

产品基于用户需求为云主机提供生命周期管理，主要基于如下层次：



- **CMP icompute** 模块对外统一提供标准化接口。接收和响应用户的云主机管理请求。
- 在云主机生命周期操作时，云平台生成云主机操作任务，形成 **WebSocket** 消息通知，实时更新操作过程中的云主机状态。

- **icompute** 通过调用 **SDK** 调用底层资源池接口实现云主机的生命周期管理。

4.1.1.3 约束

无

4.1.2 云主机创建

4.1.2.1 功能描述

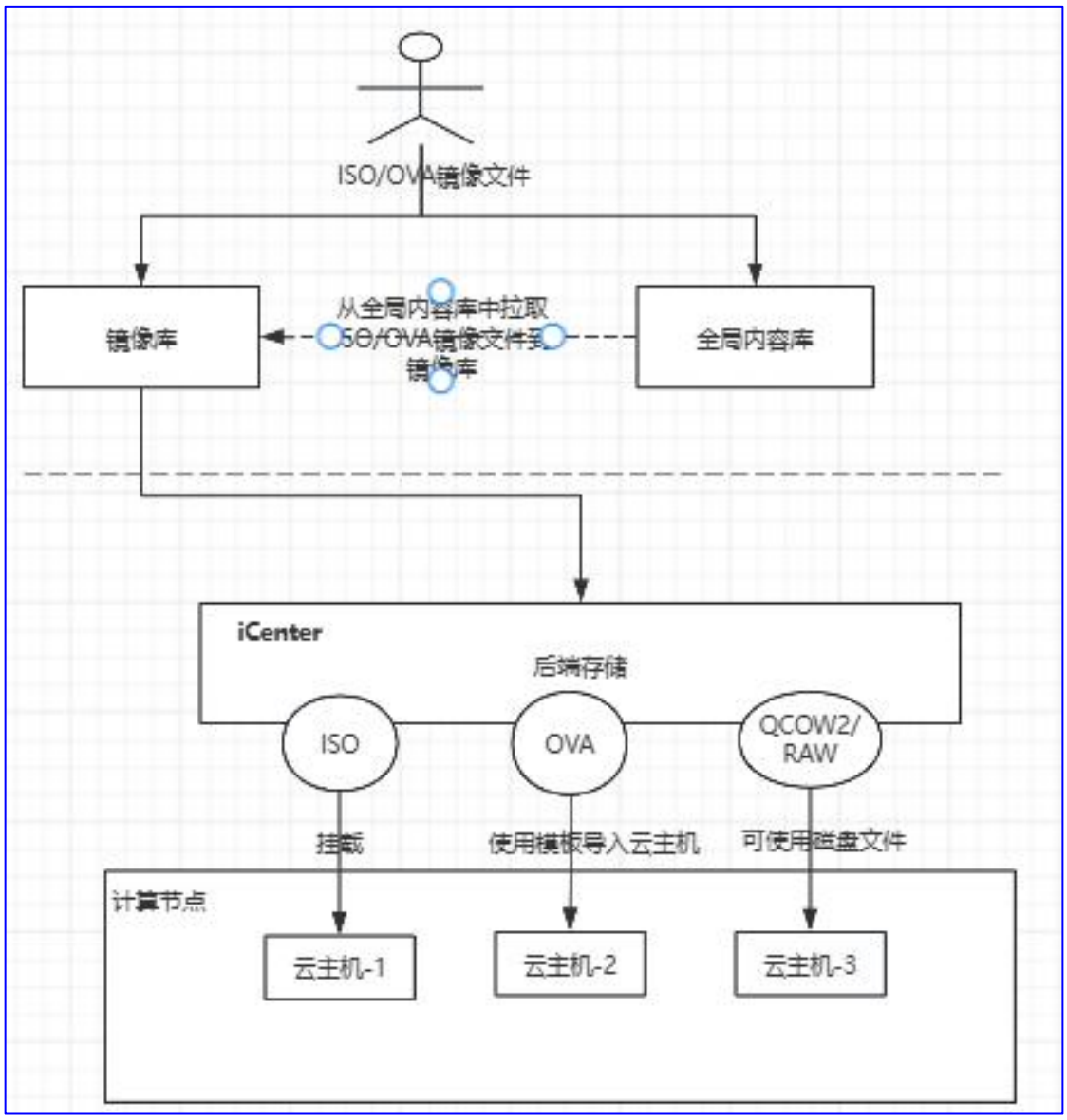
支持使用 **ISO**、**OVA**、**QCOW2** 和 **RAW** 格式的镜像创建云主机。用户可以将镜像文件上传到镜像库或全局内容库，然后在创建时指定需要使用的镜像文件，云主机自动继承镜像文件如 **CPU** 架构、操作系统、启动引导固件等元数据。选择镜像文件后，当前云主机的 **CPU** 架构就已被确定，后续只能被调度到指定架构的主机节点。支持使用备份文件创建云主机。备份文件保存有源云主机的 **CPU**、内存、网卡、磁盘等配置信息，以及备份点的磁盘数据。支持使用模板创建云主机，模板具有与源云主机相同的数据信息，可通过模板批量部署多个相同数据信息（操作系统和应用程序等）的云主机。

批量创建云主机支持从全局内容中选择 **OVA**、**ISO**、**QCOW**、**RAW** 格式的镜像文件，**OVA** 格式镜像文件会自动解析出网卡和磁盘信息，**ISO** 格式镜像文件支持用户手动添加网卡和磁盘信息，**QCOW2** 和 **RAW** 格式镜像支持添加网卡和磁盘以及配置磁盘关联文件，通过 **Excel** 模版导入数据后，会在批量创建云主机列表中展示数据的校验状态，检验失败的云主机属性可以通过页面进行修改再创建。

4.1.2.2 技术原理

ISO、**OVA**、**QCOW2**、**RAW** 格式镜像通过 **icompute** 服务中镜像管理模块将文件保存在底层 **iCenter** 的后端存储中。用户可以将自己的 **OVA** 格式的镜像文件上传到镜像库或全局内容库，云主机继承镜像文件的 **CPU** 架构、操作系统、引导启动固件属性。

使用全局内容库中的镜像时，需要选择要使用的镜像文件，并指定目标镜像库。系统会自动选择已存在相同镜像文件或者剩余可用容量最大的镜像库作为模板镜像库，用户也可手动修改使用指定镜像库；如目标镜像库中不存在该镜像库文件，则系统会先将镜像文件从全局内容库自动拷贝到目标镜像库中。若使用的是 **ISO** 镜像文件，**iCenter** 中会将 **ISO** 挂载到云主机；若使用的是 **OVA** 镜像文件，**iCenter** 则会根据模板中导入一个云主机。若是使用 **QCOW2** 和 **RAW** 镜像文件，**iCenter** 创建云主机后可使用磁盘文件中的数据。



4.1.2.3 约束

- 容灾云主机中，占位云主机仅允许查看，容灾云主机不支持编辑/删除 VPC 网卡、迁移存储、转换为模板、分配/回收使用权限、删除、快照、备份。
- 配置了多副本容灾、录影机、双活磁盘、裸磁盘的虚拟机不支持导入。
- 镜像文件所属的 CPU 架构为空时页面置灰，不支持使用该镜像文件创建云主机。

◆ 批量创建云主机的磁盘只支持新磁盘。

4.1.3 云主机控制台

4.1.3.1 功能描述

平台采用 **noVNC** 技术通过 **web** 控制台界面，可以直接对云主机进行控制，无需再使用其他工具与云主机连接。**noVNC** 是一个用于浏览器的 **VNC** 客户端，基于 **HTML5** 技术实现，它允许客户端能够以浏览器的形式接收和发送控制指令，能够切换到虚拟桌面，以及进行其它操作，实现用户和远程桌面之间的双向控制。

4.1.3.2 技术原理

noVNC 实现原理主要分为两部分：客户端和服务器端。客户端模块主要负责鼠标和键盘输入在浏览器中显示出来；服务器端负责接收、解码、发送，使用 **HTML5** 客户端和服务器之间的通信数据加密，增强安全性。

当用户通过 **Web** 浏览器访问 **noVNC** 服务时,浏览器首先加载 **noVNC** 的 **HTML5** 代码和 **JavaScript** 代码。这些代码负责建立与服务器的 **WebSocket** 连接,用于实时数据传输和控制命令交互。浏览器还负责渲染和显示远程计算机的图像和用户界面,同时捕捉鼠标和键盘事件并将其发送给服务器。**Web** 服务器接收来自浏览器的 **WebSocket** 连接请求,请根据请求的 **URL** 将其转发给后端 **VNC** 服务器。**WEB** 服务器还负责将 **noVNC** 的 **HTML5** 和 **JavaScript** 代码提供给浏览器，以及处理来自后端服务器的图像和控制数据，并将其传输给浏览器。

VNC 服务器负责捕捉用户界面，并将其发送给 **Web** 服务端，然后通过 **WebSocket** 传输给浏览器。**VNC** 服务器还会接收浏览器发送的鼠标和键盘事件，并将其转换成被控制的云主机。这样，用户在浏览器中对远程云主机的操作将直

接反映到被控制的云主机上。

4.1.3.3 约束

无

4.1.4 云主机迁移

4.1.4.1 更改主机迁移

4.1.4.1.1 功能描述

云主机在线迁移是指将一台云主机从一个宿主机迁移至另一个宿主机，而迁移过程中云主机继续执行原有指令而不会中断的一种技术。

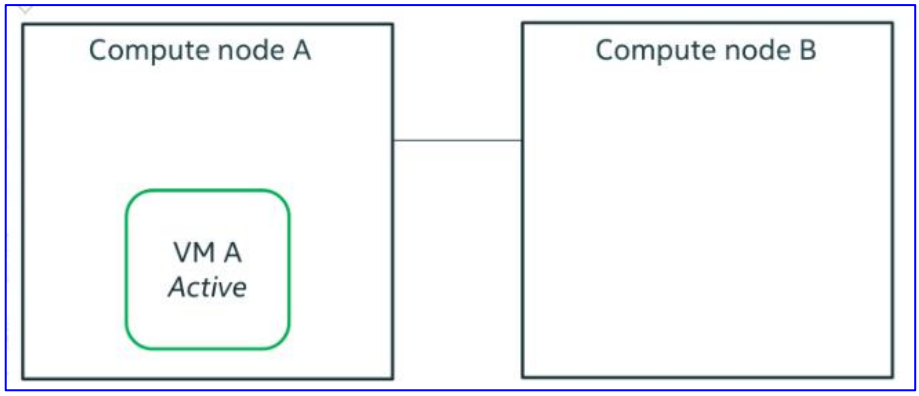
当某个物理主机资源竞争非常激烈而其他物理主机资源空闲率高时，用户可以选择在线将压力较大的云主机迁移到合适的物理主机上，期间并不影响业务的运行，这样可以保证云主机更加高效的运行。

4.1.4.1.2 技术原理

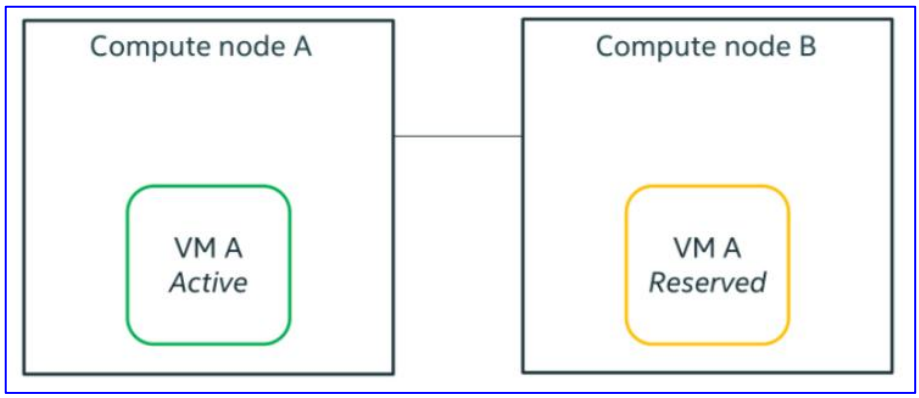
云平台支持两种在线迁移类型：**Live Migration** 和 **Block Migration**。**Live Migration** 需要实例保存共享存储中，这种迁移主要是实例的内存状态的迁移，速度会很快。**Block Migration** 除了实例内存状态要迁移外，还得迁移磁盘文件，速度会慢些，但是它不要求实例存储在共享文件系统中。

云主机在线迁移大致分五个步骤：

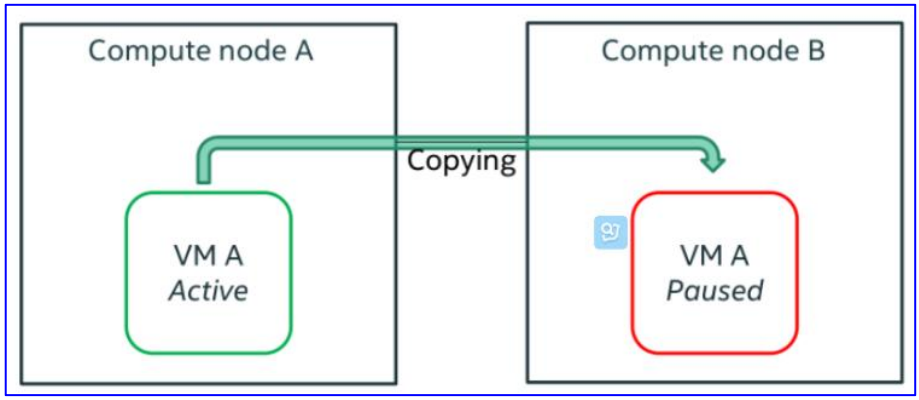
第一步：迁移准备阶段，即通过调度器，将 **node A** 上的 **VM** 要进行迁移的信息告诉 **node B**，**node B** 会开始一些准备工作（准备创建磁盘等）。



第二步：资源预留阶段，可以理解成在 **B** 上确定是否有资源来启动和 **VM A** 相同配置的云主机，并预留这个云主机。

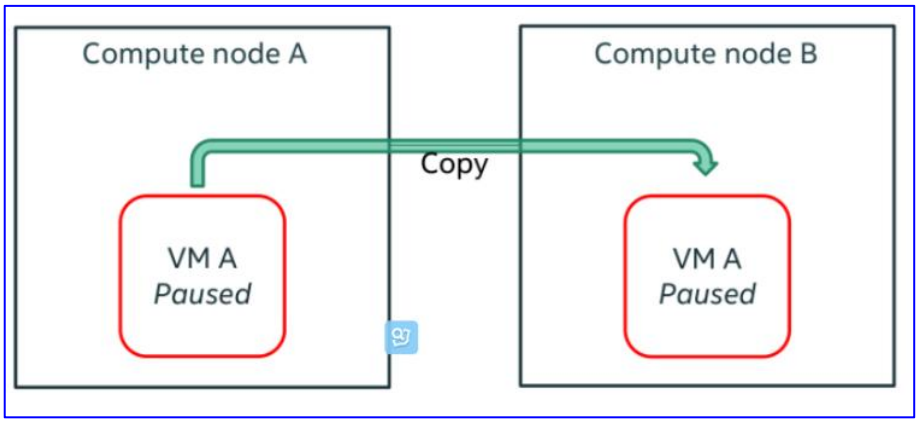


第三步：开始拷贝阶段，先将磁盘信息进行拷贝（块迁移会用到），然后将内存以及脏页进行拷贝拷贝。



第四步：停止和拷贝阶段，此时两台 **VM** 都会被暂停，会将剩余的 **pages** 和 **CPU** 状态进行拷贝。从这里可以知道，在热迁移的过程中，**VM** 不会完全都在运行状态，它会有短暂的暂停时间（注意并不是关机），在实际的线上环境中，

这么一点时间是可以接受的。



云主机迁移方案中的实时迁移是用一个预拷贝算法去迭代拷贝，在每次拷贝的过程中用内部记录的方式记录内存“脏”页，当“脏”页数据集小于一定程度时，比如 **4K** 的时候，停止云主机，把内容和寄存器迁移，由于需要停机拷贝的内容非常少，因此停机的时间非常短，不过实时迁移一般是相同体系的 **CPU** 才能相互迁移。

第五步：完成阶段，**VM A** 在 **node B** 中启动。

4.1.4.2 存储迁移

4.1.4.2.1 功能描述

存储迁移是指虚拟机迁移时，虚拟机的磁盘文件迁移到主机可以访问的其他存储上，可以是同资源池或跨资源池的，迁移过程不影响 **VM** 运行，有助于重新分配存储器负载。

4.1.4.2.2 技术原理

迁移步骤：

- 在目的存储建立与源存储一致的虚拟机磁盘结构。
- 将源存储虚拟磁盘的元数据拷贝到与之对应的目的存储的虚拟磁盘中。

- 当数据拷贝完成后，使用动态数据同步功能，将源存储虚拟磁盘的 **active** 层数据与目的存储的 **active** 层数据同步。
- 当源存储与目的存储的 **active** 层数据同步时，将存储从源系统上解锁，并锁定在目标系统上，使虚拟机的磁盘 **I/O** 重定向到目的存储的虚拟磁盘上，断开源存储磁盘 **I/O**。
- 源存储资源进行释放，完成虚拟机存储迁移。

4.1.4.3 整机迁移

4.1.4.3.1 功能描述

vMotion 提供 **VM** 灵活性和可用性，可以更改运行虚拟机的计算资源，或者同时更改虚拟机的计算和存储资源。可将正在运行的整个虚拟机从一台服务器迁移到另一台服务器上，且虚拟机保留其网络标识和连接，从而实现无缝迁移。为虚拟机提供了灵活性和可用性，满足业务和最终用户不断增长的需求。

4.1.4.3.2 技术原理

vMotion 使用的是预复制迁移（**Pre-Copy Migration**），具体原理如下：

- 系统验证目标服务器的存储器和网络设置是否正确，并保留目标服务器虚拟机的资源；
- 当虚拟机还在源服务器上运转时，将内存镜像复制到目标服务器上。在这个过程中，会监视内存的变化；
- 内存复制完成后，大部分的内存镜像已经被复制到目标服务器上，检查内存较复制之前是否发生了变化；
- 假如发生了变化，会将发生变化的内存重新复制到目标服务器中，并覆

盖先前的内存。此时仍然会监视内存的变化情况；

- 底层持续进行内存复制操作，随着复制次数的增加，所需要复制的数据就会明显减少，复制所消耗的时间就会逐渐变短，内存可能没有足够的时间发生变化。最后当源服务器与目标服务器之间的差异可以忽略不计时，内存复制操作结束；
- 内存复制完成之后，将 **VM** 的工作状态切换到目标服务器上，源服务器针对迁移的虚拟机停止工作；将存储从源系统上解锁，并锁定在目标系统上。启动目标服务器，并与存储资源和网络资源链接，同时清除源服务器上的资源。

4.1.4.4 约束

- 云主机在进行存储迁移时，不支持其他类型存储池迁移到裸设备存储池（裸设备之间也不能迁移）。
- 支持带裸设备磁盘的云主机离线迁移。在进行存储迁移操作时，裸设备磁盘可以迁移到 **CFS** 存储池或者本地存储池上。
- 在进行整机迁移操作时，含有裸设备磁盘的云主机支持部分迁移（迁移时裸设备磁盘不能更改存储池）。
- 对于使用双活磁盘的云主机，不支持云主机迁移。

4.1.5 云主机高可用&安全

4.1.5.1 云主机崩溃处理策略

4.1.5.1.1 功能描述

云主机崩溃处理策略功能主要是用来处理云主机发生崩溃事件的场景，用户

可以预先设置好崩溃处理策略策略，当云主机发生崩溃事件后，就会按照实现设置好的策略对云主机做相应的操作。

4.1.5.1.2 技术原理

通过主机定期检测虚拟机状态，当部分 **Windows** 系列操作系统的云主机出现蓝屏和部分 **Linux** 系列操作系统的云主机出现宕机，系统能够根据用户预先设置的故障处理策略进行处理，以及时恢复业务，保障业务连续性，增强系统的自动化维护手段，减少了维护人力投入。产品提供的云主机崩溃处理策略包括以下几种。

【不处理】：即云主机出现蓝屏或宕机，不对这个虚拟机做任何处理，系统即使检测到云主机发生故障，只上报告警；

【重启】：即云主机出现蓝屏或宕机，云平台将自动重启虚拟机；

【关机】：即云主机出现蓝屏或宕机，云平台将自动关闭虚拟机；

【**HA**】：即云主机出现蓝屏或宕机，云平台将自动拉起虚拟机；蓝屏时在当前主机拉起虚拟机，宕机时优先在当前主机启动，如果当前主机不满足条件，则调度到其他主机并尝试启动；

4.1.5.1.3 约束

云主机需要安装 **VM Tools**。

4.1.5.2 云主机防病毒

4.1.5.2.1 功能描述

云主机防病毒主要是指宿主机集成了生态防病毒软件，创建虚拟机时可以选

择开启防病毒，这样创建出来的虚拟机也是具备防病毒功能的。也可以对现有的云主机开启、关闭防病毒。

4.1.5.2.2 技术原理

云主机具备防病毒功能主要使用了 **ivshmem** 内存共享技术，**ivshmem-plain** 设备是 **Qemu** 提供的一种特殊设备，通过这个设备，可以实现虚机内存和主机上其它进程共存共享，应用程序可以利用此设备实现虚机内部和主机上进程间的高效数据传输。通常，虚机内部的进程作为生产者，往共享内存中写入数据，主机侧进程作为消费者，从共享内存中读取数据，虚机内部的杀毒软件驱动程序搜集虚机的行为数据放到共享内存，后端的杀毒软件分析虚机暴露的数据，判断该虚机是否行为异常甚至中毒。

4.1.5.2.3 约束

宿主机需要安装防病毒软件。

4.1.5.3 云主机高可用

4.1.5.3.1 功能描述

High Availability（**HA**）为系统中虚拟机提供了简单易用、成本效益高的高可用性功能。由于硬件故障导致的 **VM** 不会再造成灾难性的后果， **HA** 会对宕机的 **VM** 进行重新分配资源并快速重启这些虚拟机。**DRS** 则会决定放置这些虚拟机的最佳位置以满足其资源要求，其业务可在短时间内恢复正常。

4.1.5.3.2 技术原理

HA 主调度服务结合宕机事件主动上报机制实现 **VM** 的高可用性服务；

HA 主调度服务包含 **HA** 监听器和 **HA** 事件处理器；**HA** 监听器用于循环监听 **HA** 事件处理的结果，若有一次没处理完毕的宕机事件，**HA** 监听器则继续进行尝试处理；**HA** 事件处理器用于接收并处理主动上报的宕机事件；

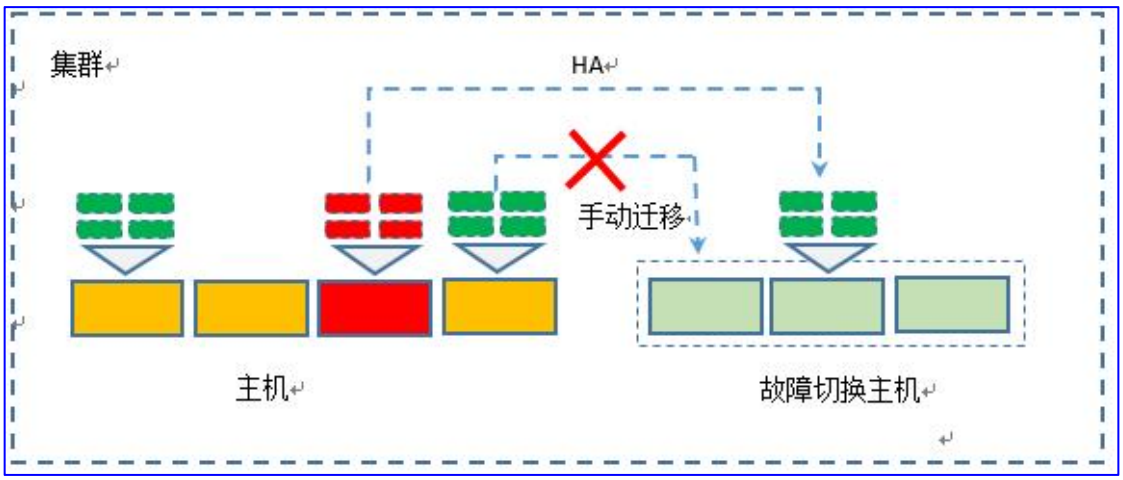
HA 宕机事件主动上报机制，服务器或虚拟机发生宕机事件，并将事件快速主动上报给 **HA** 主调度服务器的 **HA** 事件处理器。

HA 接入控制原理：

集群 **HA** 接入控制策略提供两种选项，一是 **HA** 资源预留（默认选项），二是故障切换主机。

其中 **HA** 资源预留策略通过静态定义 **CPU** 或内存资源预留百分比实现。当集群中 **CPU** 或者内存资源剩余百分比小于所配置的 **CPU** 或内存资源预留百分比，将拒绝新的虚拟机启动或者其他集群的虚拟机在线迁移到该集群，当集群中发生主机宕机或者虚拟机宕机事件，则宕机的虚拟机可以利用为 **HA** 所预留的 **CPU** 或内存资源重新启动。

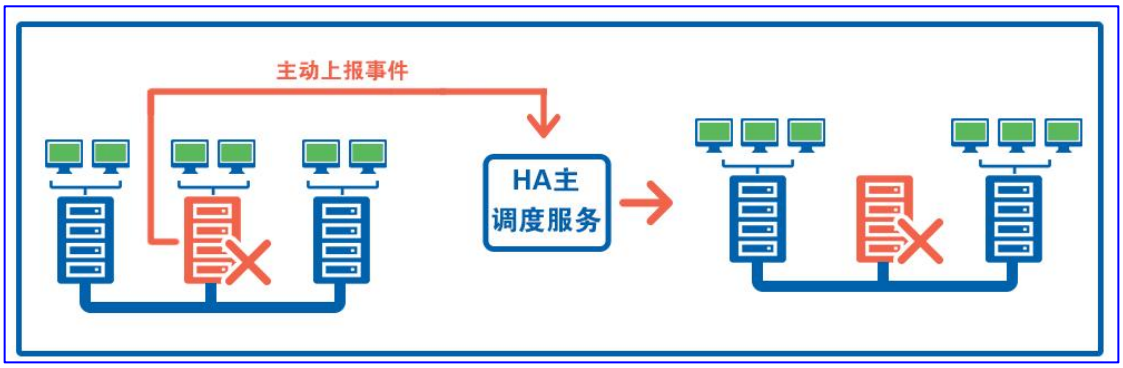
故障切换主机策略通过静态定义故障切换主机列表实现。当集群中部分主机发生故障。**HA** 将尝试在任一指定的故障切换主机上重新启动其虚拟机。为了确保故障切换主机上拥有可用的空闲容量，手动迁移虚拟机到故障切换主机的功能将禁止。可以手动迁移虚拟机到其他非故障虚拟机。



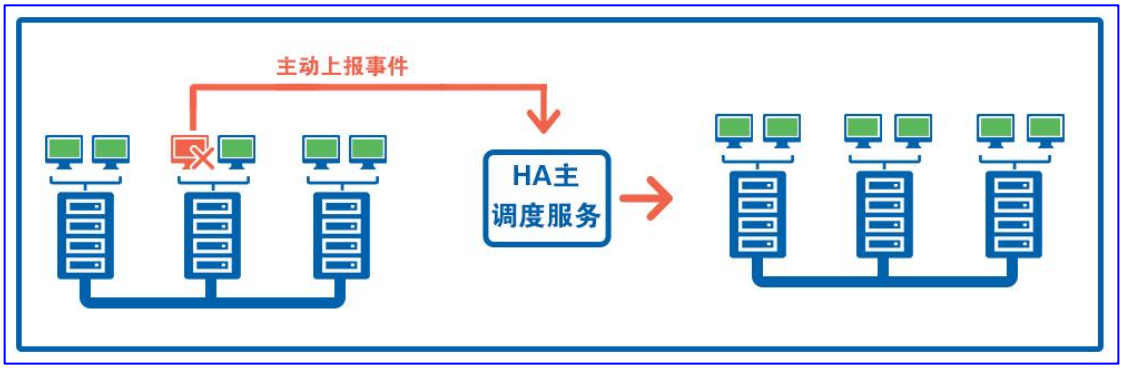
具体实现方式：

通过配置启用 **HA** 策略，在集群管理中设置 **HA** 策略。

集群内服务器发生宕机，通过主动上报机制将宕机事件上报，**HA** 主调度服务将该服务器上运行状态的虚拟机在集群内其它服务器上重新分配资源并进行重新启动。



单个运行状态的虚拟机发生宕机，通过主动上报机制将宕机事件上报，**HA** 主调度服务对宕机的虚拟机快速进行重启，。



4.1.5.3.3 虚拟机 HA 约束

无。

4.1.6 云主机纳管

4.1.6.1 功能描述

云主机纳管功能可以将底层虚拟化层正在使用的云主机纳管到云平台中来进行管理。

4.1.6.2 技术原理

纳管之前会查询底层所有的云主机，选中需要纳管的云主机之后，会将云主机纳管到云平台中，如果云主机已经在云平台中存在，则按钮置灰不可纳管，纳管云主机时如果云主机已经挂载了云硬盘，会将云硬盘一块纳管到云平台中。

4.1.7 云主机导出镜像

4.1.7.1 功能描述

用户可以将已经配置好的云主机导出为镜像文件（**ova**）到镜像库中，下次创建的时候就可以使用该 **ova** 文件创建出与源云主机一样配置的云主机。

4.1.7.2 技术原理

云主机导出镜像是基于 **qemu-img** 镜像转换工具来实现的，如果想要将云主机导出到镜像库，可以选中一个云主机、选中一个镜像库，然后通过虚拟化层使用 **qemu-img** 工具的 **convert** 命令来完成转换，默认的磁盘转换格式为 **qcow2**。

4.1.7.3 约束

无。

4.1.8 云主机快照与恢复

4.1.8.1 功能描述

通过云主机快照、内存快照与还原功能可以实现云主机中硬盘信息的安全快照。虚拟机生命周期中，通过快照和还原操作，可以对云主机数据安全保存和恢复。可以极大提高用户数据的安全性，特别是出现宕机等系统损坏的情况，通过快照恢复功能，可以快速恢复运行业务。以最大程度减少对用户业务的影响程度。

4.1.8.2 技术原理

云主机实现快照和快照恢复使用的是 **ocfs** 存储的 **reflink** 机制。**OCFS**（ **Oracle Cluster File System** ）是一种用于在集群环境下共享文件系统的技术。**OCFS** 通过提供共享文件系统来简化集群环境下的文件共享和数据管理。可以实现文件的快速复制和恢复。云主机使用的虚拟磁盘，实际上是 **OCFS2** 文件系统上的一个文件，这些文件通常的大小为 **GB** 级别。假如需要创建快照和恢复，那么就会产生一个拷贝的动作，这些拷贝技术都是基于 **reflink** 技术实现的。

4.1.8.3 约束

容灾云主机无法创建快照。

4.1.9 云主机备份

4.1.9.1 功能描述

InCloudOS 支持通过云主机备份功能，为云主机在可用存储上创建云主机某一时刻的备份。特定类型的云主机不支持云主机备份： 包含裸磁盘的云主机，包含 **NFS** 存储池磁盘的云主机。支持为启动和关闭状态的云主机创建备份，但使用裸设备存储池磁盘的云主机不支持开机状态创建备份。

InCloudOS 使用了独有的基于差分的备份方式，不依赖于云主机快照，备份不会影响云主机的性能。

支持备份的存储类型有 **CFS** 存储。

系统支持设置自动为云主机创建备份功能，通过此功能，系统可以在指定时间为指定的云主机创建备份。可以设置只创建一次或周期性执行。设置调度任务后，系统会在设置的时间自动为指定的云主机创建备份，自动备份任务的成功与否依赖备份时的主机状态和存储状态，如果主机，网络和存储状态良好，则可以成功生成虚拟机备份。

用户使用备份恢复功能，用指定的云主机备份恢复生成新云主机。恢复备份生成的云主机与源云主机配置相同，云硬盘内的数据状态为源云主机创建备份时的文件系统状态。

恢复备份产生的云主机是一个全新的云主机（拥有全新的云主机名称和 **ID**），恢复备份过程不依赖源云主机状态，生成的新云主机与源云主机互不影响。

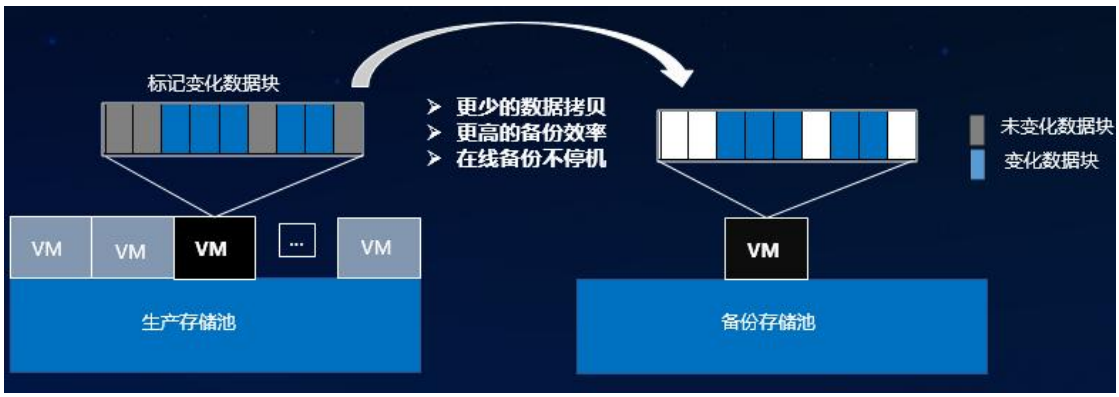
支持对备份设置备份策略，备份策略可设置最长保留时间和最大保留个数。系统将会定会对备份文件进行处理，若备份文件超过最长保留时间，将会被删除，但若全量备份超期，且其下存在增量备份未超期，则不会删除全量备份。另外，

若云主机创建了周期备份，最大保留个数的逻辑处理将会交给底层；若云主机未创建周期备份， **incloud os** 层将会根据设置的最大保留个数进行定时处理，如备份文件个数超过最大保留个数，则会删除较早创建的备份，直到满足设置的最大保留个数。

4.1.9.2 技术原理

备份机制采用 **CBT** 备份基于数据块变化追踪（**Changed Block Tracking**）技术，为云主机增量备份提供了基础，除第一次备份必须完整备份外，后续的增量备份都只需通过查询 **CBT** 位图，只需备份变化过的区块，节省了区块数据差异对比时间，所需传输的数据量也大幅减少。

CBT 备份方式的特点是数据最全面、最完整，数据一致性得到完全保护，备份的效率高，备份时间短。如果灾难发生时，可以恢复到任意备份点，不再受限于其他备份文件，每个备份点之间是相互独立。



CBT 备份

4.1.9.3 约束

- 同一个备份仅支持同时进行一个操作；
- 带裸磁盘的云主机不支持恢复为原云主机；

- 非关机状态云主机不支持进行备份恢复为云主机操作；
- 备份恢复为原云主机时，若备份中网卡之前未设置静态 **IP**，不支持设置静态 **IP**；若网卡之前设置了静态 **IP**，则不支持修改静态 **IP**，但支持切换成 **DHCP IP**；
- 云主机备份恢复操作，包括备份恢复为原云主机和使用备份创建新云主机时，不支持增加和删除网卡，仅支持编辑网卡信息；
- 云主机处于创建中、创建失败、资源池异常状态时，不支持进行备份编辑、删除操作。

4.1.10 云主机克隆

4.1.10.1 功能描述

云主机克隆，可以使客户能够以现有云主机为模板，快速复制出相同的虚拟机，也可以先创建一台云主机进行软件和系统配置，然后以该云主机为模板批量的创建出相同规格、属性和状态的云主机，实现了对虚拟机数据、属性以及状态的快速复制。

4.1.10.2 技术原理

云主机克隆将云主机快照和创建云主机两个操作在服务内部进行组合实现的，当用户执行克隆云主机操作时，云平台服务内部接收到用户请求后会调用接口为云主机创建一个临时的云主机快照，同时服务内部获取到源云主机的各项属性信息，包括云主机的规格、元数据、网络信息等，在云主机临时快照完成后，云平台利用该快照，同时将从源云主机获取到的各种属性信息来创建一台与源云主机相同的新云主机。

等到新云主机创建完成或者创建失败时，服务内部自动调用接口将临时快照进行删除，避免脏数据的残留。

4.1.10.3 约束

- 云主机存在光驱介质不支持克隆操作。
- 主备容灾云主机与服务实例云主机不支持克隆操作。

4.1.11 GPU 设备管理

4.1.11.1 功能描述

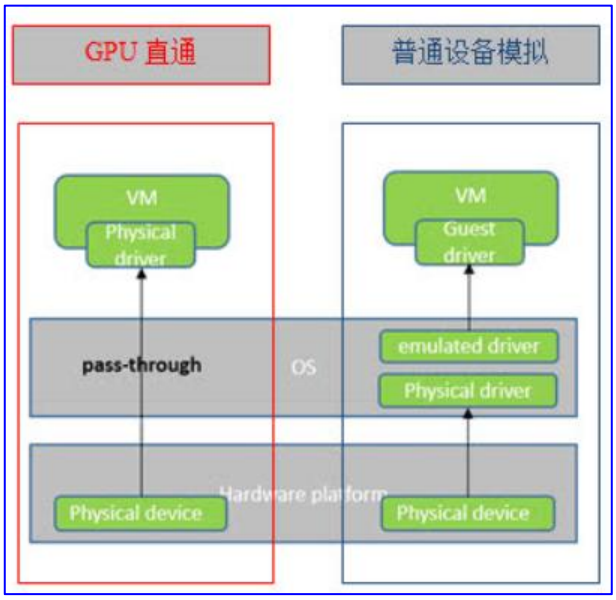
InCloud OS 允许通过为云主机直通宿主机**GPU**设备或者**GPU**切片（**vGPU**），为云主机分配 **GPU** 设备。

4.1.11.2 技术原理

(1) GPU 直通

一般情况下，普通的设备模拟需要经过 **HostOS** 的物理驱动，然后经过 **Qemu** 的模拟驱动，再转发到 **Guest OS** 内的驱动上，在 **VM** 层面看到的设备实际上是模拟的设备。

GPU 直通是也叫 **GPU pass-through** 直通。如图所示，**pass-through** 直通是将 **PCI** 插槽上的 **GPU** 设备绕过 **OS** 层直接提供给了 **VM**，在 **VM** 层面看到的是一块物理卡。



GPU 直通的技术优势如下：

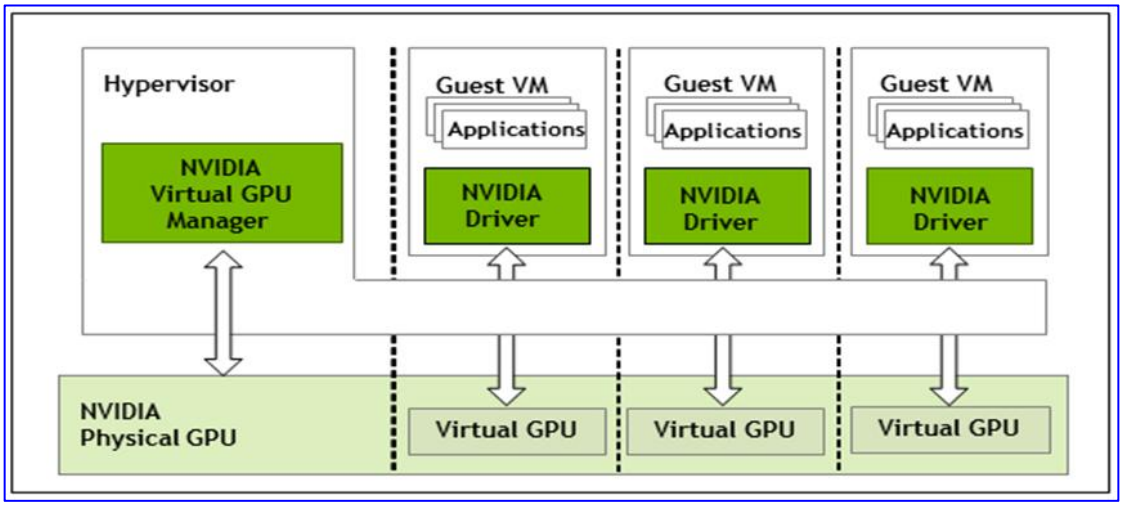
1、性能损失小。无论是 **AI** 应用、深度学习还是科学计算，使用 **GPU** 直通就是要极致利用 **GPU** 的性能，**GPU** 直通技术由于没有设备模拟和转换的过程，性能损耗极小。一般来说，**GPU** 直通虚拟机的性能与裸机上的 **GPU** 性能相比损失小于 **5%**。

2、功能兼容性好。相对于设备模拟或者 **GPU** 虚拟化技术，因为虚拟机里面直接安装的是 **GPU** 厂商的驱动，直通设备可无损使用 **GPU** 的各项特性和功能。

3、易用性。用户直接在虚拟机安装厂商驱动即可，无需关心主机侧的驱动以及其他硬件配置。

(2) vGPU

NVIDIA GPU 虚拟化技术允许多台云主机同时使用一块 **GPU** 设备的计算资源，提供卓越的图形、计算性能以及良好的软件兼容性。



虚拟 **GPU**（**vGPU**）技术通过 **GPU** 虚拟化、**GPU** 调度器、**GPU** 驱动程序和帧缓冲传输等核心组件的协同作用，为虚拟机提供高性能的图形处理能力。它的核心技术原理如下：

GPU 虚拟化：**vGPU** 通过在物理 **GPU** 上创建多个虚拟 **GPU** 实例，将物理 **GPU** 资源划分为多个虚拟切片。每个虚拟切片被分配给一个虚拟机，使其能够独立地访问和利用 **GPU** 资源。

GPU 调度器：**GPU** 调度器负责管理和调度虚拟 **GPU** 实例的资源分配。它根据虚拟机的需求和物理 **GPU** 的可用资源，动态地分配和调度 **vGPU** 切片，以确保每个虚拟机获得适当的 **GPU** 性能。

GPU 驱动程序：每个虚拟机都需要安装相应的 **GPU** 驱动程序，以便与虚拟 **GPU** 实例进行通信。**GPU** 驱动程序负责将虚拟机的图形指令转发给相应的 **vGPU** 切片，并将处理结果返回给虚拟机。

帧缓冲传输：**vGPU** 技术使用帧缓冲传输来实现图形数据的传输。当虚拟机产生图形输出时，**vGPU** 切片将渲染的图像存储在帧缓冲中，并通过虚拟化协议将帧缓冲传输给虚拟机。

vGPU 的技术优势如下：

1、提供高性能的图形处理能力：通过将物理 **GPU** 虚拟化，**vGPU** 为虚拟机提供了与物理机相似的图形处理性能和功能，使虚拟机能够运行图形密集型应用程序。

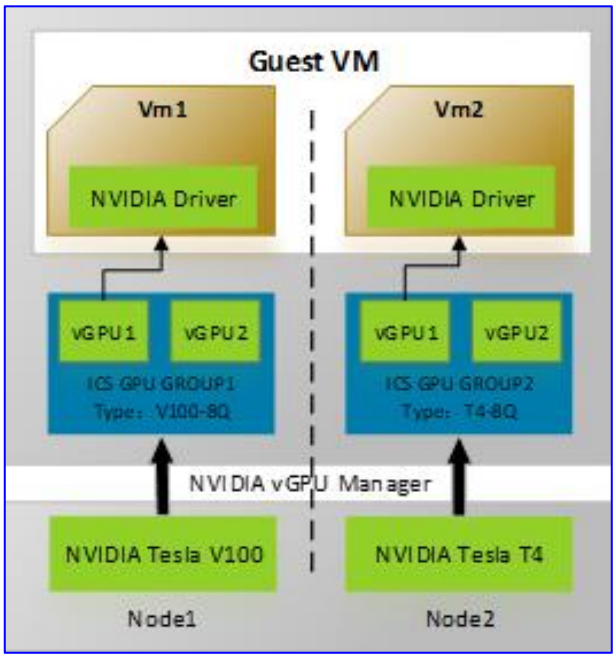
2、提高资源利用率：**vGPU** 技术允许多个虚拟机共享物理 **GPU** 资源，提高了 **GPU** 的利用率，通过动态调度和分配 **vGPU** 切片，可以确保每个虚拟机获得适当的 **GPU** 性能，避免资源浪费。

3、支持多租户环境：**vGPU** 技术允许在多租户环境中同时运行多个虚拟机，并为每个虚拟机提供独立的图形处理能力，这为云计算、虚拟桌面基础设施（**VDI**）等场景提供了更好的性能和用户体验。

(3) GPU 组与设备管理

基于 **GPU** 组的概念统一管理主机组中的 **GPU** 设备，基于主机组创建一个或多个 **GPU** 组，每个 **GPU** 组都有对应的 **GPU** 设备型号以及 **GPU** 设备使用方式，使用方式包括直通和 **vGPU** 两种。使用 **vGPU** 方式时，需要选择虚拟化的使用方式，并指定 **vGPU** 类型，每一类 **vGPU** 都对应了 **GPU** 的不同功能、**API** 的适用范围以及可生成 **vGPU** 的数量，用户可根据自身应用需求来选择某一类 **vGPU** 类型。

创建 **GPU** 组之后，可以添加 **GPU** 组对应型号的主机组 **GPU** 设备，此时需要主机组中提供 **GPU** 设备的主机安装 **NVIDIA vGPU Manager** 驱动。**GPU** 组添加 **GPU** 设备之后，如果是 **vGPU** 使用方式，会自动根据 **GPU** 组属性中的 **vGPU** 类型自动生成 **vGPU** 设备。



4.1.11.3 约束

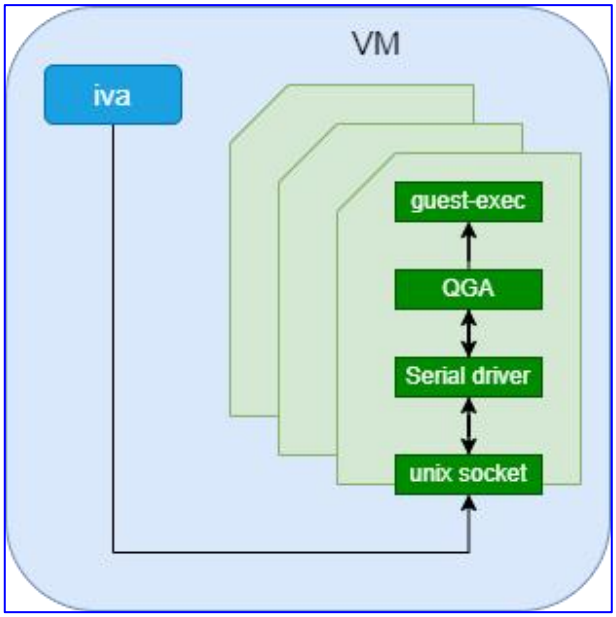
- 云主机可在关机状态下添加 **GPU** 设备。
- 一台云主机不能同时添加 **vGPU** 和物理 **GPU**。
- **windows** 云主机使用 **vGPU** 设备时，需要通过远程方式登录。

4.1.12 云主机 VMTools

4.1.12.1 功能描述

在云主机的生命周期管理中，**KVM** 虚拟化 **hypervisor** 需要与云主机 **Guest OS** 交互以实现个性化定制、高级管理功能。**VMTools** 运行于虚拟机内部，负责的功能包括从创建之初设置云主机管理员密码、网卡 **IP**，到运行中修改云主机管理员密码，到在线快照冻结 **Guest OS** 保证数据一致性，到云主机软关机、软重启保证用户数据安全，到精确监控云主机磁盘使用率、**CPU** 与内存使用率，定时进行时间同步等。

4.1.12.2 技术原理



基于 **VMTools** 实现云主机管理功能如上图，此处以云主机在线修改密码为例：用户通过图形界面下发请求到 **iva**，**iva** 通过 **unix socket** 将操作下发到对应的云主机。而云主机内部的 **qga** 服务监听串口设备，接受来自 **hypervisor** 层的调用，运行 **guest-exe** 程序以执行云主机操作系统支持的任意命令、系统调用。

4.1.12.3 约束

Windows 类型的操作系统，其 **vm-tools** 依赖 **Microsoft Visual C++ 2008**，卸载该组件会导致 **vm-tools** 无法启动。

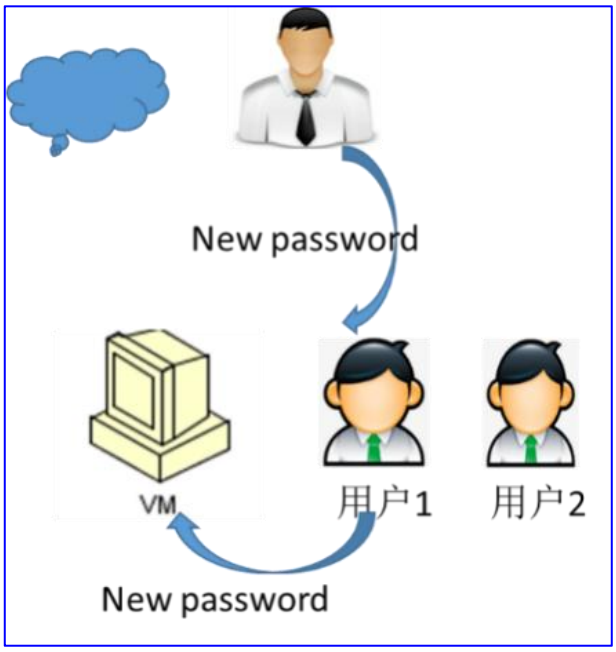
4.1.13 云主机重置密码

4.1.13.1 功能描述

在云主机使用过程中，用户可能需要在不影响正常业务的情况下修改云主机密码，并且一台云主机可能有多个用户角色权限，在修改某个用户密码时，不能影响其他用户正常访问，为此云平台开发了在线修改云主机用户密码功能。

4.1.13.2 技术原理

对于一台云主机有 A、B 两个用户角色，当需要修改用户 A 的密码时，不能影响 B 用户正常访问。云平台实现了在线指定用户名称修改云主机密码功能，修改密码后立即生效，不需要重启云主机，下次该用户进行访问时，需要使用新密码。该功能实现方式是通过 vm-tools 实现的，该服务继承了 QGA，可以实现云平台和云主机的一些交互，比如关机、重启、修改密码、执行云主机内部命令等等。



4.1.13.3 约束

云主机内部需要安装 vmtools 工具并且是开机状态。

4.1.14 一云多芯

4.1.14.1 功能描述

InCloudOS 支持多种 CPU 架构，由于不同 CPU 架构支持的操作系统不一样，因此 CMP 层在创建云主机时需根据镜像文件的 CPU 架构信息过滤操作系

统、主机等。针对一云多芯的场景，对创建云主机流程进行了优化，比如：根据选择的镜像的格式（**ISO/OVA**）来决定云主机的创建方式，不再让用户去主动选择创建方式。同时，镜像文件也增加支持设置 **CPU** 架构、芯片厂商等信息。当前 **CPU** 架构包括：**X86_64** 和 **AARCH64**，平台支持单一资源池多架构 **CPU** 混管，以满足一云多芯算力调度需求，单一资源池支持多种类型 **CPU** 芯片混合管理，包括 **ARM**、**X86**、**C86**、**SW64**、**LoongArch**。

4.1.14.2 技术原理

“一云多芯”是指使用一套云平台管理不同芯片架构的计算资源，实现异构资源的统一管理和调度，并通过云平台屏蔽底层架构差异，为用户提供体验一致的云计算服务。随着企业数字化转型和应用创新不断加速，“一云多芯”的内涵也在不断延展，不仅仅是指对不同 **CPU** 架构资源的管理，同时还包括 **GPU**、**FPGA**、**DPU** 等算力的管理；“一云多芯”已经成为 **IT** 产业链承上启下的关键纽带，除了对下管理调度异构硬件资源，提供虚拟机、裸机、容器，还要融合多种操作系统、中间件、数据库、**AI** 等服务，为上层各行业的创新应用提供稳定的运行环境以及丰富的的生态服务。

在云主机创建时，根据资源池支持的 **CPU** 架构可选择能够使用的镜像文件。而且根据镜像文件的 **CPU** 架构设置云主机本身的 **CPU** 架构，并查询 **CPU** 架构所支持的操作系统、筛选主机等，使得云主机创建完成后可以正常开机使用。

云主机全生命周期架构感知

涉及的主要计算包括云主机镜像、主机、云主机、以及基于镜像派生的云硬盘，以及云主机派生出的快照、模板、备份等。这些资源或包含基于特定架构编译的二进制可执行程序、或作为执行二进制程序的计算负载，因此需要标记架构。

涉及的主要的生命周期活动：

①镜像的上传，需要标记镜像的 **CPU** 架构、厂商（因为即使同架构不同厂商的 **CPU** 特性也有差异，也会影响云主机系统稳定）；

②云主机创建时选择资源池，资源池的架构信息来源于云平台纳管资源池主机时获得的主机架构信息的集合，因此某个资源池可能包含至少一种架构，但是一般不推荐混合架构部署；

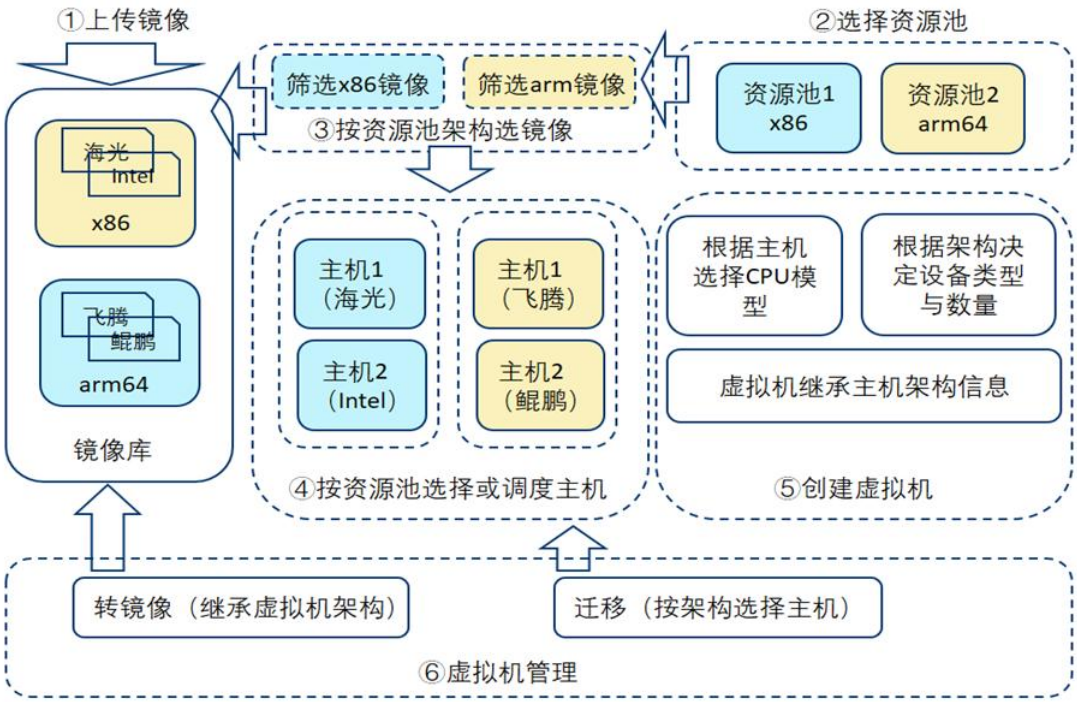
③根据选择的资源池按架构查询镜像库筛选架构合适的镜像；

④根据资源池架构筛选主机；

⑤根据选定的主机为配置合适的 **CPU** 模型，对应的设备可能也不一样（例如 **CDROM** 设备总线跟架构是紧密关联的），创建的云主机继承主机架构信息；

⑥云主机管理：云主机转为镜像、模板、快照、备份等可启动介质时，架构信息可继承传递，可以作为以后启动云主机选主机的决策条件；

⑦另外云主机迁移时，按架构匹配筛选合适主机。



4.1.14.3 约束

- 镜像文件的 CPU 架构为空时，否则页面置灰，不支持选择。镜像文件的 CPU 架构不为空时,必须与资源池的 CPU 架构一致,否则页面置灰，不支持选择；
- 云主机继承镜像文件的 CPU 架构、操作系统、启动引导固件属性
- 根据镜像文件的 CPU 架构过滤主机、查询支持的操作系统

4.1.15 弹性伸缩

4.1.15.1 功能描述

弹性伸缩（Auto Scaling，以下简称 AS）是根据用户的业务需求，通过设置伸缩规则来自动增加/缩减业务资源。当业务需求增长时，AS 自动为您增加云主机，以保证业务能力；当业务需求下降时，AS 自动为您缩减云主机，以节约成本。AS 支持自动调整弹性云服务器和带宽资源。目前产品实现如下功能：

支持自定义伸缩规则，用户可以自定义伸缩规则，支持设置定时策略或告警策略：定时策略可以执行一次、按天、星期周期性执行，可以设置不同的调整策略，包括增加、减少、调整至目标云主机个数；告警策略可根据伸缩组内实例的 CPU 使用率、内存使用率、网络流入流出率大于/小于预设值时，动态增加/减少或者调整至目标云主机个数。

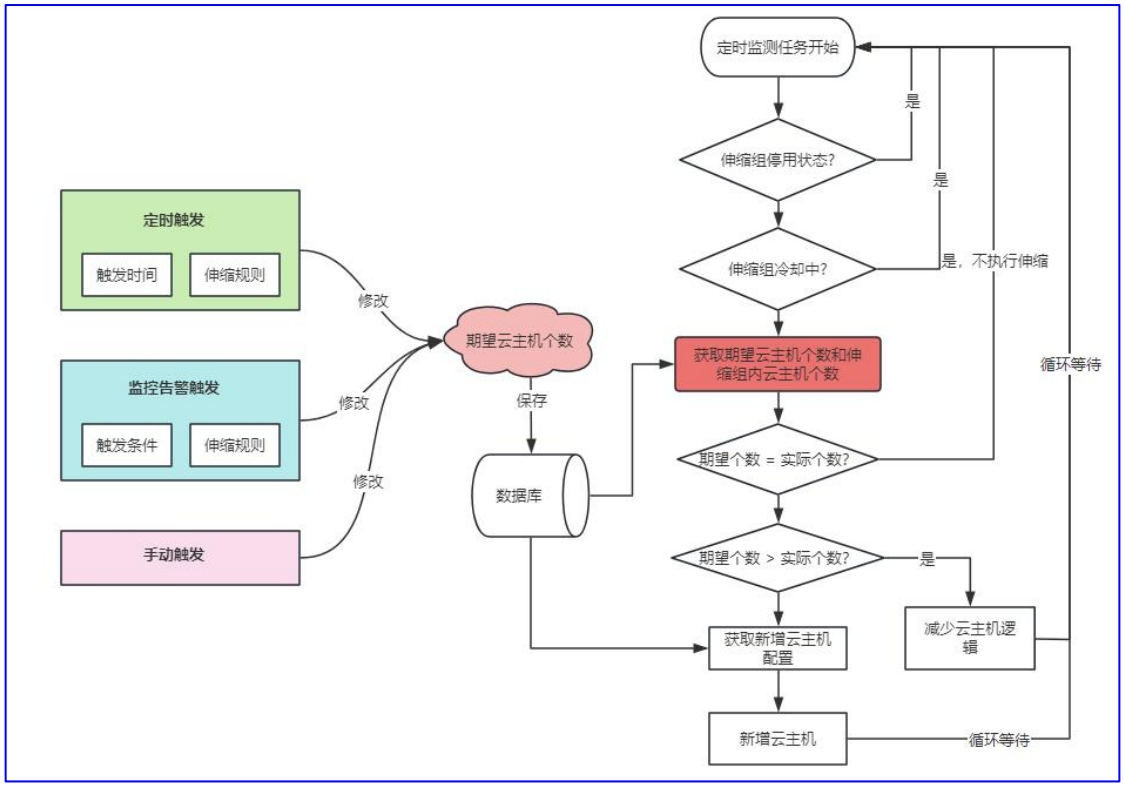
支持自定义伸缩组，可以关联伸缩规则，设置伸缩组最大云主机个数、最小云主机个数、伸缩组冷却时间。支持添加已有云主机作为成员云主机，或者选择负载均衡，此时伸缩组扩容出来的云主机自动加入到负载均衡。支持配置新增云主机的规格、网络信息，设置开机执行脚本。

如果需要立即变更伸缩组成员数量，可以手动调整伸缩组云主机个数，此时不受冷却时间限制。

支持对成员云主机设置实例保护，设置了实例保护的云主机，在自动扩缩容时不会被删除掉，用于保护用户数据。

4.1.15.2 技术原理

弹性伸缩的实现原理的核心是动态保持伸缩组内云主机数量与期望云主机个数保持一致。伸缩规则（定时、监控）和手动触发修改的是期望云主机个数。而伸缩组本身有一个定时任务监控期望云主机个数和伸缩组当前云主机个数，如果不相等就执行对应的扩缩容任务。检测任务也会检查伸缩组成员云主机的状态，将处于关机、宕机、丢失的云主机移出伸缩组，并扩容新的成员，最终达到成员云主机个数等于期望云主机个数。



4.1.15.3 约束

- 伸缩组最多关联 **20** 个伸缩规则，最大支持 **20** 个伸缩组成员。
- 云主机加入伸缩组之后，不能进行删除、移出、分配、回收、编辑操作。
- 伸缩组新增云主机配置，只能使用 **ova** 镜像模版。
- 伸缩组新增云主机配置，只能使用开启 **dhcp** 的经典网络和 **vpc** 网络，且不能指定 **ip**。
- 扩容出来的云主机开机执行脚本需要安装 **cloudinit**，并且目前只支持 **linux** 操作系统，且长度不能大于 **2000**。建议这个脚本不要太复杂，可以把一些复杂的逻辑内置在镜像模版中且调试通过，开机脚本将一些参数传给这个脚本。

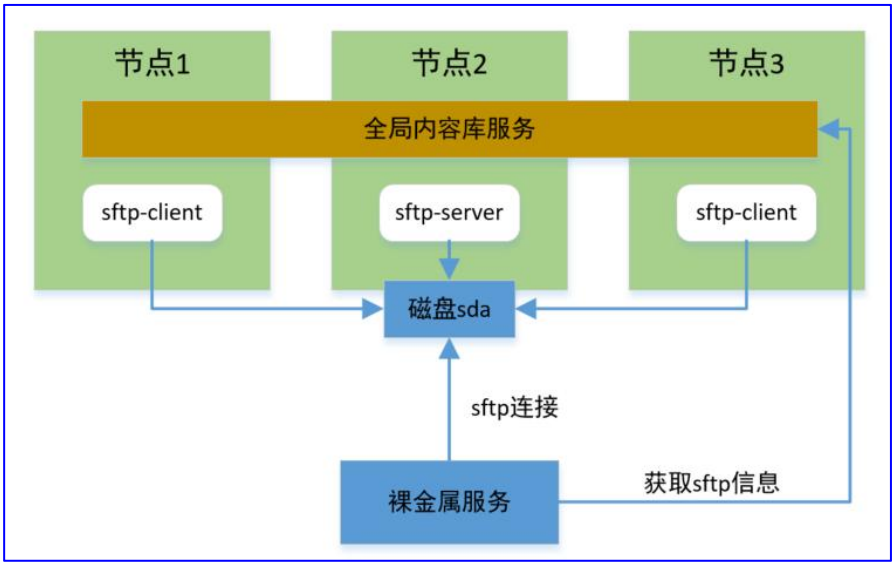
4.1.16 全局内容库

4.1.16.1 功能描述

全局内容库即是全局性的文件存储仓库，可以跨资源池为云主机提供镜像文件。在创建云主机、裸金属部署、网络负载均衡创建功能均可使用全局内容库中的文件，创建全局内容库支持使用本地盘和 **NFS** 存储。

4.1.16.2 实现原理

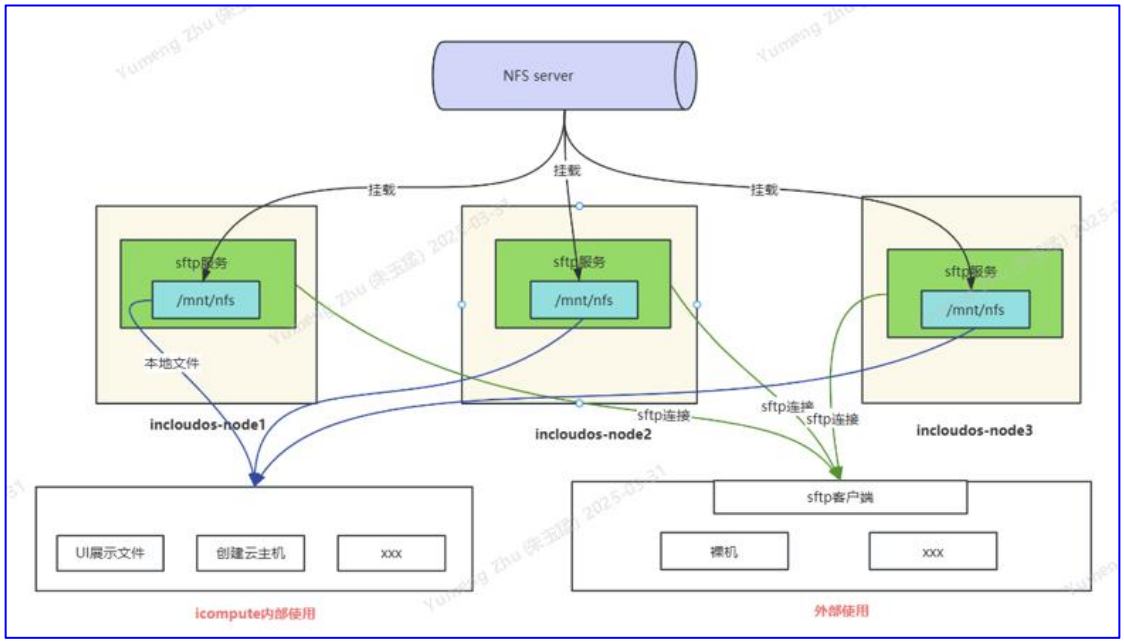
4.1.16.2.1 本地盘类型



本地盘全局内容库的实现原理：

- 扫描 **InCloud OS** 的安装节点，找到除了启动盘、光驱之外的所有的磁盘设备，用户可以在界面上选择磁盘设备、输入挂载点，创建全局内容库。
- 如果该磁盘设备没有文件系统，会先对该磁盘设备做格式化，然后挂载到挂载点上。如果有文件系统，会校验该文件系统是否可用，如果不可用抛出错误，如果可用挂载到挂载点。
- 磁盘挂载完成后，会在磁盘所在节点自动配置 **sftp** 服务，作为 **sftp** 服务端对外提供文件共享服务，并且将 **sftp** 的信息存入数据库。如果其他节点或者其他服务访问文件时，获取到 **sftp** 信息之后，直接访问 **sftp** 服务端地址即可。
- 在文件上传、删除时，如果接口访问的是本节点，直接操作本地文件，如果接口访问的其他节点，通过 **sftp** 来保存、删除文件。

4.1.16.2.2 NFS 存储类型



NFS 类型的全局内容库的实现原理如下：

- 系统会自动扫描出已安装到 InCloud OS 各节点的 NFS 信息，只有全部挂载到 icmp 节点的 NFS 才可以创建全局内容库。也可以手动输入外部的 NFS 地址，当 NFS 使用域名时，支持在界面输入域名解析地址，系统会自动将域名解析地址配置到各个节点。
- 创建全局内容库时，会把 NFS 挂载到节点的一个目录上，默认的在 /mnt/global-repo-NFS/目录下，外部的 NFS 可手动指定挂载点，已挂载的自动生成。挂载成功后，会在挂载目录自动生成 global-repo-data 目录，全局内容库的文件都是在这个文件夹下操作的，不会影响原有文件。
- NFS 挂载到挂载点后，会基于挂载点配置 sftp 服务。Sftp 的作用有两个：一个是为外部系统提供文件共享地址，另外一个是保证 NFS 存储的高可用，如果当前节点 NFS 服务不正常，则通过 sftp 访问文件，只

要有一个节点 **NFS** 配置正常，即可使用全局内容库功能。

- 系统内置 **1** 分钟一次的 **NFS** 健康检查机制，如果部分 **InCloud OS** 节点 **NFS** 网络不通或者权限配置错误会自动将该节点的文件访问方式切换为 **sftp**，如果全部节点异常则将全局内容库置为不可用状态并且显示原因。

4.1.16.3 约束

- 使用本地盘类型 **InCloud OS** 的任意安装节点要有非启动盘的其他盘符。
- 使用本地盘类型的如果是已经有文件系统的盘符，文件系统要在本节点内支持。
- 使用 **NFS** 类型的需要 **NFS** 与 **InCloud OS** 各个节点（监控节点除外）能正常通信且开放白名单.

4.1.17 镜像库

4.1.17.1 功能描述

镜像库是资源池内的存放镜像文件的仓库，目前可以上传 **ISO**、**OVA**、**qcow2**、**raw** 格式文件。支持对文件的上传、删除、编辑标签、设置 **cpu** 架构、芯片厂商、操作系统、启动引导固件等功能。镜像库支持以下类型：公有镜像库，其空间大小、文件数量与虚拟化资源池中镜像库保持一致，并且实时同步，镜像文件供属于同一虚拟化资源池的所有成员使用；私有镜像库，与 **vdc** 绑定且可以设置空间大小，仅限于绑定了此镜像仓库的 **VDC** 使用；共享镜像库，仅存储镜像元数据的逻辑镜像存储仓库，私有镜像库中的镜像被设置为共享后，仅有镜像元数据被保存到共享镜像库中，而不产生额外的物理存储空间。支持主流的操作系统，包

括 Windows Server 2003/2008/2012/2016/2019/2022， 以及 Oracle、Redhat、OpenSuse、Debian、Ubuntu、CentOS、红旗、麒麟信安、银河麒麟、统信、方德、凝思等主流 Linux OS，同时支持 Mac OS/VxWorks/安卓等嵌入式操作系统。

4.1.17.2 实现原理

镜像库：InCloud OS 层的镜像库主要是与 ics 层的镜像库做的映射，ics 层的镜像库是通过 sftp 或者其他的第三方的文件系统（NFS）实现的不同主机之间的文件共享。当 ics 层镜像库文件变化了，通过 websocket 消息实时通知 InCloud OS 层镜像库更新文件。

文件上传：InCloud OS 层的镜像库文件上传是通过分片上传的方式实现的。首先 UI 将整个文件分成小文件，然后一片一片的通过接口传给后台，后台将分片文件缓存起来，传完之后合并成一个文件，再上传到 ics 的镜像库。

4.1.17.3 约束

一个 ics 的镜像库只能对应一个公有镜像库，一个 vdc 在一个资源池下只能创建一个私有镜像库。

系统管理员只能使用公有镜像库创建云主机，vdc 管理员和用户可以使用公有、私有镜像库创建云主机。

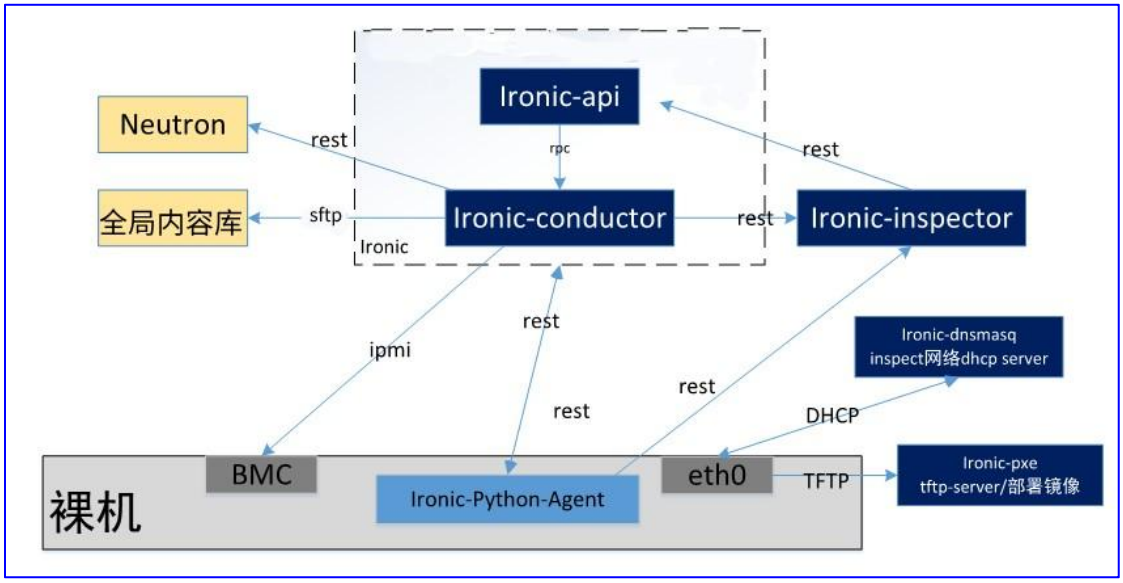
分片上传文件时，如果文件过大浏览器会有卡顿。在上传文件过程中不要刷新浏览器，否则会上传失败。

4.2 裸金属服务

在某些特定场景下，用户需要更多的硬件访问权以及更高的机器性能，比如，

高性能计算服务、数据库主机服务等，业内传统做法是将物理服务器放置在机房，每次对物理机服务器进行开关机操作或者安装操作系统等操作，都需要去机房；缺乏对机器的统一管理和调度，既浪费时间又耗费精力。为此，云平台提供裸金属服务，通过平台界面为用户提供独占式的物理服务器资源，可以远程操作资源，满足用户特定需求的可行性及高效性。

架构如下图所示，其中 **ironic** 的组件运行于裸金属资源池中，对裸金属节点进行生命周期管理：**ironic–api** 对外暴露 **API** 接口；**ironic–conductor** 是核心管理组件，通过向 **BMC** 下发 **IPMI** 命令进行带外管理，通过对 **PXE** 的配置管理和同 **IPA** 的配合，完成对裸金属的带内控制功能，如硬件信息检查和系统部署。此外，**ironic** 还同 **Neutron** 交互，对裸金属业务网卡上连交换机端口进行自动化的配置，**ironic** 从全局内容库下载用户指定的镜像模板。

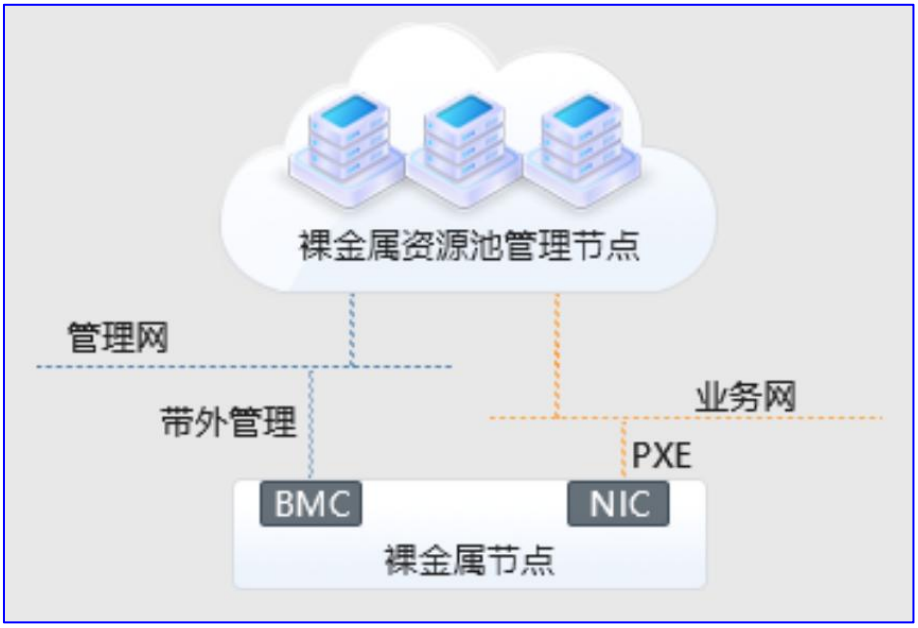


◆ 裸金属的管理模式：完全管理和精简管理

鉴于是否需要通过平台进行操作系统的部署，平台提供两种裸金属的管理模式：完全管理模式和精简管理模式；

◆ 完全管理裸金属概念及基本流程

完全管理模式是指除了能够进行带外 **BMC** 的开关机重启操作外，还可以通过业务网络 **PXE** 引导的方式，进行硬件信息收集、系统部署、磁盘清理、设备配置等等全部的功能；完全管理模式要求①裸金属具备 **BMC** 网卡，并同裸金属资源池管理网互通；②裸金属配备有业务网卡，并且业务网卡在上云平台管理时，需要初始化至资源池 **PXE** 网络二层打通，用以进行 **PXE** 引导。



完全管理模式裸金属网络架构示意图

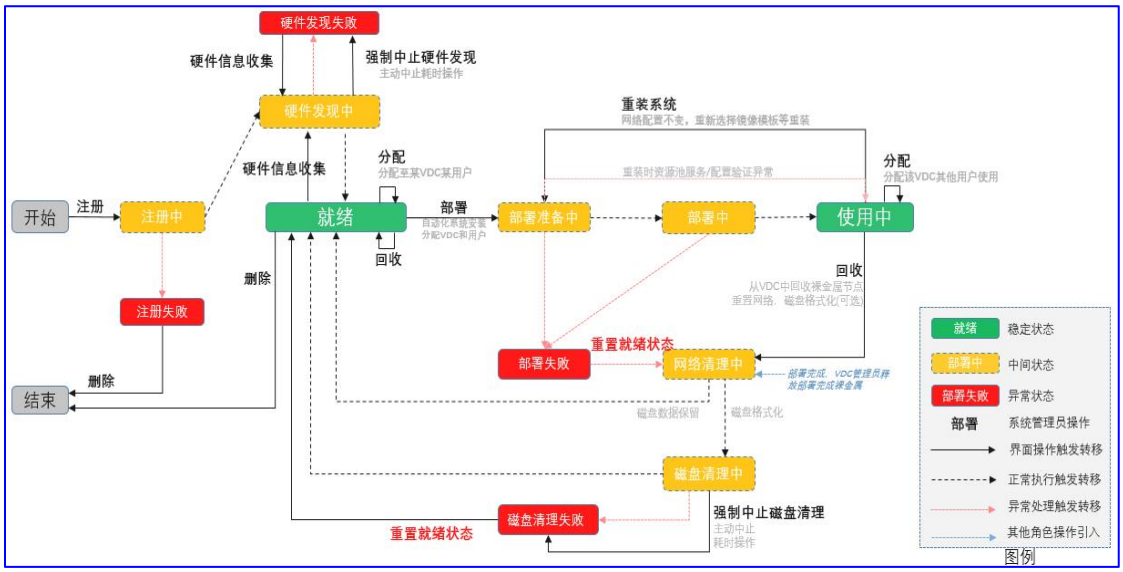
对于完全管理的裸金属，其上云的步骤主要有以下三个大的步骤：①注册与硬件发现。注册节点并收集服务器厂商、型号和序列号。完全管理模式裸金属通过业务网络 **PXE** 方式，引导代理服务（**IPA**）收集 **CPU**、内存、磁盘和网卡连接信息等。②分配与系统部署。系统管理员可以将一台未部署系统的裸金属节点分配至 **VDC** 及用户。完全管理模式可自动化部署系统和自动接入 **VPC** 网络，并且进行主机名等的系统初始化配置。③回收与系统清理。用户使用完毕后，可以从 **VDC** 回收裸金属节点，并进行网络配置和磁盘(可选)的清理，回收后可再次进行分配部署，循环使用。

完全管理模式的裸金属在平台进行生命周期管理时，涉及如下图的状态转移，

重点几个状态：

就绪：第一步注册与硬件发现后，裸金属会处于“就绪”状态，表示当前裸金属已经收集好必需的硬件信息（如网卡连接信息），可以对其进行下一步的系统操作了。

使用中：对就绪状态的裸金属完成“部署系统”操作后，裸金属转入“使用中”状态，表示通过平台已经对裸金属部署好了系统并配置好了网络，用户已经在使用裸金属的算力资源了。



完全管理模式裸金属状态转移图

◆ 精简管理裸金属概念

精简管理模式是指裸金属已经安装好了操作系统，平台对裸金属进行基于 **BMC** 网卡的开关机重启及监控功能。因此，这种管理模式仅需要裸金属配置好 **BMC** 网卡，并同资源池的管理网三层互通即可。

4.2.1 硬件信息收集

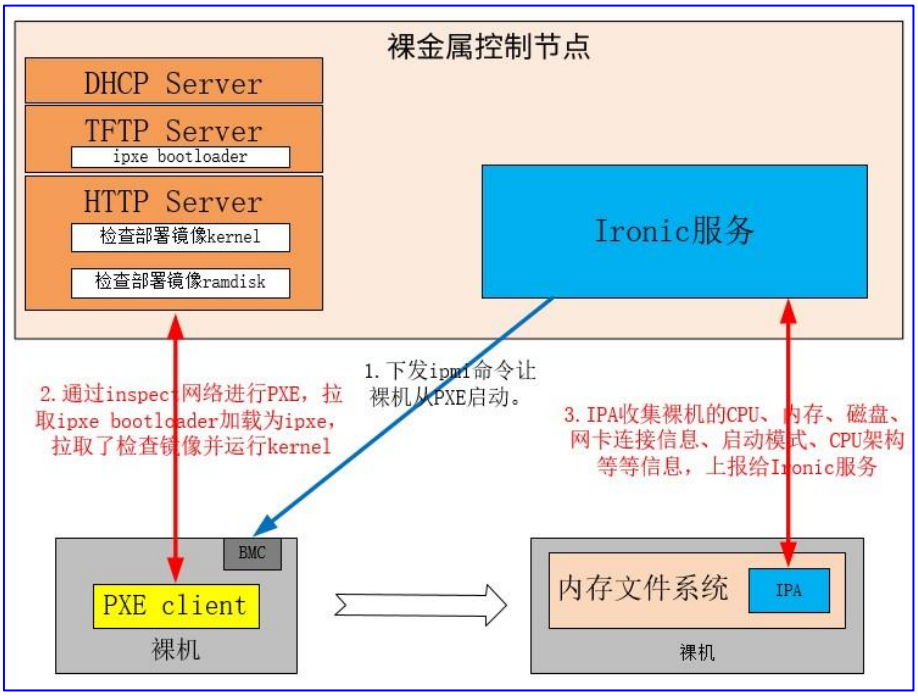
4.2.1.1 功能描述

裸金属硬件信息收集是指自动化地收集 CPU、GPU、内存、磁盘、网卡连接信息、启动模式等等设备的详情，这些设备信息是后后继步骤所必需的，如 1）CPU、GPU、内存、磁盘信息为用户选择满足业务需求的裸金属节点提供指导；2）网卡信息是系统部署时自动化配置裸金属业务网卡上连交换机端口的必要信息。

4.2.1.2 技术原理

裸金属上云/硬件信息收集前，管理员需要将裸金属的业务网卡上连交换机端口配置到所属资源池的 PXE 网络中，实现裸金属接入 PXE 网络，二层打通。

裸金属注册裸金属时，自动化地对裸金属进行硬件信息收集，基本原理为：通过 BMC 带外管理驱动裸金属进行 PXE，引导裸金属通过业务网卡 PXE 引导安装有代理服务（IPA）的内存文件系统，由代理服务收集各种硬件信息并上报至裸金属资源池的管理服务，从而实现裸金属硬件信息收集。



具体步骤为：

- **Ironic** 服务通过裸金属节点注册的 **BMC IP**、用户名、密码参数，通过 **ipmitool** 工具，向裸金属的 **BMC** 网卡下发 **IPMI** 命令，驱动裸金属下电、上电并从 **PXE** 启动。
- 当裸金属上电启动以后，从 **PXE** 进行引导。加载网卡的 **PXE** 固件进行 **bootstrap**，首先发送 **DHCP Discover** 报文，向 **dhcp server** 请求 **ip** 地址，**dhcp server** 在发放 **ip** 的时候会将 **tftp server** 的地址一同发送。裸金属节点拿到 **ip** 之后会到 **tftp server** 获取 **iPXE** 的可执行文件，从而切换至 **iPXE**。**iPXE** 通过从 **HTTP Server** 拉取 **iPXE** 的配置脚本（脚本指定了本次启动的 **kernel** 和 **initramfs**），进而从 **HTTP Server** 拉取 **ironic-agent.kernel** 和 **ironic-agent.initramfs** 进行启动。
- 当镜像启动以后，镜像中安装的 **IPA** 自动运行并执行 **inspect** 任务，**IPA** 的 **inspect** 会通过配置，会调用硬件管理器进行各种设备的信息采集，采集后，通过调用资源池管理节点的 **ironic-inspector** 服务的 **API** 接口

上报数据。**Ironic-inspector** 对数据解析后，调用 **ironic-api** 的 **API** 接口，将硬件信息更新至对应的裸金属节点中。

4.2.1.3 约束

无

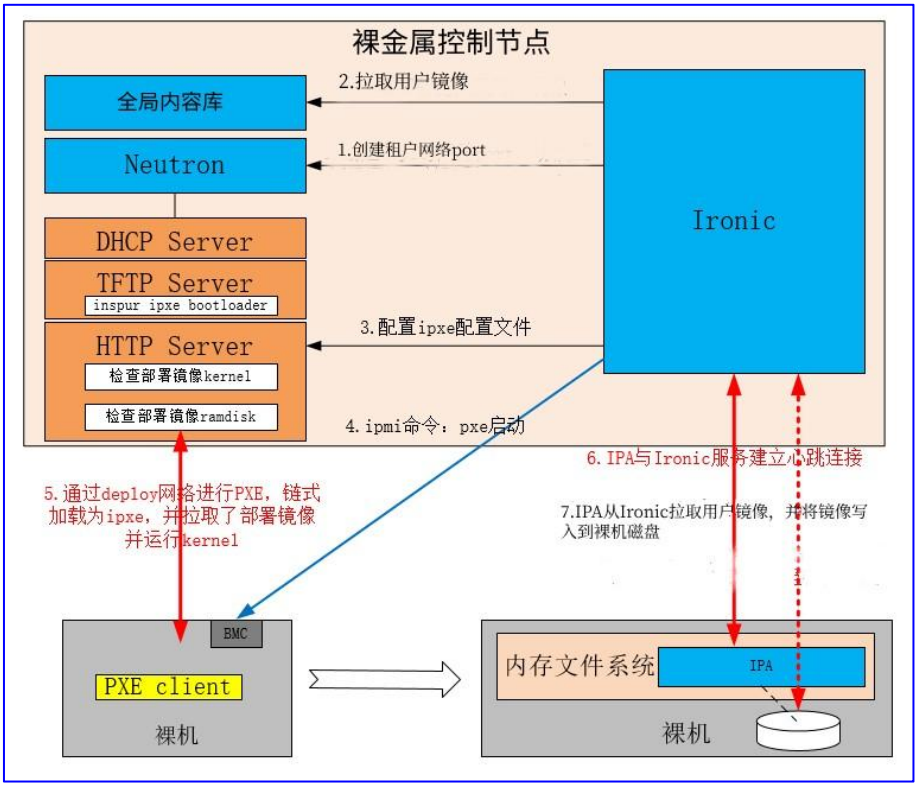
4.2.2 部署系统

4.2.2.1 功能描述

裸金属部署系统是指自动化地将用户指定的镜像模板安装安装部署至本地物理磁盘中，并且附带一些网络自动化配置和系统初始化配置的功能：**1**）支持多租户网络隔离，部署后自动化地将裸金属业务网卡接入指定 **VPC** 网络；**2**）**IP** 分配支持自动分配及手动指定两种方式；**3**）支持主机名**(hostname)**、密码的指定与配置；

4.2.2.2 技术原理

裸金属部署系统的技术原理同硬件信息收集类似，均是通过 **BMC** 带外管理驱动裸金属进行**PXE**,引导裸金属通过业务网卡**PXE**引导安装有代理服务(**IPA**)的内存文件系统， **IPA** 运行后回调 **ironic** 接口并同 **ironic** 建立心跳。**Ironic** 向 **IPA** 下发请求，由 **IPA** 从 **HTTP server** 拉取用户指定的镜像并写入物理磁盘中。并通过 **Neutron** 对交换机的管理，实现裸金属交换机端口的配置切换。



具体步骤为：

- 当用户选择某个处于“可部署”状态的裸金属节点进行部署的时候，会调用 **Ironic API** 接口准备裸金属主机，**Ironic API** 接受到消息以后，发送给 **Ironic –conductor**，**conductor** 服务调用 **Neutron** 服务接口创建 **neutron port**；并连同用户指定的主机名**(hostname)**、密码生成 **ConfigDrive** 数据；
- **Ironic–onductor** 会从全局内容库中把用户镜像下载到 **Ironic** 节点中缓存。
- **Ironic–conductor** 为裸金属 **PXE/iPXE** 准备配置；
- **Ironic–conductor** 通过 **ipmitool** 向裸金属节点下发 **IPMI** 命令，驱动裸金属进行 **PXE** 启动；
- 裸金属基于业务网络的 **PXE** 逻辑网络进行 **PXE** 启动，获取 **ip** 地址并获取 **IPA** 镜像（**ironic–agent.kernel** 和 **ironic–agent.initramfs**）；

- IPA 镜像启动后， IPA 会调用 ironic-api 接口进行回调，ironic-conductor 服务与 IPA 建立心跳连接；
- Ironic-conductor 服务向 IPA 下发请求，令其进行镜像部署。IPA 以 HTTP 方式从 Ironic Conductor 拉取第 2 步缓存的用户镜像，并写入裸金属节点的本地磁盘；IPA 将 ConfigDrive 写入磁盘的最后 64M 的分区中；ironic-conductor 调用 Neutron 接口，更新关联的 port 属性，写入网卡连接信息，触发 Neutron 通过 genericswitch 驱动将裸金属交换机端口切换至指定的 VPC 中。最后 ironic-conductor 通过 IPMI 接口设置从硬盘启动，启动后，由初始化工具 cloudinit 读取 ConfigDrive，自动化生成网卡的配置文件、设置主机名、修改密码。

4.2.2.3 约束

无

4.2.3 磁盘清理

4.2.3.1 功能描述

裸金属支持本地物理磁盘清理，及进行文件系统的格式化，用户在回收完全管理的裸金属时可以勾选快速格式化对磁盘进行磁盘清理。

4.2.3.2 技术原理

裸金属物理磁盘采用控制代理带内清除文件系统的方式实现：在用户在删除云物理机并选择了“清理物理磁盘”后，ironic 会通过驱动裸金属通过清理网络（复用 PXE 网络）进行 pxe，裸金属在 pxe 过程中会拉取 ironic 服务为其下载

的部署镜像，进行引导启动。当部署镜像上电以后，**IPA** 回调 **ironic** 服务建立心跳连接，**ironic** 下发清理磁盘的命令至 **IPA**，由 **IPA** 执行 **wipefs** 对磁盘中的 **metadate** 数据进行清理。

4.2.3.3 约束

无

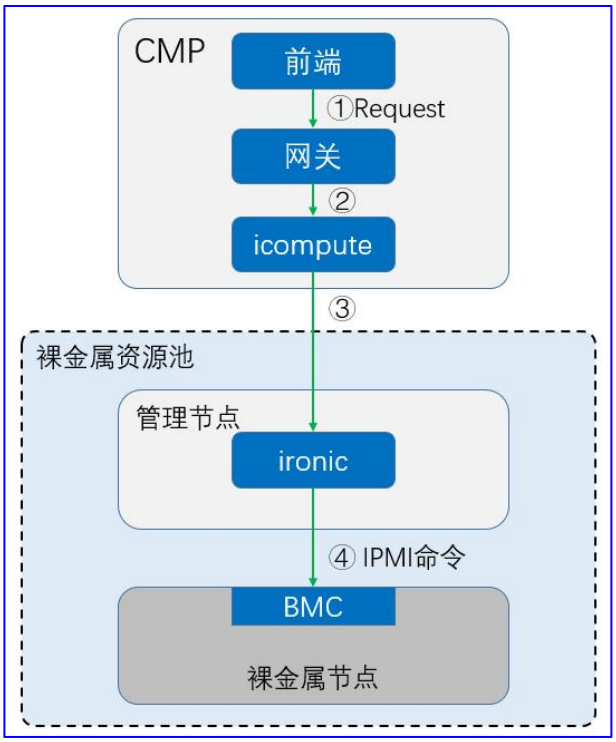
4.2.4 精简管理

4.2.4.1 功能描述

在某些特定场景下，用户对于已经上业务的物理服务器，可以在不重装系统，尽量不重启机器的情况下，实现对外部服务器进行统一管理与分配，监控物理服务器性能，评估其资源占用率等功能。

4.2.4.2 技术原理

以精简管理方式上云，通过 **BMC** 地址来识别为真实物理服务器，并且通过带外方式提供电源管理服务。



裸金属精简管理流程如上图所示，流程如下：

- 在 **CMP** 前端页面，输入需要精简管理裸金属的 **BMC IP** 地址，**BMC** 用户，**BMC** 密码等信息。
- **CMP** 网关将请求转发 **ibaremetal** 模块。
- **icompute** 调用 **ironic** 的添加裸金属接口发起纳管操作。
- **ironic** 根据校验 **BMC** 信息，并在 **node** 表中增加一条记录。
- **CMP** 接收到 **ironic** 成功响应后，**icompute** 模块在 **node** 表里同样增加一条记录。

4.2.4.3 约束

无

4.2.5 GPU 裸金属

4.2.5.1 功能描述

GPU 裸金属是在裸金属服务基础上，针对智算场景提供的带有 GPU 卡的裸金属管理服务。具体提供功能包括：

- 裸金属硬件信息收集支持收集 GPU 卡信息(此版本仅支持英伟达 GPU)，包含 GPU 卡个数、厂商、型号、显存；
- 裸金属列表支持通过 GPU 字段进行模糊查询，方便用户快速查找带有某种 GPU 的裸金属服务器；
- 支持给裸金属自定义硬件配置，在非兼容 GPU 卡的情形下，系统管理员可以指定裸金属自定义 GPU 设备，向用户提供展示 GPU 卡的厂商、型号等信息，方便用户分辨选择 GPU 裸金属；
- 在首页中展示了裸金属资源池中的 GPU 卡数量和使用量，方便用户掌握 GPU 在裸金属下的使用情况。

4.2.5.2 技术原理

针对裸金属硬件信息收集时收集 GPU 卡信息，其技术原理为在 IPA 镜像中打入了特定 GPU 卡的驱动（如 NVIDIA 的 GPU 驱动），并在 IPA 中扩展了一个对应厂商的 GPU 硬件管理器，此硬件管理器通过相应的 python 依赖包收集该厂商 GPU 卡设备信息。因此，在硬件信息收集阶段，裸金属 PXE 至 IPA 镜像后，IPA 代理通过执行该硬件管理器收集并上报 GPU 卡信息至 ironic 服务，实现 GPU 卡的硬件信息收集。

4.2.5.3 约束

当前仅支持自动收集 **NVIDIA GPU** 信息。

4.2.6 裸金属规格管理

4.2.6.1 功能描述

裸金属规格是对裸金属硬件配置的画像及分类，包含 **cpu**、内存、磁盘、**GPU** 卡、以太网卡、**IB** 卡属性，用于在用户申请裸金属时提供易于阅读的配置类别，方便用户选择裸金属，并且基于裸金属规格对裸金属进行计费。

4.2.6.2 技术原理

裸金属规格的生成及刷新是在裸金属资源池管理界面的“收集裸金属规格”，该功能是在 **CMP** 层对完全管理模式的裸金属进行硬件信息画像及分类，按照 **cpu**、内存、磁盘、**GPU** 卡、以太网卡、**IB** 卡进行比对分类，自动创建出新规格，并刷新各个裸金属规格关联的裸金属数量。

4.2.6.3 约束

只对完全管理的裸金属进行刻画规格。

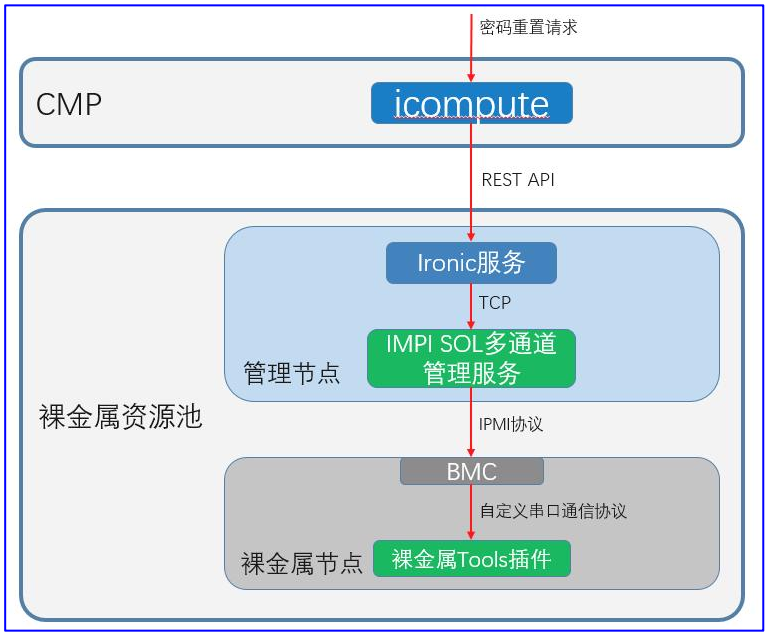
4.2.7 重置密码

4.2.7.1 功能描述

重置密码是指针对完全管理模式下已经部署好系统的裸金属进行密码重置。平台有基本的密码复杂度要求，对于具体的操作系统类型，在重置时需要满足特殊的要求。

4.2.7.2 技术原理

裸金属操作系统的（一键式）重置密码基于 BMC 带外网络完成，通过自研的 IPMI SOL 多通道串口通信机制实现，不需要带内网络打通。该机制主要由运行于资源池管理节点的多通道管理服务、运行于裸金属操作系统的插件组成，两个组件通过 BMC SOL 链路进行通信。IPMI SOL 多通道管理服务用于在 IPMI 仅限的单一会话中将 SOL 通信链路虚拟出多个会话通道，并提供 TCP 连接建立新通道，通道的数据经过自研协议进行组织，并通过 ipmitool 工具封装进向 BMC 发送的 IPMI 报文中；裸金属插件监听串口，并基于根据自研协议进行解析，调用对应通道的处理逻辑，完成相应的任务，裸金属插件会在节点部署系统时自动地注入进裸金属操作系统运行。



具体到重置密码过程，大体流程为：1）icompute 接收重置密码的请求后，向 ironic 服务下发重置密码请求；2）ironic-api 通过 rpc 消息调用相应的 ironic-conductor 组件向裸金属重置密码，ironic-conductor 向本节点的 IPMI SOL 多通道管理服务建立 TCP 连接，发送重置密码的 TCP 报文。3）IPMI SOL

多通道管理服务基于同目标裸金属已建立好的 IPMI session，将 TCP 报文转换为 IPMI 报文，发送至裸金属 BMC。4）裸金属 BMC 基于 IPMI SOL 特性（Serial Over LAN，IPMI 2.0 协议支持），将 IPMI 报文转为串口数据，发送至串口；5）裸金属插件读取串口数据，并执行重置密码命令，完成密码重置。

4.2.7.3 约束

仅支持完全管理模式的裸金属。

4.2.8 重装系统

4.2.8.1 功能描述

重装系统是指对完全管理模式且已经部署好系统的裸金属进行系统的重新安装。支持重新选择镜像，支持重新设置主机名和系统的密码。网络配置不变。

4.2.8.2 技术原理

同裸金属部署系统类似，裸金属重装系统的原理也是通过 BMC 带外管理驱动裸金属进行 PXE，引导裸金属通过业务网卡 PXE 引导安装有代理服务（IPA）的内存文件系统， IPA 运行后回调 ironic 接口并同 ironic 建立心跳。Ironic 向 IPA 下发请求，由 IPA 从 HTTP server 拉取用户指定的镜像并写入物理磁盘中。并通过 SDN 对交换机的管理，实现裸金属交换机端口的配置切换。详细步骤参考裸金属部署。

4.2.8.3 约束

仅支持完全管理裸金属。

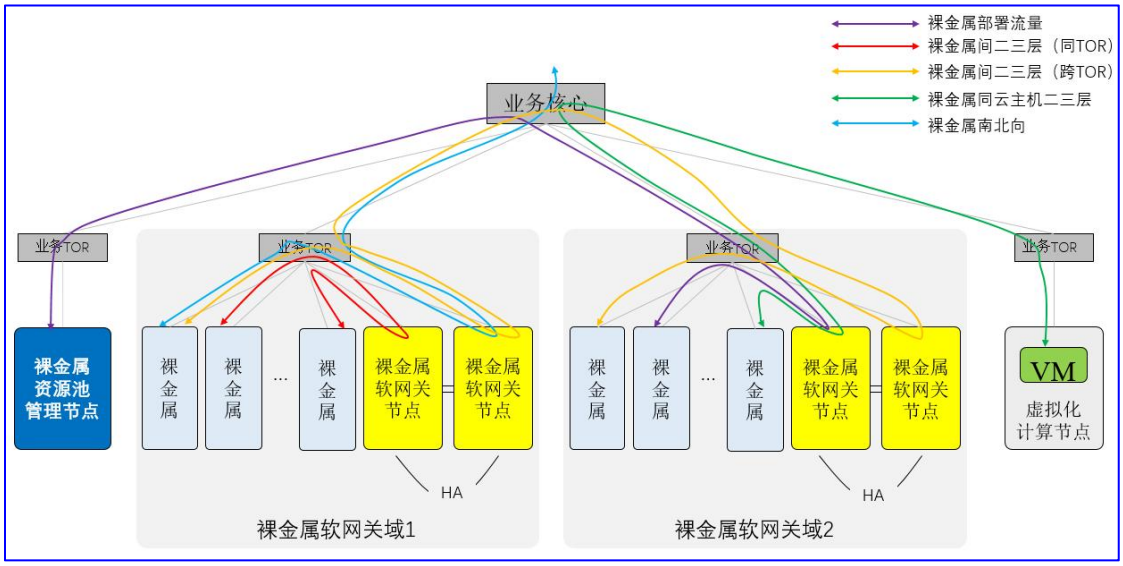
4.2.9 裸金属 overlay 网络

4.2.9.1 裸金属网关

4.2.9.1.1 功能描述

为裸金属提供网关服务。一个裸金属网关可以为多个裸金属提供服务。实现 **overlay** 网络下，裸金属与其他裸金属或者虚拟机之间 **overlay** 通信，所有流量都需要经过裸金属网关。

4.2.9.1.2 技术原理



整体组网架构

在一个裸金属资源池内，通过划分创建裸金属高可用网关组来实现 **overlay** 组网下裸金属的注册、部署系统基本功能，以及部署系统完成的裸金属可通过 **overlay** 隧道实现裸金属间二三层互通、裸金属与云主机间二三层互通的基本网络功能。

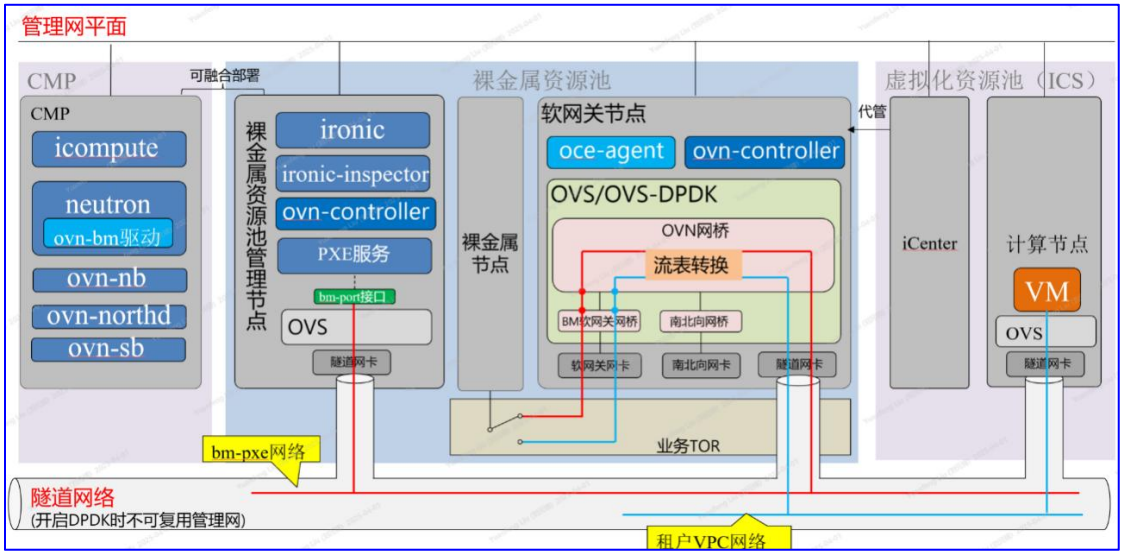
（一）裸金属网关组网下的裸金属流量模型

1. 同 **TOR** 下裸金属与裸金属之间的二三层互通都需要经过该 **Tor** 下的裸

金属网关节点。

2. 跨 **TOR** 下裸金属与裸金属之间的二三层需要分别经过各自直连 **Tor** 下的裸金属网关节点。

3. 跨 **TOR** 下裸金属与虚拟机之间的二三层互通需要经过裸金属直连的 **Tor** 下裸金属网关节点并通过业务核心交换机送到虚拟机所在的 **Tor** 节点。



（二）裸金属注册硬件信息发现流程

1. 裸金属准备阶段：

- 创建 **Geneve** 类型 **pxe** 网络, **vni** 为 **10001**，接入 **vlan 4100**
- 创建高可用网关组，添加 **Tor** 交换机以及裸金属网关节点，并配置 **vlan** 分配范围。此时，每个高可用网关组下 网关节点自动创建至各个裸金属资源池管理节点的 **rep-port**(经 **neutron ovn-bm** 绑定，**oce-agent** 将自动创建 **patch port**,下发流表至网关节点 **ovs**，**vlan tag 4100**，并实现 **VLAN** 至 **geneve** 转换)，打通网关节点到裸金属资源池管理节点上 **bm-port** 的逻辑链路。
- 将裸金属接入 **Tor** 的端口的 **vlan** 修改为 **access 4100.**

2. 裸金属注册时 **ironic** 流程：

- **ironic** 向裸金属节点下发 **ipmi** 命令，令裸金属从 **PXE** 启动
- 裸金属 **PXE**：分析跟踪 **DHCP discover** 报文：
- **DHCP discover** 属于广播报文，**UNTAG**，进入接入裸金属接入交换机，打上 **pxe** 网络 **VLAN** 号，如 **4100**。
- 交换机将广播报文转发至 **VLAN4100** 所有端口，包括网关节点 **BM** 网关网桥的上连端口，进入 **BM** 网关网桥的 **pipeline**。
- 之后报文在 **BM** 网关网桥上去掉 **TAG4100** 后，通过 **rep-port** 的 **patch port** 转发至 **OVN** 网桥。报文在 **OVN** 网桥根据 **OVS** 流表封装为 **Geneve** 报文，广播至多个控制节点 **Geneve VTEP**。
- 控制节点根据流表处理报文，转发至 **bm-port** 接口。监听 **bm-port** 的 **dnsmasq** 服务接收并处理 **DHCP** 请求返回 **DHCP offer** 报文。

由此往下，裸金属即可完成 **DHCP**、**TFTP**、**HTTP** 整个 **iPXE** 启动过程，拉取到 **kernel**、**initramfs** 文件启动，最终实现硬件信息检查。

(三)裸金属网关组网下裸金属系统部署流程：

1. CMP 触发一个裸金属部署系统流程之后，**Ironic** 通过 **ipmi** 命令使裸金属从 **PXE** 启动，裸金属通过网关节点流表和 **bm-pxe** 网络 **DHCP** 到地址，并完成后续的 **PXE**、内存文件系统启动和用户镜像的拷贝流程。

2. Ironic 向 **tenant port** 添加绑定信息 **binding:profile**，完成 **bm-pxe** 网络和租户 **VPC** 网络的切换。**Neutron-server** 做 **ml2** 驱动的绑定完成 **tenant port** 绑定信息的 **update**。**Neutron server** 调用 **ovn-bm** 驱动做 **port** 绑定的 **update**，根据 **port binding:profile** 中的上联交换机，找到所属高可用网关组中的裸金属网关节

点，分配低层 VLAN 号，genericswitch 将交换机端口切换至该低层 VLAN；OCE-Agent 创建 tenant 代表口;ovn-controller 下发 tenant 相应流表。完成网关节点到租户 vpc 网络的逻辑打通。

4.2.9.1.3 约束

只支持完全管理模式下的裸金属。

4.2.9.2 裸金属安全组

4.2.9.2.1 功能描述

通过裸金属网关使用 Geneve 类型 VPC 子网为裸金属部署系统后，关联了 Geneve VPC 子网的裸金属网卡可以配置安全组。安全组支持：

- 定制 TCP 规则
- 定制 UDP 规则
- 定制 ICMP 规则
- 自定义 IP 协议规则
- 自定义 DNS 规则
- 支持黑白名单功能
- 支持优先级
- 支持绑定地址组资源
- 支持配置端口集
- 支持有、无状态安全防护

4.2.9.2.2 技术原理

裸金属安全组实现原理同云主机类似，区别在于虚拟机的安全组相关流表下发在所在主机节点上，而裸金属则是通过自研 **L2** 驱动 **ovn-bm**，并运用层次化端口绑定技术，配合自研的运行于裸金属网关节点的 **ovn** 扩展代理（**oce-agent**）控制裸金属代表口（**patch port**）的创建删除，触发裸金属网关节点的 **ovn-controller** 将相关流表（包括安全组功能）下发至裸金属网关节点上。而在数据面上，通过层次化端口绑定机制为裸金属分配了网关域内唯一的 **VLAN** 号，并由 **generic-switch** 驱动对裸金属接入端口进行了动态修改，由此实现将裸金属流量通过 **Trunk** 口“引流”至网关节点，并通过 **patch port** 引入 **ovn** 网桥，经由 **OVS** 流表实现安全组的功能。

OVN、**OVS** 层面的安全组实现逻辑同云主机相同，逻辑如下：

- **OVN** 通过 **Port Group** 实现安全组，虚拟机接口加入的 **Port Group** 中。
- **OVN** 通过 **ACL** 实现安全组规则，**ACL** 会加入到 **Port Group** 中。
- **OVN** 下发流表到 **OpenVSwitch**，实现具体的安全组功能。
- 裸金属代表口加入安全组后，进出代表口的流量需要经过安全组检查。

4.2.9.2.3 约束

只支持对裸金属网关下关联 **Geneve** 类型 **VPC** 子网的网卡配置安全组。

4.2.9.3 裸金属浮动 IP

4.2.9.3.1 功能描述

裸金属网关下关联 **Geneve** 类型 **VPC** 子网的网卡支持绑定浮动 **IP**。**VPC** 子网中的云资源绑定浮动 **IP** 后可以与外界互相通信，其本质是一种 **NAT** 技术。浮

动 IP 支持 IPv4 和 IPv6，IPv4 的浮动 IP 支持绑定 IPv4 的 port，IPv6 的浮动 IP 支持绑定 IPv6 的 port。

4.2.9.3.2 技术原理

裸金属绑定浮动 IP 的技术原理同安全组类似，其实现前提均是通过将裸金属流量“引入”裸金属网关节点，并通过代表口（patch port）将裸金属的网络模型同云主机拉齐。裸金属网关节点之于裸金属代表口相当于宿主机之于虚拟机的 TAP 设备。

而在 SDN 层面的技术原理为：软 SDN 实现浮动 IP 时，创建 OVN 的 nat 资源，nat 的类型为 dnat_and_snat。OVN 按照 nat 信息转换成相应的 ovs 流表。基于 ovs 流表实现 nat 的转换。对于目标地址是浮动 IP 的报文，更换目标 IP 为裸金属 IP，进行转发。对于源地址是裸金属 IP，出去外部的流量，更换源地址为浮动 IP。

4.2.9.3.3 约束

只支持对裸金属网关下关联 Geneve 类型 VPC 子网的网卡绑定浮动 IP，并且每台裸金属仅支持绑定 1 个浮动 IP。

4.2.10 裸金属纯文本控制台

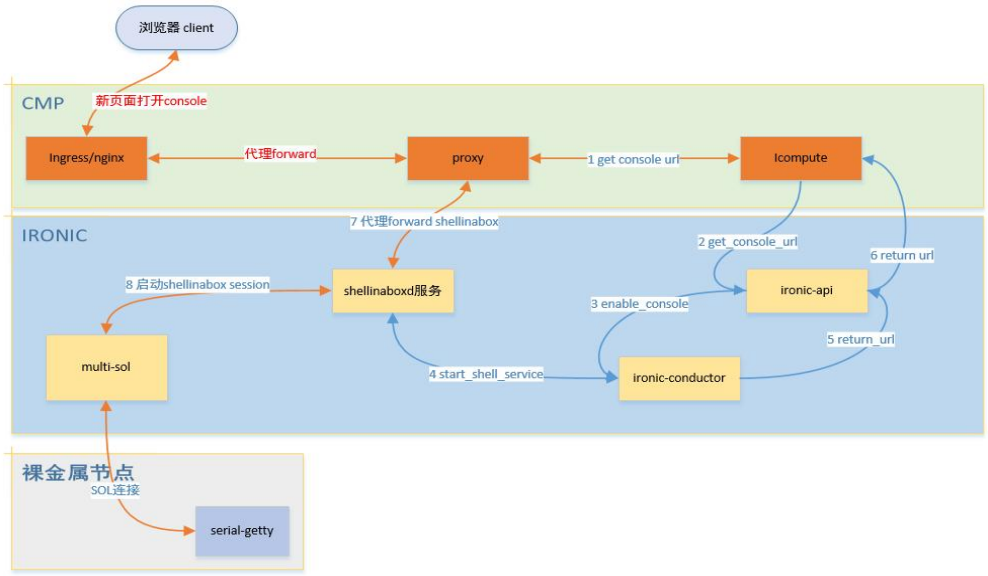
4.2.10.1 功能描述

裸金属纯文本控制台是指基于 BMC SOL（Serial Over LAN）实现的裸金属纯文本 Console，方便用户通过浏览器登录裸金属 Linux 类型的客户系统进行操作管理。与旧版本不同，InCloudOS V8.2.0 版本的裸金属控制台在 CMP 层增加

代理服务 **bm-proxy**，用于转发裸金属控制台的数据，访问时暴露的 **URL** 为 **CMP** 地址及裸金属节点 **id**，解决了旧版本裸金属控制台 **URL** 直接暴露裸金属资源池管理节点 **IP** 端口以及 **URL** 无认证的安全问题，且不再要求浏览器必须直接可访问裸金属资源池，更加安全和易用。

4.2.10.2 技术原理

在 **CMP** 部署 **bm-proxy** 服务，**web** 客户端访问 **cmp** 地址**/bm_console/**裸金属 **ID** 时，**ingress-control** 会重定向到 **bm-proxy** 服务，**bm-proxy** 内部调用 **icompute-api** 获取裸金属节点的文本控制台 **url**，并将此裸金属 **ID** 的文本控制台的数据转发到对应的 **shellinaboxd** 服务上。



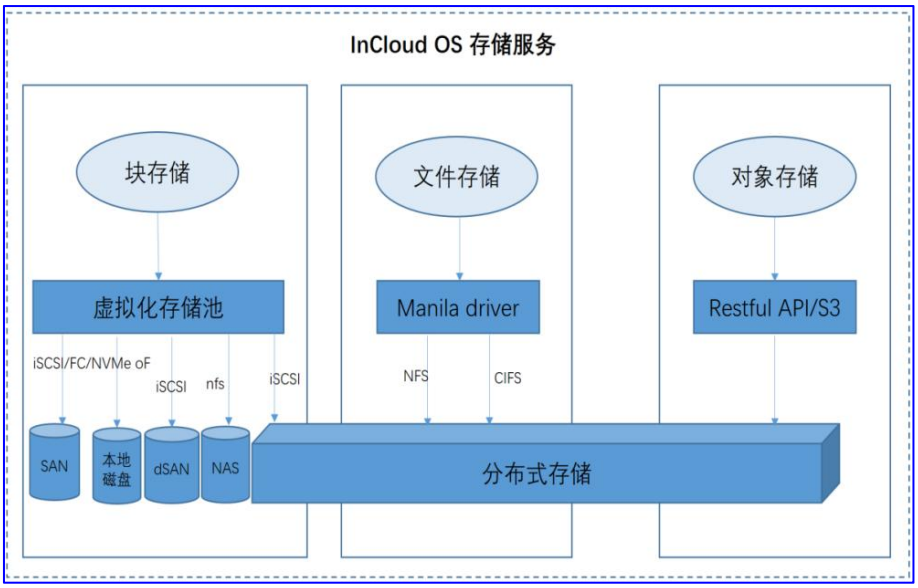
bm-proxy 识别 **url** 中的 **node_id**，如果需要调用 **icompute** 获取裸金属节点的文本控制台 **url**，调用 **icompute** 获取 **url** 后，标记裸金属节点和此 **bm-proxy** 绑定，并更新到其他 **bm-proxy** 上，**ingress** 流到其他 **bm-proxy** 上的数据，会转发到和裸金属节点绑定的 **bm-proxy** 上，以此保证 **shellinaboxd** 服务端保持一个 **IP** 的客户端访问。

4.2.10.3 约束

仅支持完全管理裸金属。

4.3 存储服务

随着大中型企业快速的发展，企业用户规模不断扩大，不但需要满足用户存储大量文本、图像和视频等要求，还要保证数据的稳定性和安全性，因此对于存储的异构多样性要求越来越高，**InCloud OS** 存储服务架构图如下：



云平台存储服务支持块存储、文件存储和对象存储三种存储类型，三种存储类型各有优势，相互补充，用户可以根据实际的业务需要，选择相应的存储类型。

块存储主要是基于文件系统（如 **ocfs** 文件系统）或 **Block** 块的模式，支持将任意大小的文件或块的方式挂载到云主机上，为云主机提供持久化的存储设备。**InCloud OS** 的虚拟化存储池是存储设备与资源抽象为数据存储池，数据存储池是一个逻辑资源池，它将不同存储设备的差异进行隐藏，提供统一抽象的模型提供存储能力与接口，目前可被管理和抽象的存储设备有：

- 主机上的本地磁盘

- SAN（Storage Area Network）存储（包括 iSCSI、FC SAN 以及 NVMe over Fabrics）上提供的块设备，如 LUN（Logical Unit Number）、NVMe 等
- NAS（Network Attached Storage）存储上划分的文件系统
- 分布式存储池

文件存储提供了可共享访问、弹性扩展、高可靠以及高性能的共享文件存储服务。支持将文件存储同时挂载给多个云主机，允许同时进行访问，避免了大量的数据拷贝与同步成本，达到真正的资源共享。支持 NFS 和 CIFS 两种主流协议，NFS 可以通过网络让不同操作系统的计算机可以共享数据，CIFS 主要是针对 Microsoft Windows 操作系统的本地网络文件系统。

对象存储是兼具块存储高速直接访问磁盘和文件存储易共享的特点。它主要用于非结构化数据的存储，核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备（Object-based Storage Device，OSD）构建存储系统，能够自动管理其上的数据分布。

4.3.1 块存储

InCloud OS 块存储服务提供的云硬盘，主要用来给云主机提供块存储空间，用户可以在线创建云硬盘并挂载给云主机，云硬盘的使用方式和传统服务器硬盘完全一致。同时云硬盘具有更高的数据可靠性，更高的 I/O 吞吐能力和更加简单易用等特点，适用于文件系统、数据库或者其他需要块存储设备的系统软件或应用。

4.3.1.1 块存储池管理

4.3.1.1.1 功能描述

由于存储设备的能力、接口协议等差异性，利用 **InCloud OS** 存储虚拟化技术，可以将不同的存储设备转换为统一管理的存储池，存储池可理解为一种逻辑容器，它以文件或块的形式来存取云主机的相关资源，如磁盘镜像、快照和备份等数据。**InCloud OS** 针对用户不同的存储设备和不同的业务需求，提供了多种数据存储类型，包括：高可用存储池（**CFS** 共享模式）、本地存储池（**CFS** 专项模式）、**NFS** 存储池和高速高可用存储池（裸设备存储池）等，还支持外接分布式存储。

高可用存储池是采用集群文件系统，它提供共享和专享两种模式使用模式。共享高可用存储池可以被集群中的多个主机访问使用，为 **HA** 和 **DRS** 等高级功能的实现提供了支持；专享数据存储是充分利用服务器上的本地硬盘资源，格式化完成的数据存储只允许被此服务器主机进行访问。

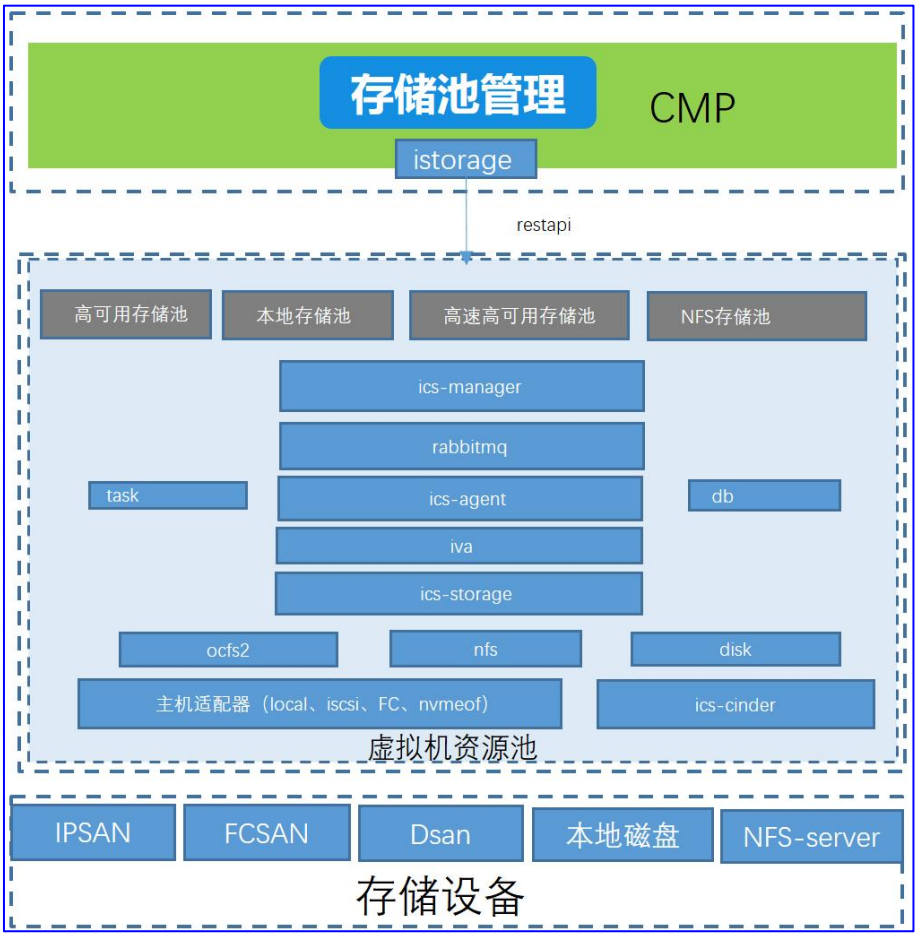
NFS 存储池是提供访问 **NAS** 服务器的功能，使用 **NFS** 数据存储之前需要按照特定的配置、网络和 **NFS** 数据存储准则。**NAS** 服务器需要配置主机能访问的网络连接。同时在配置方面，**NAS** 服务器需要支持 **NFS v3**、**NFS v4.1** 两种协议版本之一，同时支持启动 **no_root_squash** 权限控制，并确保共享目录添加读/写权限。用户不能使用不同的 **NFS** 协议版本挂载同一个数据存储，因为 **NFS v3** 和 **NFS v4** 客户端不使用相同的锁定协议可能导致用户数据不一致。

高速高可用存储池是基于小 **LUN** 的高性能虚拟磁盘技术方案，通过将存储设备上创建的 **pool** 以 **iSCSI** 或 **FC** 的形式挂载给主机，然后以 **pool** 为单位构建裸设备存储池。用户可以基于裸设备存储池创建裸设备磁盘，每个裸设备磁盘对

应了 **pool** 里面的一个 **LUN**。这种类型的虚拟磁盘不需要在存储设备上创建集群文件系统，从而有效降低 **I/O** 路径性能损耗。

存储池标签管理是针对云平台对接多套虚拟化资源池或单资源池对接多套存储池场景，通过为存储池设置资源标签，如：高性能存储池、中性能存储池、低性能存储池等，方便管理人员能够根据标筛选存储池，也可通过标签识别合适的存储池，用于资源的发放、迁移、删除等管理操作，提升用户的易用性。

4.3.1.1.2 技术原理



存储池管理架构图如上图所示，用户通过 **CMP** 控制台管理虚拟化存储池，虚拟化存储池管理是控制台发送请求到 **ics–manager** 服务，并将请求参数整合处理后发送到 **ics–agent**，然后转发到 **iva**，最后通过 **ics–storage** 完成存储管理操作。

虚拟化存储池使用的关键技术有：

- 存储池柔性卸载

  基于存储池柔性卸载特性，当存储设备无法下发 **IO** 时，不会引发主机重启，从而保证客户业务在极端情况下不会因为主机重启而中断。当主机连接不到存储设备心跳盘，则该主机会卸载相应的存储池，云主机通过 **HA** 机制在其他可正常访问的存储设备的主机上重启。

- 存储域通信模式

  高可用存储池的节点位于同一个存储域中，通过节点间通信完成对共享资源的并发访问控制。存储域支持三种通信模式：网络通信模式、磁盘通信模式和混合通信模式。网络通信模式适用于网络稳定的环境，性能好但对网络波动容忍度差。磁盘通信模式不依赖于网络，但性能要略差于网络通信模式，且对集群规模有一定限制。混合通信模式也称网络**/**磁盘主备通信模式，优先使用网络通信模式，若遇到网络故障则自动切换到磁盘通信模式。混合通信模式适用于普通网络环境，兼顾网络通信模式性能的同时又能够容忍网络波动，大幅提升高可用存储池的稳定性。

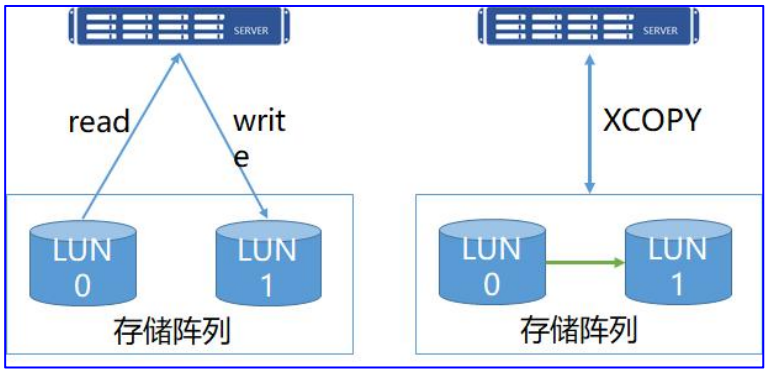
- 存储池加速

传统的存储池内或者存储池间的数据拷贝需要首先将数据拷贝到主机内存中，然后再写入目标存储池。这个过程需要占用 **CPU** 和内存资源，同时又要占用存储链路带宽，且数据拷贝速率受限于存储链路带宽。

SCSI SPC–3 提出了新的 **SCSI** 增强指令 **XCOPY(Extended COPY)**，可支持存储阵列内的数据拷贝卸载，数据可直接在存储阵列内进行数据拷贝，而无需经

由存储链路通过 CPU 进行数据中转。

InCloud OS 存储池硬件加速特性基于 **XCOPY SCSI** 指令，可实现云硬盘文件拷贝卸载，既提高了数据拷贝的速率，又能降低系统的整体开销。此外，对于不支持 **XCOPY SCSI** 指令的存储阵列，云硬盘文件拷贝可自动识别并转换为传统拷贝方式，以保持系统的兼容性。存储池加速示意图如下所示：



◆ 存储池扩容

存储池的使用量达到一定的阈值时，为了不影响系统的性能，需要向系统动态的增加存储池的容量，以保证系统的高可用性，即存储池扩容。存储池的扩容不影响原存储池的使用，新添加的磁盘空间可立即作为存储池可用空间直接使用。

为了保证存储池扩容的安全性，提供了维护模式方案，用户将存储池进入维护模式，中断业务，扩容完成后，退出维护模式自动恢复业务。

◆ 缓存存储池

缓存存储池是由本地存储池转换而来，且一个虚拟化资源池上最多存在一个缓存存储池。缓存存储池为虚拟化容灾的持续数据保护（**Continuous Data Protection**，简称 **CDP**）提供 **IO** 数据缓存的作用。虚拟化资源池集群通过植入后端存储过滤驱动，来实时捕获虚拟磁盘所有 **IO** 访问操作，后将数据暂存到缓存存储池上，用于数据的传输，因此要求缓存存储池的读写性能不低于生产存储池。

4.3.1.1.3 约束

无

4.3.1.2 云硬盘全生命周期管理

4.3.1.2.1 功能描述

云硬盘是一种高可靠、低成本、高可用、可定制化的存储设备，可作为云主机的独立可扩展硬盘使用。云硬盘是平台不可或缺的一部分，为云主机提供持久的存储资源，不仅可以用作系统盘存放镜像文件来启动云主机，还可以作为云主机的数据盘。

- 云硬盘创建

磁盘格式支持 **RAW** 和 **QCOW2**、磁盘策略支持精简制备和非精简制备、当磁盘格式 **QCOW2** 时，可设置簇大小为 **64K/128K/256K/512K/1M/2M**、共享属性和自定义大小。

- 云硬盘列表分页查询和模糊匹配

支持根据用户设置的每页展示数量，快速展示云硬盘信息。支持多方位展示云硬盘信息，包括：名称、状态、关联对象、大小、磁盘格式、**WWID** 编号、磁盘策略、共享、加锁、副本数、容灾、存储池、所属资源池、存储类型、虚拟数据中心、用户、可用区、描述、**ID**（云平台记录的唯一标识）、**UUID**（虚拟化资源池记录的唯一标识）、创建时间和操作，使得用户实时监控对云硬盘的使用情况。支持用户按照名称、存储类型、状态、虚拟数据中心、用户、容灾、所属资源池、存储池、大小、磁盘格式、磁盘策略、关联对象、共享、加锁、**ID** 和 **UUID** 等筛选条件快速查询云硬盘，并且支持名称的模糊查询，支持多种状态

批量筛选，满足典型的应用需求场景。

◆ 云硬盘详情全方位展示

摘要、任务和操作日志，其中摘要包含：名称、状态、大小、可用区、所属资源池、存储池、存储类型、虚拟数据中心、用户、存储路径、磁盘策略、磁盘格式、共享、加锁、副本数、容灾、描述、**ID**（在云管理平台上唯一标识）、**UUID**（在虚拟化资源池上唯一标识）、创建时间、修改时间和管理云主机信息。

◆ 云硬盘删除和强制删除

指的是不仅在数据库中删除云硬盘记录，同时会清理虚拟化资源池中虚拟磁盘，并且可选择地是否清理存储池中虚拟磁盘的数据。云硬盘强制删除属于高危操作，适用于云硬盘完全故障的场景。

◆ 云硬盘编辑

编辑云硬盘名称、是否共享和描述；要求云硬盘是可使用状态，或云硬盘是已挂载状态且云主机是关闭状态时。

◆ 云硬盘扩容

是用于调整云硬盘大小，支持离线扩容和在线扩容，在线扩容时要求云主机已展示 **vmtools** 工具。

◆ 云硬盘纳管

是一种云硬盘管理手段，支持通过云平台将虚拟化资源池中已存在的虚拟磁盘，通过纳管的方式统一到云平台内进行管理，常用于虚拟化资源池的利旧或云迁移场景。

◆ 云硬盘移除

是云硬盘纳管的反向操作，目的是仅在云管理平台删除云硬盘，虚拟化资源

池中的虚拟磁盘不会被删除，可用替换云管理平台场景。

◆ 云硬盘分配

是调整云硬盘所属虚拟数据中心和所属用户，常用于部门重新规划或调整的场景，实现无需通过迁移数据，实现资源的重新归类和调整。

◆ 云硬盘回收

是云硬盘分配的反向操作，系统管理员回收云硬盘操作后，云硬盘不属于任一虚拟数据中心，并且所属用户为当前操作用户；虚拟数据中心管理员回收云硬盘操作后，云硬盘所属用户为当前操作用户。

◆ 云硬盘移动

是指将一个云硬盘从一个存储池迁移至另一个存储池。用户可以根据业务负载情况更灵活的选择使用不同的存储池，极大提高了存储资源的充分利用。

◆ 云硬盘复制

是指将一个云硬盘从一个存储池复制至另一个存储池，实现数据的快速备份，支持在同一个存储池内复制，也支持跨存储间的复制。

◆ 云硬盘快照

是基于云硬盘某一时刻的数据进行“即时备份”，不直接复制全部数据，而是先记录当前硬盘的元数据，与原始硬盘共享相同的数据块，当原始硬盘的某块数据有写入操作时，先将该数据块的旧版本复制到快照的存储空间中，再对原始硬盘写入新数据，既能快速快照，又能高效利用存储空间。

◆ 云硬盘备份

是将云硬盘数据独立存储到与原硬盘隔离的备份存储池，进行存储，在需要的时候恢复原云硬盘或者创建新的云硬盘进行业务数据的恢复，保障数据安全，

备份过程中会首先生成一个临时在线快照，基于临时在线快照创建临时卷，并执行拷贝数据操作，备份动作完成后，删除临时卷与临时在线快照，实现云主机不关机的情况下对其某个单独的云硬盘进行在线备份的能力，产品支持云硬盘即时备份和周期备份两种方式进行云因盘备份。

◆ 云硬盘回收站管理，包括如下关键操作：

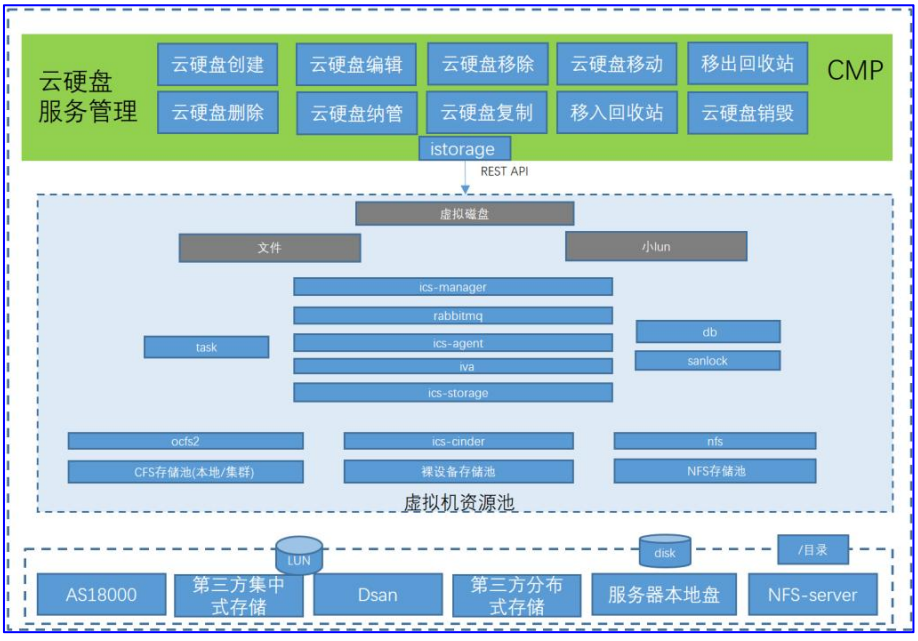
云硬盘移入回收站：云硬盘删除或强制删除时，可选择移入回收站，允许云硬盘在系统中存留一端时间，避免用户误操作导致数据丢失风险。

云硬盘从回收站恢复： 在系统设置的自动销毁时间前，可从回收站中恢复云硬盘，用于保护用户数据。

云硬盘销毁： 支持在回收站中自动销毁和手动销毁云硬盘，用于彻底删除云硬盘数据场景。

清空回收站： 支持一键清空回收站中云硬盘数据，方便用户操作。

4.3.1.2.2 技术原理



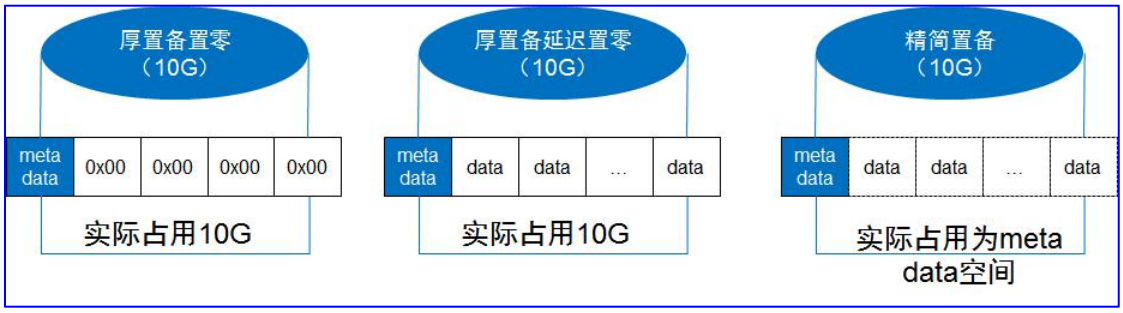
云硬盘管理架构图如上图所示，用户通过 **CMP** 控制台管理虚拟化资源池，

云硬盘服务管理是控制台发送请求到 **ics-manager** 服务，并将请求参数整合处理后发送到 **ics-agent**，然后转发到 **iva**，接着通过 **ics-storage** 实现不同存储池的接口统一，根据请求的参数分别调用 **ocfs**、**ics-cinder** 和 **NFS** 的虚拟磁盘接口，最终完成存储资源管理操作。云硬盘服务管理使用的关键技术有：

- 厚制备和精简制备

区别于传统的厚置备置零，**InCloud OS** 云硬盘管理提供的厚置备延迟置零类型采取预分配方式，在创建时为云硬盘分配所需空间但不会擦除物理设备上的数据，仅当有 **I/O** 操作时才置零，然后完成 **I/O**，既可以预分配空间，又提高了空间分配效率。

精简置备并未在云硬盘创建的时候为其分配所要求的全部空间，只在磁盘文件的元数据区的文件大小字段写入了虚拟的文件大小，而元数据区实际记录磁盘块索引的字段为空，故并未实际分配对应的磁盘块。在云主机向磁盘中写入数据的时候，才按需、动态的在物理存储空间中分配对应的存储空间。三种分配方式的对比如下图所示：

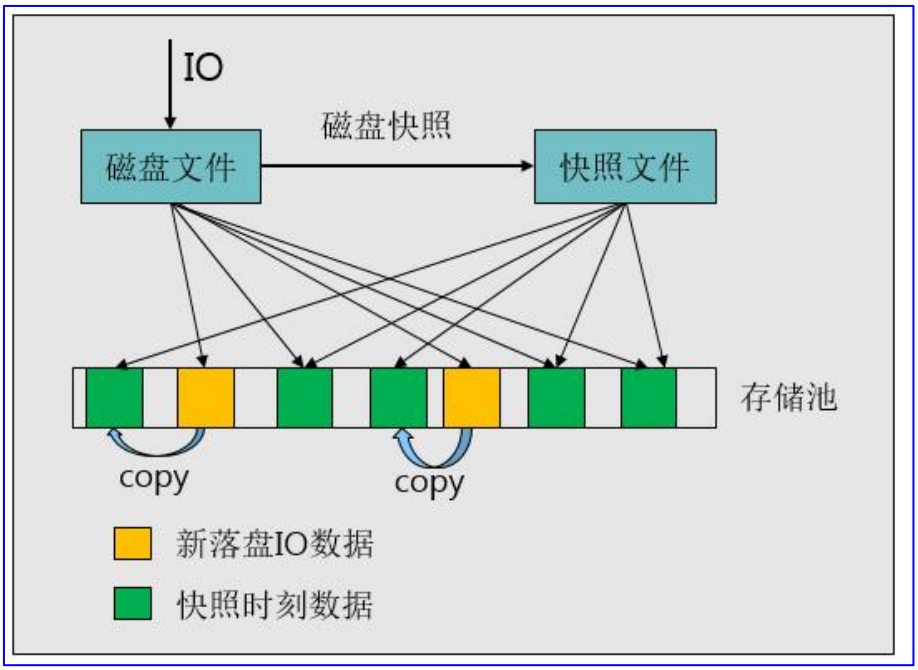


- 磁盘簇大小
- 云硬盘文件是由多个固定大小的单元组织构成，这些单元被称为 **cluster**（簇），我们这里可以根据需要，在云硬盘时，自定义 **cluster size**（簇大小），取值范围为：**64K**、**128K**、**256K**、**512K**、**1M**、**2M** 。**cluster**

size 的大小决定了磁盘 **L1**、**L2 cache** 检索的效率，**cluster size** 越大的话，检索效率越高，磁盘性能越好。目前可配置 **qcow2** 格式的虚拟磁盘。

- 磁盘快照

InCloud OS 云主机快照是通过磁盘快照实现，磁盘快照采用写入时复制技术**(Copy on Write)**，当某个 **block** 将被改写的时候，该 **block** 首先被 **copy** 到其他地方（即拷贝到快照系统指定的某个位置），然后再在它原来的位置进行改写。这样能保证每个快照文件数据的完整性，各个快照之间是相互独立的，不会因为某一个快照损坏导致无法恢复，可以实现快照快速创建、恢复、删除等功能，便于云主机快照的管理。



- 回收站管理

用户可以选择将待删除的云硬盘放入回收站，以备用户在必要时恢复云硬盘中的数据。可以设置云硬盘在回收站中的保留时长，超过该时长，则系统会彻底销毁云硬盘。

进入回收站的云硬盘依然保留分配给配额和存储数据，只有云硬盘被销毁后，其占用的配额和数据才会被释放。

4.3.1.2.3 约束

- 为了数据的一致性，使用共享云硬盘要求集群文件系统。
- 为了考虑数据的一致性，不单独实现单个云硬盘快照，可通过云主机快照实现云硬盘中数据的保护。
- 云硬盘强制删除属于高危操作，强制删除不一定能成功删除，例如，针对正常使用的云硬盘，要求从云主机中移除才能正常删除。

4.3.2 文件存储

InCloud OS 文件存储使用软件定义层 **manila** 组件抽象和封装统一的文件存储管理接口，并通过不同的 **driver** 对接多种类型的文件存储集群进行管理，提供文件存储服务，可支持 **NFS** 和 **CIFS** 协议，创建的文件存储，可以在云主机或裸机内通过挂载文件存储来使用网络存储空间，从而达到扩容的目的。多个云主机或裸机挂载同一个文件存储即可实现文件共享，并同时支持文件存储快照、**ACL** 控制，打造稳定可靠的文件存储服务，为共享海量数据场景提供有力支持。

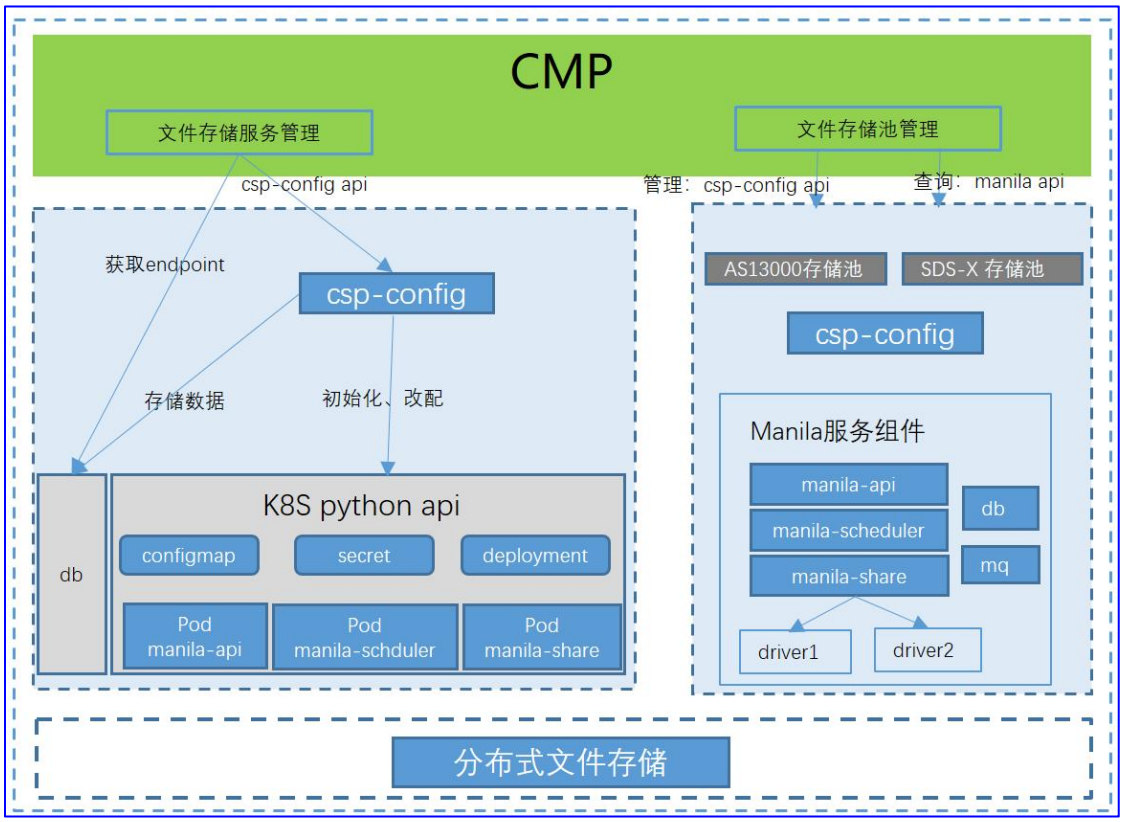
文件存储管理通过云平台调用 **Manila** 提供的 **RestfulAPI**，实现在分布式文件存储上创建和管理文件存储，提供给云主机或裸机或容器使用，并支持在界面上实现云主机可视化挂载和卸载文件存储操作，方便用户的操作。

4.3.2.1 文件存储池管理

4.3.2.1.1 功能描述

为了满足部署的灵活性和快速性等要求，支持在部署 **CSP–config** 服务时顺带部署软件定义层 **manila** 组件并自动拉起服务。并支持在 **CMP** 界面上进行文件存储池生命周期管理，包括增加存储池、编辑存储池、删除存储池和设置存储池标签管理。

4.3.2.1.2 技术原理



整体上抽象出软件定义层 **csp–config** 部署组件用于管理 **manila** 组件，实现文件存储服务和文件存储池的界面化管理，方便用户运维，核心流程如下：

- 云平台管理层 **CMP** 通过 **UI** 调用 **csp–config api** 接口，间接调用 **K8S python api** 请求，完成 **manila** 三个组件 **manila–api**、**manila–scheduler**

和 manila-share 的初始化部署，其中 manila-share 经过改造后，内置一个 fake drver，不依赖后端存储集群；并保存 manila-api 的 endpoint 到数据中，用于 CMP 获取并调用。

- 云平台管理层 CMP 通过 UI 调用 csp-config api 接口，间接调用 K8S python api 请求，完成文件存储池添加和删除等管理操作，并内置后端存储集群连通性、用户名和密码正确性的校验机制，和支持获取负载均衡、存储池/目录等特殊配置字段实现 CMP 界面自动填充，并支持在 CMP 界面创建目录和设置配置的操作，极大地降低客户运维的复杂度。

4.3.2.1.3 约束

无

4.3.2.2 文件存储全生命周期管理

4.3.2.2.1 功能描述

平台通过软件定义层 Manila 提供文件存储服务，可支持 NFS 和 CIFS 协议，通过 Manila 创建的文件存储，可以在云主机内通过挂载文件存储来使用网络存储空间，从而达到扩容的目的。多个云主机挂载同一个文件存储即可实现文件共享，并同时支持快照、ACL 控制。

- 文件存储实例创建

文件存储平台可以对接多个提供文件存储服务的后端存储集群，首次使用某一存储池时，会自动创建一个文件存储类型，并指定后端名称来对应后端存储集群，用户可以根据不同的需求来选择不同存储池，用于创建文件存储，无需特殊关注文件存储类型。

支持来源新建与文件存储快照创建文件存储并支持 **NFS** 与 **CIFS** 协议，文件存储实例命名会根据用户的输入自动命名，并支持中文字符。

◆ 文件存储实例删除

支持删除文件存储实例，当用户不在需要某个文件存储实例时，可以删除该文件存储实例，删除文件存储也会同时删除该文件存储内已经存放的用户数据，需要二次确认。

◆ 文件存储实例编辑

支持对已创建的文件存储进行编辑，支持修改文件存储的名称、大小和描述，方便用户对不同的文件存储进行个性化管理，便于用户识别、归类文件存储列表。

随着系统的持续使用、业务的增加，会造成某些事先规划的资源大小不满足具体应用，支持对文件存储编辑大小进行扩容操作。扩容操作能够提升单个文件存储实例的容量上限，以适应业务的需求。

◆ 文件存储实例列举

平台可以展示当前用户已经创建出的所有文件存储实例，并同时展示各个文件存储实例相应的名称、状态、容量、已使用量、所属存储池、所属存储类型、所属存储集群、协议、所属虚拟数据中心、所属用户、描述、创建时间和操作等信息。用户可以通过以上信息来管理各个文件存储实例。支持用户按照名称、状态、所属虚拟数据中心、所属用户、大小、所属存储池、协议等筛选条件快速查询文件存储实例，并且支持名称的模糊查询，满足典型的应用需求场景。

◆ 文件存储实例详情展示

支持查看文件存储实例详情，包括摘要(基本信息和路径)、快照、任务和操作日志，其中，摘要中的基本信息包含：名称、状态、所属存储池、所属虚拟数

据中心、所属用户、描述、大小、所属存储类型、所属存储集群、协议、创建时间、来源、底层标识等信息；摘要重的路径指的是所有存储集群 **export** 出的路径，包含域名和负载均衡地址。快照是基于该文件存储实例创建的文件存储快照列表信息。任务是记录关键的异步任务的进度和最终状态，方便用户实时关注资源管理动态。操作日志是记录关于该文件存储实例的所有操作记录，用于追溯和审计。

◆ 文件存储实例快照管理

文件存储被用于挂载给云主机或者其他宿主机，用户在使用过程中，会在文件存储内存放用户数据，为保障数据安全性提供文件存储快照功能，为当前时间点文件存储内的数据做一个副本或者静态影像，以防重要数据丢失后需要进行数据恢复。

快照名称支持中文以及添加中文描述信息。平台支持通过文件存储快照创建一个新文件存储实例。创建完成后，新的文件存储实例就包含了原文件存储实例打快照时的所有数据，实现用户数据得到及时恢复。

◆ 文件存储用户管理

支持文件存储用户的配置管理，用于访问 **CIFS** 协议类型的文件存储时进行身份认证。支持文件存储用户的创建和删除、用户密码重置。

◆ 文件存储实例访问权限管理

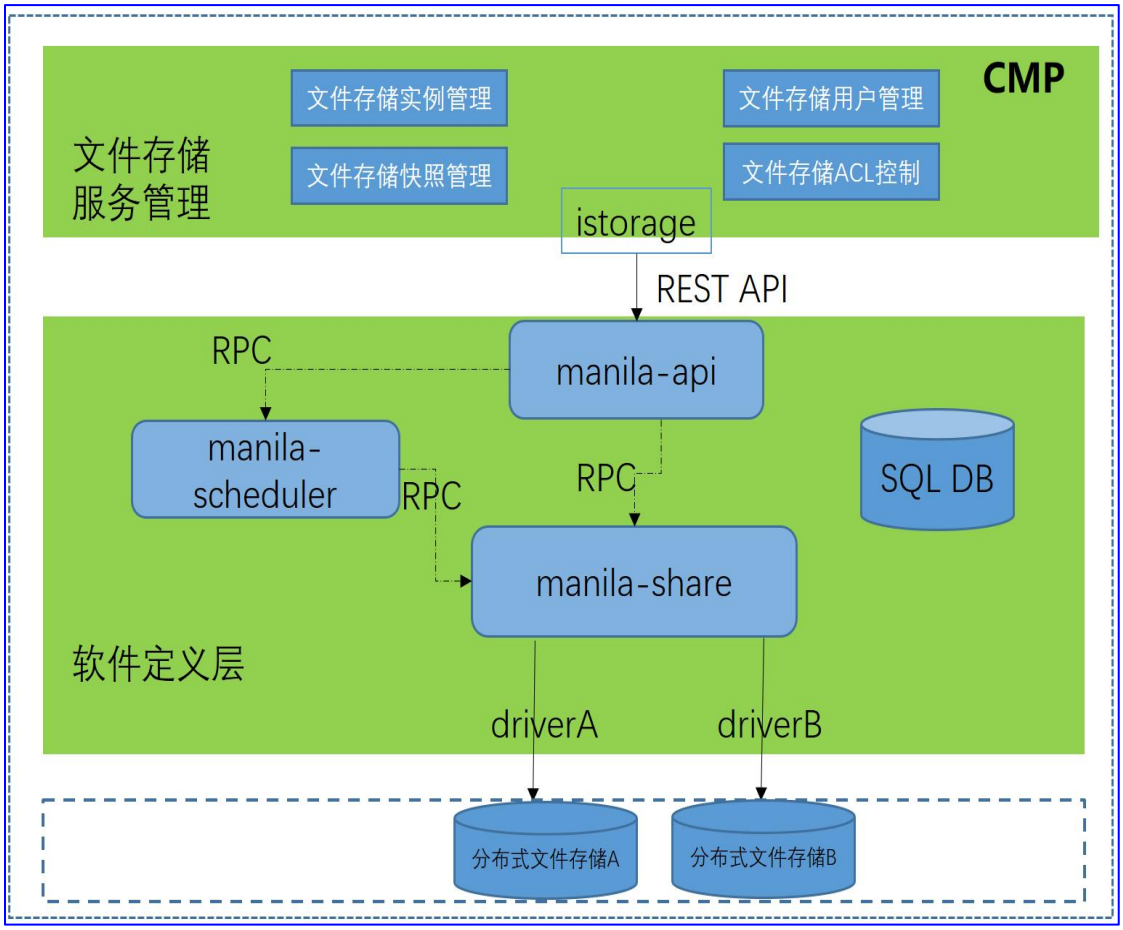
对于 **NFS** 协议的文件存储，通过访问权限管理可以设定指定 **IP** 的客户端具有何种权限来访问该 **NFS** 文件存储内的数据，其中权限类型包括：只读、读写。对于 **NFS** 协议的文件存储，对任何 **IP** 默认是无权访问。只有经过给 **IP** 授权以后，才能使用该 **IP** 登录 **NFS** 文件存储。

对于 **CIFS** 协议的文件存储，通过访问权限管理可以设定指定用户具有何种权限来访问该 **CIFS** 文件存储内的数据，其中权限类型包括：只读、读写。对于 **CIFS** 协议的文件存储，对任何用户默认是无权访问。只有经过给用户授权以后，才能使用该用户登录 **CIFS** 文件存储。

◆ 文件存储实例分配和回收

支持更改文件存储实例的所属虚拟数据中心和所属用户，方便文件存储实例跨虚拟数据中心和用户进行管理。

4.3.2.2.2 技术原理



文件存储架构图如上图所示，整体上是基于软件定义层的 **manila** 服务暴露 **Rest api** 接口，负责文件存储实例、文件存储快照和文件存储用户/密码功能的管

理，供云管理平台 CMP 调用和管理。具体地，用户在 CMP UI 界面的操作请求，会进行解析并转化为相关 Rest api 请求发送给 manila api 组件，manila-api 层将 Rest api 请求入参整合处理后发给 manila scheduler 层，manila scheduler 层会根据传入的文件存储类型调度到绑定的文件存储后端集群上，最后，manila share 层将请求发给相应的后端存储驱动 storage diver，驱动会调用后端存储系统完成相应的请求。

4.3.2.2.3 约束

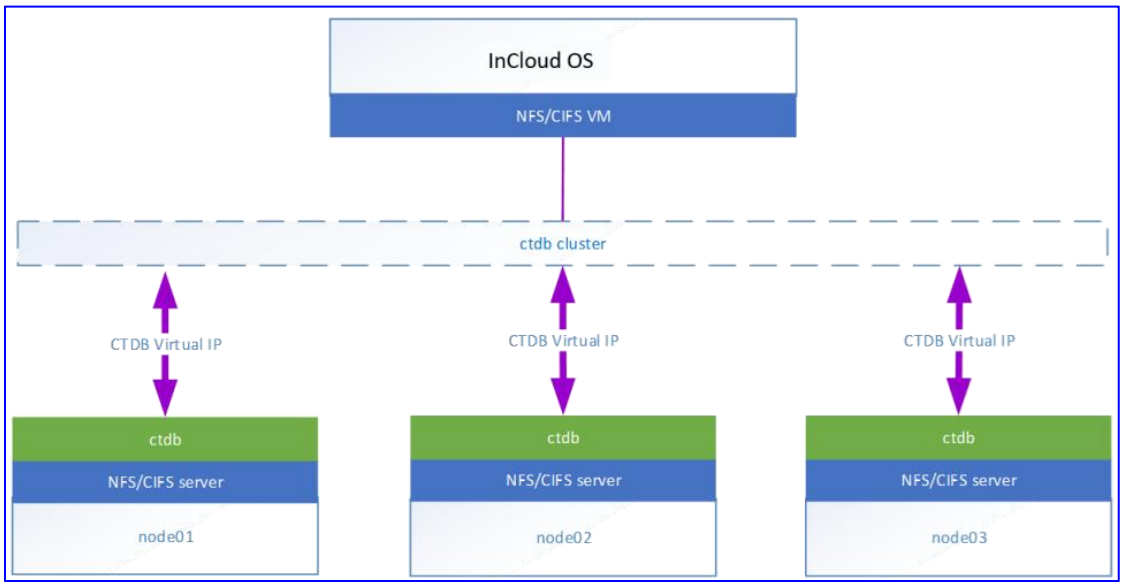
- 为了获得较好的兼容性，NFS 协议的文件存储实例建议在 Linux 系统下挂载，CIFS 协议的文件存储实例建议在 Window 系统下挂载。
- 删除文件存储实例之前，应确保该文件存储实例没有被挂载到客户端，负责会导致文件存储路径在云主机内残留。

4.3.2.3 文件存储高可用

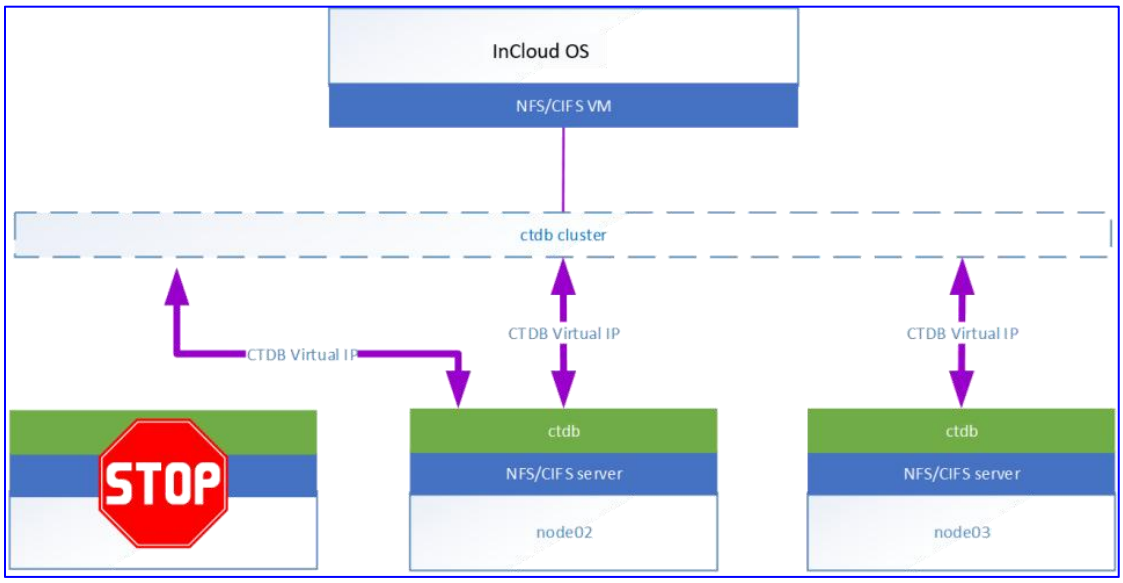
4.3.2.3.1 功能描述

云主机或裸金属中业务支持通过高可用集群方式访问文件存储路径，利用高可用集群，可实现访问文件存储时的高可用、负载均衡特性、及文件存储某个节点服务宕机后的拉起、恢复功能。

- 技术原理



CTDB 集群正常情况



云主机和裸金属中业务访问文件存储实例路径，是 **ctdb** 服务所提供的虚拟 **IP**，**ctdb** 内部转发请求至对应的文件存储服务节点，当某个节点故障、或者文件存储服务故障，则由 **ctdb** 集群功能将请求转发至另外正常节点。同时 **ctdb** 会尝试拉起故障节点的服务、负责恢复后的 **ctdb** 集群重新均衡操作。

由于 **ctdb** 服务同样也提供负载均衡功能，当的访问量较多时，由 **ctdb** 将请求均衡到各个节点，提升访问的性能。

除了对接使用 **CTDB** 集群提供的虚拟 **IP**，平台也能够对接使用 **NAS** 存储集

群的 **pdns** 服务提供的域名，只需要在客户端配置对应的 **DNS**，即可通过含有域名的 **share** 导出路径进行挂载。

4.3.2.3.2 约束

使用文件存储路径时，建议使用域名方式，并需要在**/etc/resolv.conf** 中配置 **nameserver ip**(文件存储负载均衡地址)。

4.3.3 对象存储

InCloud OS 对象存储服务是一种可容纳海量数据且支持 **HTTP/HTTPS** 协议访问的分布式存储服务，支持基于 **RESTful API** 存储和管理任意类型的文件，适合各类企业、网站使用。**InCloud OS** 对象存储服务支持租户专属对象存储、共享对象存储，具有 **UI** 可视化的账号管理、应用接入密钥展示、存储桶管理、对象管理等功能模块。

对象存储是通过云平台调用管软的 **RestfulAPI** 和 **S3** 接口，在分布式对象存储上管理账号、存储桶和对象资源；同时支持云主机或裸机或容器上承载的业务调用 **S3** 接口进行数据的上传、下载和备份等操作。

4.3.3.1 功能描述

对象存储适宜用来组织非结构化数据，不仅仅是“云存储”数据，还包括备份归档数据、海量图片数据、医疗影像数据、日志数据、视频监控数据和超算分析数据等。适合普通用户、网站、企业和开发者用户，平台 **UI** 控制台提供了丰富的功能，如下文所述。

- 增加对象存储站点

对象存储站点整体上是通过直接接入物理对象存储的方式提供对象存储服

务,可单独配置 **S3** 网关地址和 **S3** 网关端口用于用户客户端直接上传和下载对象存储数据，支持更新物理存储凭证信息。

◆ 对象存储站点列表展示

对象存储站点列表支持展示对象存储站点的名称、状态、类型、版本、管理 **IP**、管理端口、总容量、可用容量、存储使用率、存储超分比、创建时间和操作等属性。

◆ 对象存储站点详情展示

对象存储站点详情支持展示站点的摘要信息、对象存储账号、对象策略和操作日志信息，其中摘要包含基本信息（名称、状态、类型、版本、管理 **IP**、管理端口、创建时间）、资源统计（总容量、已使用量、账号个数、桶个数、对象个数）、该站点下存储桶相关信息，并增加存储桶管理入口；对象存储账号包含账号列表和账号操作入口；对象策略包含策略列表和策略操作入口。

◆ 移除对象存储站点

对象存储平台支持用户删除一个空物理存储，出于安全性考虑，只有空物理存储才允许被删除。

◆ 编辑对象存储站点

编辑对象存储站点支持编辑名称、**S3** 网关地址、**S3** 网关端口和访问凭证信息，访问凭证是用于当底层物理存储集群管理密码被修改后，需要在对象存储平台同步更新对象存储访问凭证场景。

◆ 对象存储站点标签管理

支持为对象存储站点分配标签、回收标签，针对云平台对接多个对象存储站点场景，方便用户进行筛选等管理。

◆ 增加对象存储账号

创建的新存储账号时，需要指定类型：非共享或共享（非共享仅在 **VDC** 内的指定用户下使用该账号，共享可再多个 **VDC** 下的用户公用该账号），设置配额：最大桶个数、最大对象个数、最大容量。创建成功后生成每个账号对应的 **AK** 和 **SK**。用户可以使用 **AK** 和 **SK** 通过对象存储网关直接进行文件操作。

◆ 纳管对象存储账号

支持纳管已接入对象存储站点的底层对象存储集群上已有的账号和账号的 **ACL** 权限、存储桶及 **ACL** 权限、对象及对象 **ACL**、桶策略等资源，需要选择对象存储站点、底层对象存储集群上已有的账号、数据中心和用户信息，实现对象存储信息的统一管理，方便利旧项目上已有的对象存储集群管理。

◆ 对象存储账号列表展示

支持展示存储账号的名称、状态、存储类型、站点名称、类型、所属虚拟数据中心、容量配额、已用容量、存储桶数量、桶已用数量、对象存储配额、对象存储已用数量、用户、创建时间和操作，并支持按照所属虚拟数据中心、类型、名称、状态、存储类型和站点名称进行筛选。

◆ 删除对象存储账号

支持用户删除对象存储账号，并经过二次确认后，同时删除对象存储账号下存储桶内的对象资源。

◆ 修改对象存储账号配额

云平台支持为每个对象存储账号设置配额，配额信息包括：容量配额，支持 **MiB**、**GiB**、**TiB** 和 **PIB**（**1–1024**）粒度；对象存储配额，支持无限制类型和定值类型（**1–50000000**）；桶数量配额，支持支持无限制类型和定值类型（**1–99999**）。

◆ 对象存储账号详情展示

对象存储账号详情中支持展示账号基本信息（名称、类型、虚拟数据中心、公钥/私钥、**S3** 网关地址、存储类型、站点名称、状态、容量配额、桶数量、对象数量配额、所有者、所属存储租户、访问权限、创建时间）、该账号下存储桶列表及存储桶操作入口。

支持展示通过 **SDK** 完成如创建存储桶、删除存储桶等操作的代码示例，包含 **JAVA**、**PYTHON**、**C++**语言。

◆ 对象策略管理

支持对象存储策略增加、编辑和删除，在创建存储桶时选择对象存储策略，能将存储桶创建于对象物理存储的不同存储池内，提升数据的故障隔离性。

◆ 增加存储桶

存储桶是对象存储中对象的容器，存储桶名在同一对象存储账号下全局唯一，且不支持修改桶名，支持选择对象存储账号、自定义名称、设置配额（容量、对象数量）、选择对象策略。

◆ 删除存储桶

支持用户删除存储桶，删除存储桶时，桶内资源会一并被删除。

◆ 编辑桶配额

云平台支持为每个存储桶设置配额，配额信息包括：桶内最大对象个数、桶最大容量。即用户在桶内上传对象时，要遵循桶配额的限制，当超出配额限制后，用户上传对象或者创建目录操作会失败并提示超出配额限制。

桶配额的容量和最大容纳对象个数只要有任何一个触发配额限制，就会导致对象上传失败。

◆ 对象多版本支持

云平台存储桶支持多版本,开启多版本后,桶中的对象都以多版本形式存储,版本数量无限制。

◆ 存储桶详情

存储桶详情中支持展示桶基本信息(名称、状态、对象策略、访问权限、版本控制、创建时间、总容量/已使用容量、对象数量)、该存储桶下对象列表、共享列表和上传列表。

支持展示通过 **AWS SDK** 完成如上传文件、删除文件、下载文件等操作的代码示例,包含 **JAVA**、**PYTHON**、**C++**语言。

◆ 桶 **ACL** 管理

支持为存储桶设置 **ACL** 权限控制,包括私有、公共读、公共读写。其中私有是赋予存储桶的所有者完全控制权限;公共读是赋予所有用户可以匿名查看存储桶中的文件;公共读写是赋予所有用户在存储桶内读取和写入对象。

◆ 存储桶对象生命周期规则

支持为桶内的对象设置生命周期规则,当桶内的对象到期之后,自动删除桶内的对象。生命周期规则可按照整个桶的对象或者桶内某个前缀的对象进行设置,可以设置当前版本的到期时间,也可以设置历史版本的到期时间。

◆ 桶策略管理

通过桶策略实现对存储桶和对象更精细化的权限控制管理,支持增加策略、编辑策略和删除策略。并且增加策略支持可视化视图和 **JSON** 视图,可视化视图能够覆盖绝大多数权限控制场景,**JSON** 视图适用于适用对象存储的开发人员,属于更高级的权限控制。

◆ 增加目录

支持在桶内创建文件夹，即目录。通过目录来归类用户数据，方便用户对数据的管理、归纳。

◆ 上传对象

支持用户直接在 **UI** 控制台上传对象，并且在上传对象前，允许重命名对象名称，单次只能上传一个对象。当然用户可以开启多个控制台窗口进行多个对象并行上传。

◆ 删除对象

云平台支持删除对象操作，支持单个删除和批量删除，以及目录递归删除，安全起见，在删除之前会弹窗提示，用户二次确认后可才会继续执行删除任务。

◆ 对象 **ACL** 管理

支持对象 **ACL** 控制管理，置为为对象设置继承桶、私有、公共读、公共读写权限。继承桶权限是赋予对象和存储桶的 **ACL** 一样的权限控制；私有是赋予对象的所有者完全的控制权限；公共读赋予所有用户读权限，不能够进行写入；公共读写时赋予所有用户读和写权限。

◆ 共享对象

云平台支持特定的对象生成一个临时 **URL**，该 **URL** 仅在指定的时间段有效，管理员可在有效期内将 **URL** 进行分享，该 **URL** 可以直接下载该对象文件。

◆ 下载对象

云平台支持手动下载对象，保存到本地服务器目录下。

◆ 搜索对象

云平台支持通过指定文件名前缀快速搜索并定位目标文件。

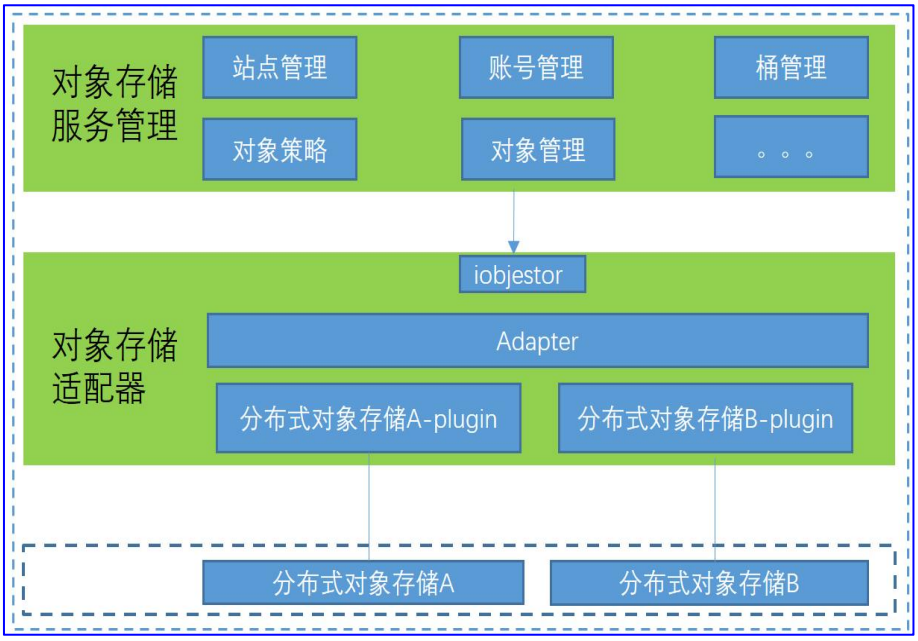
◆ 资源任务管理

支持展示异步操作任务的状态和进度展示，方便用户实时关注资源管理动态，并支持按照操作类型、状态、创建时间进行筛选。

◆ 资源操作日志

记录资源从创建到删除的整个生命周期内所有成功操作的详细记录，用于追溯和审计。

4.3.3.2 技术原理



InCloud OS 界面实现对象存储集群的管理、对象存储账号管理、存储桶管理和对象管理等功能；在适配器 **Driver** 层基于 **RestApi** 接口以及 **S3** 接口，实现和底层对象存储集群交互。其中，对象存储账号、存储桶以及对象的创建、删除等管理操作，通过 **RestApi** 接口直接访问物理存储，要求云平台和对象存储集群的管理网联通；对象的上传、下载等操作通过 **S3** 接口访问物理存储，要求云平台和对象存储集群的数据网联通，若不具备条件，支持使用管理网传输。租户的应用可以从云平台对象存储账号详情页面处获取存储桶的接入密钥，并通过集成

对象存储通用 SDK 实现对象的上传、下载等功能。

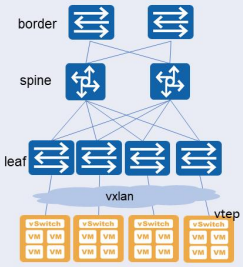
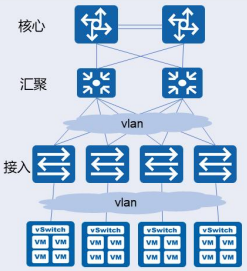
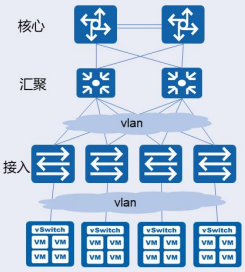
4.3.3.3 约束

- 通过界面上传文件仅支持大小 4GiB 以内，若更大的文件需要通过 S3 客户端进行上传。

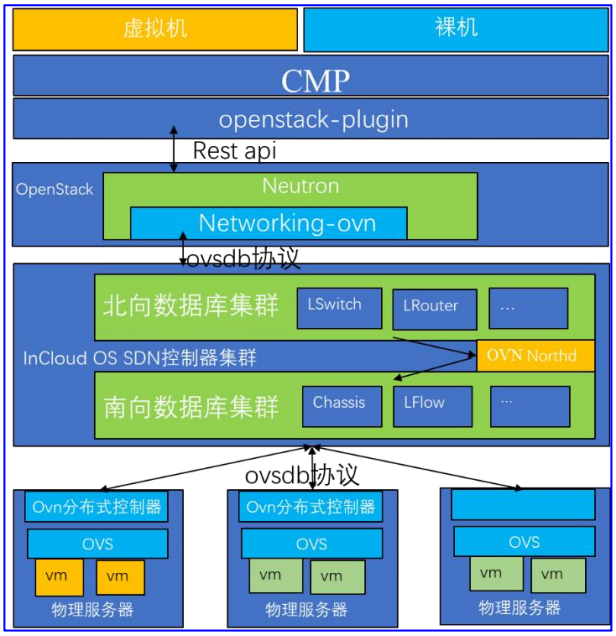
4.4 网络服务

4.4.1 网络技术方案概述

InCloudOS 支持 2 种网络技术路线、3 种网络模型，其中技术路线包括基于软 SDN 的 VPC 网络与经典网络，其中 VPC 网络又分为互联模式 VPC 与精简模式 VPC，2 种 VPC 模型与经典网络构成 3 种网络模型，三者区别如下：

	软SDN-互联模式VPC	软SDN-精简模式VPC	经典网络
描述	Overlay网络模型，基于软件实现隧道网络端点	VLAN网络，云平台提供二层及安全组、LB、DNS等能力，三层、防火墙依赖硬件配置	传统VLAN网络，仅提供二层网络能力
拓扑示意			
与硬件网络设备耦合	解耦	解耦	解耦
功能丰富度	高	中（提供二层网络及安全组、LB等能力）	低（仅提供二层网络能力）
性能	线速	线速	线速
自动化程度	高	中	低

4.4.1.1 软 SDN 技术架构



- 控制面统一使用 **ovsdb** 作为 **RPC** 通信协议
- 使用 **networking–ovn** 插件与北向数据库交互，完成网络信息配置
- 通过 **neutron** 完成虚拟网卡的生命周期管理
- 基于 **OVN+OVS** 实现虚拟机、裸金属数据互联互通

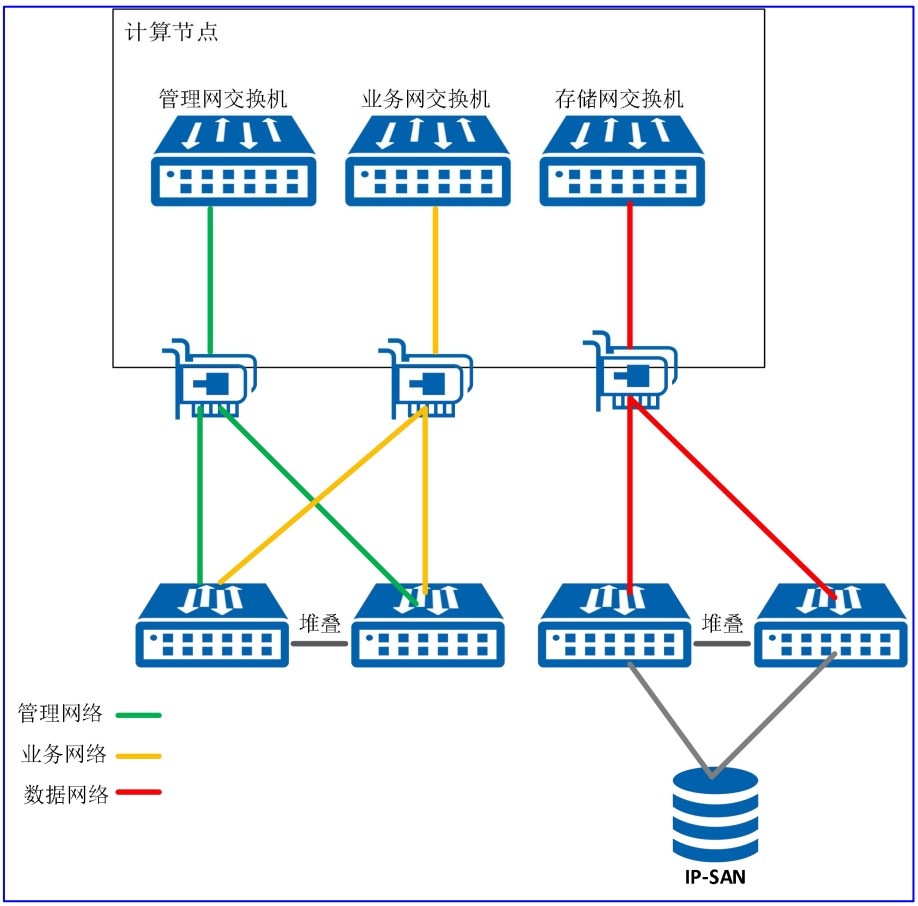
4.4.1.2 经典网络技术原理

经典网络服务通过虚拟交换机 **MAC** 转发表的学习来进行流量转发，与传统物理交换机转发原理类似，可以为虚拟机划分不同的 **VLAN**，不同 **VLAN** 之间通信需要依赖外部物理网络进行三层转发。

经典网络采用 **VLAN** 部署方式，用户需要提前规划好业务网络相关 **vlan** 信息，并在物理交换机放行对应的 **VLAN ID**。通常情况下接入交换机的端口可以配置为 **access** 或 **trunk** 两种模式，当配置为 **access** 时，**vlan** 标签由物理交换机负责处理；当配置为 **trunk** 时，注意同时要放行对应的 **vlan**，**vlan** 标签由虚拟交

换机负责处理。

经典网络使用 **vlan** 传统网络，组网模型需要考虑管理、存储、业务三网的隔离、互不干扰，同时外部交换机需要做堆叠或 **M–LAG** 进行高可用配置。



4.4.1.3 功能对比

功能	说明	经典网络	精简模式 VPC	互联模式 VPC
使用范围	是否租户共享	全局共享	租户间隔离，可针对子网设置共享	租户间隔离，可针对子网设置共享
跨资源池	是否跨多个虚拟化资源池	×	√	√

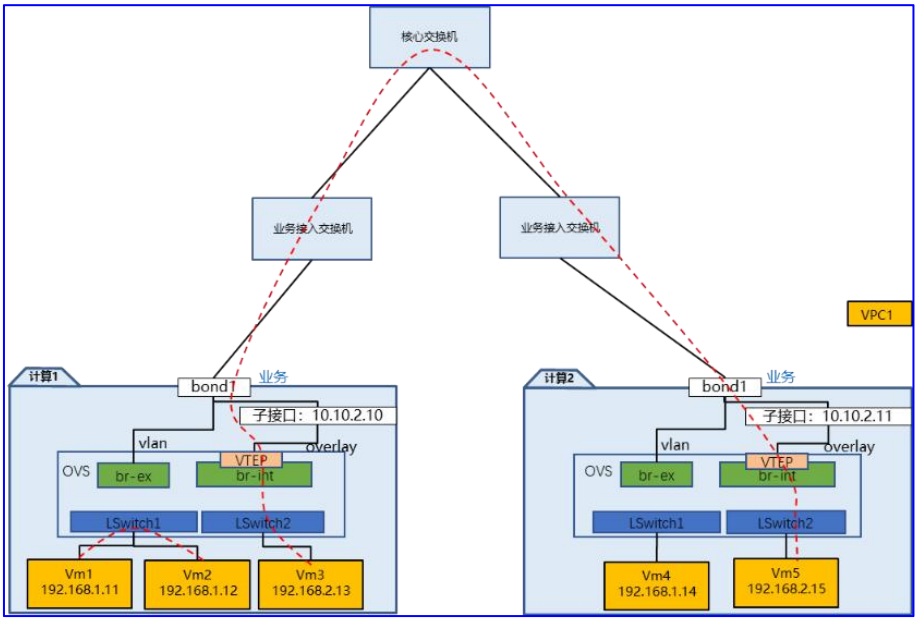
二层隔离	不同业务网络间二层隔离	√	√	√
虚拟路由器	三层路由器，分布式部署在所有计算节点，用于子网间互联	×	×	√
NAT 网关	网络地址转换服务，用于 **VPC** 南北向通信	×	×	√
对等连接	实现 **VPC** 之间的网络连接	×	×	√
安全组	防护虚拟网卡进出流量安全	×	√	√
防火墙	保护 **VPC** 内部子网之间流量及进出 **VPC** 流量安全，作用于虚拟路由器上	×	×	√
浮动 **IP**	通过绑定浮动 **IP**，实现内外网互通	×	×	√
负载均衡	访问流量根据转发策略分发到后端资源池	负载均衡服务本身依赖	支持	支持

		SDN，经典网络 IP 可以添加为后端服务的非 VPC 成员		
DNS	域名系统	部署 SDN 后，可以将经典网络 IP 添加到 DNS 系统	√	√
IPSec VPN	IPSec Vpn 服务	×	×	√

4.4.1.4 流量模型

4.4.1.4.1 互联模式 VPC

❖ **L2 互通**



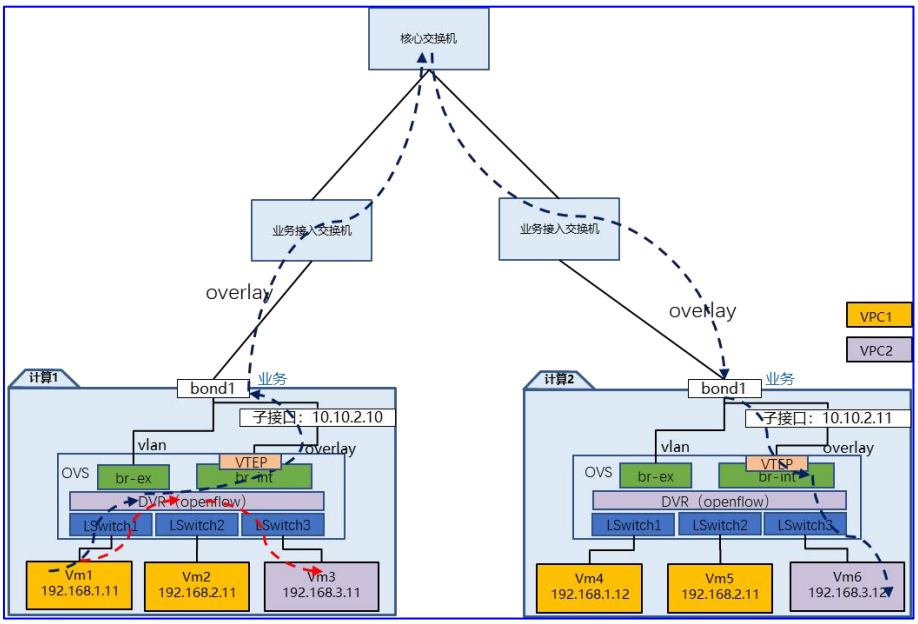
场景 1：同节点 L2 互通（VM1–VM2）

- VM1 出口流量 ovs 流表转换后直接转发至 VM2，不需要出主机

场景 2：跨节点 L2 互通（VM3–VM5）

- VM3 出口流量转发至计算节点 1 的 br–int，封装后经 bond1 网卡进入 underlay 网络
- 经交换机转发后，进入计算节点 2 的 bond1 网卡，经 br–int 解封装转换后转发至 VM5

❖ 东西向 L3 互通



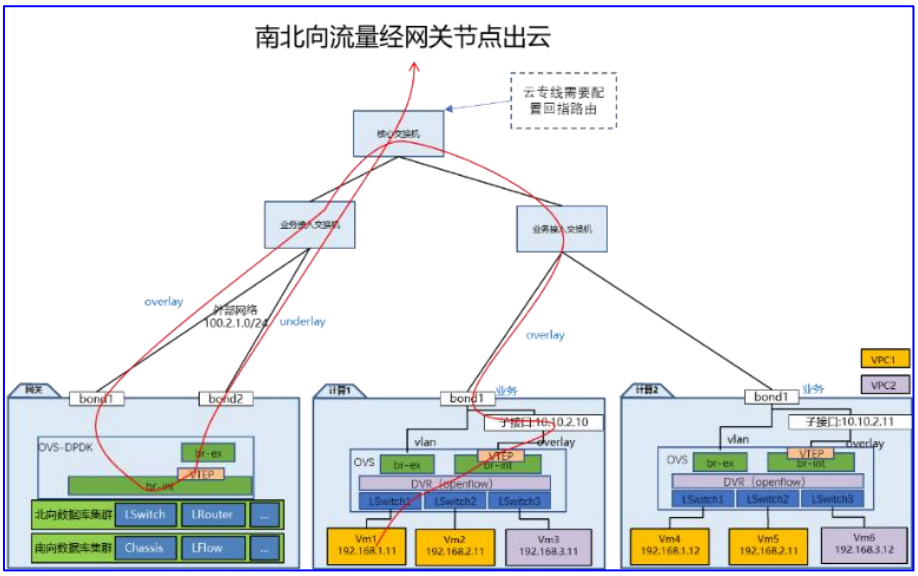
场景 3：同节点、跨网段互通（VM1 访问 VM3）

- VM1 出口流量经 DVR 流表转换后直接转发至 VM3，不需要出主机

场景 4：跨节点、跨网段访问（VM1 访问 VM6）

- VM1 出口流量经 DVR 流表转换后转发至计算节点 1 的 br–int，封装后经 bond1 网卡进入 underlay 网络
- 经交换机转发后，进入计算节点 2 的 bond1 网卡，经 br–int 解封装、DVR 流表转换后转发至 VM6

❖ 南北向——经网络节点出云



场景 **5**：**VM1** 经 **NAT** 网关访问外部网络

- **VM1** 流量经计算节点 **br–int** 封装后转为 **overlay**，并转发到网络节点
- 经网络节点 **NAT** 网关(**vRouter**)转换为外部 **underlay** 网络 **IP**(**vRouter** 网关）
- 经 **bond2** 网卡发送到外部交换机

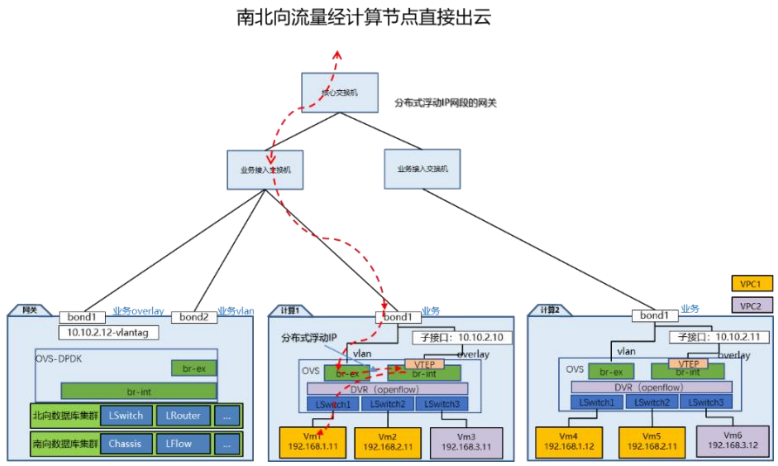
场景 **6**：**VM1** 经集中式浮动 **IP** 访问外部网络

- **VM1** 流量经计算节点 **br–int** 封装后转为 **overlay**，并转发到网络节点
- 经网络节点 **ovs** 流表转换为对应的浮动 **IP**
- 经 **bond2** 网卡发送到外部交换机

场景 **7**：**VM1** 经云专线（三层转发）访问外部网络

- **VM1** 流量经计算节点 **br–int** 封装后转为 **overlay**，并转发到网络节点
- 经网络节点 **ovs** 流表实现三层转发，转换为对应的外部网络 **IP**
- 经 **bond2** 网卡发送到外部交换机

❖ 南北向——经计算节点出云

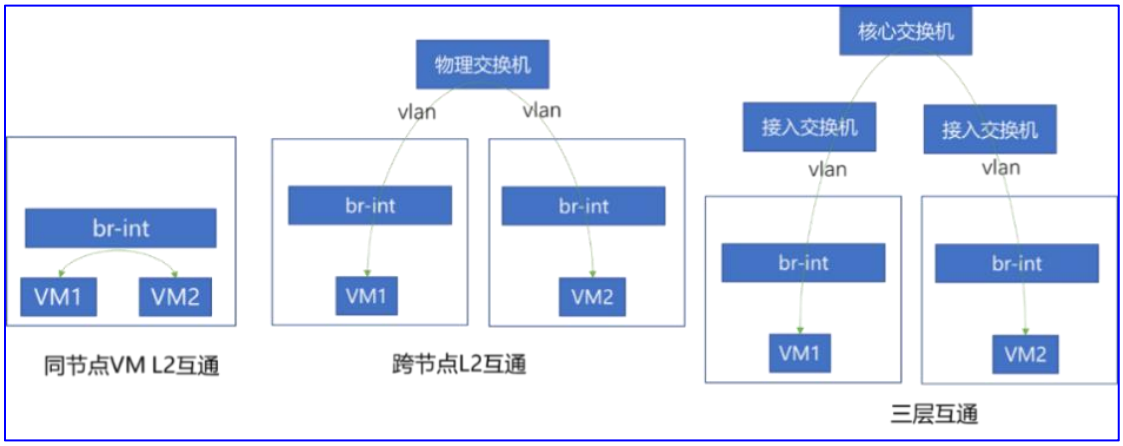


场景 8：VM1 经分布式浮动 IP 访问外部流量

- 前置条件：需要创建外部 VLAN 网络来承载分布式浮动 IP，并在核心交换机配置外部网络的网关， 且该外部 VLAN 网络能下到计算节点上连业务交换机；计算节点 bond1 网卡默认为 trunk 口，且配置子接口做 vtep IP
- VM1 经本节点 DVR 流表，转换为对应的浮动 IP(外部网段)
- 直接由计算节点的业务网卡出云

4.4.1.4.2 精简模式 VPC

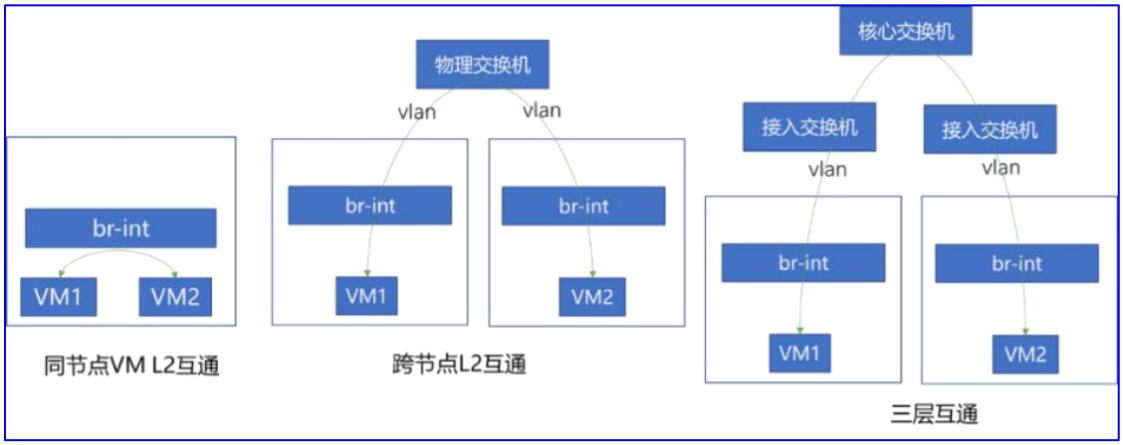
- 精简模式 VPC 中，云平台只提供二层网络功能，包括子网、安全组、QoS 等，三层网络（vRouter、NAT、浮动 IP）及防火墙等功能不可用，需要在硬件交换机上进行配置。



- 同节点 **L2** 互通：经计算节点内虚拟交换机转发互通。
- 跨节点 **L2** 互通：经虚拟机–虚拟交换机出计算节点后，经物理网络转发到对端计算节点，并经对端虚拟交换机到达对端虚机，完成转发。
- **L3** 互通：经虚拟机–虚机交换机出计算节点，经物理网络上配置的三层路由转发，转为对端的 **vlan** 网络，并到达对端计算节点，经虚拟交换机到达对端虚机，完成转发。

4.4.1.4.3 经典网络

- 经典网络中，云平台只提供二层网络功能，不支持子网、路由、安全组/防火墙等特性，三层网络及安全防护需要在硬件网络设备上进行配置。其流量模型如下：



- 同节点 **L2** 互通：经计算节点内虚拟交换机转发互通。

- 跨节点 **L2** 互通：经虚拟机–虚拟交换机出计算节点后，经物理网络转发到对端计算节点，并经对端虚拟交换机到达对端虚机，完成转发。
- **L3** 互通：经虚拟机–虚机交换机出计算节点，经物理网络上配置的三层路由转发，转为对端的 **vlan** 网络，并到达对端计算节点，经虚拟交换机到达对端虚机，完成转发。

4.4.2 SDN 配置

4.4.2.1 SDN 配置

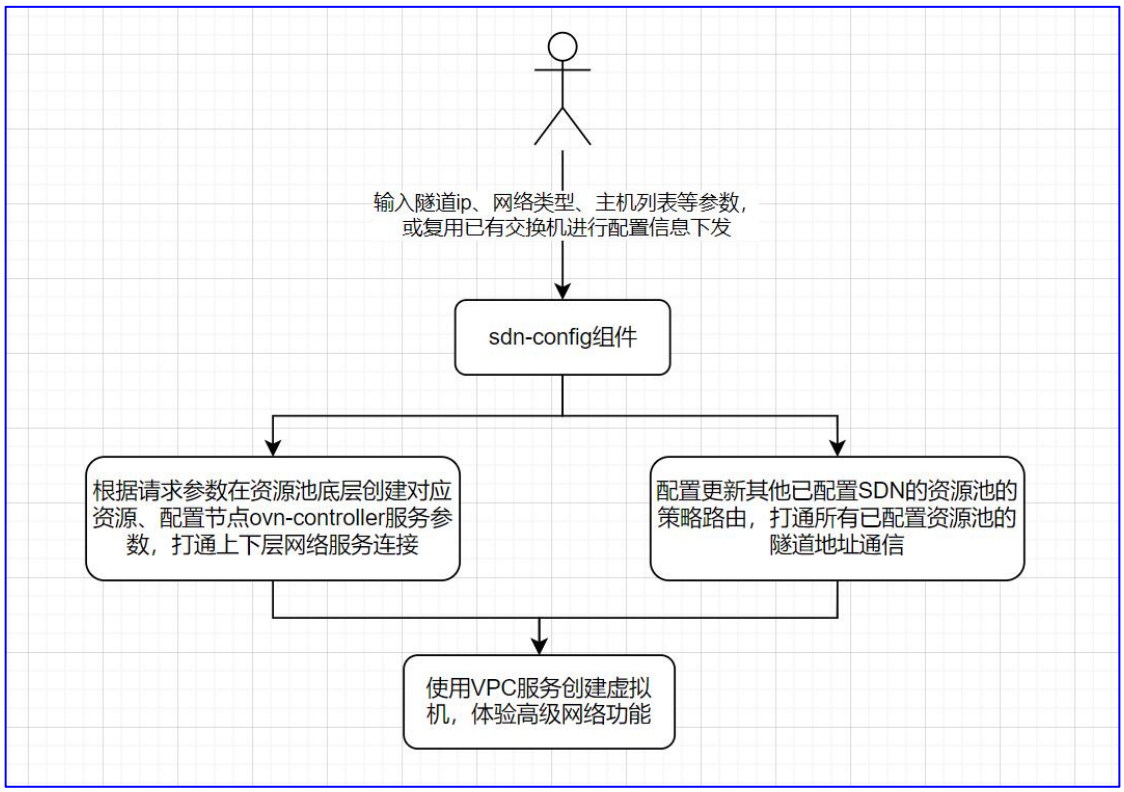
4.4.2.1.1 功能描述

平台通过 **SDN** 配置功能，实现虚拟化资源池分布式控制器、计算节点业务网络上行链路的自动化、可视化配置，该功能为 **VPC** 网络使用的基础。

用户在需要使用 **VPC** 网络时，需要通过 **SDN** 配置功能对资源池进行 **SDN** 配置，配置完成后即可使用 **VPC** 创建云主机，并使用 **NAT** 网关、安全组等高级网络功能。

4.4.2.1.2 技术原理

系统通过配置 **SDN** 所传参数对资源池底层执行对应配置，包含但不限于创建用于连接 **VPC** 与计算服务的介质、配置用于通信的上行链路及底层连接的 **ovn** 地址等。使用流程如下图：



4.4.2.1.3 约束

无

4.4.2.2 外部网络多网段

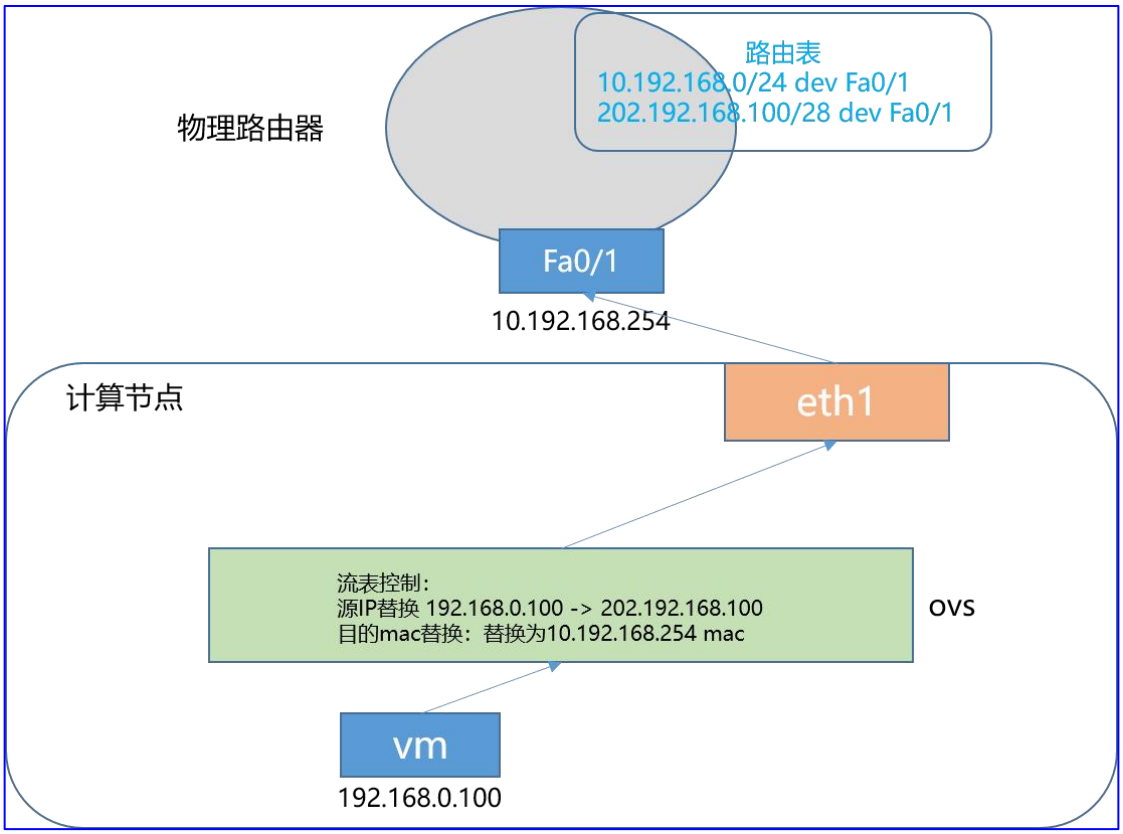
4.4.2.2.1 功能描述

外部网络多网段，每个网段可以有不同的用途。如网段 1 配置云内私网地址，IP 用于虚拟路由器接口，网段 2 配置公网网段，IP 用于分配浮动 IP，实现云平台内部不占用公网 IP。

4.4.2.2.2 技术原理

软件 SDN 将多个网段划分到一个二层中，虚拟机报文经过 ovs 流表实现 SNAT 替换成一个网段 B 的 IP 后，可以直接二层发送到另一个网段 A 的网关。返回报文通过在物理路由器配置路由，将网段 B 的报文，经过网段 A 的网关接

口发出，到达计算节点，进过 ovs 流表转换目的地址，到达虚拟机。实现流量互通。



4.4.2.3 内大网

4.4.2.3.1 功能描述

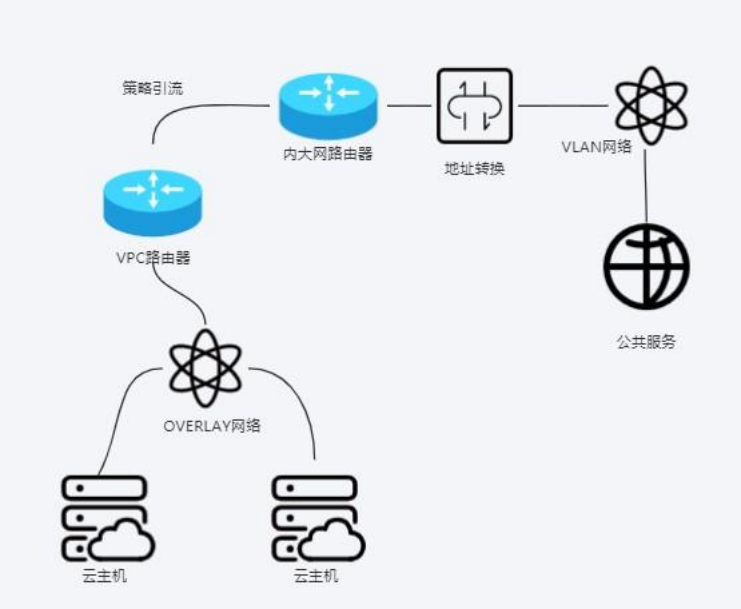
内大网可配置一个 **underlay** 网络，定义一组访问网段。使用互联 **VPC** 子网的云主机，可通过绑定内大网，通过内大网定义的 **VLAN** 网络，访问配置的访问网段。

使用互联 **VPC** 子网的云主机，无法直接通过互联 **VPC** 子网，直接访问部署到 **underlay** 网络中的公共服务（如对象存储、文件存储等）。需要经过一系列的浮动 **IP** 配置或者再添加一个精简 **VPC** 的网卡，会带一些流量隔离和操作复杂问题。云主机绑定内大网后，可在其他流量不受影响的前提下，将访问特定网段

的流量引入内大网所在 **VLAN**，实现访问公共服务。

4.4.2.3.2 技术原理

软件 **SDN** 在互联 **VPC** 绑定内大网后，将 **VPC** 的默认路由器连接到内大网路由器。通过策略路由的方式，将访问公共服务的流量引入到内大网路由器。内大网路由器通过 **SNAT** 转换，将源 **IP** 替换成内大网网段的 **IP**，将报文发送到物理 **VLAN** 网络中，实现访问公共服务。



4.4.3 VPC

4.4.3.1 VPC

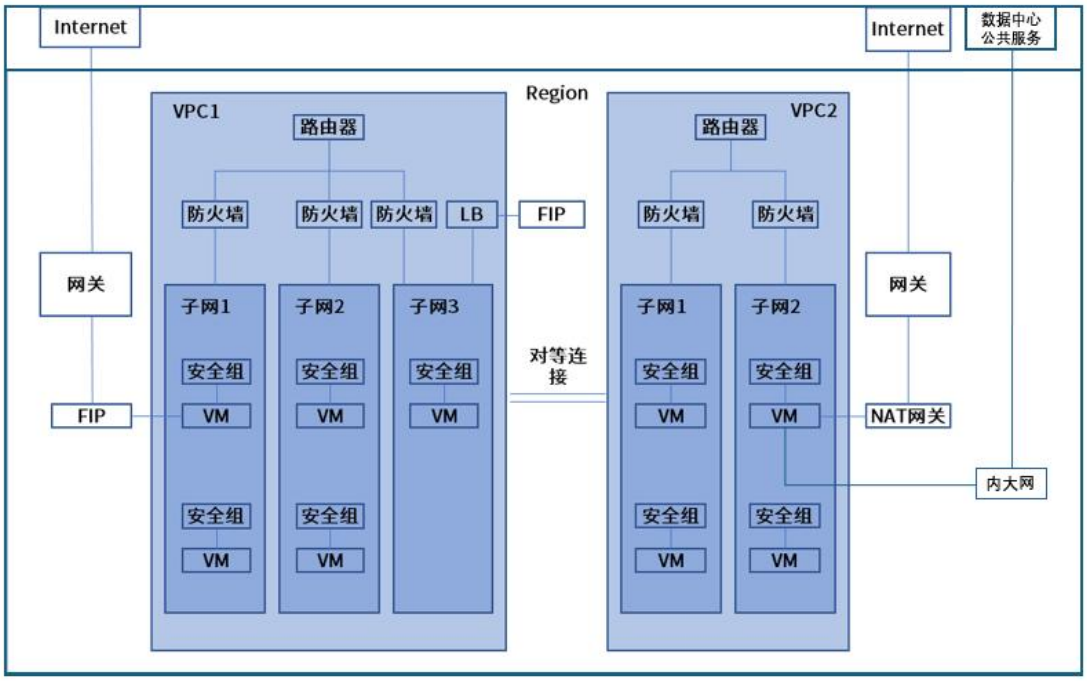
4.4.3.1.1 功能描述

虚拟私有云（**Virtual Private Cloud**）是一个逻辑隔离的网络空间。租户可拥有一个或多个 **VPC**，**VPC** 之间网络隔离。用户可使用 **VPC** 自主构建管理隔离的虚拟网络环境，提升用户在私有云中资源的安全性。通过 **VPC**，用户可以完全掌控自己的虚拟网络，包括绑定内大网、创建子网、创建路由、防火墙、配

置 DHCP、开启组播等，使用户能够在保证网络隔离的同时，更加科学的划分网络范围，部署灵活、管理方便，满足不同的业务需求，统一 VPC 框架下，虚拟化资源及容器资源均可以使用虚拟私有云及其下的资源，形成互联互通的整体模型，利用虚拟私有云的拓扑图，用户能够统揽关联的多种云资源，支持快速明确资源占有、资源归属，在云平台中，通过引导页为用户提供图文并茂的阐述，增强了易用性，降低了用户学习门槛。

VPC 存在两种模式：互联模式 VPC 和精简模式 VPC。互联模式 VPC 适用于方案 1 场景，VPC 创建成功之后，会创建出默认路由器，VPC 内子网会自动连接到该路由器上。精简模式 VPC 中三层转发需要依赖物理路由器实现，适用传统网络路线；互联模式 VPC 内可包含私有云上的多种网络资源，如网络、路由器、防火墙、NAT 网关等。

4.4.3.1.2 互联模式 VPC 技术原理

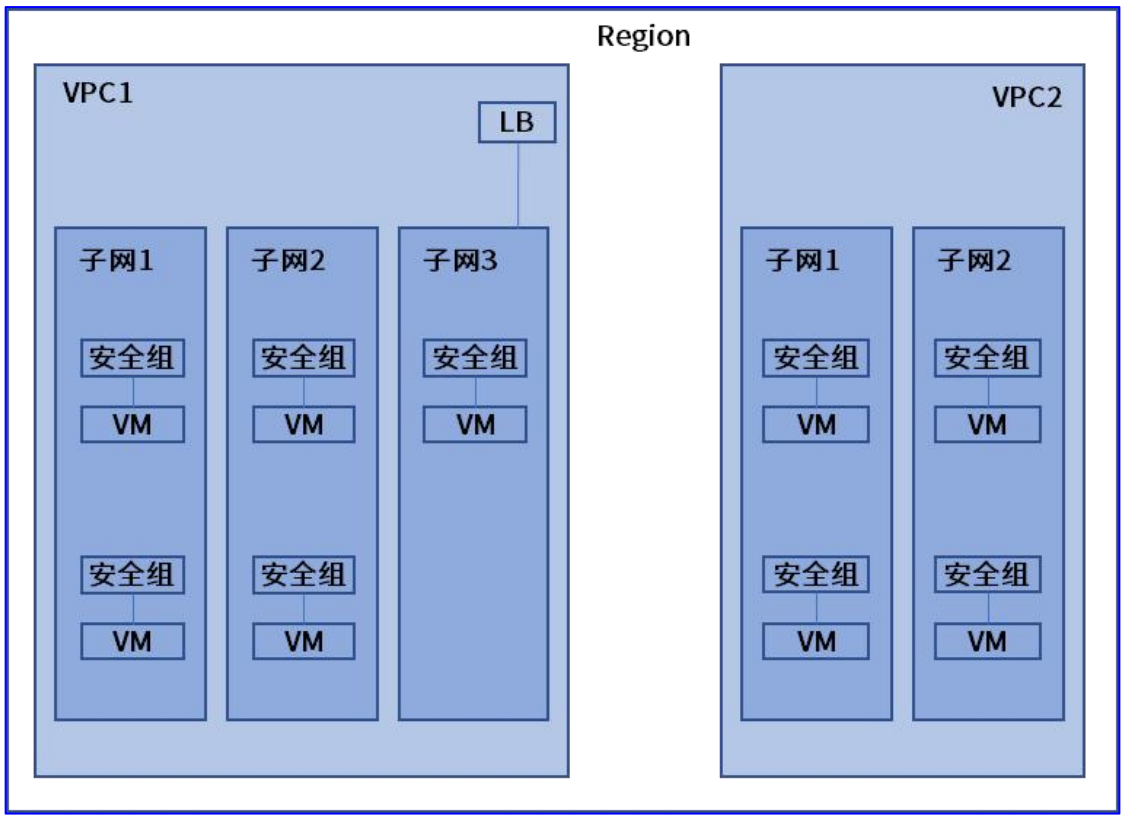


VPC 是一套逻辑隔离的、由用户自主配置和管理的虚拟网络环境，旨在提

升用户资源的安全性，简化用户的网络部署。

VPC 内网络资源处于单一资源池中，互联模式 **VPC** 内部通过路由技术，实现网络的三层互通，**VPC** 之间通过对等连接技术实现网络互联。

4.4.3.1.3 精简模式 VPC 技术原理



精简模式 **VPC** 下，用户只能使用云平台的二层网络功能，三层网络依赖硬件交换机手动配置实现。因此只能使用子网、安全组等功能，虚拟路由器、防火墙、**NAT** 网络等依赖三层的功能都不可用。

4.4.3.2 分布式路由

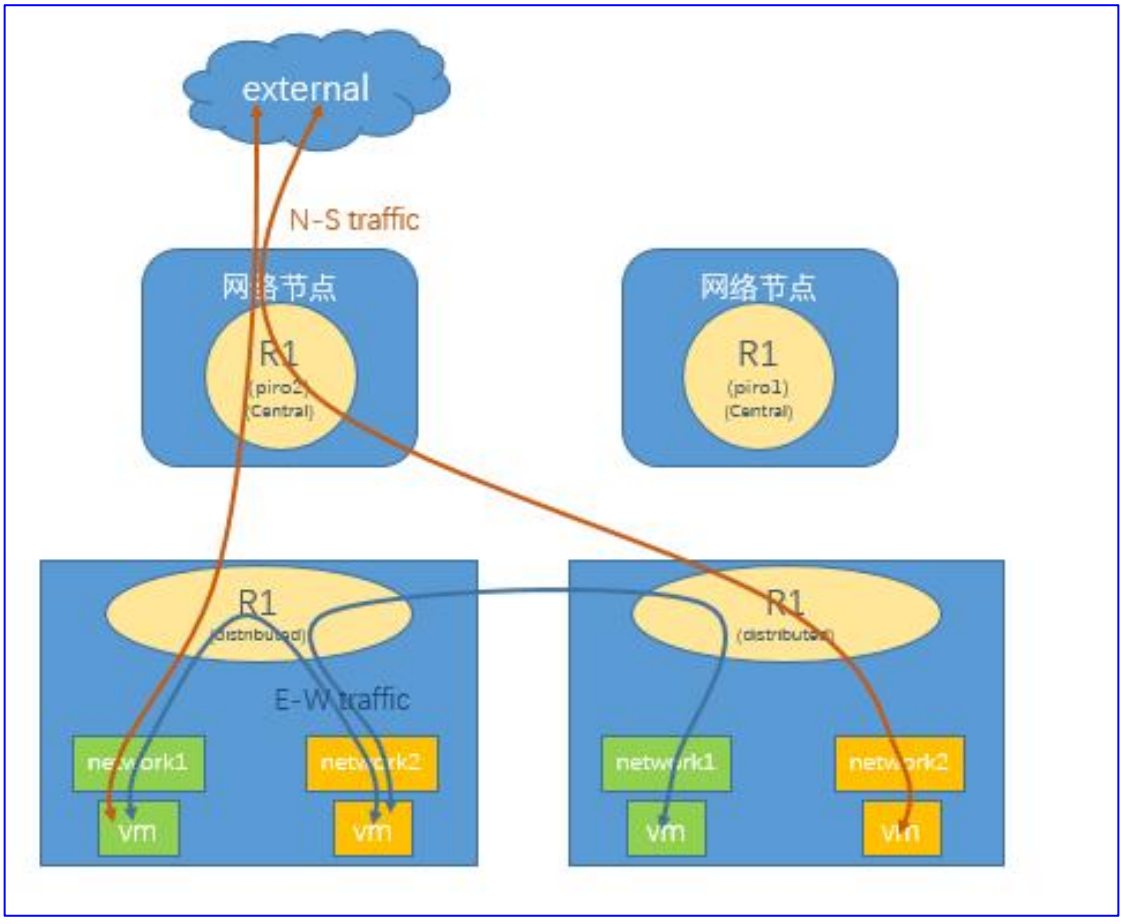
4.4.3.2.1 功能描述

在平台的网络环境中，跨子网的云主机通信需要通过虚拟路由器。这既包括不同子网的云主机之间的通信，又包括云主机与外网之间的通信。

传统网络模型下， 虚拟路由器会部署在集中式的网络节点或交换机下， 从而产生了两个问题，其一是网络节点/集中式三层交换机将成为整个网络的瓶颈，其二是单点失败的问题。针对上述问题，**InCloudOS** 提供分布式路由，基于分布式及高可用设计将网络服务及流量分布到各计算节点，解决上述问题。

4.4.3.2.2 技术原理

软 **SDN** 通过分步式路由解决上述问题，东西向（**E–W**）上的流量不需要将数据包发送到网络节点，而是由计算节点本地的分布式路由直接进行跨子网的转发；南北向会经网络节点出局，网络节点采用主备高可用部署模式，在主节点故障时会立刻切换到备节点上，保障用户业务正常运行。



◆ 对于东西向的流量，流量会直接在计算节点之间传递。

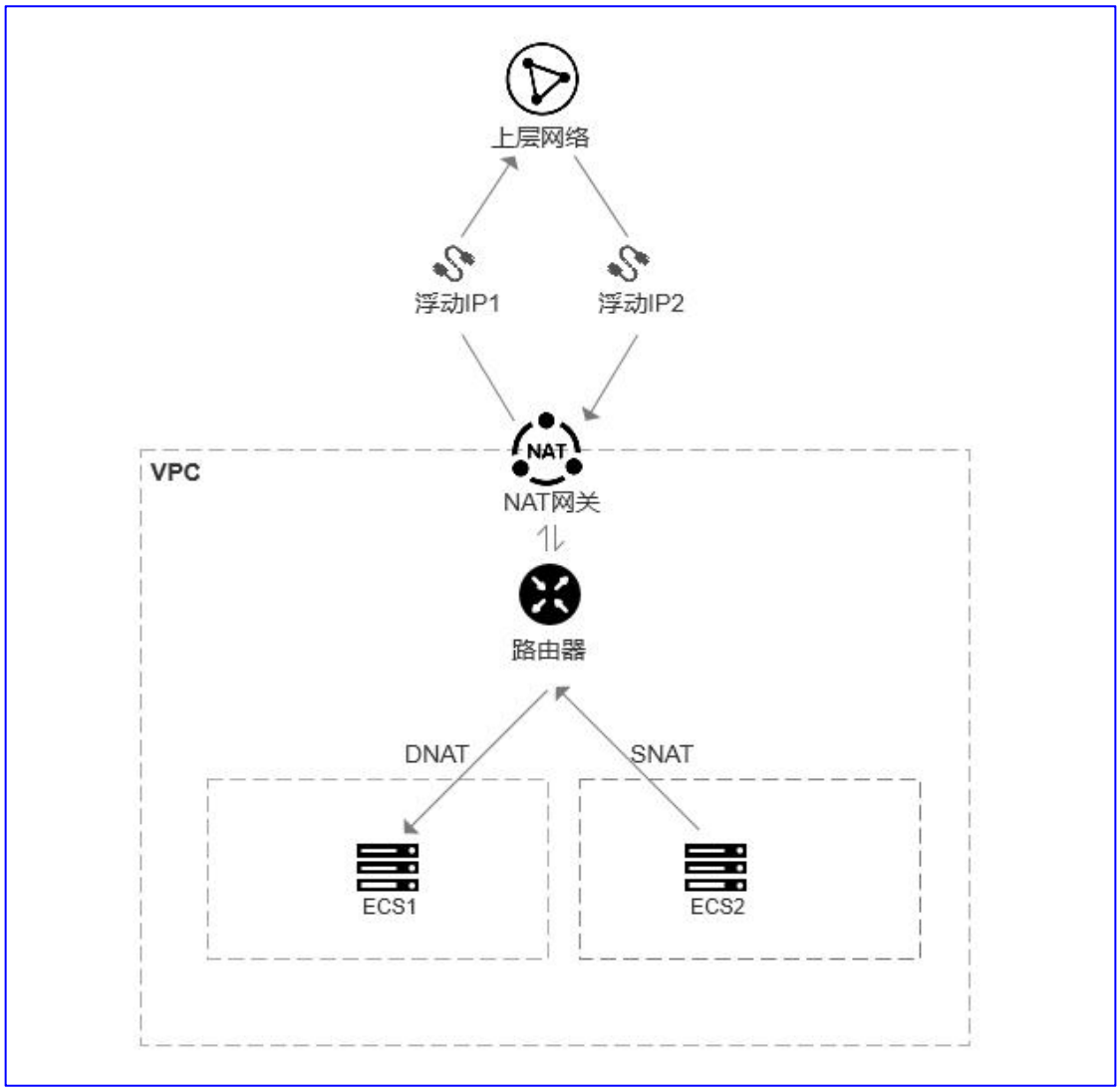
◆ 对于南北向的流量，经网络节点出局，主节点故障后流量备节点接管。

4.4.3.3 NAT 网关

4.4.3.3.1 功能描述

NAT 网关提供网络地址转换服务，包含 **DNAT**（目的地址转换）与 **SNAT**（源地址转换）。**DNAT** 可以用于实现一个外部网络地址的端口转发到 **VPC** 内应用的某个服务端口，支持 **IPv4** 和 **IPv6**，极大地拓展了网络的兼容性与适用性;**SNAT** 可以用于实现多台云主机共享一个外部 **IP** 地址访问外部(路由的外层)网络。

4.4.3.3.2 技术原理



虚拟私有云中存在一个默认路由器，用于将云内网络联通起来。通过在默认路由器上关联外部网关，通过添加 **SNAT** 规则，使得云内云主机可以将地址转换为网关 **IP**，为上层网络提供服务。通过添加 **DNAT** 规则，使得浮动 **IP** 的指定端口可以映射为云内 **IP** 的指定端口，实现上层网络访问云内云主机。

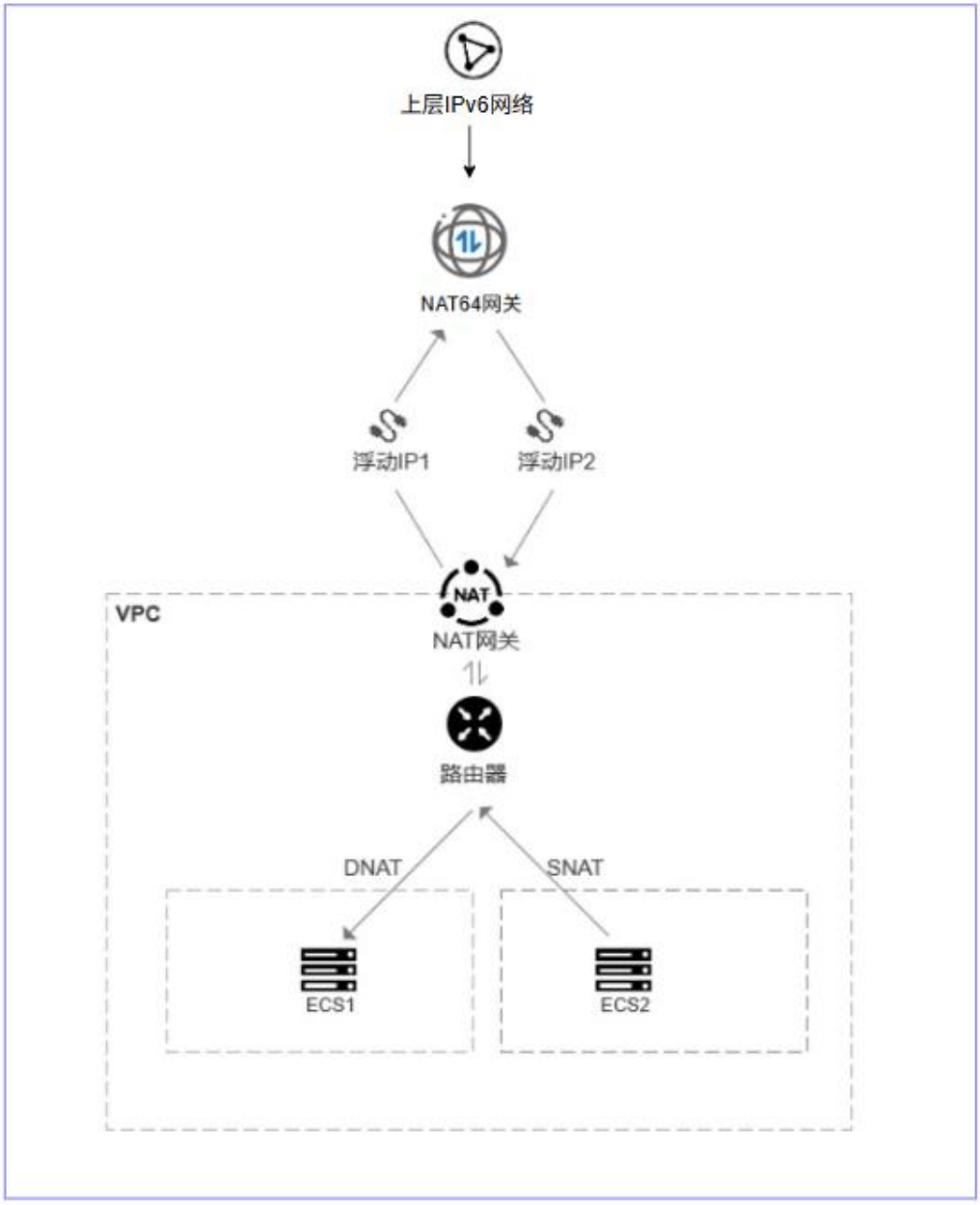
4.4.3.4 NAT64 网关

4.4.3.4.1 功能描述

NAT64 网关提供双栈网络地址转换服务，用于将 **IPv6** 地址转换为 **IPv4** 地

址，打通 **IPv6** 客户端无缝访问 **IPv4** 服务端的业务通信，提升云平台双栈网络架构弹性扩展能力。

4.4.3.4.2 技术原理



IPv6 客户端访问封装过的 **IPv6** 地址时报文会先到达 **NAT64** 网关，由内置服务进行双栈地址转换后将业务报文转发至对应 **IPv4** 服务端中，实现上层 **IPv6** 客户端能够跨协议栈访问到后层 **IPv4** 服务端。

4.4.3.5 浮动 IP

4.4.3.5.1 功能描述

浮动 IP（又称弹性 IP）是外部公有网络中的地址。VPC 子网中的云资源绑定浮动 IP 后可以与外界互相通信，其本质是一种 NAT 技术。浮动 IP 支持 IPv4 和 IPv6，IPv4 的浮动 IP 支持绑定 IPv4 的 port，IPv6 的浮动 IP 支持绑定 IPv6 的 port。

4.4.3.5.2 技术原理

软 SDN 实现浮动 IP 时，创建 OVN 的 nat 资源，nat 的类型为 dnat_and_snat。OVN 按照 nat 信息转换成相应的 ovs 流表。基于 ovs 流表实现 nat 的转换。对于目标地址是浮动 IP 的报文，更换目标 IP 为虚拟机 IP，进行转发。对于源地址是虚拟机 IP，出去外部的流量，更换源地址为浮动 IP。

4.4.3.6 对等连接

4.4.3.6.1 功能描述

对等连接（Peering Connection）是一种网络连接方式，用于在不同的虚拟私有云（VPC）之间建立直接的网络通信通道。通过对等连接，用户可以实现不同 VPC 之间的内网互通，无需通过外部网络或 VPN，确保数据传输的安全性和高效性。对等连接还支持配置关联子网，用户可以灵活选择两端 VPC 内哪些子网进行互通，进一步提升网络管理的灵活性和精细化程度。

对等连接主要用于以下场景：

1. 跨账户 VPC 互通：支持在同一账户或不同账户下的 VPC 之间建立连接，

方便组织内部不同部门或合作伙伴之间的资源共享。

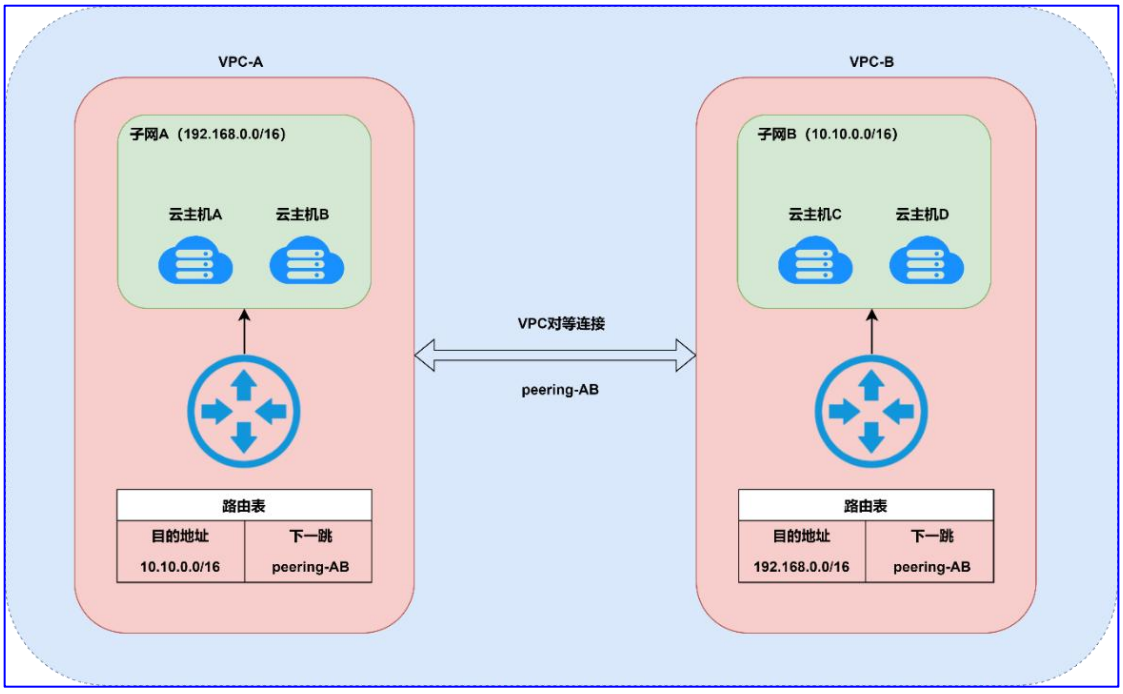
2. 多区域业务互通：在不同区域内，通过对等连接实现 **VPC** 之间的互通，支持跨区域的业务部署和数据同步。例如，可以在不同区域部署业务应用和数据库，通过对等连接实现低延迟、高带宽的数据传输，确保业务的高效运行。

3. 高可用性和冗余：通过对等连接，用户可以构建高可用性和冗余的网络架构，确保业务的连续性和稳定性。

4. 简化网络架构：对等连接简化了网络架构，减少了网络配置的复杂性和运维成本。

对等连接的使用场景广泛，包括组织内部系统的互联、跨部门数据共享、灾备系统的构建等。通过对等连接，用户可以灵活地构建和管理其云上网络，实现资源的高效利用和业务的快速部署。

4.4.3.6.2 技术原理



虚拟私有云之间通常网络隔离，但是通过路由技术可以实现三层互通，对等

连接便利用了这一特性，使得虚拟私有云之间的网络能够互联互通。以下是其主要技术原理：

1. 路由配置：在建立对等连接后，双方 **VPC** 需要配置相应的路由表，将目标 **VPC** 的 **CIDR**（**Classless Inter-Domain Routing**）块添加到本地路由表中，以实现流量的正确转发。在本系统内只需要指定两端子网，即可实现自动化的路由配置，简化了用户的操作。

2. 网络隔离：对等连接通过严格的网络隔离机制，确保不同 **VPC** 之间的流量不会相互干扰。每个对等连接都是独立的，且只能在授权的 **VPC** 之间进行通信，保证了网络的安全性。

3. 安全性：对等连接的数据传输不经过外部网络，减少了数据泄露的风险。同时，用户可以结合安全组和网络 **ACL**（访问控制列表）等安全策略，进一步增强网络的安全性。

4. 关联子网配置：在对等连接中，用户可以配置关联子网，指定哪些子网可以通过对等连接进行互通。这一机制允许用户精细化管理网络流量，确保只有授权的子网能够进行通信，提升了网络的安全性和管理效率。

通过对等连接，用户可以实现云上资源的高效互通和管理，构建安全、稳定、高性能的云网络环境。

4.4.3.7 IP 地址预留

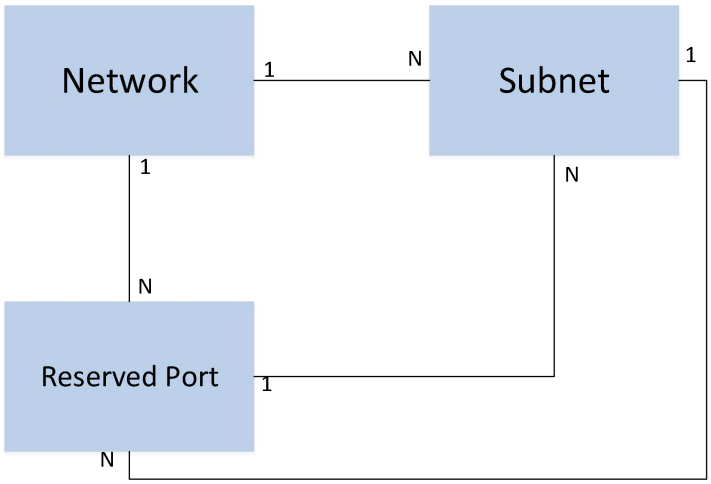
4.4.3.7.1 功能描述

平台支持在 **VPC** 子网中对部分 **IP** 地址的预留，该类预留 **IP** 不会再分配给其它资源，保障不会与其他服务自主配置的 **IP** 产生冲突，如多台云主机配置

Keepalived 作主备时，预留出的 **IP** 地址可以用作 **VIP** 管理多台云主机。

4.4.3.7.2 技术原理

IP 地址预留是在网络子网中创建一部分预留端口，并绑定 **IP** 地址和对应的 **MAC** 地址，同时在数据库中会自动生成对应的表项，且不会在宿主机上创建 **TAP** 设备，网络子网预留端口的关系如下图所示。



预留端口和普通端口的区别在于二者在配置端口创建时的参数有所不同，预留端口的创建在 **Device_owner** 为指定的 **network：reserved**，同时端口安全和管理状态都是默认状态置 **False**。

4.4.3.8 QoS

4.4.3.8.1 功能描述

平台实现了端口配置 **QoS**，对流量进行带宽限制的功能。**QoS** 具有 **policy** 和 **rule** 两个概念，**rule** 是具体的规则，而 **policy** 是 **rule** 的集合，可以将 **policy** 关联到虚拟端口或浮动 **IP** 实现带宽限制。

4.4.3.8.2 技术原理

ovn 作为 Neutron ml2 的 mechanism driver，具有对虚拟端口和浮动 IP 进行 QoS 配置的功能。Neutron 接收 api 请求，将 QoS 信息传递给 ovn，ovn 将 QoS 转化为流表作用于 ovs 的网桥上。端口带宽限制是通过 meter 流表下发到虚拟机端口所在网桥实现的，当端口带宽超过限速后就会对报文执行 drop 操作。

4.4.3.9 安全服务引流

4.4.3.9.1 功能描述

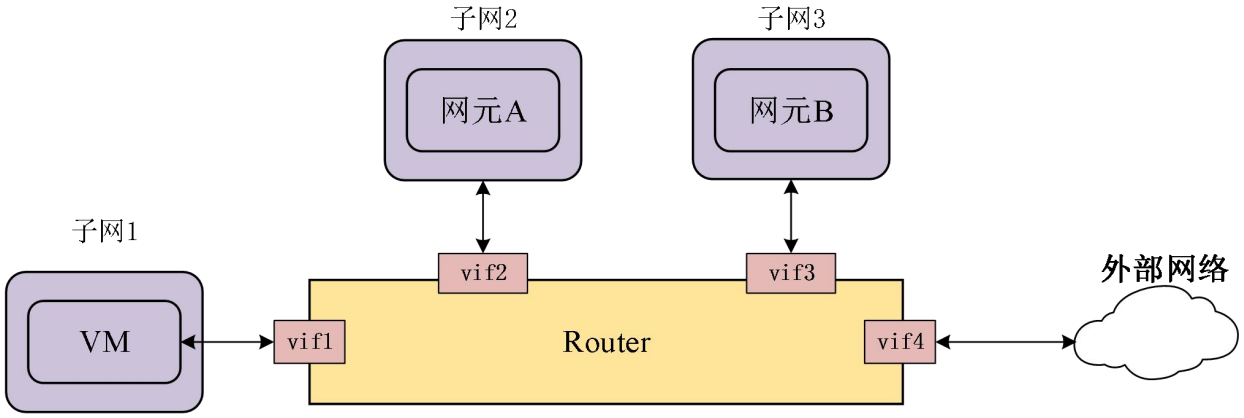
平台支持安全服务引流功能，为 VPC 内云主机提供安全防护。安全服务引流将多种安全服务实例（如云防火墙、云 WAF、云堡垒机、云主机安全等）按业务需求编排成有序流程，对云平台 VPC 中的网络流量进行 “串联式” 安全检测与防护。安全服务引流解决了传统安全防护 “单点部署、分散管理、难以适配云弹性”的痛点，实现了安全能力的灵活扩展、按需调用和统一管控。

安全服务引流支持南北向、东西向的流量引流。对于南北向流量，支持多级网元编排和网元单双臂模式设置；对于东西向流量，支持指定子网流出的流量引流到安全网元。

4.4.3.9.2 技术原理

安全服务引流基于 ovn 的策略路由功能实现，策略路由匹配子网 CIDR 和路由接口 mac 地址，将匹配到的流量进行重路由到安全网元，经过安全网元处理后，再进入路由转发流程到达目的地址。对于有多种网元的情况，添加策略路由匹配上一级网元处理后的流量，并重路由到下一级的安全网元；对于网元有单双臂模式设置需求的场景，按照流量方向识别需要经过的网元，并基于网元地址下

发策略路由。



4.4.4 访问控制

4.4.4.1 安全组

4.4.4.1.1 功能描述

平台通过结合和优化 **Open vSwitch** 流表，提供安全组功能以支持云主机级别的安全防护措施。

安全组支持：

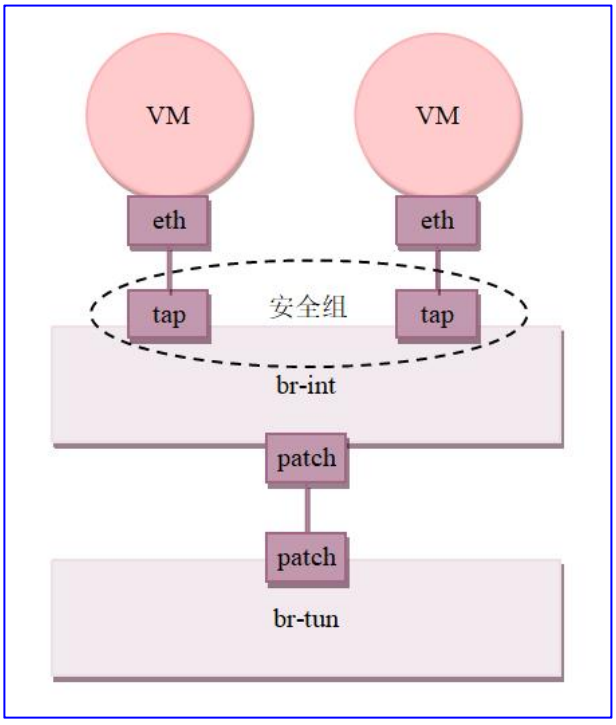
- 定制 **TCP** 规则
- 定制 **UDP** 规则
- 定制 **ICMP** 规则
- 自定义 **IP** 协议规则
- 自定义 **DNS** 规则
- 支持黑白名单功能

- 支持优先级
- 支持绑定地址组资源
- 支持配置端口集
- 支持有、无状态安全防护

4.4.4.1.2 技术原理

安全组实现逻辑如下：

- **OVN** 通过 **Port Group** 实现安全组，虚拟机接口加入的 **Port Group** 中。
- **OVN** 通过 **ACL** 实现安全组规则，**ACL** 会加入到 **Port Group** 中。
- **OVN** 下发流表到 **OpenVSwitch**，实现具体的安全组功能。
- **TAP** 口加入安全组后，进出 **TAP** 口的流量需要经过安全组检查。



安全组支持用户任意组合安全组规则，安全控制云主机入和出方向的所有流量；在云主机迁移（云主机 **HA**、手动迁移）等各种场景下，安全组规则也会伴随云主机自动迁移。

4.4.4.2 防火墙

4.4.4.2.1 功能描述

平台支持防火墙的配置管理，云平台主要通过以下概念来实现对防火墙的管理：

- 规则（**Rule**）：允许用户指定所要匹配的名称、描述、针对的协议（**TCP**，**UDP**，**ANY**）、行为（**Allow**，**Drop**）、源/目的 **IP** 地址/子网和端口号/端口号区间。
- 防火墙（**Firewall**）：规则的逻辑集合，可以关联作用到的虚拟路由器。
- 与安全组的规则的区别：防火墙应用于 **vRouter**、防护 **vRouter** 的流量，安全组则应用于虚拟机的虚拟网卡、防护虚拟网卡进出流量安全。

4.4.4.2.2 技术原理

平台防火墙是基于 **OVN** 的分布式防火墙，通过 **OVN** 中的策略路由功能实现。源于 **OVN** 的分布式架构，基于策略路由功能实现的防火墙也是分布式架构，防火墙规则存在于多个节点，通过向虚拟路由器上添加策略路由规则来实现对进出数据网络的网络包进行控制的。

4.4.5 网间互联

4.4.5.1 云专线

4.4.5.1.1 功能描述

平台通过云专线功能，实现 **vpc** 网络与云外网络的打通。云专线包括专线网关和关联子网两个模块。

专线网关：用户可以通过云专线网关动态扩展专线网关数量，进而借由 ECMP 技术，实现提升云专线带宽的目的；

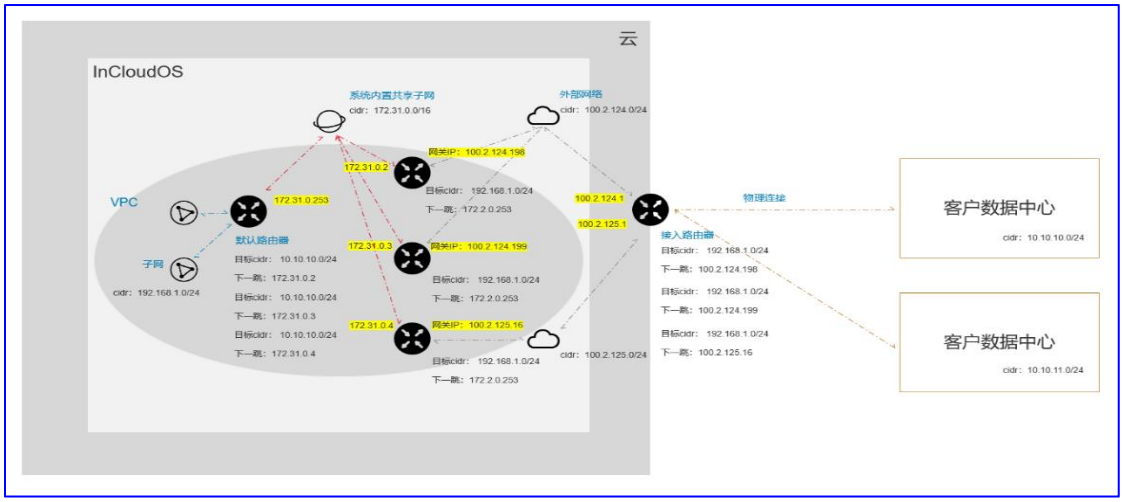
关联子网：配置云内子网和云外 CIDR，都支持多选。

4.4.5.1.2 技术原理

云专线通过路由器、共享子网、外部网络这三个组件组合实现。

首先，在 VPC 内创建自定义路由器，将默认路由器和自定义路由器连接到系统内置共享子网。然后为该路由器选择和接入路由器网络连通的外部网络设置为网关。不同路由器可以选择同一个外部网络设置为网关，也可以选择不同外部网络设置为网关。最后，在默认路由器、自定义路由器上配置好静态路由。

通过默认路由器开启 ECMP，北向数据包可以负载到多个自定义路由器上，进而提高云内到云外的带宽。同理，通过接入路由器开启 ECMP，可以提高云外到云内的带宽。



4.4.5.2 VPN

4.4.5.2.1 功能描述

VPN（Virtual Private Network）又叫做虚拟专用网络。它是在公用网络上

建立的专用网络，进行加密通讯。平台提供 VPN 功能，VPC 内云主机可以通过 IPSec 隧道与对端云主机或设备进行安全加密的通信。

VPN 功能包含基本信息、IKE 策略、IPSec 策略、站点连接四个管理维度。

基本信息：指定 VPN 的名称、虚拟数据中心、VPC 网络、描述，VPC 网络中的默认路由器为 VPC 提供访问远端站点的出口网关，可建立 VPC 内网络与远端站点之间的安全加密通信。

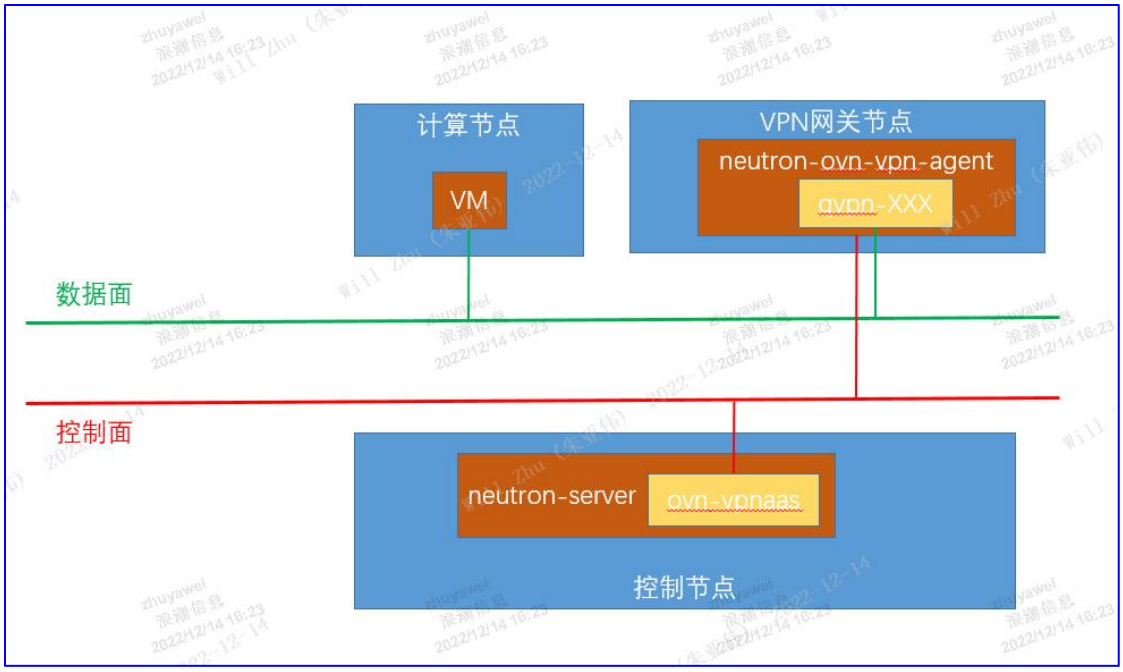
IKE 策略：Internet Key Exchange，因特网密钥交换，该策略可以为通信双方提供建立 IPSec 隧道时协商阶段所需要的加密和认证算法。

IPSec 策略：该策略可以为通信双方提供 IPSec 隧道在数据传输阶段所使用的协议、封装模式、加密和认证算法。

站点连接：IPSec 站点连接是通信双方之间建立连接后的 VPN 通道，建立 IPSec 站点连接后，通信双方才可以通过 VPN 进行安全加密通信。

4.4.5.2.2 技术原理

在 neutron-server 中扩展 ovn-vpnaas 插件，通过 ovn-vpnaas 插件将 VPN 配置数据下发到 neutron-ovn-vpn-agent 容器中，创建 qvpn-XXX 网络命名空间。在 qvpn-XXX 网络命名空间中，添加 vpn 网关接口，启动相应进程进行 IKE 协商，生成 IPSec 隧道，进行 VPN 通信。同时在 qvpn-XXX 网络命名空间和逻辑路由器之间添加网络将相关 VPN 流量引到 qvpn-XXX 网络命名空间中。



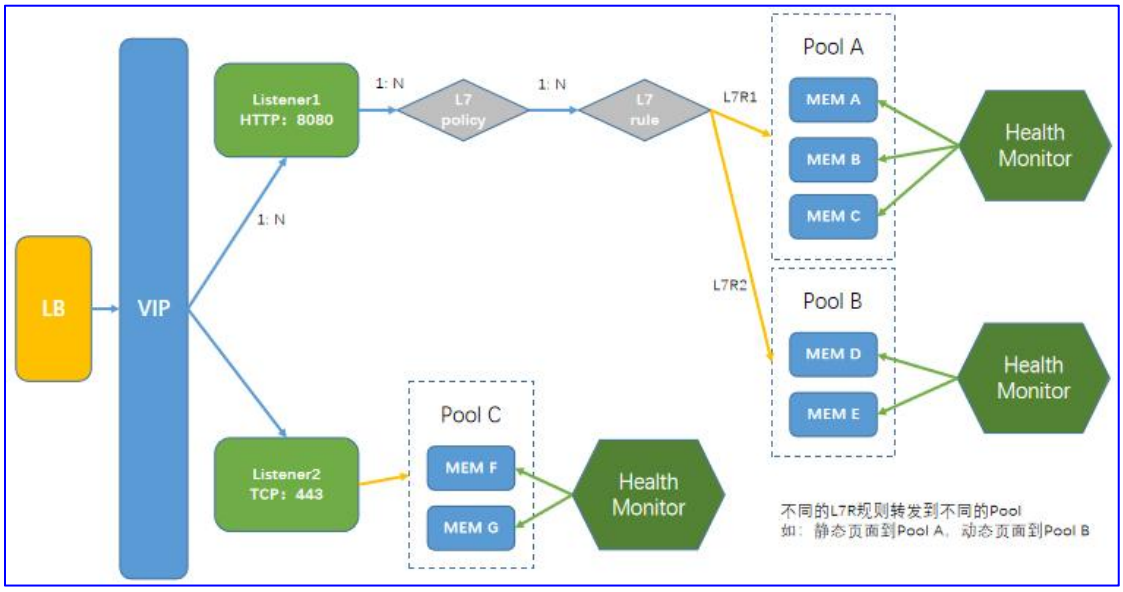
4.4.6 负载均衡

4.4.6.1 功能描述

负载均衡（**Server Load Balancer**）是将访问流量根据转发策略分发到后端服务器组（服务器组，可包含云物理机，云主机，服务实例）的流量分发控制服务。负载均衡扩展了应用的服务能力，增强了应用的可用性。负载均衡通过设置虚拟服务地址,将添加的服务器组成员虚拟成一个高性能、高可用的应用服务池，并根据转发规则，将来自客户端的请求分发给服务器组中的成员。负载均衡器对于 **IPv4** 与 **IPv6** 都有着较好的兼容能力与流量处理能力，支持使用 **IPv4** 或 **IPv6** 的网络创建负载均衡器，并且将流量分发至不同的服务器组成员。负载均衡支持通过启用和停用实现流量转发服务的正常处理与终止执行,控制流量通道的开启与关闭。

支持 **L7** 负载均衡，**L7** 负载名称来自 **OSI** 七层模型，意指基于七层数据分发请求到后端服务器组。**L7** 负载均衡又称为请求交换、应用负载均衡、基于内

容的路由、基于内容的负载均衡等，**L7** 负载是特定于 **HTTP** 或 **terminated HTTPS** 一种负载均衡特性，根据用户配置的一个或多个第 **7** 层策略，不同的客户端请求被路由到不同的后端服务器组，或者基于既定的路由策略，重定向到不同的 **URL**。

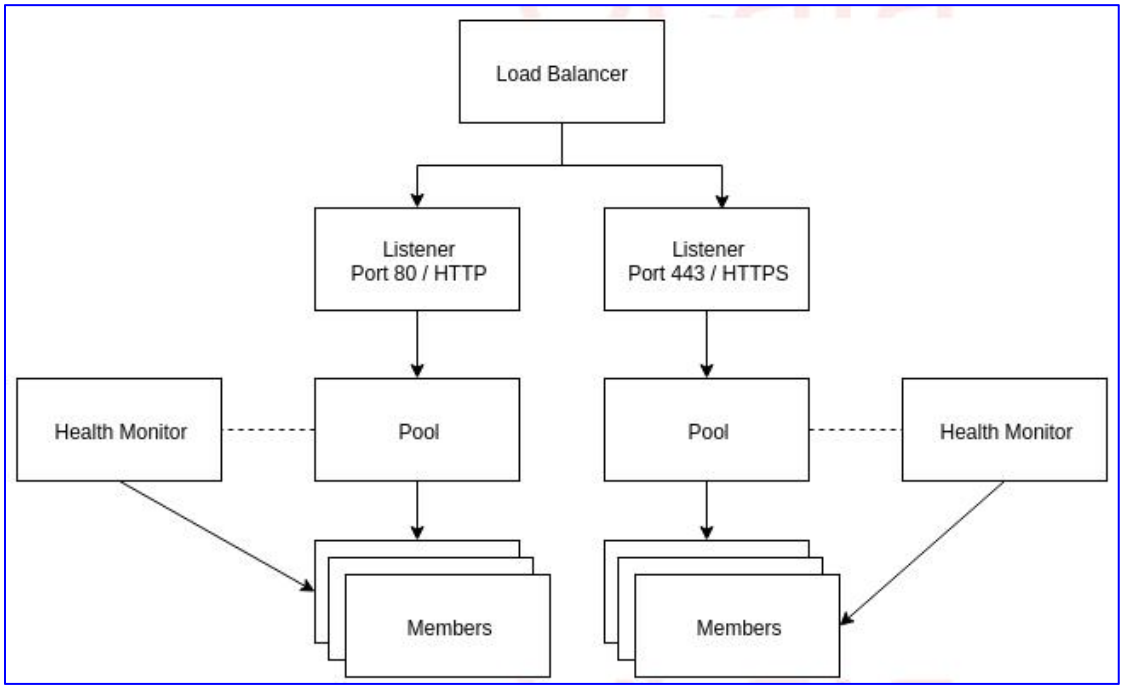


支持 **TERMINATED_HTTPS**，借助这一功能，可以将 **TLS** 流量的解密/加密工作从应用程序服务器分流至网络负载均衡器，从而帮助您优化后端应用程序服务器的性能，同时确保工作负载的安全，支持的证书类型包含国密、**rsa** 等。此外，网络负载均衡器会将客户端的源 **IP** 保存到后端应用程序，同时在负载均衡器上终止 **TLS**。负载均衡默认检查资源池中的云主机的健康状态，自动隔离异常状态的云主机，消除了单台云主机的单点故障，提高了应用的整体服务能力。平台提供向多个云主机中运行的应用提供负载均衡的方法，还提供 **API** 来快速方便地部署负载均衡器。该功能可应用于运营商级别的可靠、可扩展负载均衡场景，采用自研的高可用负载均衡方案提供底层支撑。

支持高性能负载均衡能力，通过开启 **haproxy** 多进程模式以及内核参数的调优，实现了计算资源的充分利用以及请求消息的快速响应，四层、七层新建速率和吞吐等重要指标都大幅提升。

相关概念：

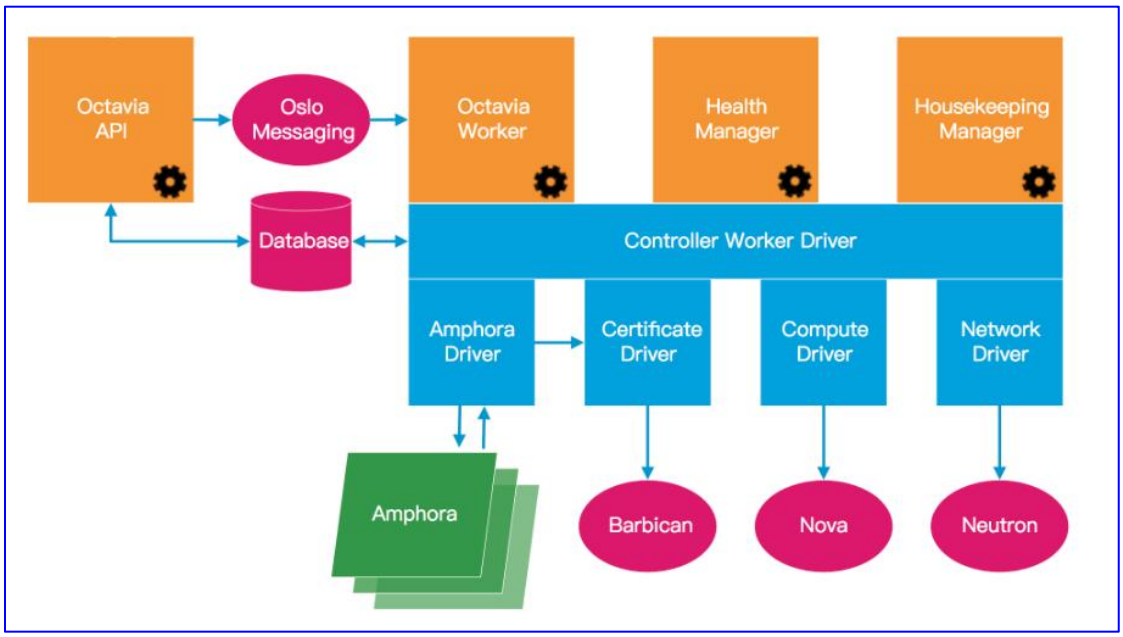
- **Load Balancer**：虚拟负载均衡器，具有对外提供统一服务的 **VIP**。
- **Listener**：监听 **VIP** 的不同端口。
- **Pool**：**Pool Member** 的集合，代表负载后端的云主机池。在以 **HAProxy** 为 **Driver** 的情况下，一个 **Pool** 对应着在一个独立的 **network NAMESPACE** 中运行的 **HAProxy** 进程所管理的 **backend**。
- **Member**：运行服务的云主机，可加入到 **pool** 中。
- **Health Monitor**：健康监测器，监测 **Member** 的存活状态，支持 **HTTP**、**TCP** 和 **PING** 等多种检测方法。当发现 **Member** 有异常时，会操作 **Pool** 的分发列表。



4.4.6.2 技术原理

平台默认使用 **Octavia + HAProxy** 实现负载均衡功能，每个负载均衡服务会对应一个或多个云主机。**Octavia** 根据用户提出的负载均衡要求在提供负载均衡

服务的云主机中生成符合要求的 **HAProxy** 配置文件并启动 **HAProxy**，然后由 **HAProxy** 进行负载均衡。



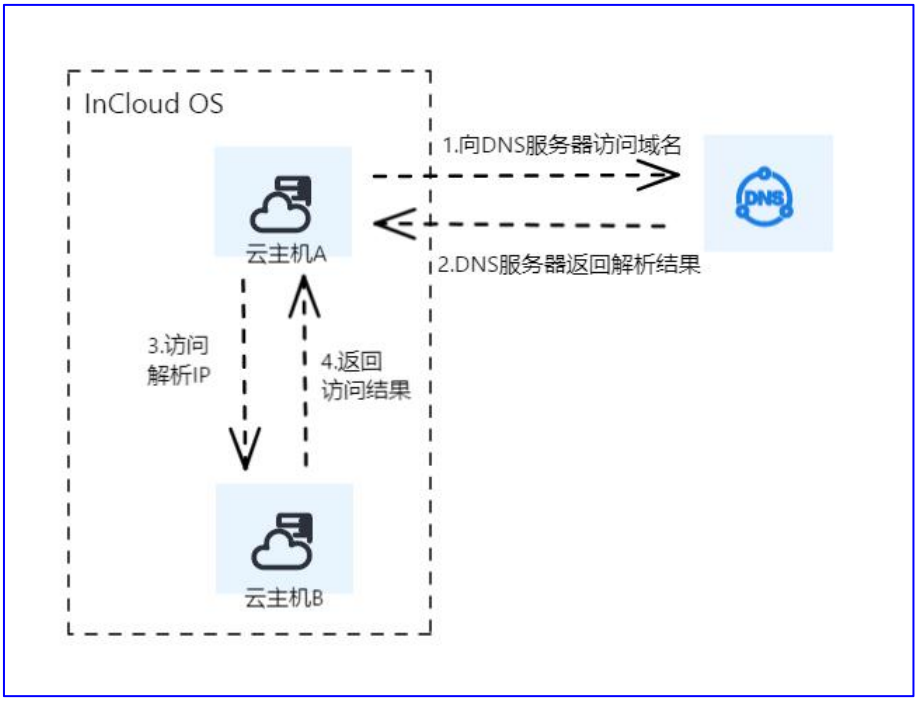
提供负载均衡服务的 **Amphora** 云主机一般会绑定三个租户网络，**management network** 被 **Octavia** 管理组件用来与 **amphora-agent** 通信，下发配置信息、监控负载均衡服务的运行状态。**VIP network** 实际提供业务的访问，客户端通过此网络访问 **Amphora** 云主机绑定的 **VIP**，当业务请求到达 **VIP**，负载均衡服务通过预先配置的负载方法（轮询、最小连接数等），将请求分配给指定的后端成员。**tenant network** 用于 **Amphora** 云主机及业务云主机的连接，当业务请求由负载均衡分配给指定的后端成员后，流量通过此网络由 **Amphora** 云主机发送到实际的业务云主机。平台当中提供了独立的 **Amphora** 云主机管理服务，支持查看节点列表、节点详情、关机、软重启、进入云主机控制台，与对待普通云主机无异，可通过关机来让负载均衡器触发自动重建 **Amphora** 云主机。

4.4.7 云解析 DNS

4.4.7.1 功能描述

DNS（**Domain Name System**）又叫做域名系统，它是一个将域名和 **IP** 地址相互映射的一个分布式数据库，能够使用户通过直接访问域名来访问网站或者 **Web** 应用程序。

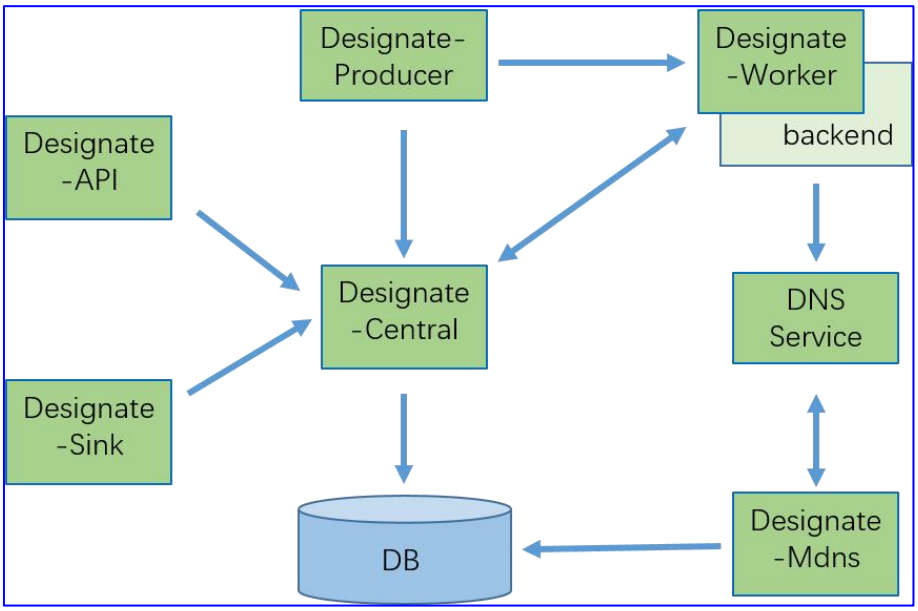
平台提供 **DNS** 服务，云主机访问域名时，集群内对接的 **DNS** 服务器对该域名进行解析，向云主机返回被访问域名对应的 **IP** 地址，云主机可根据解析 **IP** 访问目标云主机或相应服务。



平台基于 **Bind9** 提供 **DNS** 服务，用户可创建多种类型的域名，会自动将其创建到对接的 **DNS** 服务器中，用户可在子网中配置 **DNS** 服务器地址或直接在云主机中手动配置对接 **DNS** 服务器地址（可配置单个或多个），当访问域名时会自动前往 **DNS** 服务器中进行地址映射转换。

4.4.7.2 技术原理

平台基于 **Designate** 提供 **DNS** 服务。**Designate** 对外提供 **zone** 和记录集的增删查改 **RestAPI** 接口，用户调用接口后，**Designate** 会通过 **rndc** 技术远程操控 **DNS** 服务器中记录集的增删查改，然后用户访问 **DNS** 服务器来获取域名及 **IP** 地址的映射关系。



其中 **Designate–api** 组件负责接收 **API** 请求并转发给 **Designate–central** 组件进行处理，**Designate–central** 接收到请求到之后更新数据库，然后将请求下发给 **Designate–worker** 组件，**Designate–worker** 组件负责对接 **DNS** 服务器以及下发接收到的命令至 **DNS** 服务器中进行域名资源的增删查改，**DNS** 服务器处理完接收到的命令之后又会向 **Designate–mdns** 组件查询同步，从而达到 **DNS** 服务器和 **Designate** 数据库信息一致性。**Designate–producer** 组件负责大型作业调用服务以及对 **zone** 状态和数据库数据的维护工作。

4.4.8 经典网络

4.4.8.1 功能描述

经典网络是一种符合传统虚拟化用户使用习惯的二层网络功能，可以为虚拟机构建不同的隔离域，同一隔离域内的虚拟机可以互通，经典网络资源范围限于单个虚拟化资源池。

经典网络的 **VLAN** 值，可配置 **1–4095** 中的任意值，其中 **4095** 表示 **trunk all**，即允许所有 **VLAN** 通过。经典网络开启 **DHCP** 后可配置网络地址、地址分配池、网关地址。当用户未指定网关地址时，系统会自动分配一个分配地址池外的 **IP** 地址。

经典网络不仅支持删除及新建，同时也支持移除和纳管。被移除的经典网络在云平台界面中不再展示，但在底层虚拟化资源池上仍然存在，可以通过重新纳管该经典网络，再次将它导入到云平台中。

4.4.9 流量镜像

4.4.9.1 功能描述

流量镜像（**Traffic Mirroring**）是一项重要的网络流量管理服务，能把出入云主机网卡的流量完整拷贝一份，定向传输到另一台指定云主机或云外网络设备，方便用户进行网络监控、数据分析、数据备份等操作。

该服务有两大显著优势：一是不影响源云主机的网络转发，保障其正常业务运行；二是能保持报文内容（含报文头）的原始性，为后续分析提供可靠数据。

流量镜像主要用于以下场景：

1. 网络监控与安全审计：安全团队可将关键云主机流量镜像到监控主机，

实时监测异常行为和安全威胁，保障网络安全。

2. 性能优化与故障排查：运维人员通过镜像故障云主机流量，分析流量信息，定位网络问题，提升网络性能。

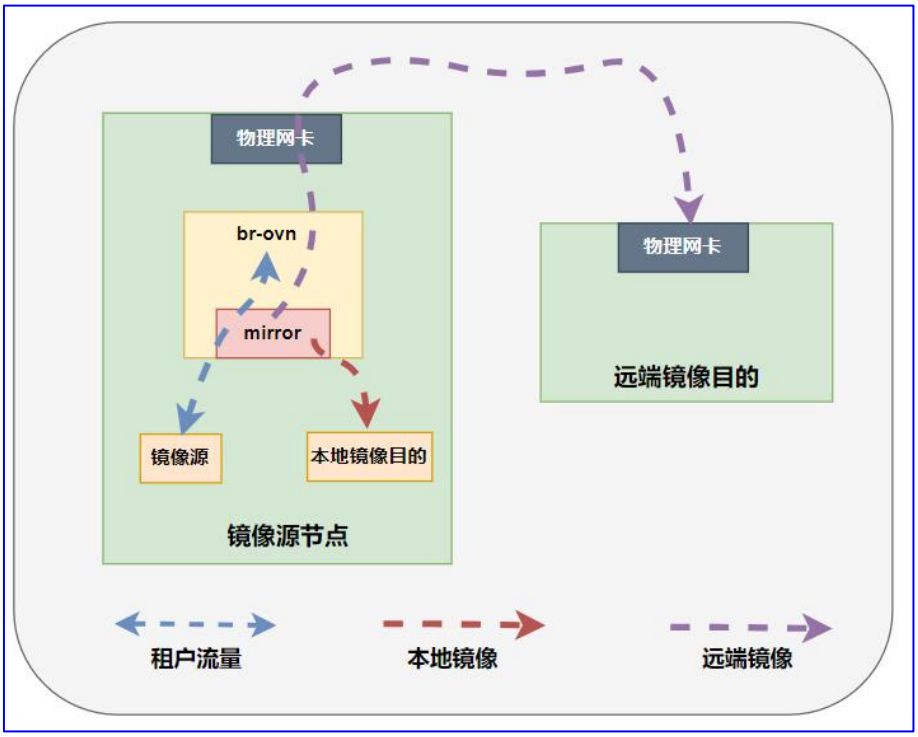
3. 数据备份与容灾：对关键业务数据，将云主机流量镜像到备份主机实现实时备份，确保灾难时数据可恢复，保障业务连续性。

4. 大数据分析与业务洞察：把流量数据镜像到分析平台，挖掘商业价值，为企业决策等提供数据支持。

流量镜像广泛应用于各行业，无论是保障金融交易安全、优化电商体验，还是确保医疗数据存储和支持在线教学等场景都能发挥重要作用。用户借助此服务，能更高效地管理云网络，推动业务发展。

4.4.9.2 技术原理

基于 **OVN** 的虚拟化网络环境，云主机发出的流量会送到 **br–ovn** 网桥，送到云主机的流量也会经过 **br–ovn** 网桥送出。流量镜像在 **br–ovn** 网桥下添加 **mirror** 流表，将所有与镜像源云主机有关的流量镜像到本地另一云主机或远端某主机中。



4.5 容器服务

4.5.1 应用中心

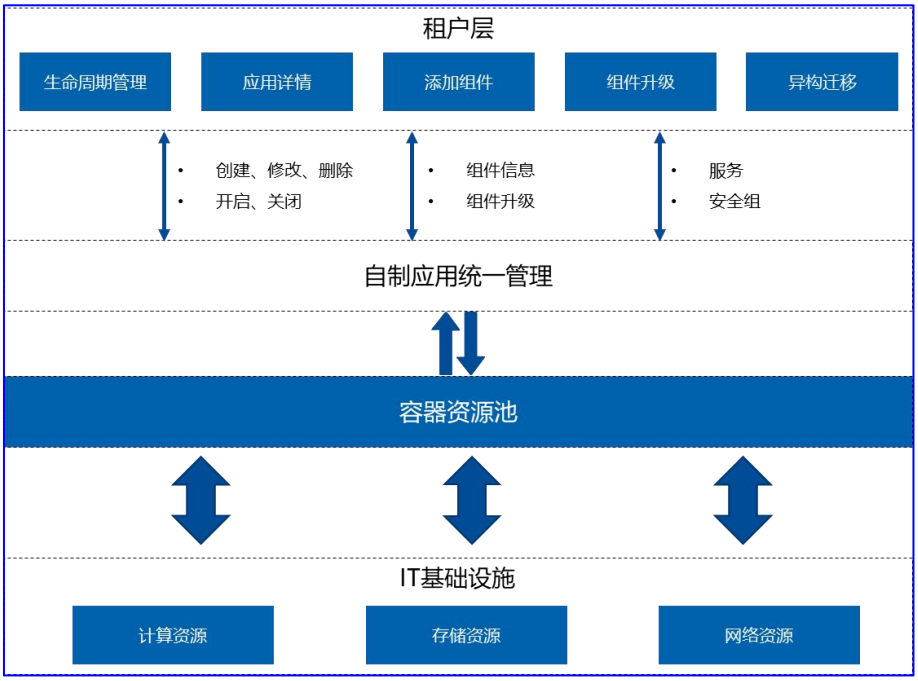
4.5.1.1 自制应用管理

4.5.1.1.1 功能描述

自制应用是由一个或多个工作负载（**workload**）、服务（**service**）、存储卷（**pvc**）等 **Kubernetes** 原生资源构成的整体应用，通过自制应用逻辑上管理一个完整的业务系统，例如 **OA**、**ERP** 等。

自制应用提供容器应用的全生命周期管理，支撑应用的创建、状态变更、应用组件部署、服务或网络策略管理等一系列功能，用户可根据实际需求通过页面操作应用组件的部署、整合多个应用组件间的关联关系。自制应用提供更加易用

的 **Kubernetes** 工作负载（**Deployment**、**Statefulset** 等）的快速部署和存储卷、环境变量、健康检查等高级功能，同时结合系统多层级、多维度的角色设置实现应用的分层级、分权限管理。



自制应用功能架构

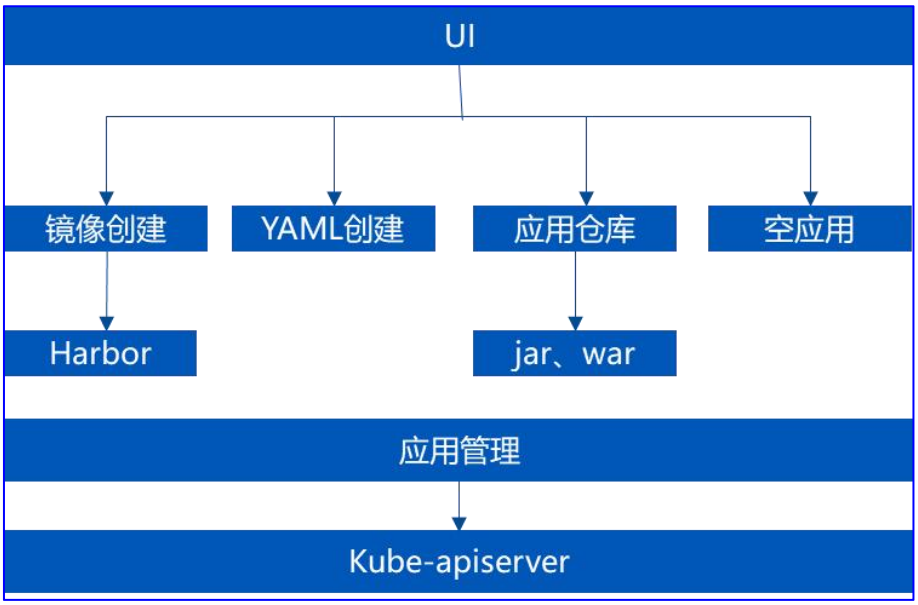
应用管理服务功能包括：

- 应用生命周期管理：创建、修改、开启、关闭、删除
- 应用信息全方位展示：基本信息、组件信息、应用拓扑
- 应用网络管理：服务、网络策略
- 组件变更
- 应用异构迁移

4.5.1.1.2 技术原理

设计 **Kubernetes** 自定义资源（**CR**）**Release** 表示自制应用，进行符合容器部署最佳实践的引导性部署，并对常用参数给出默认提示值，引导不熟悉

Kubernetes 底层的用户能够快速完成容器化应用部署，创建完毕自制应用后可以通过添加组件继续向该应用添加属于该应用的其它组件。



- 创建空应用用于创建不包含组件的应用，用于后续添加组件。
- 通过镜像创建组件通过选择客户上传的容器镜像，并进行对应资源规格、存储卷、容器端口等设置，并完成该组件对应访问服务设置，最终应用管理组件调用容器资源池 **kube-apiserver** 完成基于镜像的自制应用创建；
- 通过 **YAML** 创建，直接调用 **kube-apiserver** 完成资源创建；
- 通过应用仓库创建，可以选择用户上传的 **jar** 包、或 **war** 包，通过 **initContainer** 技术将应用宝拷贝到内置的 **tomcat** 或 **openjdk** 镜像的对应路劲，通过内置 **jar** 命令启动对应的组件。

4.5.1.1.3 约束

容器应用跨架构迁移需要 **x86** 和 **arm** 混合部署的容器资源池中进行迁移。

4.5.1.2 Helm 应用管理

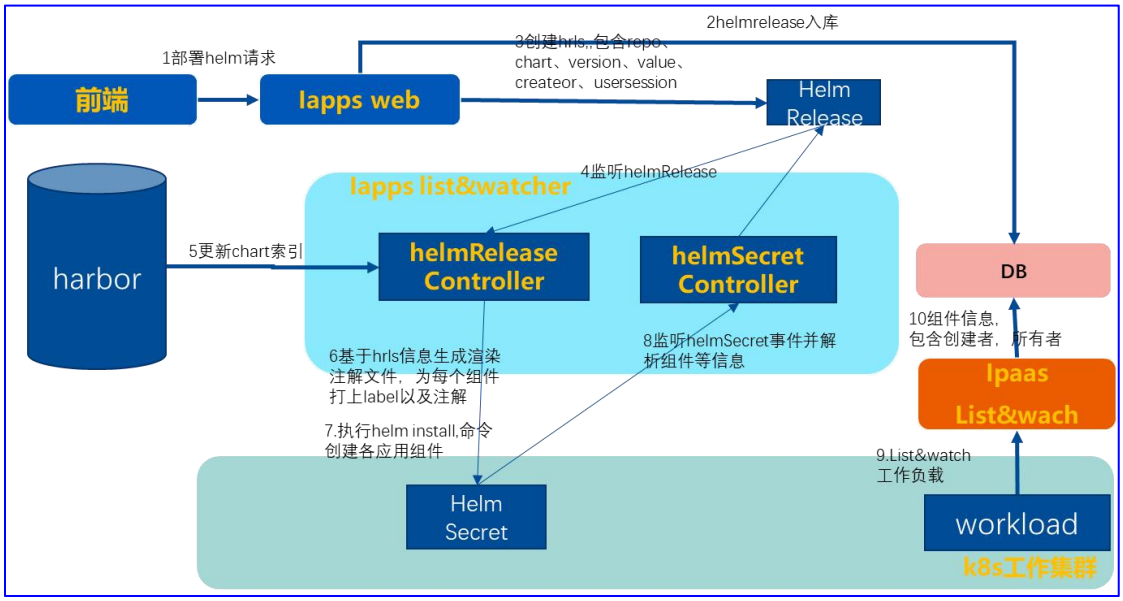
4.5.1.2.1 功能描述

Helm 应用管理提供基于标准 **helm Chart** 模板进行应用的创建、升级、改配、详情查看、监控等标准 **Helm** 应用的生命周期管理。该功能主要面向如下场景：

场景一：熟悉容器的客户/**ISV** 其业务系统的交付部署方式基于 **K8S**，则可直接将 **chart** 模板上传到容器服务/应用商店，通过本功能进行一键化部署和管理。

场景二：面向云原生场景，客户一些通用能力组件，例如：中间件、数据库、技术中台组件等，可基于镜像部署的基础上，逐步形成 **Helm** 模板，上传至应用商店，后续新项目或者搭建新的环境可快速部署。

4.5.1.2.2 技术原理



- 定义 **HelmRlease** 资源用来记录 **Helm** 应用创建的应用模板信息以及用户信息并记录 **helm** 应用的部署状态。
- 通过控制器 **helmReleaseController** 实现了对 **Helm** 应用的异步创建，解决同步创建 **helm** 应用超时问题。

- 通过注解 **meta.helm.sh/release-name** 注解标识组件所属应用，通过 **app.kubernetes.io/managed-by: Helm**（**helm** 自动生成）标识过该组件通过 **helm** 创建。
- 通过 **helmSecretController** 对 **helm** 生成的特殊类型 **secret** 进行监听，将 **helm** 创建的应用与底层 **cmp** 定义的 **release** 资源进行关联，实现纳管集群中原有 **helm** 应用自动同步到 **cmp** 层的功能。

4.5.1.2.3 约束

- **Helm Chart** 模板创建的应用详情查看等功能，基于标准 **chart** 模板部署出的应用页面展示正常，不报错（如遇到当前 **UI** 不支持的资源类型、属性类型或者和目前产品有冲突的属性不做展示）。
- **Helm Chart** 模板创建的应用支持更新能够被 **UI** 管理的资源和属性（通过镜像创建的应用），其它多余资源和属性保持不变。

4.5.1.3 一云多芯应用

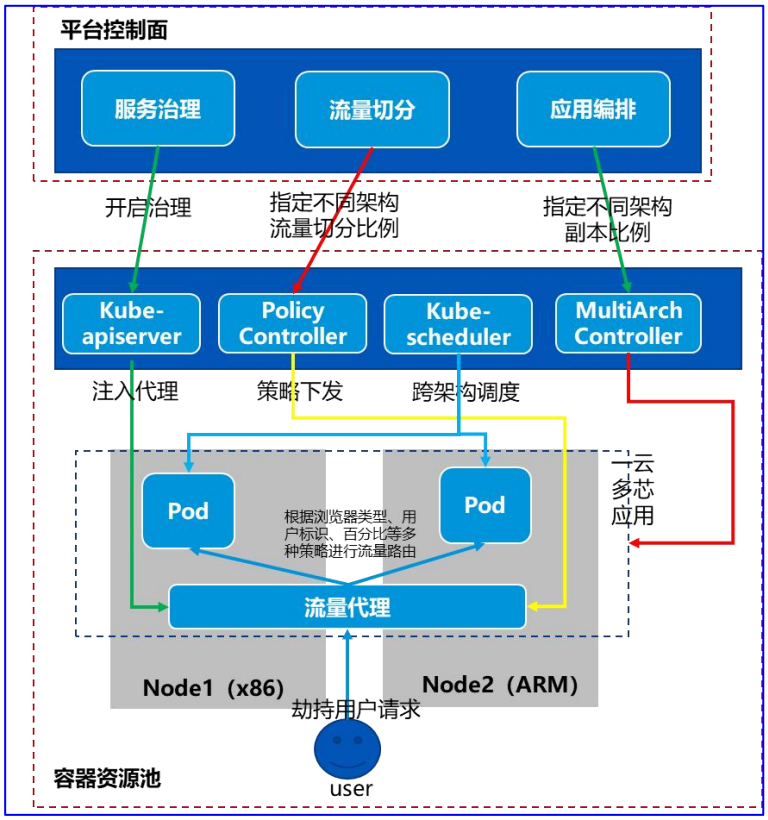
4.5.1.3.1 功能描述

一云多芯应用是指同一应用的不同实例分别运行在不同 **CPU** 架构的主机上，并实现统一的服务访问，支持跨架构统一编排、一键部署、低成本自由切换，支持按架构自定义副本数和流量切分比例的应用。一云多芯应用功能特性如下：

- 跨架构编排：统一设置一云多芯应用属性，一次完成同一应用在不同架构实例的部署，支持应用跨两种及以上架构进行部署。
- 跨架构差异化部署：支持设置不同架构的容器镜像、实例数、不同架构按照自定义流量比例和请求内容的流量切分策略。

- 滚动升级：支持设置各架构最小可用实力数和每次更新的实例数，通过滚动的方式完成应用升级。
- 故障转移：跨架构故障转移，指节点故障后该节点运行的容器可跨架构进行重建；等算力节点优先调度，指节点故障后该节点运行的容器优先调度到规格算力相等的节点上。
- 一云多芯应用异构迁移：基于主机规格算力测算应用算力，可展示一云多芯应用的总算力和各实例算力。异构迁移支持等实力数迁移和等规格算力迁移，等实力数迁移可指定跨架构迁移的实力数进行固定实力数迁移；等规格算力迁移指从源架构迁移到目标架构时保障应用总算力一致，高性能节点迁移至低性节点时可动态调整副本数保障应用算力一致。

4.5.1.3.2 技术原理



1、容器应用编排调度基于自定义 CR 和多芯控制器 MultiArchController

实现一云多芯应用跨架构编排和生命周期管理

2、基于 **scheduler** 插件扩展机制实现按 **CPU** 架构自定义副本数调度，**scheduler** 扩展插件监听 **Pod** 创建事件，自定义 **CPU** 架构按比例调度算法实现 **Pod** 到指定架构主机的有向调度，并对 **Pod** 添加架构标识标签。

3、基于服务治理技术，根据 **Pod** 的架构标识标签，实现不同架构应用副本东西向流量按比例分发，同时结合 **ingress** 实现入口(南北向)流量按比例分发。

4、结合动态控制不同架构间副本和流量切分比例，实现一云多芯应用的跨架构自动快速切换。

4.5.1.3.3 约束

一云多芯应用只能创建在计算节点由不同 **CPU** 架构的主机组成的容器资源池。

4.5.1.4 应用商店

4.5.1.4.1 功能描述

应用商店提供包括开源软件、行业软件、基础数据库及中间件在内的应用模板，形成生态汇聚，实现多组件复杂应用的快速、一致性下发、差异化配置部署，推动业务系统快速、复制化上云。应用商店提供了一组基于 **Helm** 的应用模版和可视化的快捷应用部署功能，用户可以根据实际需求通过页面操作，配置简单的参数，实现快速部署应用。

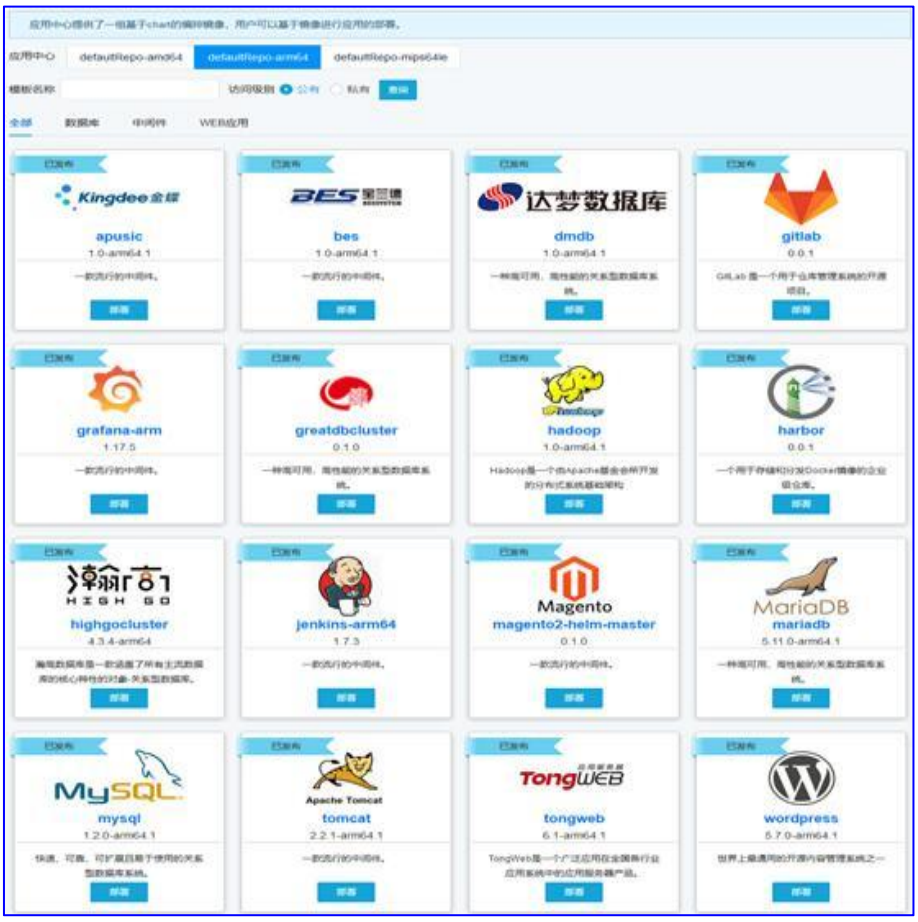
应用商店功能包括：

- **Chart** 模板上传：支持标准 **chart** 模板上传、更新、删除，可快速上传开源、**ISV**、用户的标准 **chart** 模板和管理。

- 便捷式部署应用：通过已编排好的应用模版，配置简单的参数，即可快速部署应用。减少用户在部署相似应用时的繁琐操作，满足用户对重复、快速部署的需求。
- 应用模版版本管理：应用模版下架、新增模版分类、删除模版分类、基本信息修改。
- 模版全方位信息展示：基本信息、应用信息。
- 直观的应用模版管理：以图形化形式展示系统已上架的应用模版，支持自定义分组、为应用设置可见范围；支持模版分类和模糊查询，支持模版下架功能，对于已下架的功能，用户可在应用蓝图中编辑修改模版内容。

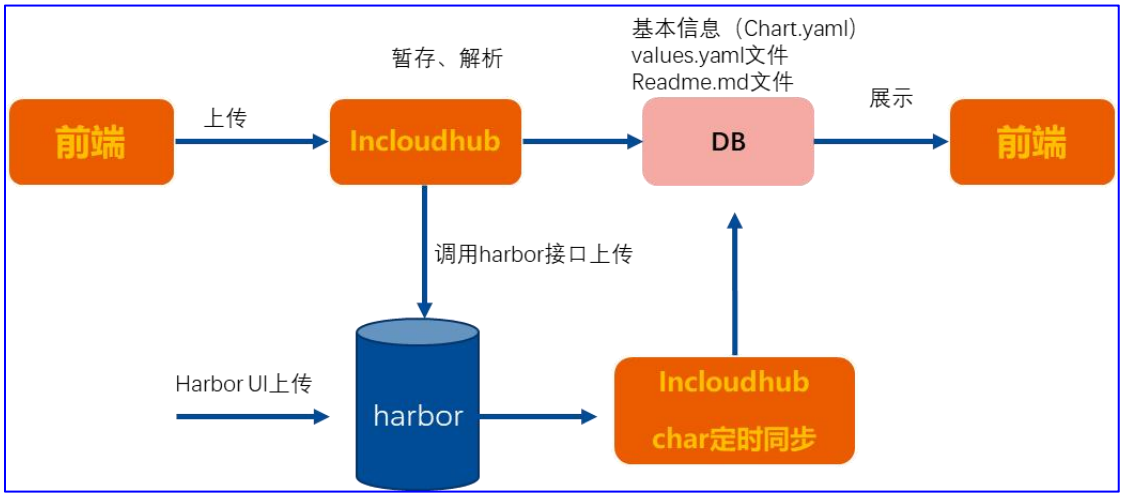
4.5.1.4.2 技术原理

应用商店



平台提供大量常用开源软件、国产应用软件试用版、部分商业软件试用版，用户可快速获取服务，实现应用一键部署，差异化应用可通过参数快速配置，形成复制化高效部署，减少人工部署的差异化，屏蔽底层复杂的资源管理和分配，实现应用快速部署和上层服务的统一管理和底层物理资源的自动调度。

Chart 模板上传



- 用户上传 **chart** 文件至 **incloudhub** 暂存。
- **incloudhub** 调用 **helm lint** 命令校验文件格式是否符合 **helm** 标准。
- 根据文件生成对应的 **MD5** 码。
- 解析 **Chart.yaml**、**Values.yaml**、**Readme.MD** 文件。
- 通过 **chart.yaml** 文件解析获取模板的基本信息、名称、版本等。
- 将基本信息、**md5** 值、**Values.yaml**、**Readme.MD** 文件保存到数据库中。
- 将文件传到 **harbor** 仓库中。

4.5.1.4.3 约束

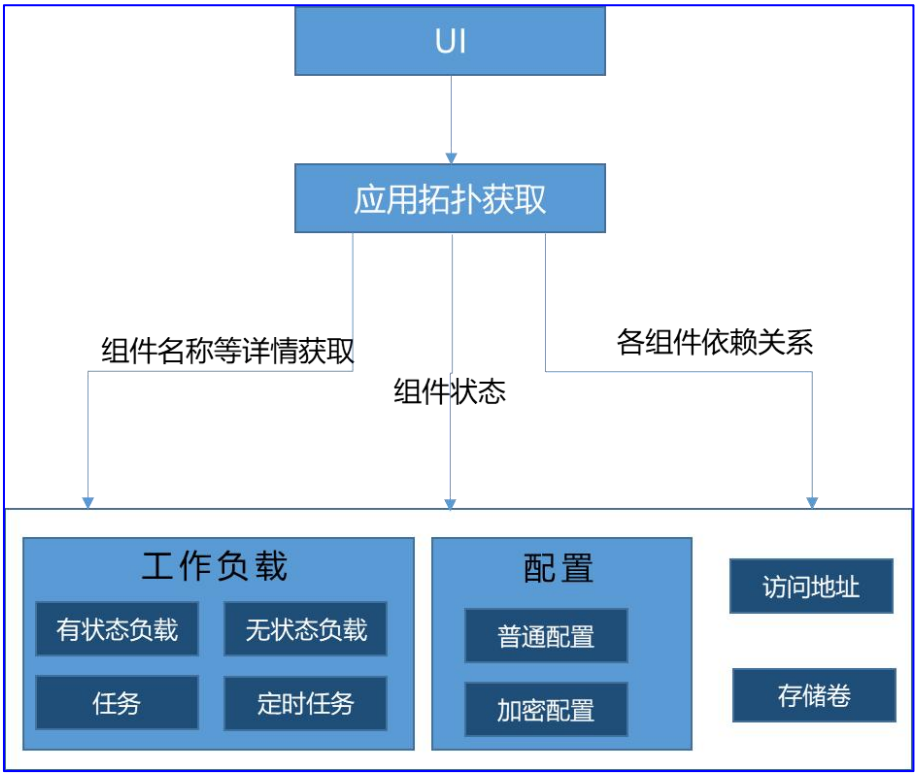
无。

4.5.1.5 应用拓扑

4.5.1.5.1 功能描述

以图形化形式直观展示应用组件间复杂的关联关系，支持可拖拽动态展示，支持组件信息的快速查看。拓扑关系中可以展示应用内的各组件的名称、类型、状态以及组件间的依赖关系。

4.5.1.5.2 技术原理



- **UI**：基于 **echart** 的力导向图实现，自动根据节点关系进行合理的布局。
- 应用拓扑服务：负责获取应用下所有组件、配置、网络、存储等资源，获取各组件依赖关系，组装报文输出。
- 资源数据存储：容器资源池平台存储各组件详情、组件依赖关系、配置、网络组件、存储卷等资源的属性。

4.5.1.5.3 约束

无。

4.5.2 工作负载

4.5.2.1 工作负载生命周期管理

4.5.2.1.1 功能描述

工作负载是 **Kubernetes** 的抽象资源，包括无状态负载、有状态负载、任务、定时任务等类型。工作负载管理提供多种工作负载的管理，支撑工作负载的状态展示及变更、升级及回滚、网络设置、调度策略设置、日志及控制台、容器组及容器信息查看等功能。

工作负载管理整合容器的状态管理、升级与回滚、访问控制、调度策略、资源监控、日志、控制台等功能，并能够支持可视化设置副本数、自动弹性伸缩、健康检查、镜像拉取策略等特性，能够最大程度的实现应用软件的快速部署与升级、智能监控、自动运维等一系列操作，保障应用软件服务的可持续性、高可用性，提高资源利用率，减少用户在运维管理上的人力与时间成本。

工作负载服务功能包括：

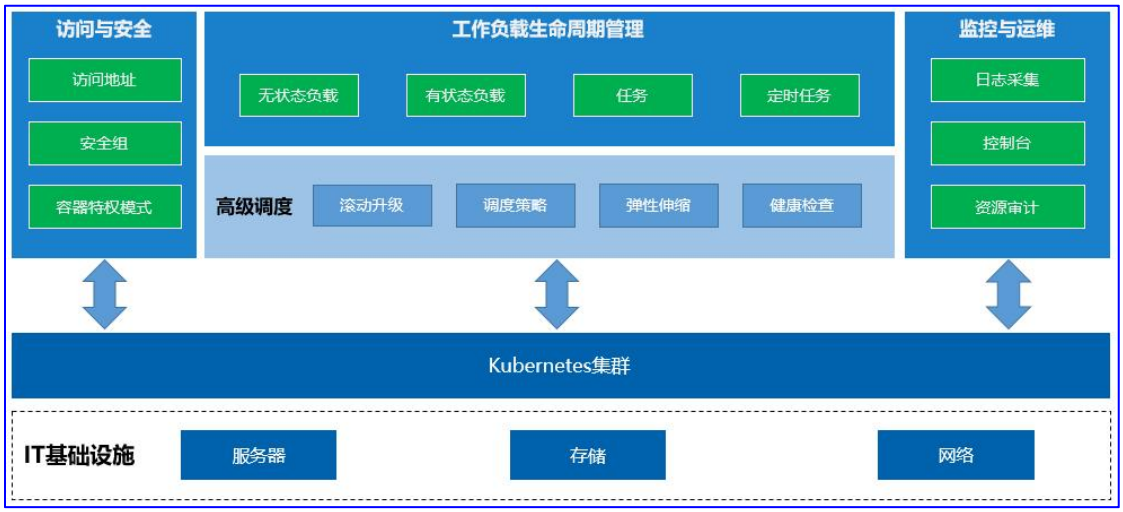
- 无状态负载管理：**Kubernetes** 无状态负载抽象资源管理，支持修改、升级、回滚、开启、关闭、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、网络设置、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计、资源使用历史曲线等功能。
- 有状态负载管理：**Kubernetes** 有状态负载抽象资源管理，支持修改、升级、回滚、开启、关闭、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、网络设置、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计、资源使用历史曲线等功能。

- 任务：**Kubernetes** 任务抽象资源管理，支持修改、存储卷配置、重新执行、删除、运行状态查看、容器组及容器管理、事件查看、环境变量查看、网络策略、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计等功能。
- 定时任务：**Kubernetes** 定时任务抽象资源管理，支持修改、存储卷配置、启用、停用、删除、运行状态查看、任务记录、事件查看、环境变量查看、网络策略、调度策略管理、控制台日志及采集日志展示、容器控制台、资源审计等功能。
- 自动水平弹性伸缩：云平台提供工作负载水平弹性伸缩能力，能够在工作负载的 **CPU/**内存使用量达到设置好的阈值时自动增加副本数，在使用量过低时自动减少副本数，帮助应用自动应对波峰波谷，保证工作负载高可用的前提下提高资源使用率。
- 多维度健康检查：云平台提供了一套完整的工作负载健康检查及重启机制，保障服务的持续运行，包括进程级的健康检查及业务级健康检查，能够自动定期检查容器进程、业务请求是否正常，并在异常情况下重新启动容器。
- 灵活的调度策略：支持客户根据自身的实际需求为工作负载设置灵活的节点调度策略，包括指定节点、按照节点标签选择、根据与其他工作负载亲和关系选择等，云平台根据设置的策略将容器调度到符合要求的节点上。
- 可控制的特权模式容器：针对某些特殊场景，例如监控类、安全类应用，云平台允许创建特权模式的工作负载。在特权模式下，**Docker** 容器将

完全拥有内核的管理权限。但与此同时也会有安全隐患，因此管理员在使用特权模式时，一定要对应用有足够的了解。

- 零停服的升级策略：云平台支持工作负载滚动更新的升级策略，在一次完整的发布过程中，合理地分成多个批次，每次发布一个批次，成功后再发布下一个批次，直到把所有旧的批次全部更新到新版本。在整个滚动过程期间，保证始终有可用的副本在运行，从而平滑的发布新版本，实现零停服**(without an outage)**、用户零感知。
- 全面的日志采集及展示：云平台支持容器控制台日志及自定义文件日志的采集，并通过直观、便捷的操作支撑日志的展示和查询，帮助运维人员快速查找定位应用异常问题。
- 挂载临时卷：云平台提供了为工作负载挂载临时卷的功能，支持硬盘和内存两种方式的临时卷，临时卷生命周期与 **pod** 生命周期相同，**pod** 销毁后临时卷也被删除，挂载时若选择内存作为临时卷类型，则支持临时卷可使用的内存大小。

4.5.2.1.2 技术原理



工作负载管理服务提供基于 **Kubernetes** 集群的抽象资源管理能力，支撑工作负载的生命周期管理、网络策略、监控与运维，并具备滚动升级、调度策略设置、弹性伸缩、健康检查等自动运维能力。

4.5.2.1.3 约束

无。

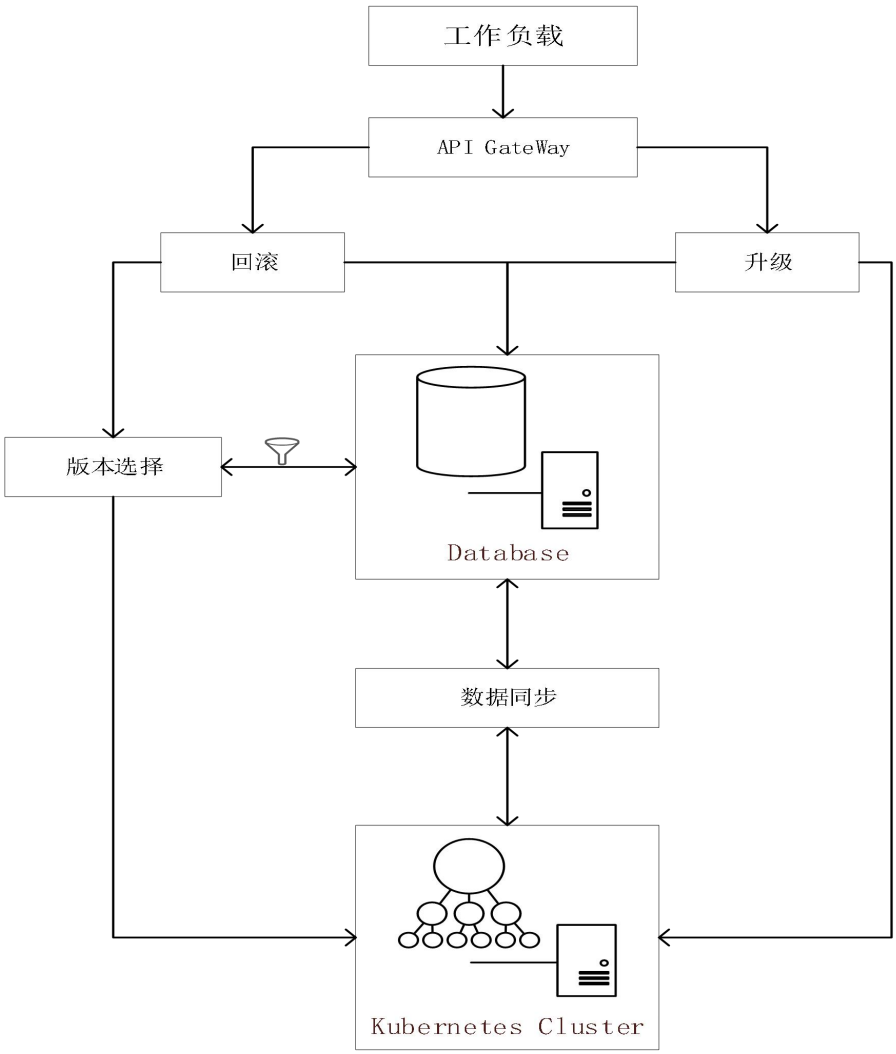
4.5.2.2 升级与回滚

4.5.2.2.1 功能描述

升级功能支持无状态负载、有状态负载、任务、定时任务的升级，升级内容主要包含容器属性（镜像、**CPU/**内存限制、容器端口、启动命令、参数、环境变量等）、存储卷配置、健康检查配置和调度策略。升级时需要记录版本升级记录。

回滚支持展示无状态负载、有状态负载、任务、定时任务的升级记录，支持用户选择某个版本进行回滚。

4.5.2.2.2 技术原理



- 通过 **Kubernetes** 客户端调用 **kube-apiserver** 获取工作负载底层各属性信息。
- 升级镜像，调用镜像仓库功能获取各镜像与版本信息。
- 升级存储卷配置，调用 **kube-apiserver** 接口获取目前已挂载存储卷信息，调用存储管理功能获取目前没有被挂载的存储卷。
- 通过 **Kubernetes** 客户端调用 **kube-apiserver** 修改 **Kubernetes** 集群内工作负载属性与配置。
- **Kubernetes** 按照既定升级策略进行版本的升级与回滚。

4.5.2.2.3 约束

无。

4.5.2.3 容器文件管理

4.5.2.3.1 功能描述

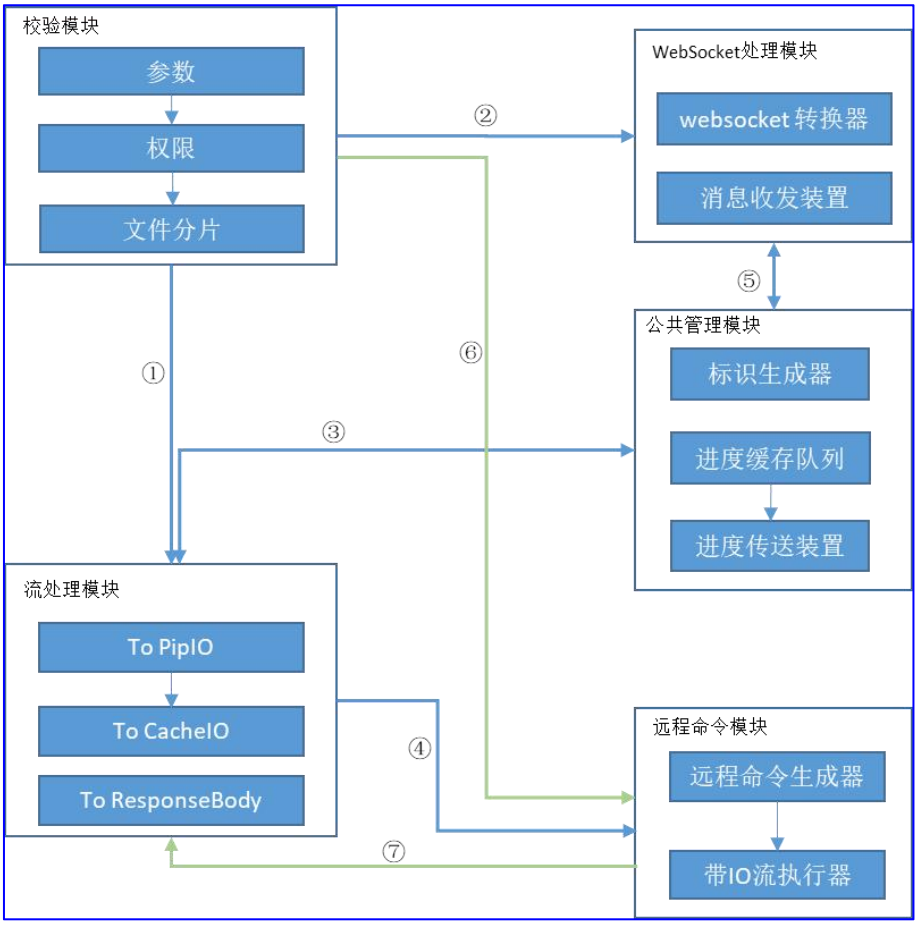
容器文件管理支持在浏览器查看容器内文件系统，可以方便的浏览容器内的目录和文件。支持新建文件夹、上传、下载和删除文件。主要面向以下场景使用：

1、开发测试场景中，开发人家在定位问题时，需要查看日志文件进行问题诊断、分析。

2、开发测试场景中，开发人员或用户需要在容器内部更新现有文件或者添加新的文件用于业务调试。

3、对于数据分析/机器学习的用户，其应用容器在运行一些数据处理程序或者训练模型，在处理完成后，用户可能需要下载生成的文件或训练完成的模型文件。

4.5.2.3.2 技术原理



- 通过参数校验后，将参数转发到流处理模块；
- 通过参数校验后，将参数转发到流处理模块；
- 从公共模块中获取唯一的 **key**，作为传输缓存标识，并初始化缓存计数器；
- **IO** 流处理模块将 **body** 里的分片文件二进制流，转成 **PipIO**，再封装成计数的 **CacheIO**，转发到远程命令模块；执行器则会带着这些 **IO** 流，调用 **apiserver** 进行文件传输；
- **websocket** 模块从公共管理模块中，获取 **key** 值，并读取缓存的进度信息，然后推送给 **UI**；

- 通过参数校验后，将参数转发到远程命令模块，在命令生成器中，拼接好传入的命令参数，并通过执行器调用 **apiserver** 执行远程命令；
- 将执行命令的标准输出流，重定向到请求响应，完成下载、查询等远端数据操作指令。

4.5.2.3.3 约束

无。

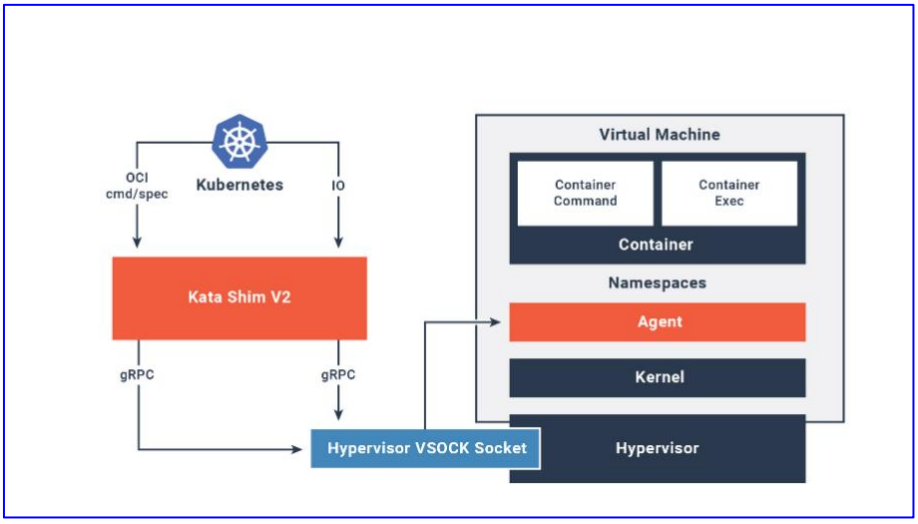
4.5.2.4 安全容器

4.5.2.4.1 功能描述

安全容器通过轻量型虚拟机技术构建一个安全的容器运行时，将普通容器的两层隔离（**namespace** 和 **Cgroup** 的限制）扩展到三层隔离，可以有效防止恶意租户对宿主机内核的直接攻击，并大幅减少对机器上其他业务产生的风险。

安全容器支持无状态负载、有状态负载以轻量级虚拟机方式运行实例，支持工作负载进行容器基本配置的改配、卷挂/卸载、健康检查以及除特权模式外的安全参数设置。

4.5.2.4.2 技术原理



- **kubelet** 通过接口调用 **kata-shim-v2** 创建 **pod/container**。
- **kata-shim-v2** 为每个 **pod/container** 创建一个 **QEMU/KVM** 虚拟机。
- 虚拟机中启动一个极简的 **kernel**，最终运行一个用户态程序 **kata-agent**。**kata-agent** 通过 **VSOCK** 向 **host** 上暴露一个 **gRPC** 接口，**kata-shim-v2** 通过这个接口连接上 **kata-agent**，进而实现对虚拟机内部的管理。这个 **gRPC** 接口既用于传输管理命令，也用于传输 **stdin**，**stdout**，**stderr** 等数据流。
- **kata-agent** 在虚拟机中通过 **libcontainer** 管理 **container**。启动 **container** 所需要的 **OCI bundle** 可以通过 **block device** 或者 **mount point** 的形式从 **host** 上挂上去。

4.5.2.4.3 约束

1. 安全容器不支持与 **Docker** 运行时节点混合部署。

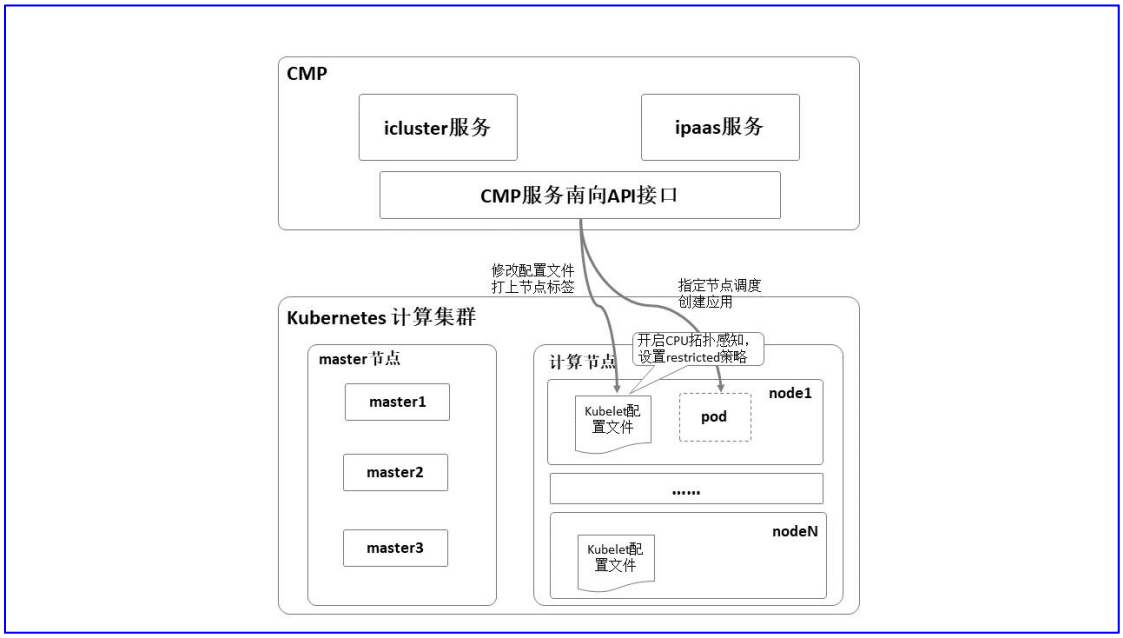
2. 受限于轻量虚拟机，容器安全暂不支持使用加速设备，例如 **GPU**、**FPGA**、大页内存；不支持安全容器内的日志采集；不支持开启特权模式。

4.5.2.5 NUMA 拓扑感知调度

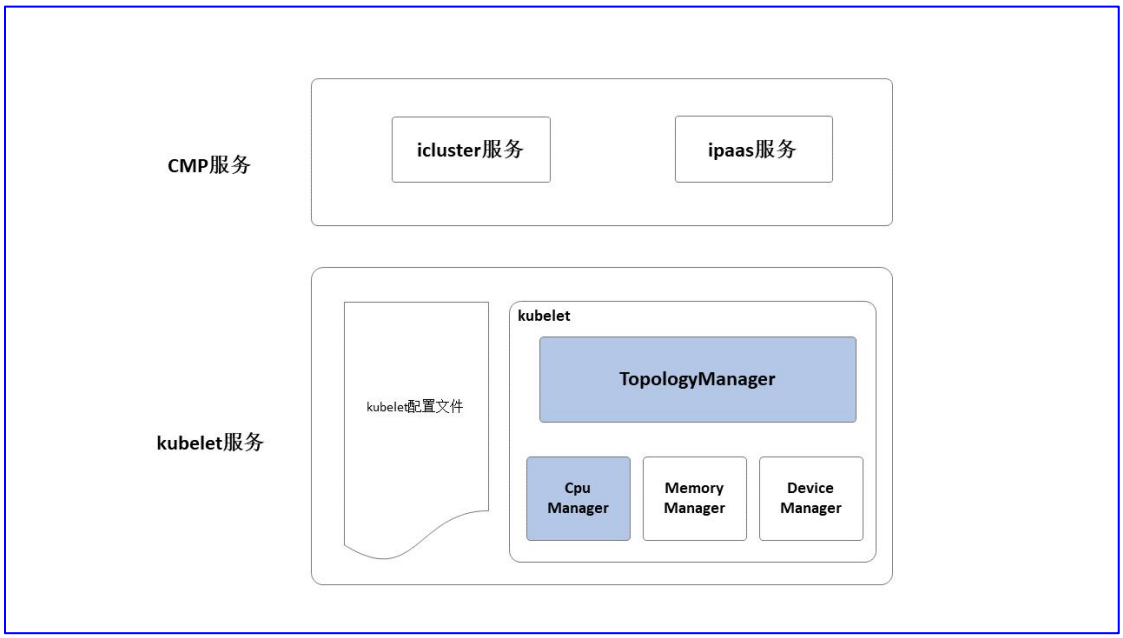
4.5.2.5.1 功能描述

支持设置和展示节点 NUMA 策略，支持配置的 NUMA 亲和策略包括：尽量 NUMA 亲和（best-effort）、必须 NUMA 亲和（restricted）、单 NUMA 亲和（single-numa-node）。支持创建工作负载和修改工作负载时选择 NUMA 亲和并配置其对应的亲和策略。

4.5.2.5.2 技术原理

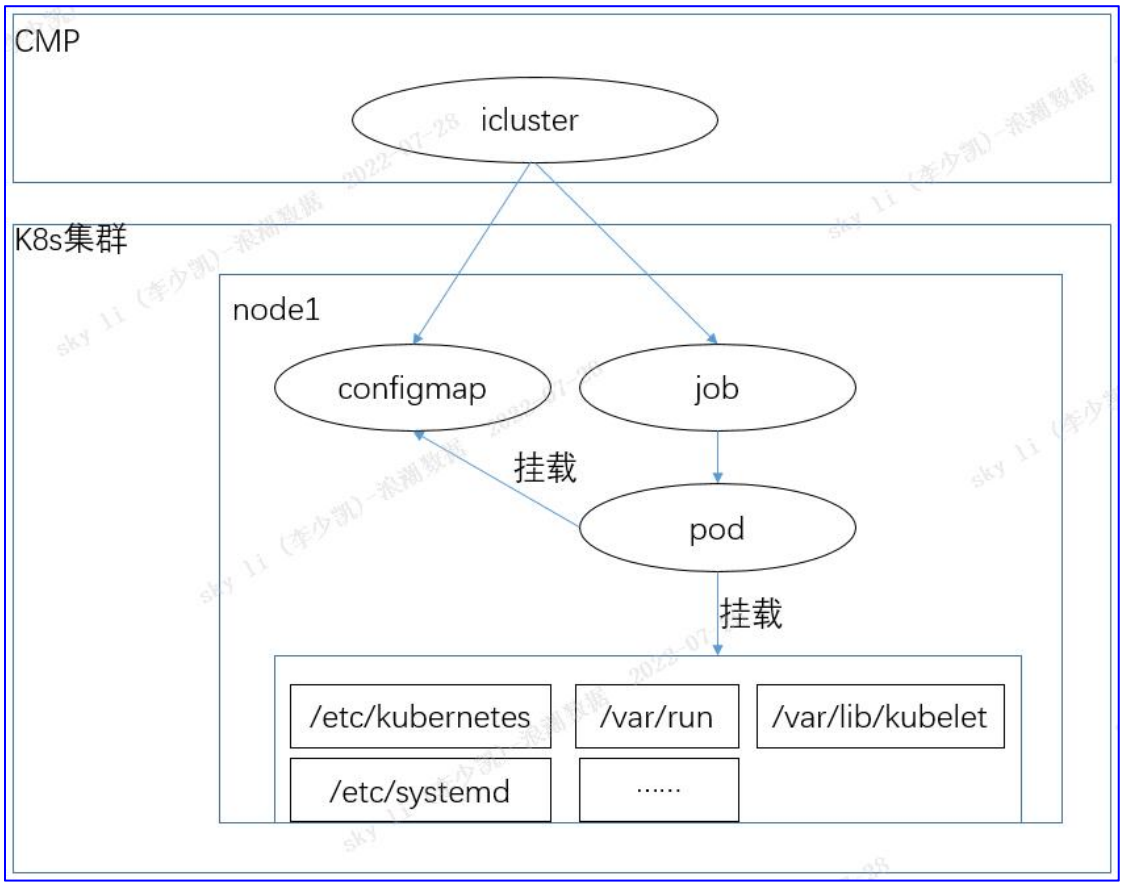


容器集群的 NUMA 拓扑感知功能在 CMP 层主要涉及两个服务 icluster 服务和 ipaas 服务；icluster 服务用来开启关闭节点的 NUMA 拓扑感知以及重启节点 kubelet 服务，并为开启 NUMA 拓扑感知的节点打上标签；ipaas 服务用来指定开启 NUMA 拓扑感知的节点，并在指定节点上创建应用。



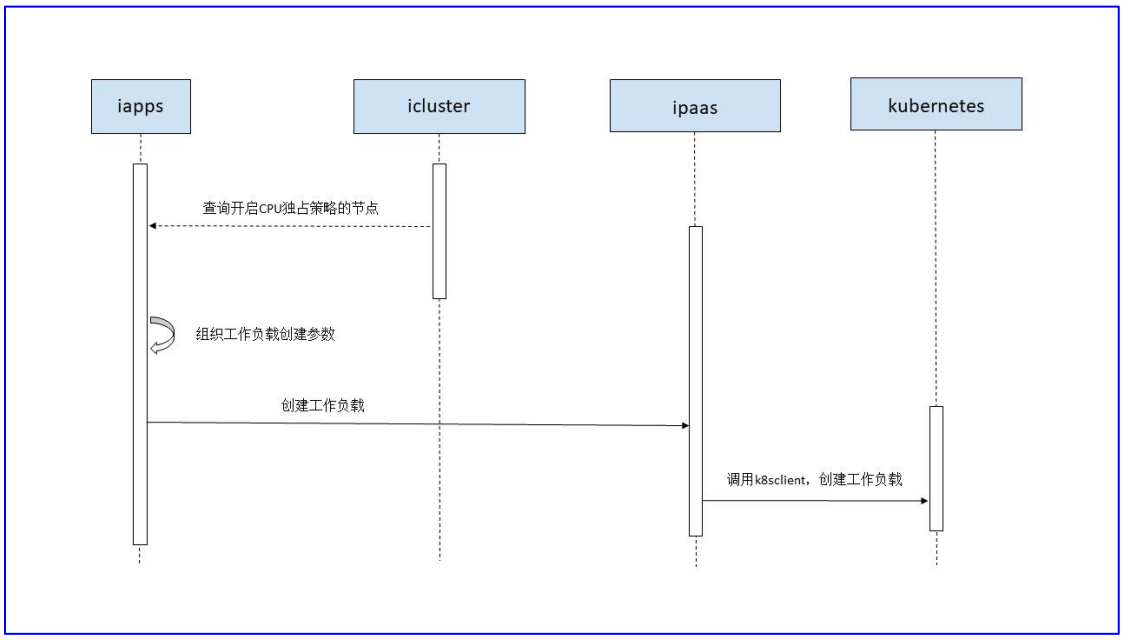
涉及的 **CMP** 服务 **icluster** 和 **ipaas** 服务通过 **restful** 服务设置 **NUMA** 拓扑感知的开启、关闭以及应用服务的创建；对于 **kubelet** 服务底层实现 **NUMA** 拓扑感知主要涉及到 **TopologyManager**、**Cpu Manager**、**Memory Manager**、**DeviceManager** 等模块；**Cpu Manager**、**Memory Manager**、**DeviceManager** 实现 **TopologyManager** 提供的接口，负责提供 **NUMA** 拓扑；**TopologyManager** 将 **Cpu Manager**、**Memory Manager**、**Device Manager** 提供的 **NUMA** 拓扑进行选择和组合，在创建容器时最终实现应用对 **NUMA** 拓扑的亲和。

节点 **NUMA** 亲和设置：



icluster 服务会在目标节点启动 **configmap** 与 **job**，**job** 会启动 **pod** 并挂载 **configmap** 与目标节点的部分目录，以达到备份、修改 **kubelet** 配置文件以及重启 **kubelet** 服务的主要目的。

创建工作负载时支持选择 **NUMA** 亲和策略：



创建应用并开启 **CPU NUMA** 亲和、涉及的模块主要有 **icluster** 和 **ipaas**，具体的流程如上图所示。

如果创建工作负载所在集群的节点开启 **NUMA** 拓扑亲和，则通过节点列表过滤出开启节点 **NUMA** 拓扑亲和的节点，设置节点亲和来达到工作负载尽可能做到 **NUMA** 亲和的效果。

该功能重点主要体现在，基于节点列表功能，可以通过 **NUMA** 亲和策略（尽量亲和、必须亲和、必须单 **NUMA** 亲和）、或者标签（**NUMA** 亲和标签、**CPU** 亲和开启标签、内存亲和开启标签），过滤出业务想要 **NUMA** 亲和的节点。

4.5.2.5.3 约束

- 仅 **CPU** 规格为整数核且 **CPU** 内存资源请求值（**request**）等于限制值（**limit**），即 **QoS** 为 **Guaranteed** 的 **Pod** 支持拓扑感知。
- **NUMA** 拓扑亲和调度只支持 **Numa node** 小于 **8** 的宿主机节点。
- 被调度到开启 **NUMA** 拓扑感知的节点后， **Pod** 可能会出现启动失败（**TopologyAffinityError**、 **UnexpectedAdmissionError**）的情况，

kubelet 会一直在当前计算节点重复创建 **pod** 直至节点崩溃。

4.5.3 GPU 管理和调度

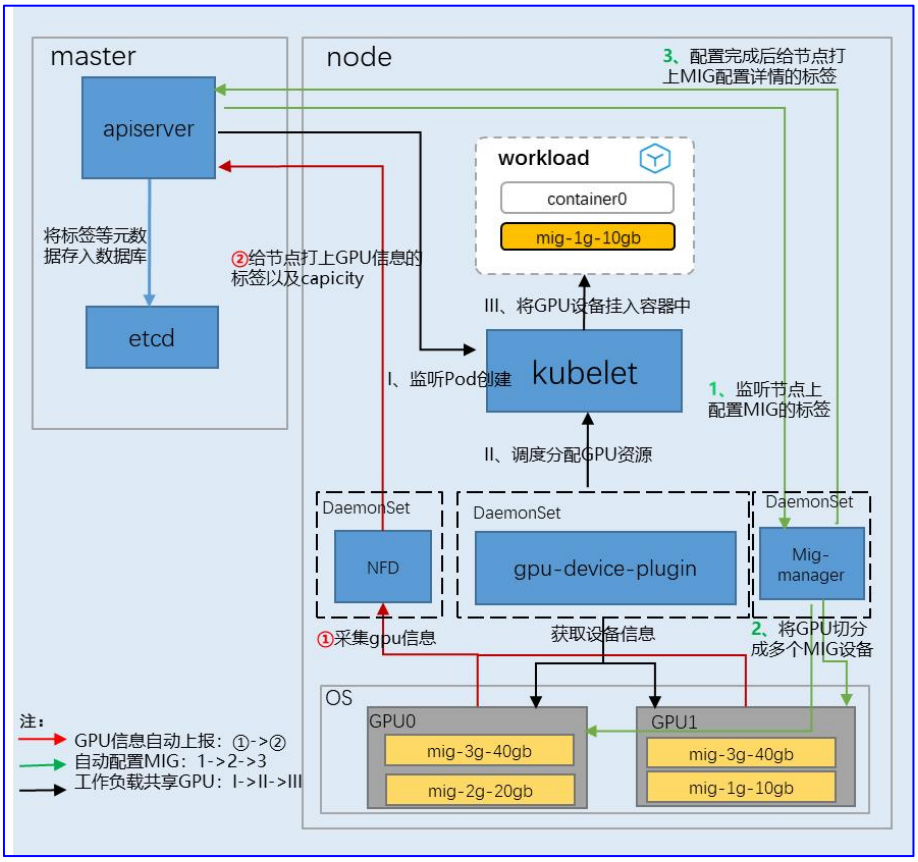
4.5.3.1 功能描述

容器服务支持精细化的 **GPU** 管理和调度，可以自动化识别 **GPU** 厂商型号、设置精细化的 **GPU** 配合，支持 **GPU** 按卡独占、按照显存共享隔离、按 **MIG** 共享隔离等多种 **GPU** 使用方式。

- 灵活的 **GPU** 配额管理模型，支持 **GPU** 独享、**GPU MIG** 共享、**GPU** 共享等多种 **GPU** 使用模式。
- 基于 **MIG** 技术实现 **GPU** 共享，以多实例的形式提供给 **POD** 使用，每个实例都有独立于各自的显存、缓存和计算核心。
- 基于 **CUDA** 劫持的 **GPU** 资源共享技术方案实现 **GPU** 资源的共享，实现容器应用内部实际运行时 **GPU** 核心和显存的限制，同时对容器应用无感知。

4.5.3.2 技术原理

4.5.3.2.1 基于 MIG 技术的 GPU 共享



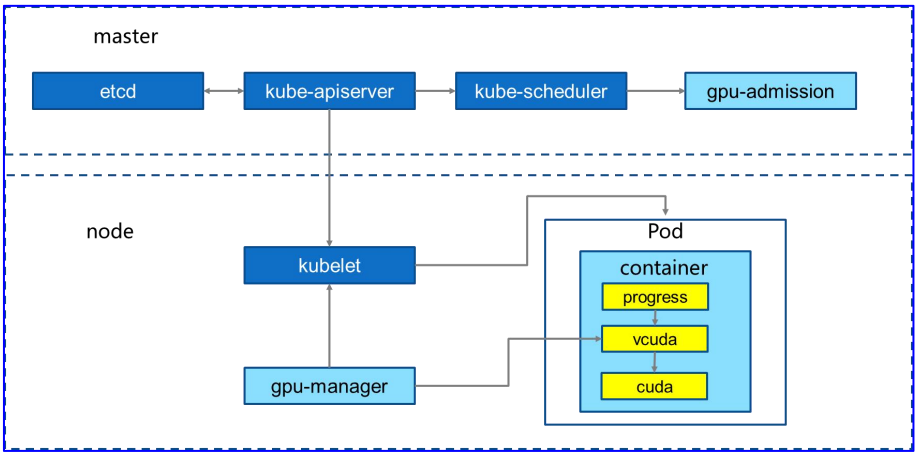
基于 MIG 技术的 GPU 共享主要包括以下几部分设计：

1）基于 NFD 的 GPU 信息采集：GPU 集群部署 node-feature-discovery，通过查询设备文件等方式，定时获取节点上 GPU 厂商、型号等信息。

2）基于 mig-manager 的 MIG 自动配置：基于 mig-manager 控制器，通过监听节点标签更新事件，自动解析 MIG 配置策略， 自动将 GPU 切分为多个 MIG 设备

3）基于 devicePlugin 的设备扩展：通过 devicePlugin 机制，发现节点 GPU MIG 设备，并上报本节点 GPU 容量、可分配量等信息；创建容器时，devicePlugin 通过设备挂载的方式将 MIG 设备自动映射到容器中。

4.5.3.2.2 基于 CUDA 劫持的 GPU 共享



核心原理是开发 **libcuda** 库封装，通过劫持 **cuda driver API** 的调用来做资源限制。**DevicePlugin** 在给 **POD** 挂载 **GPU** 设备和驱动文件时将改写后的 **cuda so** 文件挂载到业务 **POD** 的**/usr/local/nvidia** 目录。

基于 **CUDA** 劫持的 **GPU** 资源共享方案的技术架构如上图所示，主要包括三个核心部分：

1）vcuda：**vcuda** 是对 **nvidia-ml** 和 **libcuda** 库的封装库。**vcuda** 通过劫持容器内用户程序的 **cuda** 调用限制当前容器内的进程对 **GPU** 核心和显存的使用数量；

2）gpu-manager：是标准的 **K8S device plugin**，实现了 **GPU** 资源上报和分配等功能，将 **GPU** 的算力注册为 **inspur.com/vcuda-core**，将显存注册为 **inspur.com/vcuda-memory**。**gpu-manager** 支持共享和独占两种模式，当负载里 **inspur.com/vcuda-core request** 值在 **0~100** 情况下，采用共享模式调度，优先将碎片资源集中到一张卡上，当负载里的 **inspur.com/vcuda-core request** 为 **100** 的倍数时，采用独占模式。

3）gpu-admission：是一个 **K8S Scheduler extender**，实现了 **Scheduler** 的

predicates 接口，**kube-scheduler** 在调度 **inspur.com/vcuda-core** 资源请求的 **Pod** 时，**predicates** 阶段会调用 **gpu-admission** 的 **predicates** 接口对节点进行过滤和绑定，同时 **gpu-admission** 提供了 **GPU** 资源池调度功能，解决不同类型的 **GPU** 在 **namespace** 下的配额问题。

4.5.4 容器网络

容器网络管理服务提供应用或工作负载的访问地址创建、修改、删除、变更内外部访问、设置外部访问方式、配置域名信息等一系列功能，提供应用或工作负载的安全组的创建、配置安全组规则、关联或移除组件等一系列功能。

4.5.4.1 服务

4.5.4.1.1 功能描述

服务是应用组件可以对外访问或提供对外访问以及访问限制等一系列属性的集合，在实际使用场景中对应应用或工作负载的访问地址与安全组功能。

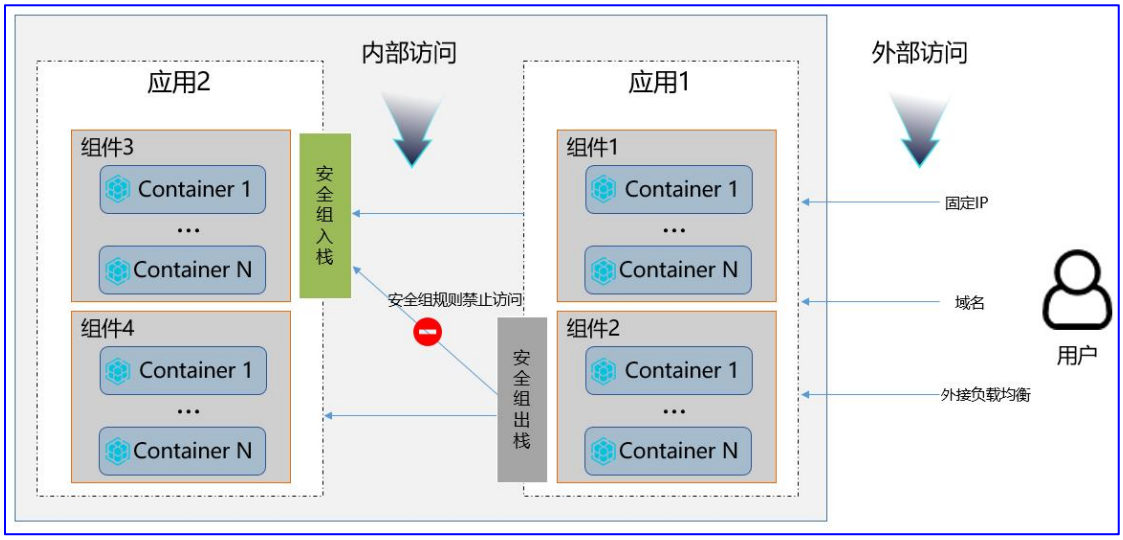
服务整合多类型服务包括内部访问、外部固定 **IP** 访问、外接负载均衡访问的创建，并支持服务的类型转换，同时支持详细地址的增删改，支持实时获取容器端口进行相应的端口映射关系修改，能够最大程度减少用户配置应用的访问入口的时间成本。

服务功能包括：

- 服务创建：支持通过表单、**yaml** 两种方式创建服务，创建服务时支持服务类型选择，包括内部访问类型、外部固定 **IP** 访问类型、外接负载均衡访问类型，支持指定标签选择器，指定标签选择器时支持手动输入同时支持通过组件指定。

- 服务修改：支持通过表单、**yaml** 两种方式修改服务，修改内容包括服务类型、负载均衡策略等。
- 端口操作：支持端口的添加、删除、修改，内容包括端口协议、服务端口号等
- 服务列表：支持多个筛选条件获取服务列表，展示内容包括服务名称、状态、访问方式、**IP** 类型、用户、资源池、应用空间等信息。
- 服务删除：支持服务删除。

4.5.4.1.2 技术原理



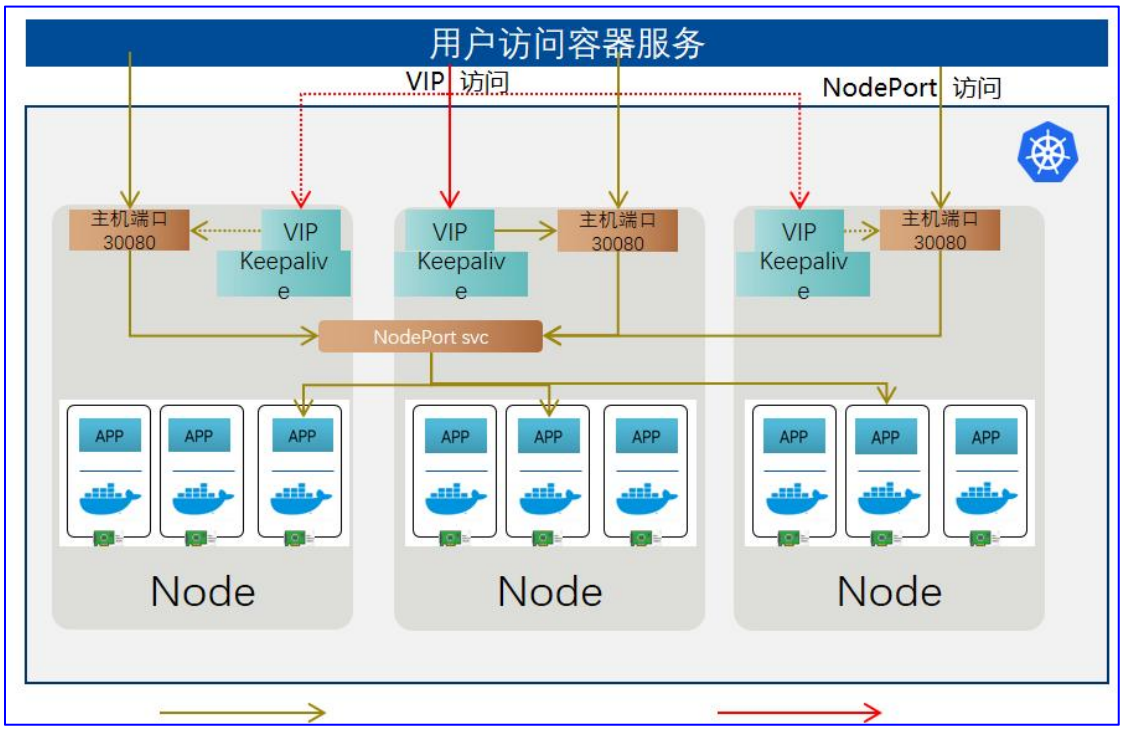
容器资源池通过多样的服务设置，为用户提供细粒度的访问管理。

访问方式包括：

- **1**、内部访问：应用通过 **Kubernetes service** 访问应用，该模式下 **service** 是 **ClusterIP** 类型，通过集群的内部 **IP** 暴露服务，服务只能够在集群内部可以访问。
- **2**、对外固定 **IP** 访问：应用通过 **NodePort** 方式在每个 **Node** 上分配一个端口作为外部访问入口，然后通过平台的集群 **VIP** 方式对外提供访

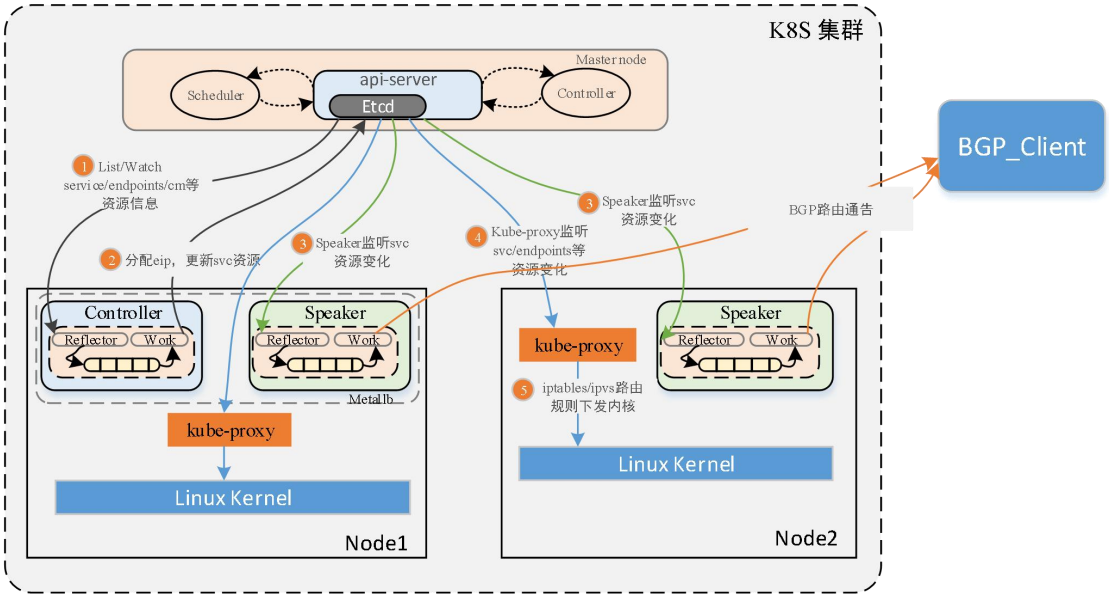
问。

- 在容器资源池部署时，会选取部分节点作为集群南北向网络节点， 部署 **Keepalive** 的服务，创建主备高可用的 **VIP**，通过 **VIP** 代替 节点 **IP + NodePort** 的方式供集群外访问入口，用户不需要考虑访问入口的高可用，一个 **VIP+**不同端口暴露所有服务。



- **3**、负载均衡 **IP** 访问：容器集群外部以设置 **IP** 形式暴露，使用固定 **IP** 直接访问，并支持为服务设置会话亲和性。

容器服务基于 **MetalLB** 来实现负载均衡 **IP** 访问，其总体架构如下：



Metallb 包括了 **controller** 和 **speaker** 两个组件，其中

- **controller** 组件采用 **Deployment** 部署方式，只会在一个节点上部署。该组件 **list/watch apiServer** 的 **service**、**endpoints**、**configmap** 等资源，在创建 **LoadBalancer** 类型的 **service** 时，**controller** 组件会为 **service** 分配 **external–IP** 并更新 **service**。
- **speaker** 组件采用 **DaemonSet** 方式部署。该组件 **list/watch apiServer** 的 **service**、**endpoints**、**configmap** 等资源，获取 **LoadBalancer** 类型的 **service**，并对 **external–IP** 对外通告。

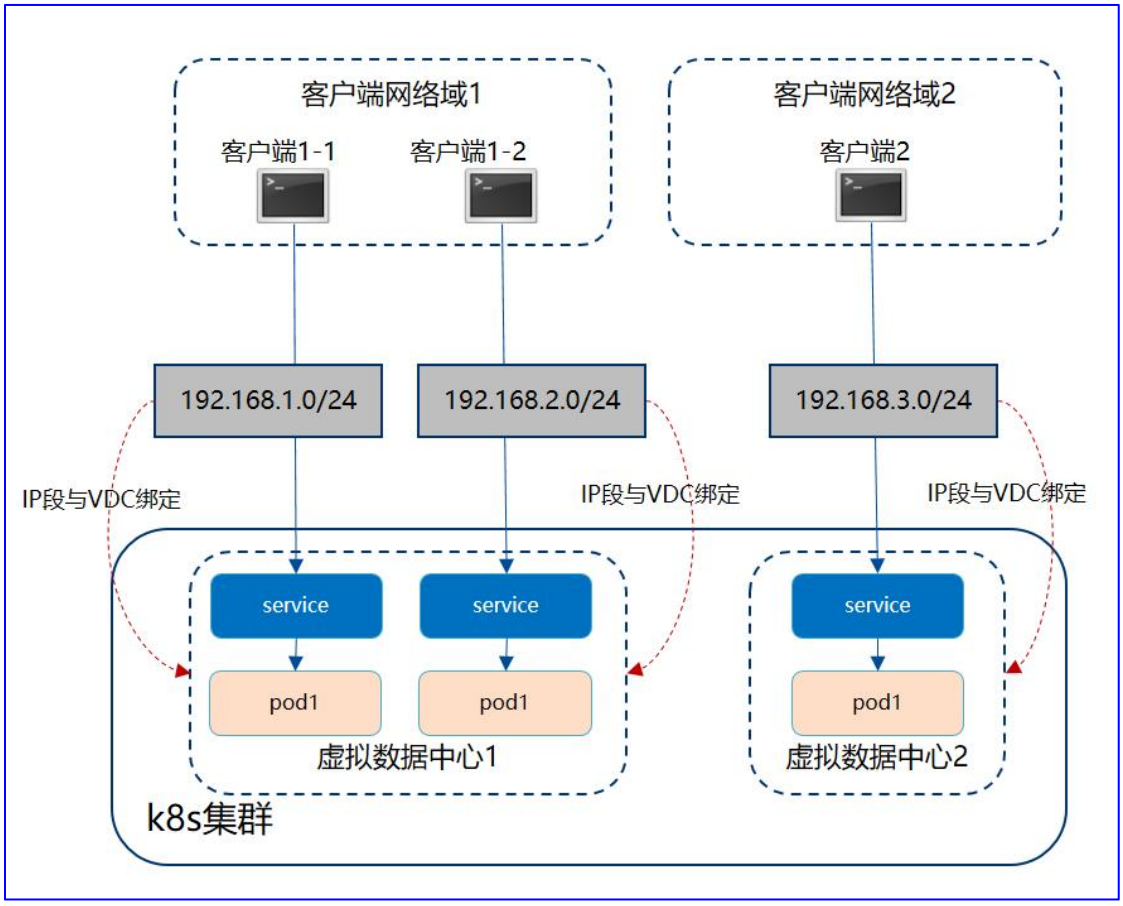
基于 **MetalLB**，容器服务提供两种形式的对外通过方式：**Layer2** 和 **BGP** 两种工作模式。

- **Layer2** 模式：在该模式下，**Metallb Speaker** 扫描宿主机上所有接口并对接口进行过滤（接口 **up**，可发送 **ARP/NDP**）。然后在接口上创建 **newARPresponder/newNDPresponder**，实现 **ARP/NDP** 对外通告。
- **BGP** 模式：**Speaker** 在获取 **ConfigMap** 资源时，会根据 **FRR** 配置模板

生成 **FRR** 配置文件并触发 **FRR** 加载配置动作，实现 **BGP** 邻居建立等功能。在 **Speaker** 获取 **svc** 后，获取 **svc** 的 **Status.LoadBalancer.Ingress.IPs** 信息，并由 **FRR BGP** 实现路由动态通告。

容器服务支持租户级负载均衡子网，可以为不同虚拟数据中心的应用分配不同的负载均衡 **IP** 地址段，以支持来自不同网络域的客户端对不同租户应用使用对应网段进行访问的需求，从而满足多租户下网络精细划分、网络隔离及安全等需求。

容器服务支持负载均衡 **IP** 复用，可以为多个服务设置同一个负载均衡 **IP**，从而减少对 **IP** 资源的消耗。



4.5.4.1.3 约束

无。

4.5.4.2 应用路由

4.5.4.2.1 功能描述

应用路由是一种基于七层的 **HTTP** 和 **HTTPS** 协议转发，通过域名和路径实现对资源池内应用访问，支持平台级别和租户级应用路由。

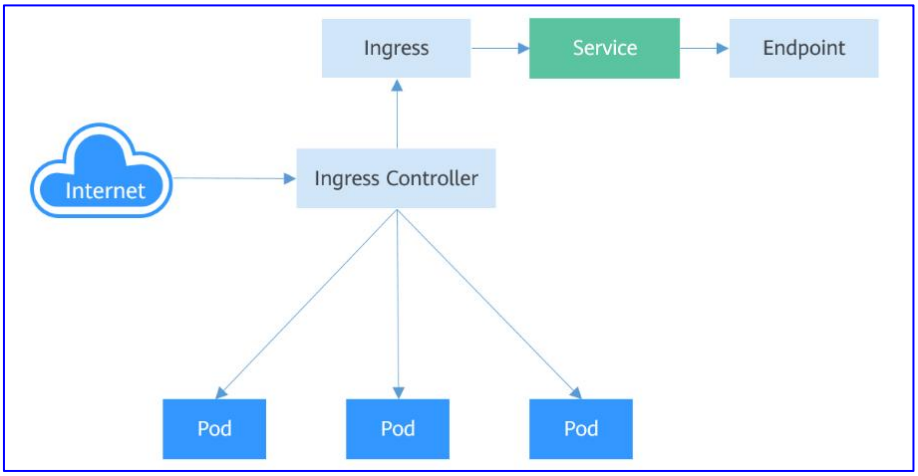
应用路由功能包括：

- 应用路由创建：支持通过表单和 **yaml** 两种方式创建应用路由，创建应用路由支持 **HTTP**、**HTTPS** 协议、支持设置 **TLS** 加密方式、支持设置多域名多路径，通过域名和路径实现更细粒度的访问划分。
- 应用路由列表查看：支持多种查询条件筛选应用路由列表，展示应用路由的名称、**URL** 访问地址、创建时间等信息。
- 应用路由修改：支持通过表单和 **yaml** 两种方式修改应用路由，修改应用路由支持修改 **HTTP/HTTP** 协议、访问域名以及路径等。
- 应用路由删除：支持应用路由的批量删除。
- 应用路由详情：以列表形式展示应用路由的 **URL** 访问地址、**TLS** 证书信息以及 **nginx** 注解信息。

4.5.4.2.2 技术原理

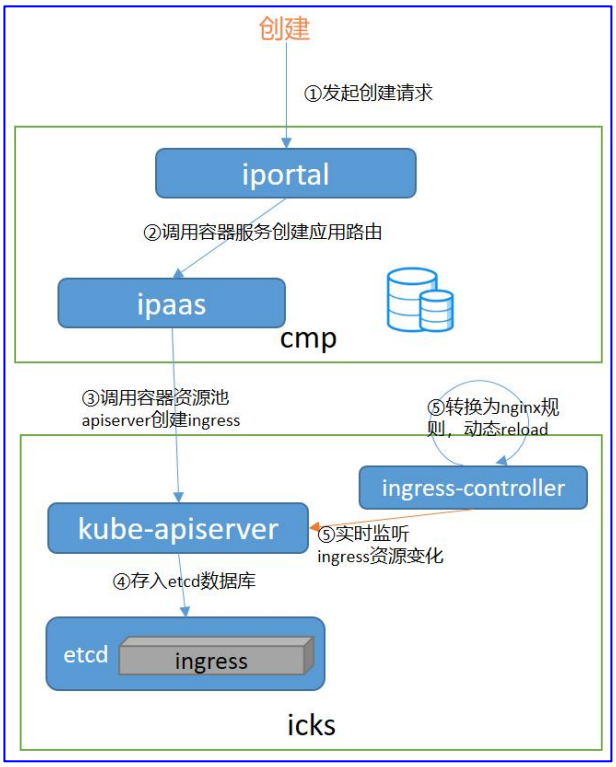
应用路由基于 **K8S Ingress** 资源来实现。

K8s Ingress 实现需要借助于 **ingress controller** 组件，其实现机制如下图所示：



外部请求首先到达 **Ingress Controller**，**Ingress Controller** 根据 **Ingress** 的路由规则，查找到对应的 **Service**，进而通过 **Endpoint** 查询到 **Pod** 的 **IP** 地址，然后将请求转发给后端 **Pod**，从而实现对资源池内应用的访问。

创建应用路由的主体流程如下：



- 页面通过表单或 **UI** 填写应用路由信息，提交创建请求。
- **UI** 理请求信息，调用容器服务。
- 容器服务接收到请求后，进行相关的权限等信息校验。校验完成后调用

容器资源池 **kube-apiserver**，创建 **ingress**。

- **Ingrss-controller** 监听 **ingress** 资源变化，在获取 **ingress** 创建事件后，检查 **ingress** 资源配置并生成 **Nginx** 的配置。
- **kube-apiserver** 将数据存入 **etcd**。

4.5.4.2.3 约束

无

4.5.4.3 网络策略

4.5.4.3.1 功能描述

网络策略用来实现容器服务间的网络访问控制，能够从应用组件、网络协议、网络端口三个层面限定应用的入方向和出方向流量，精准细粒度控制应用网络隔离策略，并支持图形化直观展示网络策略的应用范围、入方向、出方向规则。

网络策略功能包括：

- 网络策略创建：支持通过表单、**yaml** 两种方式创建网络策略，并能够基于模板快速创建网络策略。
- 网络策略列表查看：支持多种查询条件筛选网络策略列表，展示网络策略的名称、归属、创建时间等信息。
- 网络策略修改：支持通过表单、**yaml** 两种方式修改网络策略。表单修改支持对应用组件、出方向规则、入方向规则逐条修改，精准控制改动内容。
- 网络策略删除：通过表单删除网络策略。
- 网络策略详情拓扑展示：以图形化形式直观展示网络策略的组件的访问

控制逻辑，包括入方向规则、作用范围、出方向规则等。

4.5.4.3.2 技术原理

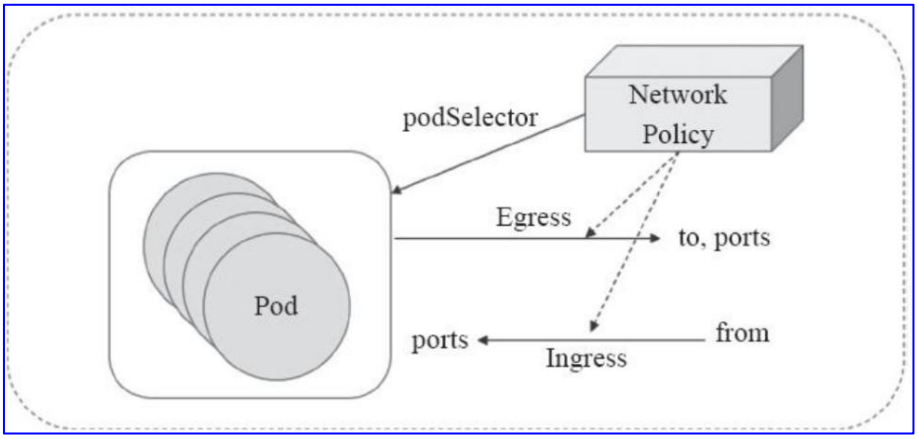
网络策略基于 **K8S Networkpolicy** 来实现，其遵循以下标准约定：

（**1**）如果没有网络策略适用于 **Pod**，则允许所有进出该 **Pod** 的流量。

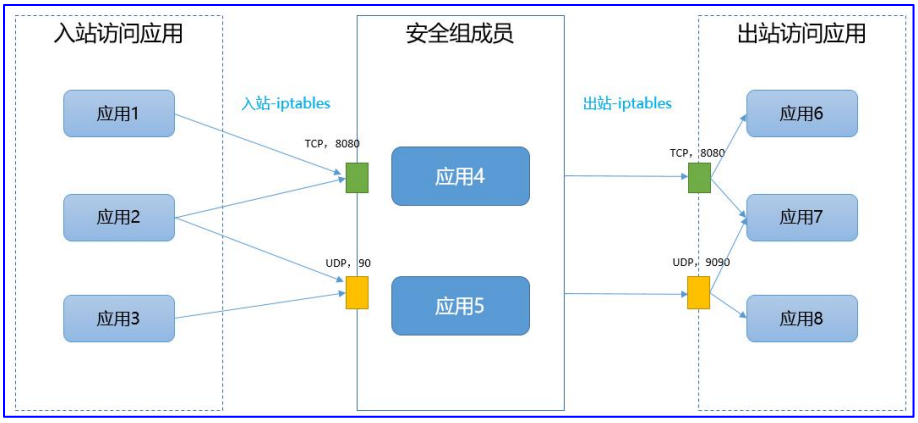
（**2**）如果一个或多个网络策略应用于类型为 **ingress** 的 **Pod**，则仅允许这些策略明确允许的入口流量。

（**3**）如果一个或多个网络策略应用于类型为 **egress** 的 **Pod**，则仅允许这些策略明确允许的出口流量。

NetworkPolicy 通过 **podSelector** 选择一组 **Pod** 资源作为控制对象，是定义在一组 **Pod** 资源之上用于管理入站流量，或出站流量的一组规则，如下图：

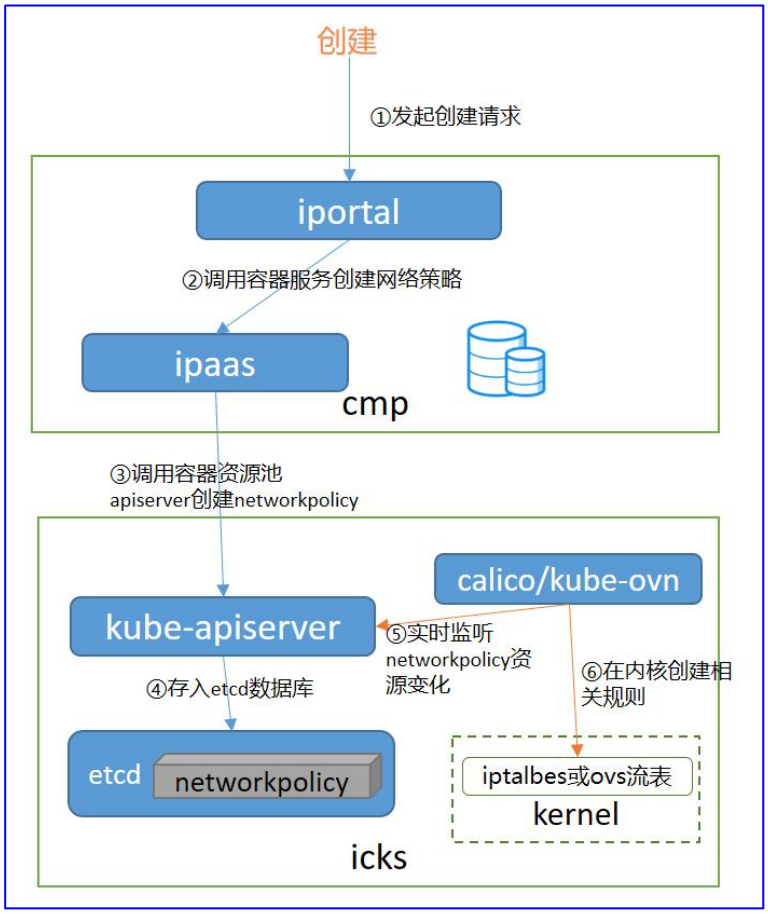


NetworkPolicy 的应用效果如下：



网络策略的具体实现依赖于 **CNI** 插件，默认采用 **Calico CNI** 插件，基于 **iptables** 实现网络策略的入站、出站规则设置。

容器网络创建网络策略主体流程如下：



- 页面通过表单或 **UI** 填写网络策略信息，提交创建请求。
- **Ui** 整理请求信息，调用容器服务。
- 容器服务接收到请求后，进行相关的权限等信息校验。校验完成后调用容器资源池 **kube–apiserver**，创建 **networkpolicy**。
- **kube–apiserver** 将数据存入 **etcd**。
- **CNI** 插件如 **Calico** 通过 **list/watch** 机制，实时监听到 **networkpolicy** 资源的创建操作，获得相关信息。
- **CNI** 插件根据获取的 **networpolicy** 信息，调用内核函数创建 **iptables**

规则或 ovs 流表，从而实现网络控制。

4.5.4.3.3 约束

无。

4.5.4.4 IP 池

4.5.4.4.1 功能描述

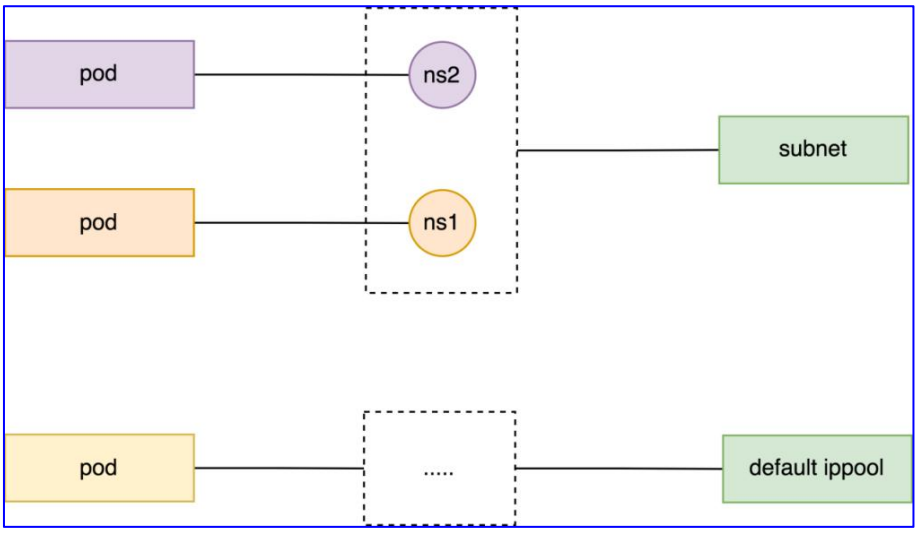
- 容器平台支持为不同的应用空间分配单独的子网并可以预留部分 IP 地址， 实现客户对网络地址更精细化的管理，以及更强的隔离性。
- 容器 IP 池功能功能包括：
- 容器 IP 池子网的创建：支持通过表单方式创建容器子网。
- 容器 IP 池子网列表查看：支持多种查询条件筛选容器子网列表，展示容器子网的名称、网段、绑定应用空间等信息。
- 容器子网修改：支持通过表单方式修改容器子网。表单修改支持对绑定的应用空间进行解绑和添加。
- 子网删除：支持通过表单方式删除容器子网。
- 容器网络详情展示：展示容器子网的名称、网段、绑定应用空间等信息。

4.5.4.4.2 技术原理

平台支持在 Calico 网络插件下实现子网管理。

Calico ：一个或多个 Namespace 可以被绑定到一个子网中，这些 Namespace 下的 Pod 将会从该子网中分配 IP，并使用子网下的网络配置。如果 Pod 绑定的 Namespace 没有绑定子网，将使用默认子网 default–ippool 为

其分配 IP 地址。



◆ 约束

无。

4.5.4.5 容器网络双栈

4.5.4.5.1 功能描述

容器平台支持 IPV4 和 IPV6 双栈，意味着容器组 Pod、服务 Service 和节点 Node 可以获取 IPv4 地址和 IPv6 地址， 客户端可以通过 IPV4 或 IPV6 地址直接访问集群中的服务或容器组，满足用户对容器网络双栈的需求。

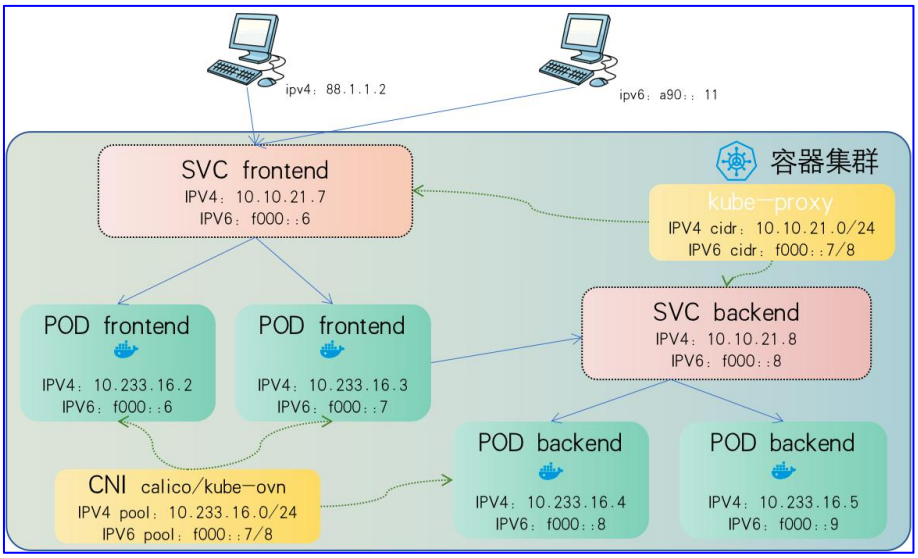
4.5.4.5.2 技术原理

在容器集群部之前，要为部署集群的所有节点开启 IPV6，并配置 IPV6 地址。

容器集群网络模型的核心要求之一是每个 Pod 都拥有自己的 IP 地址并可以使用该 IP 地址进行通信，容器集群通过 CNI 插件(calico)实现了的 IPAM 功能，在 CNI 插件中配置 POD 允许的 IP4/IPV6 CIDR，容器集群创建 POD 时，

kubelet 模块通过容器运行时调用 **CNI** 插件完成 **IP4/IPV6** 地址的分配，使 **POD** 可以同时具备 **IPV4** 和 **IPV6** 地址，并完成 **IPV4/IPV6** 两个地址的容器集群组网，使得容器集群内的 **POD** 可以通过 **IPv4** 或 **IPv6** 网络直达。

集群部署时配置集群 **Service** 的 **CIDR** 参数：**service-cluster-ip-range**，同时配置 **IPV4** 和 **IPV6** 的 **CIDR**，两者以逗号分隔，用户在创建 **Service** 时，集群可以为 **Service** 分配 **IPV4** 和 **IPV6** 两个地址，**kube-proxy** 模块同时为 **Service** 创建 **IPV4/IPV6** 负载访问对应后端 **pod** 的 **iptables** 或 **ipvs** 规则，实现用户通过 **IPV4/IPV6** 地址直接访问 **Service**。



◆ 约束

需要容器集群节点开启 **IPV6**。

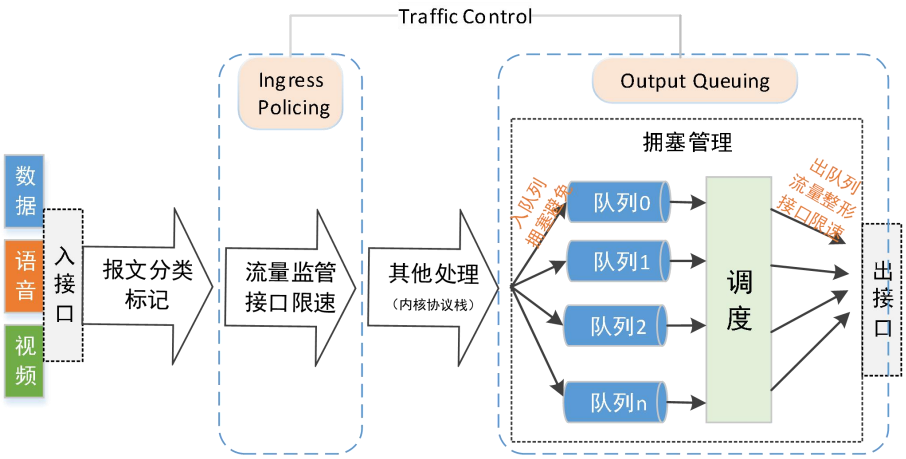
4.5.4.6 QoS

4.5.4.6.1 功能描述

平台支持对 **Pod** 网卡出入带宽的限速配置，从而实现对网络带宽资源的精细化控制。

4.5.4.6.2 技术原理

对 **Pod** 网卡出入带宽的限制依赖于 **Linux** 内核的 **Traffic Control** 模块实现，**Qos** 实现机制如下图所示：



- 报文分类与标记。依据用户设置的过滤规则，利用分类器（**filter**）将数据包分为不同的类别（**class**）或设置为不同的优先级。
- 接口限速、流量监管和流量整形。将业务流量限制在特定带宽内，当业务流量超过额定的带宽时，超过的流量将被丢弃或者缓存。
- 拥塞避免和拥塞管理。拥塞管理在网络发生拥塞时，采用队列技术，将数据报文放入队列中缓存，并采取队列调度算法将数据报文发送出去

4.5.4.6.3 约束

无。

4.5.4.7 固定 IP

4.5.4.7.1 功能描述

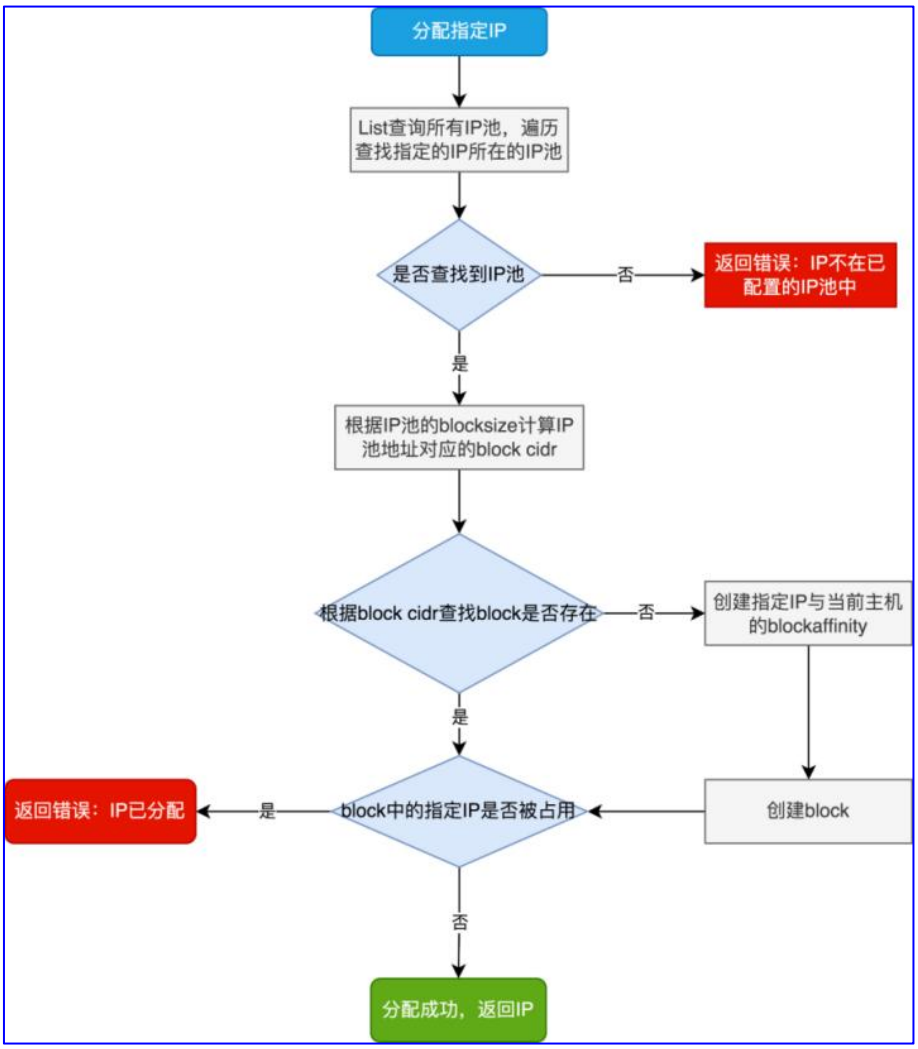
- 容器平台支持为 **Pod** 分配固定的 **IP** 地址段或单个 **IP**，可以使 **IP** 地址作为业务组件实例唯一的标识。**Pod** 重建后，**IP** 地址段或单个 **IP** 可保

持不变。

4.5.4.7.2 技术原理

固定 IP 的具体实现依赖于 CNI 插件，默认采用 Calico CNI 插件。

Calico 分配指定 IP 的逻辑流程：



4.5.4.7.3 约束

固定 IP 必须在应用空间所属的子网内。

4.5.4.8 RDMA

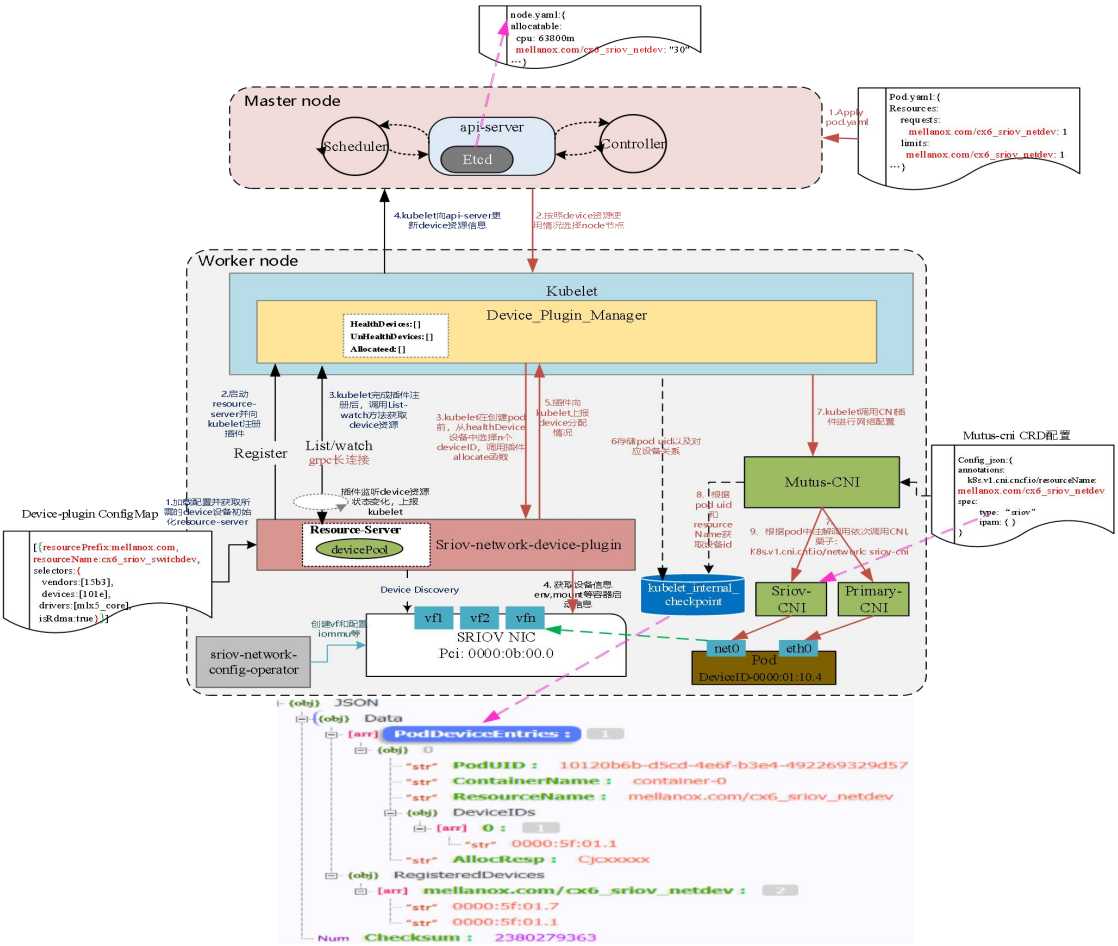
4.5.4.8.1 功能描述

- 容器服务支持为应用添加 **RDMA** 高性能网络，能够满足 **HPC**、**AI**、存储、大数据等基于 **RDMA** 编程体系的应用间高速数据读取需求。

4.5.4.8.2 技术原理

基于 **SR–IOV+**容器多网卡技术，提供给容器 **RDMA** 网络+标准 **K8S** 网络的双重能力。

总体技术架构如下：



RDMA 具体配置主要分 **4** 步，实现流程如下：

- 智能网卡自动配置：配置过程使用以 **daemonset** 模式部署 **Sriov-Network-Config** 组件，**Sriov-Network-Config** 根据 **supported-nic-ids** 的网卡配置信息获取过滤宿主机网卡，实时 **watch SriovNetworkNodeState** 的 **CR** 并加载内核 **iommu**，生成相应个数 **VF**，配置完成后自动重启本节点 **SR-IOV-Device-Plugin** 的 **pod** 来实现 **VF** 快速上报。**SriovNetworkNodeState** 的 **CR** 中期望值发生变化时，**Sriov-Network-Config** 会驱逐节点 **pod** 并重启节点，来实现 **VF** 重新自动配置。
- **VF** 发现上报：使用以 **daemonset** 模式部署 **SR-IOV-Device-Plugin** 组件实现 发现识别上报 **VF** 信息。**SR-IOV-Device-Plugin** 通过网卡 **vendors**、**devices** 和 **drivers** 的信息来过滤发现识别上报 **VF** 信息。
- **VF** 分配：**pod** 创建时，**K8S Scheduler** 根据 **pod VF** 个数需要，将 **pod** 调度到 **VF** 节点。**Kubelet** 监听到 **Pod** 创建事件，从内存中为 **Pod** 分配足够的 **deviceID**，并连同 **podUID**、**ResourceName** 等信息存储到本机的**/var/lib/kubelet/device-plugins/kubelet_internal_checkpoint** 文件中，这个文件供 **Mutus-CNI** 查询并将 **deviceID** 传递给 **SRIOV-CNI** 使用。
- **VF** 挂载：**Kubelet** 调用 **Mutus-CNI** 为 **Pod** 配置网络信息。**Mutus-CNI** 根据 **podUID** 与 **ResourceName** 查询 **deviceID** 信息，并连同 **Kubelet** 传递的入参依次调用 **Calico** 和 **Sriov-CNI** 来配置网络。**Calico** 为 **Pod** 生成 **eth0** 主网络并实现 **K8S** 定义的 **service/networkPolicy** 等网络概念，而 **Sriov-CNI** 主要将 **VF** 挂载到 **pod** 中。

4.5.4.8.3 约束

无

4.5.4.9 自定义 DNS 配置

4.5.4.9.1 功能描述

平台支持为容器组设置自定义 DNS 规则，以满足容器通过资源池外部域名服务器访问其他应用等场景。

4.5.4.9.2 技术原理

容器服务会在容器组创建过程中，由 kubelet 为其生成 DNS 相关配置。该配置来源包括两个：kubelet 默认配置，以及容器组的自定义 DNS 配置。容器服务支持在页面创建应用及组件时，填写自定义 DNS 配置，主要内容包括：

属性名称	数据类型	可选项
hostName	string	optional
subDomain	string	optional
dnsConfig	PodDNSConfig	optional
dnsPolicy	DNSPolicy	None/Default/ClusterFirst/ClusterFirstWithHostnet
SetHostnameAsFQDN	bool	Default to false. optional
HostAliases	[]HostAlias	optional

- **hostname**：在默认情况下，**Pod** 的名称将被系统设置为容器环境内的主机名称 **(hostname)**，但通过副本控制器创建的 **Pod** 名称会有一段随机后缀名，无法固定，此时可以通过在 **Pod yaml** 配置中设置 **hostname** 字段定义容器环境的主机名。同时，可以设置 **subdomain** 字段定义容器环境的子域名。如果在 **pod** 中显式的指定这个字段，则使用 **hostname** 命令可以返回指定的值。
- **subdomain**：设置 **subdomain** 字段定义容器的子域名。
- **dnsConfig**：支持多个属性，包括：

属性	描述
nameservers	将用作 **Pod** 的 **DNS** 服务器的 **IP** 地址列表。最多可以指定 **3** 个 **IP** 地址。当 **Pod** 的 **dnsPolicy** 设置为 **None** 时，列表必须至少包含一个 **IP** 地址，否则此属性是可选的。列出的 **DNS** 的 **IP** 列表将合并到基于 **dnsPolicy** 生成的域名解析文件的 **nameserver** 字段中，并删除重复的地址
searches	**Pod** 中主机名查找的 **DNS** 搜索域列表。此属性是可选的。指定后，提供的列表将合并到从所选 **DNS** 策略生成的基本搜索域名中，并删除重复的域名。**Kubernetes** 最多允许 **6** 个搜索域。
options	可选的对象列表，其中每个对象可以具有 **name** 属性（必需）和 **value** 属性（可选）。此属性中的内容将合并到从指定的 **DNS** 策略生成的选项中，并删

	除重复的条目。

- **dnsPolicy**：**pod** 的主要 **dns** 规则，包括：

策略	描述
Default	继承 **Pod** 所在宿主机的域名解析设置
ClusterFirst	优先使用 **Kubernetes** 环境的 **DNS** 服务
ClusterFirstWithHostNet	强制在 **hostNetWork** 网络模式下使用 **ClusterFirst** 策略（默认使用 **Default** 策略）
None	忽略 **Kubernetes** 集群的 **DNS** 配置,需要手工通过 **dnsConfig** 自定义 **DNS** 配置

- **SetHostnameAsFQDN**：当 **Pod** 配置为具有全限定域名（**FQDN**）时，其主机名是短主机名。例如，如果你有一个具有完全限定域名 **busybox-1.busybox-subdomain.my-namespace.svc.cluster-domain.example** 的 **Pod**，则默认情况下，该 **Pod** 内的 **hostname** 命令返回 **busybox-1**，而 **hostname --fqdn** 命令返回 **FQDN**。当你在 **Pod** 规约中设置了 **setHostnameAsFQDN: true** 时，**kubelet** 会将 **Pod** 的全限定域名（**FQDN**）作为该 **Pod** 的主机名记录到 **Pod** 所在名字空间。在这种情况下,**hostname** 和 **hostname --fqdn** 都会返回 **Pod** 的全限定域名。
- **HostAliases**：容器中的程序需要其他形式的域名解析，通过 **.spec.hostAliases** 为 **Pod** 配置 **hostAliases**，会自动将配置的解析记录写入 **Pod** 的**/etc/hosts** 文件中。

4.5.4.9.3 约束

当 **dnspolicy** 为 **none**，此时的 **dnsConfig** 不能为空。

4.5.4.10 跨集群服务发现

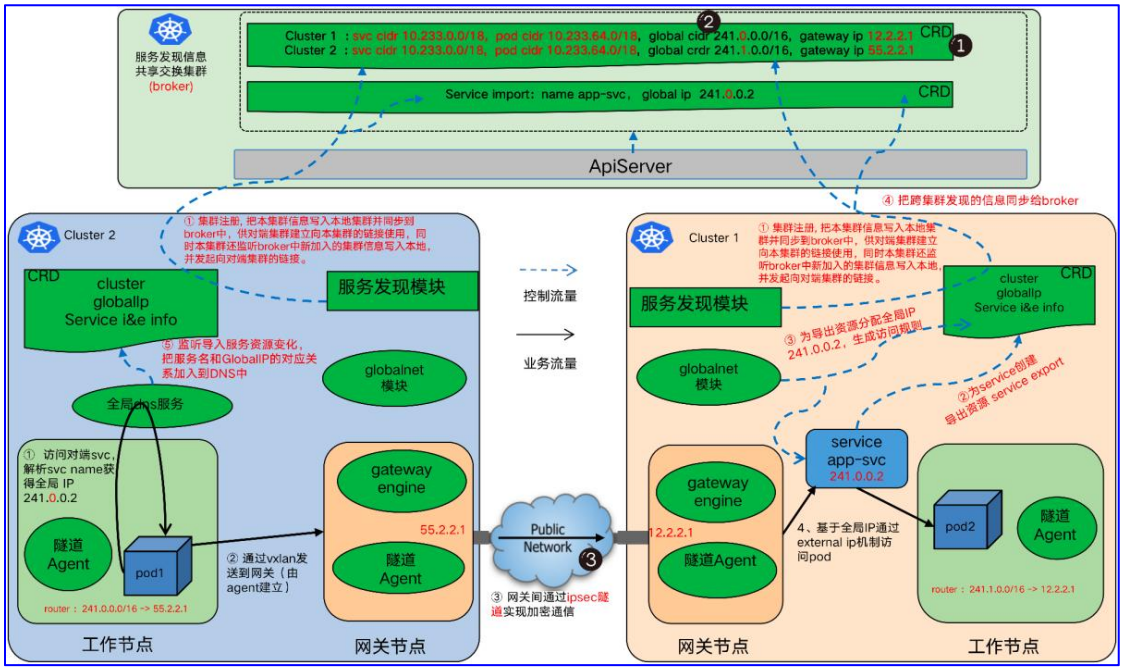
4.5.4.10.1 功能描述

平台提供跨资源池的互联互通和服务发现能力， 安装平台多资源池服务发现组件，创建多资源池服务， 支持通过指定的 **Service** 名称访问非本地集群的容器应用。

4.5.4.10.2 技术原理

基于**Submariner**将为部署在需要相互通信的多个**Kubernetes**集群中的容器服务提供网络连接和服务发现能力，解决 **Kubernetes** 集群之间的访问孤岛问题，支持 **IPSec** 隧道加密网络连接。

总体技术架构如下：



- **Gateway：** 运行于 **gateway** 节点，负责建立、管理到其他集群的安全隧

道，实现跨集群加密的网络连接

- **Route Agent:** 运行于每个节点，负责设置本地的路由、**iptables** 规则，将跨集群流量从普通节点转发至 **Gateway** 节点
- **Broker**：逻辑上的概念，提供了一组 **API** 供参与的集群调用，通过 **API** 可以推送本地集群的资源信息也可以获取其他集群的资源信息
- **Lighthouse agent**：负责服务发现功能，与 **broker** 的 **api server** 直接交互。在本地集群 **export service** 时，将 **service** 信息推送至 **broker**，并且从 **broker** 拉取其他集群 **export service** 的信息
- **Lighthouse coredns**：负责 **export** 出来的 **service** 的域名解析
- **Globalnet** （可选）**:**处理具有重叠 **CIDR** 的集群的互连
- **ServiceExport** 指明需要跨集群暴露的服务，使所有集群都能发现这个服务
- **ServiceImport** 互联集群中发现 **ServiceExport** 后会由 **ligthhouse** 在本集群创建，记录了原 **service** 的信息，包括来自哪个集群，对应的 **serviceName** 等

4.5.4.10.3 约束

不同集群 **gateway** 网关节点之间能相互通信。

4.5.5 容器存储

4.5.5.1 多种存储类型对接

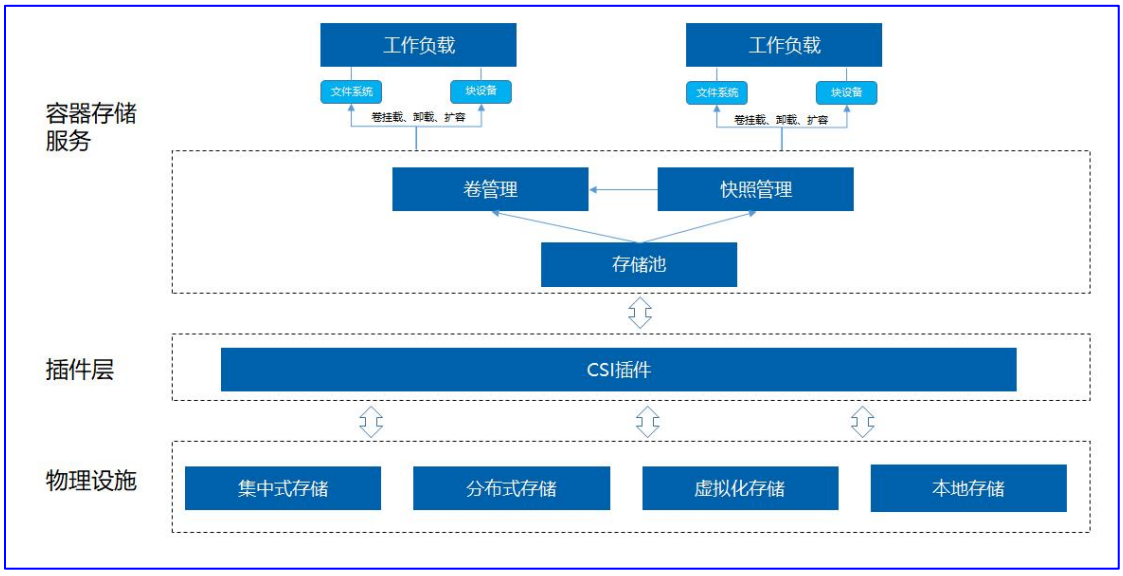
4.5.5.1.1 功能描述

随着政府、企事业单位信息化的发展，信息化系统用户规模不断扩大，不但

需要满足用户存储大量文本、图像和视频等要求，还要保证数据的稳定性和安全性，因此对于容器平台纳管多种类型存储要求越来越高。

容器服务存储池功能支持对接多种类型的存储，目前支持本地存储、分布式存储、集中式存储。

4.5.5.1.2 技术原理



集中式存储，从概念上可以看出来是具有集中性的，也就是整个存储是集中在一个系统中的。但集中式存储并不是一个单独的设备，是集中在一套系统当中的多个设备。

分布式存储系统是大量普通 PC 服务器通过 Internet 互联，对外作为一个整体提供存储服务。分布式存储系统具有如下几个特性：

- 可拓展：分布式存储系统可以扩展到几百台甚至几千台的集群规模，随着集群规模的增长，系统整体性能表现为线性增长。
- 低成本：分布式存储系统的自动容错、自动负载均衡机制使其可以构建在普通 PC 机之上。线性扩展能力也使得增加、减少机器非常方便，可以实现自动运维。

- 高性能：无论是针对整个集群还是单台服务器，都要求分布式存储系统具备高性能。
- 易用：分布式存储系统需要能够提供易用的对外接口，同时要求具备完善的监控、运维工具，并能够方便地与其他系统集成。

容器服务将支持接入的多种类型的存储按照存储提供的形式，分为块存储、文件存储、本地存储。

块存储是一种主流存储类型，主要是基于 **Block** 块的模式，容器服务支持通过插件将任意大小的块设备挂载到容器组内，为容器组提供持久化存储空间。目前支持两种主流形态，一种是基于 **SAN** 的传统集中式存储，另一种是基于 **IP** 的分布式存储，比如浪潮 **AS13000**，经过多方位的大规模和性能测试，对上述两种主流形态的支持度很高，同时也支持通过 **Kubernetes** 标准插件对接第三方存储。

文件存储也是主流存储类型，提供了可共享访问、弹性扩展、高可靠以及高性能的共享文件存储服务。容器服务支持将文件存储同时挂载给多个容器组，允许同时进行访问避免了大量的数据拷贝与同步成本，达到真正的资源共享。

本地存储做为远端存储系统的一种补充，支持将主机上的磁盘或分区组成 **VG**，划分 **LV** 与容器 **PV** 关联，挂载到容器应用并初始化文件系统进行业务数据持久化。容器服务管理的本地存储支持单节点读写访问模式，不支持卷扩容。

4.5.5.1.3 约束

无。

4.5.5.2 存储卷管理

4.5.5.2.1 功能描述

存储卷管理提供存储卷的全生命周期管理，支撑存储卷的创建、挂载、卸载、扩容、存储卷共享、卷模板、查看事件、资源审计、监控使用量、使用率、历史曲线等一系列功能。

容器服务提供可视化的存储卷管理功能，能够屏蔽不同厂商、不同类型的存储产品复杂的设置及对接过程。

根据存储文件系统的管理方式以及卷的特性，将存储分为块存储、文件存储、本地存储，更加方便用户查找使用。

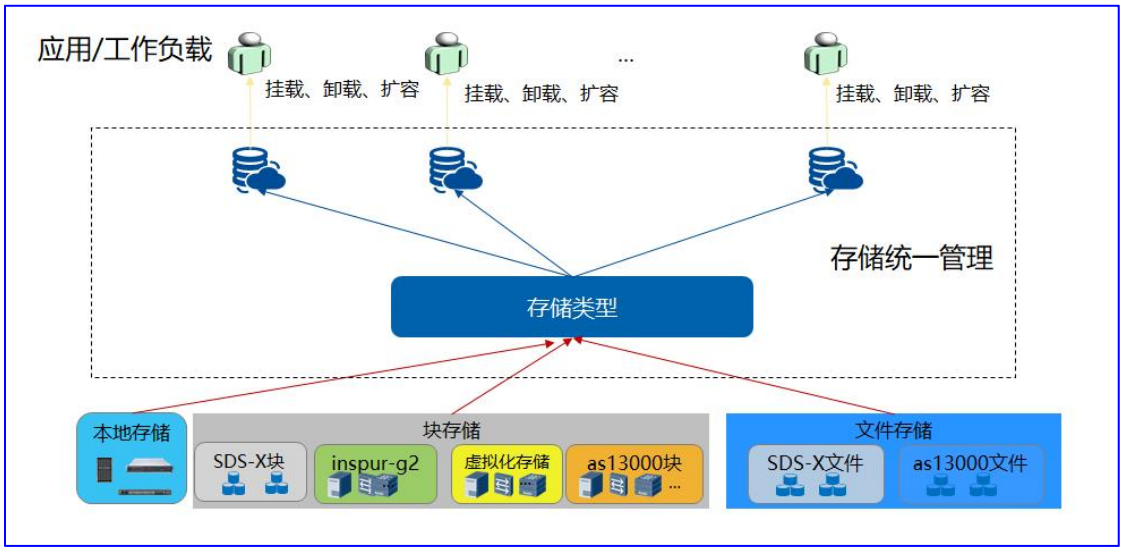
支持应用挂载、卸载存储卷，以满足不同数据持久化功能。支持存储卷扩容功能，满足不断增加的应用数据需求。支持存储卷使用量、使用率动态的展示，用户直观的了解存储卷基本概况。

存储卷管理功能包括：

- 存储卷生命周期管理：创建、修改、删除、挂载、卸载。
- 存储池生命周期管理：存储池创建、修改、删除。
- 应用信息全方位展示：基本信息、存储卷使用量、存储卷使用率、资源审计、存储池使用量、使用率信息。
- 卷模板：支持使用卷模板为有状态负载应用动态创建存储卷。
- 主动卸载、挂载卷：存储卷对应用主动发起挂载、卸载动作。
- 存储卷在线扩容：支持未挂载、已挂载的存储卷扩容，支持通过修改存储卷容量，实现在不影响业务系统以及持久化数据情况下，快捷的实现存储卷扩容。扩容时业务不中断，数据不丢失，满足高可用业务需求。

- 存储卷使用率、使用量、历史曲线：展示存储卷使用量、使用率，方便用户实时的查询当前存储的使用情况，分析存储使用情况。
- 存储卷共享：支持一个存储卷同时为多个应用提供数据存储服务

4.5.5.2.2 技术原理



容器集群通过存储类型、存储卷的管理功能，屏蔽底层不同厂商、不同类型的存储设备的对接和使用，为上层工作负载提供卷的挂载、卸载、扩容、使用量监控等功能。

4.5.5.2.3 约束

- 在同一个虚拟数据中心中，不可存在相同名称的存储卷。
- 存储卷在被应用组件使用的时候，无法被删除，只有未被使用的存储卷才可以被删除。
- 访问模式为单节点读写的存储卷不能挂载给不同的应用组件。
- 应用组件实例数大于 **1**，不能挂载访问模式单节点读写存储卷。
- 应用组件开启弹性伸缩或者升级策略为滚动升级时不能挂载访问模式

单节点读写存储卷。

4.5.5.3 存储卷快照

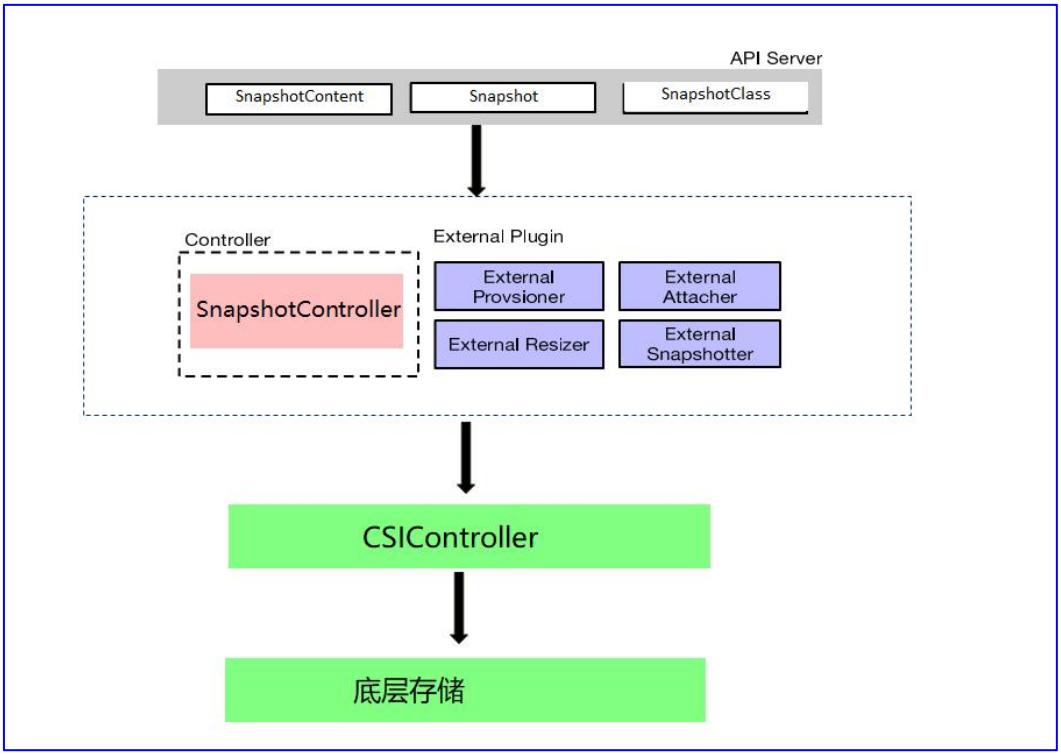
4.5.5.3.1 功能描述

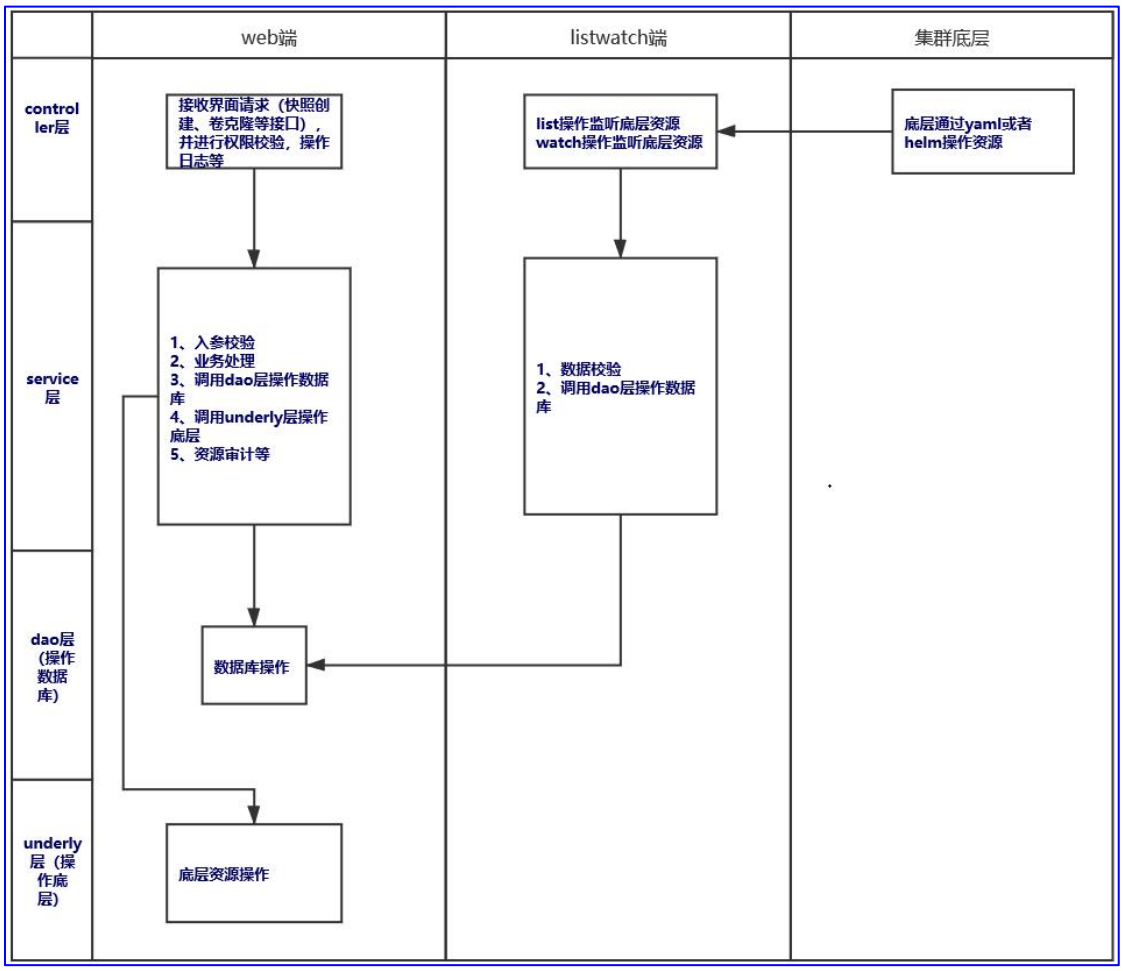
随着大中型企业快速的发展，企业用户规模不断扩大，数据的增长速度也越来越快，需要缩短备份窗口，以在线的方式对存储数据进行保护，提高数据保护过程中的应用感知能力。快照是对块存储卷某时刻数据状态的记录，可以作为一种数据容灾方案。当数据丢失时，可通过快照及时将业务数据完整地恢复到快照创建点的状态

可以实现块存储快照的生命周期管理，包括快照创建、删除、修改和基于快照创建新卷。

4.5.5.3.2 技术原理

基于 **K8S** 快照资源和 **CSI** 插件原理进行深度研发，交互原理如下图所示：





4.5.5.3.3 约束

无

4.5.5.4 存储卷克隆

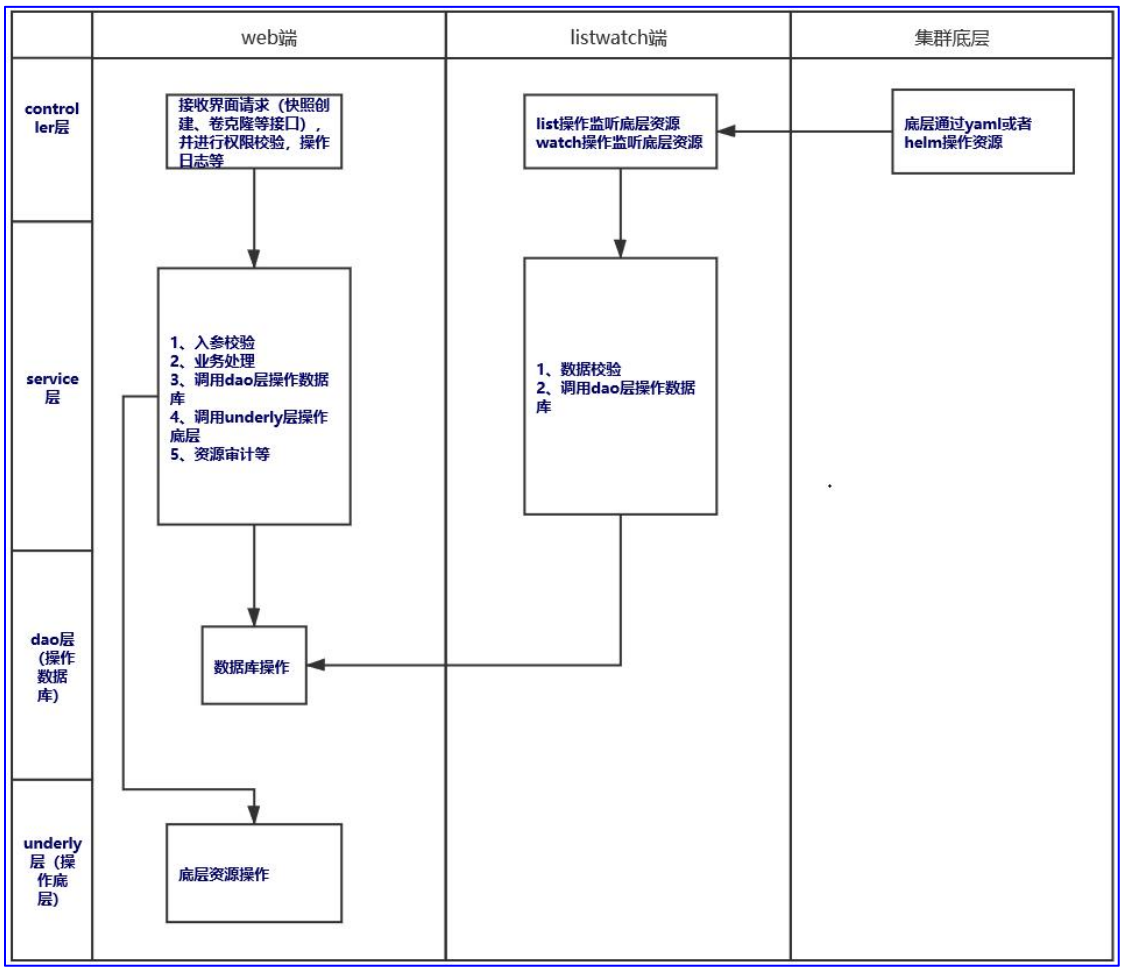
4.5.5.4.1 功能描述

克隆是基于存储卷已经写入数据进行复制。容器服务支持块存储卷进行克隆。支持块存储进行克隆，克隆时可以选择新存储卷的名称和描述信息。

4.5.5.4.2 技术原理

基于 **K8S** 卷创建卷流程和 **CSI** 插件原理进行深度研发，交互原理如下图所

示：



4.5.5.4.3 约束

无。

4.5.5.5 块存储卷弹性扩容

4.5.5.5.1 功能描述

容器块存储卷弹性扩容设计的目标旨在实现容器存储卷的动态扩容功能，帮助用户解决手动扩容卷的高运维成本，因扩容滞后导致的业务中断，以及存储资源使用率等问题，容器块存储卷弹性扩容要实现如下特性：

- 当存储卷使用率到达设定的阈值后，可自动触发容器卷扩容至目标容量，

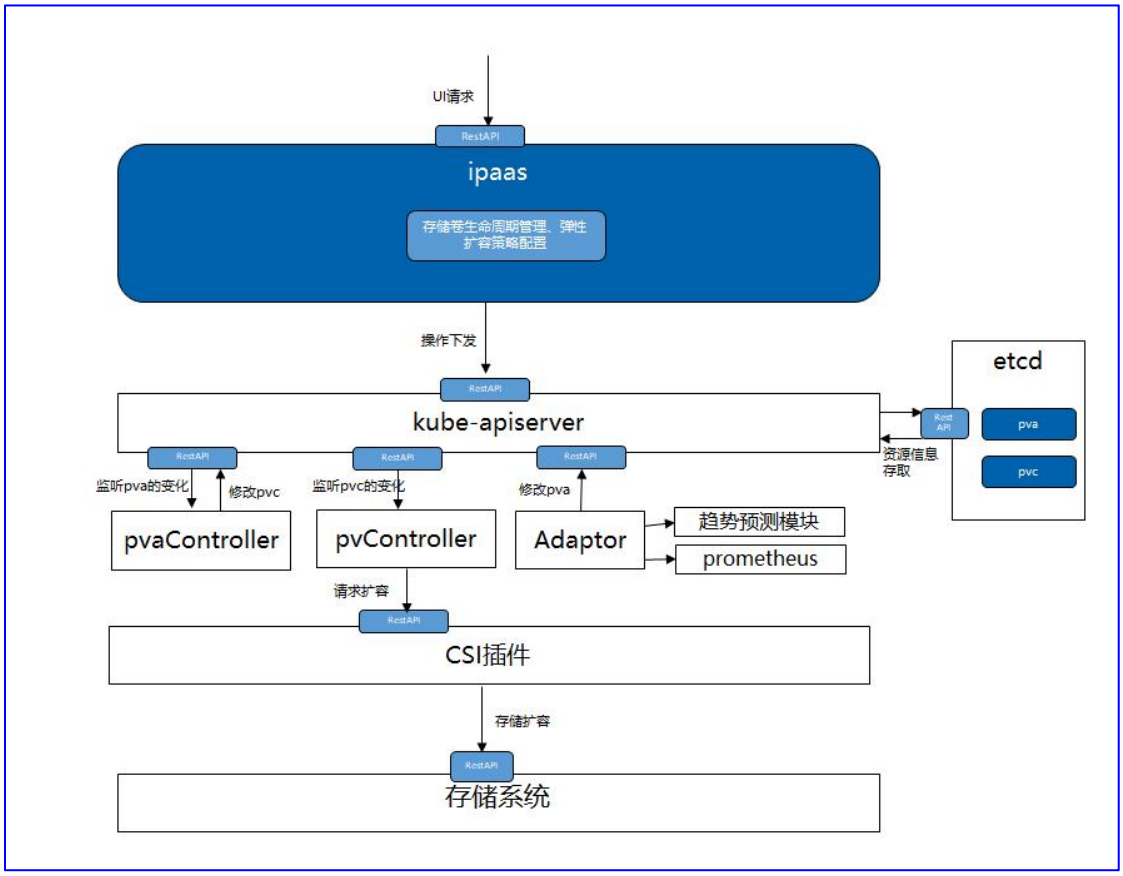
能支持按固定容量扩容或按照扩容前容量的百分比为扩容步长进行扩容。

- 根据智能预测算法预测的剩余容量耗尽时长，可自动触发容器卷扩容至目智能预测算法推算出的足够设定天数使用的容量。

- 能根据设定的时间，自动触发容器卷的扩容至目标容量，能支持按固定容量扩容或按照扩容前容量的百分比为扩容步长进行扩容。

- 能对自动扩容操作进行事件记录。

4.5.5.5.2 技术原理

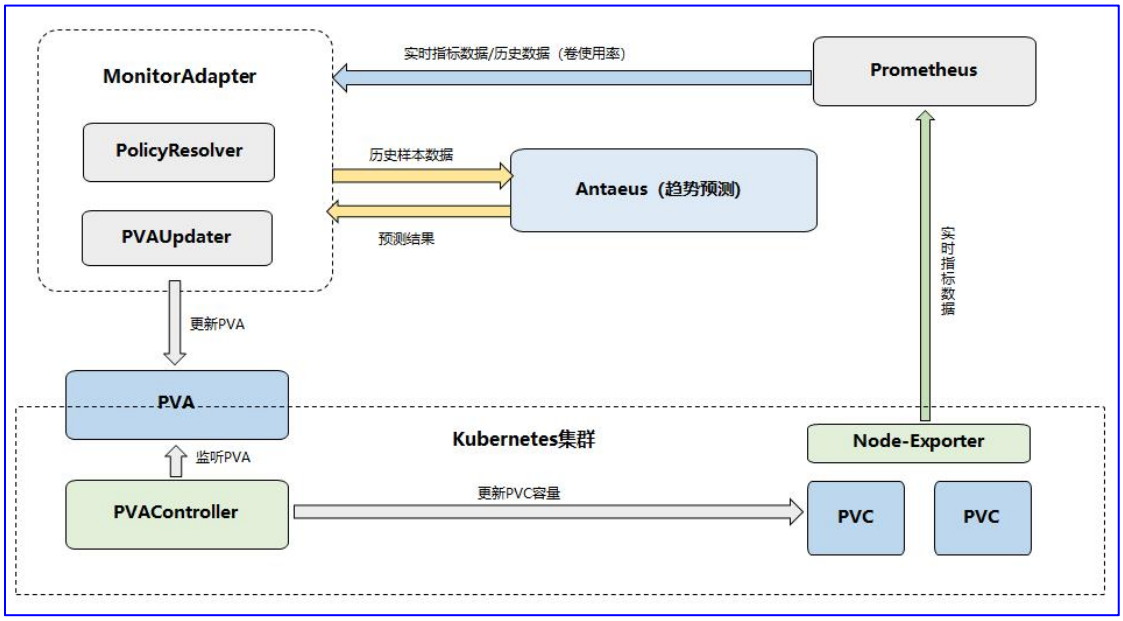


- 通过自研 **ipaas** 模块承担前端请求，进行存储卷资源 **PVC** 以及自定义的动态扩容资源 **PVA** 的管理。

- **Adaptor** 与 **prometheus** 和趋势预测模块进行交互获取使用率数据并对容量进行预测，根据结果判断是否需要扩容并更新到 **PVA** 资源中。

- 自研 **PvaController** 监听 **PVA** 的变化，根据是否扩容来修改容器卷资源 **PVC**。
- **PvController** 监听 **PVC** 的变化，若容量已经发生变法，通知 **CSI** 进行扩容。

关键技术点



基于 **K8S** 声明式 **API** 技术，扩展 **K8S** 的资源类型（**CRD**），用来表征自动扩容的内容。

- 通过 **prometheus** 进行采集并保存容器卷使用率，根据自动扩容策略，**MonitorAdapter** 模块定时去获取卷的实时使用率和历史使用率数据作为自动扩容的数据源。
- 通过 **sidecar** 模式，将 **MonitorAdapter** 模块和趋势预测模块部署在一个 **pod** 内,使用 **uds** 进行通信将历史数据通过 **stl**、三次指数平滑、**arima** 三种预测算法转化为容量预测数据用来进行智能扩容。
- 基于 **list/watch** 技术，实现资源 **controller** 用来监听 **CRD** 的变化，获取扩容数据通过完成容器卷扩容。

- 基于 **K8S Event** 管理技术，将 **CRD** 实例创建、修改、以及扩容等关键操作生成事件用来进行审计和操作记录。

4.5.5.5.3 约束

- 实现动态扩容，依赖承载容器卷的后端存储系统支持存储卷的扩容。
- 本地存储卷、文件存储卷以及 **Hostpath**、**Emptydir**、**ConfigMap**、**Secret** 等不支持弹性扩容。

4.5.6 配置中心

4.5.6.1.1 功能描述

配置中心是 **Kubernetes** 的两种特殊类型的存储卷的集合，包括普通配置和加密配置，是从集群外部向 **Pod** 内部的应用注入配置信息的方式。

普通配置实现向容器中提供配置文件或环境变量来实现不同配置，从而实现了镜像配置与镜像本身解耦，使容器应用做到不依赖于环境配置。

加密配置在普通配置的基础上解决了密码、**token**、密钥等敏感数据的配置问题。

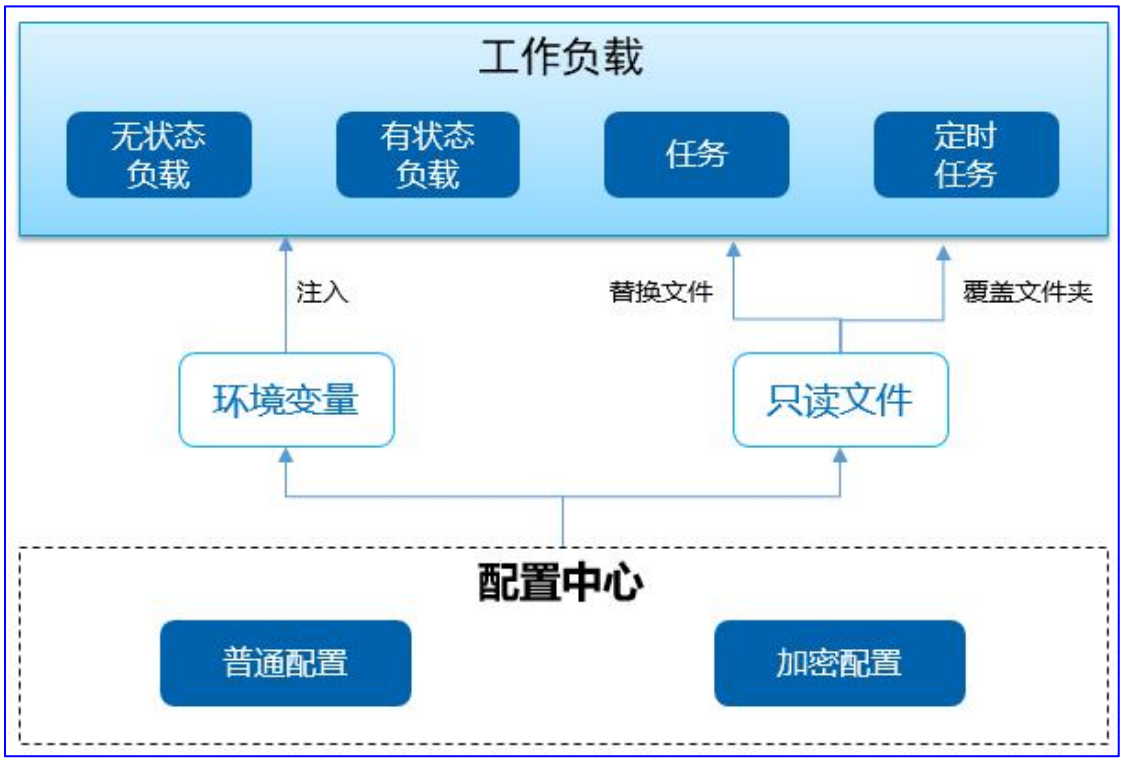
配置中心为工作负载提供了最基础的资源配置功能，能够最大程度的方便用户添加和修改配置文件，减少用户在运维管理上的人力与时间成本。

配置中心服务功能包括：

- 普通配置：**Kubernetes** 普通配置，支持手动输入添加和导入文件添加，修改和删除等功能。在工作负载中，支持将普通配置的键值对配置到容器环境变量中，以及支持通过替换文件和覆盖文件的方式将普通配置挂载到容器上。

- 加密配置：**Kubernetes** 加密配置，包含两种加密类型：普通类型、**TLS**（安全传输层协议），支持手动输入添加和导入文件添加，修改和删除等功能。多用于配置密码、**Token**、私钥等敏感数据，在工作负载中，支持将普通配置的键值对配置到容器环境变量中，以及支持通过替换文件和覆盖文件的方式将普通配置挂载到容器上。
- 支持用户手动输入或者文件导入的方式，来定义配置信息。
- 支持将配置信息中想要使用的键值对，通过覆盖文件或替换文件的方式添加到容器的挂载卷上。
- 通过覆盖文件添加的挂载卷支持热加载功能，无需重启工作负载，即可生效。

4.5.6.1.2 技术原理



配置中心服务提供基于 **Kubernetes** 集群的基础配置功能，配置可以通过环

境变量的方式注入到工作负载的容器中，也可以通过替换文件或覆盖文件夹的形式挂载到容器上。

4.5.6.1.3 约束

无。

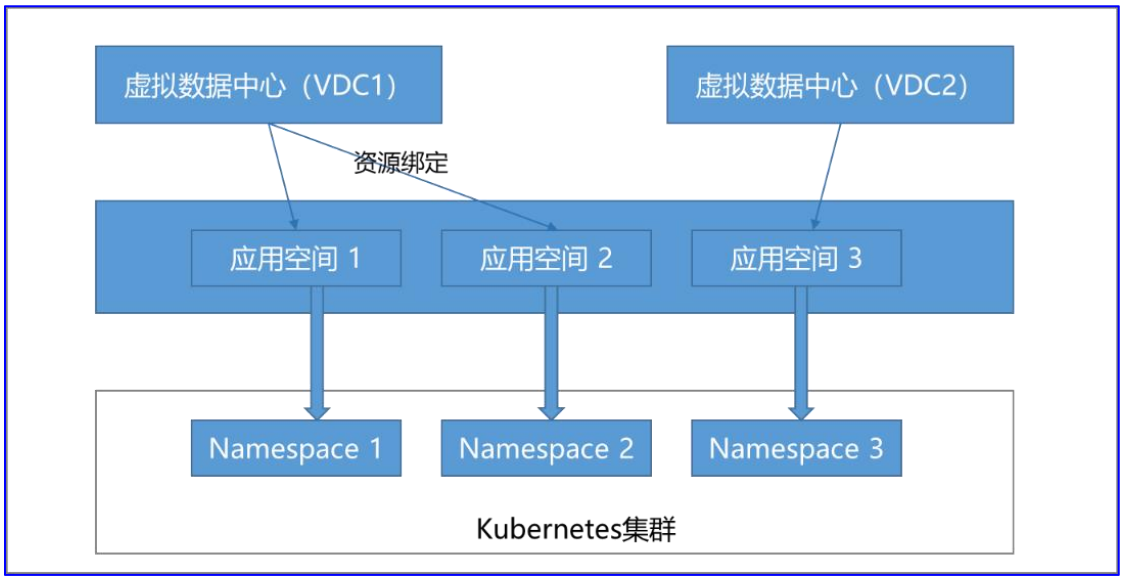
4.5.7 应用空间

4.5.7.1.1 功能描述

应用空间管理用于管理 **Kubernetes** 底层 **namespace** 资源，支持应用空间创建、删除以及绑定虚拟数据中心。支持通过该功能进行底层 **namespace** 与上层 **VDC** 绑定，支持创建空间时选择是否开启网络隔离。

4.5.7.1.2 技术原理

应用空间与 **Kubernetes** 底层 **namespace** 为一对一对应关系，通过 **kube-apiserver** 进行 **namespace** 创建与删除。



应用空间网络隔离

应用空间网络隔离基于 **k8s networkpolicy** 实现，其本质是根据 **networkpolicy** 出、入方向规则，在资源池节点上添加相对应的 **iptables** 规则，从而实现对应用空间内应用的访问控制。当选择开启网络隔离时，会自动生成 **1** 条 **networkpolicy**，并展示在网络管理>网络策略功能中。该条 **networkpolicy** 主要定义内容包括如下：

（**1**）实现同一应用空间内部应用间网络互通，不同应用空间应用间网络隔离。

（**2**）允许管理组件访问：允许带有 **accessAllNamespaces** 标签的 **namespace** 来访问，包括 **ingress-nginx/monitoring/istio-system** 等。

（**3**）开通 **dns** 服务：允许访问 **kube-system** 下的 **coredns** 及固定 **IP** 的 **nodelocaldns**

（**4**）允许访问 **istio-system** 下的组件，比如 **istiod**，以允许服务治理。

4.5.7.1.3 约束

无。

4.5.8 本地备份

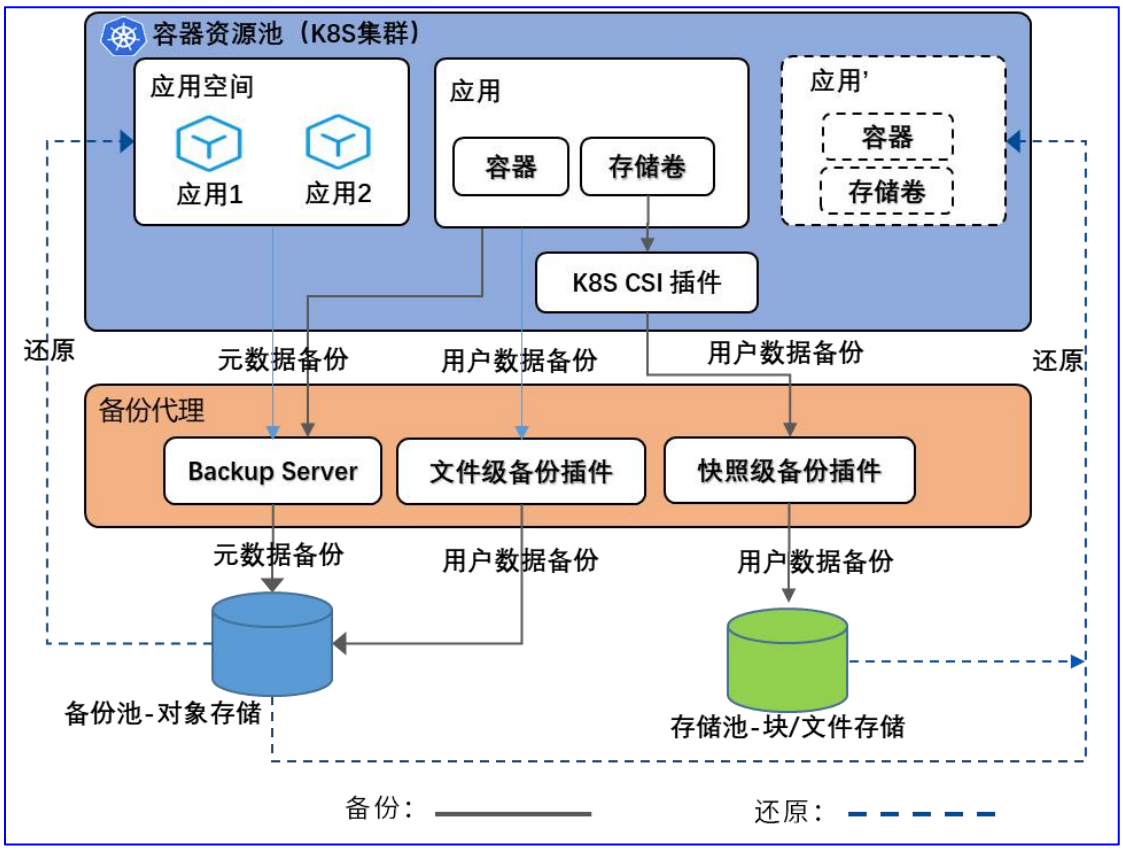
4.5.8.1 功能描述

越来越多的业务系统会部署到基于 **K8S** 的容器中，当业务系统在数据损坏、误删等异常场景时，如何快速恢复客户业务系统，并能够保留业务数据，是用户急迫的关键需求。本平台的容器应用备份恢复功能，能够将容器业务系统整体进行备份，包括元数据和业务数据，也可以对整个 **Namespace** 的业务系统进行备份，在需要时，能够进行一键快速恢复。同时，为了满足不同的业务场景，支持

只备份元数据或者卷数据，也可同时备份两者。

4.5.8.2 技术原理

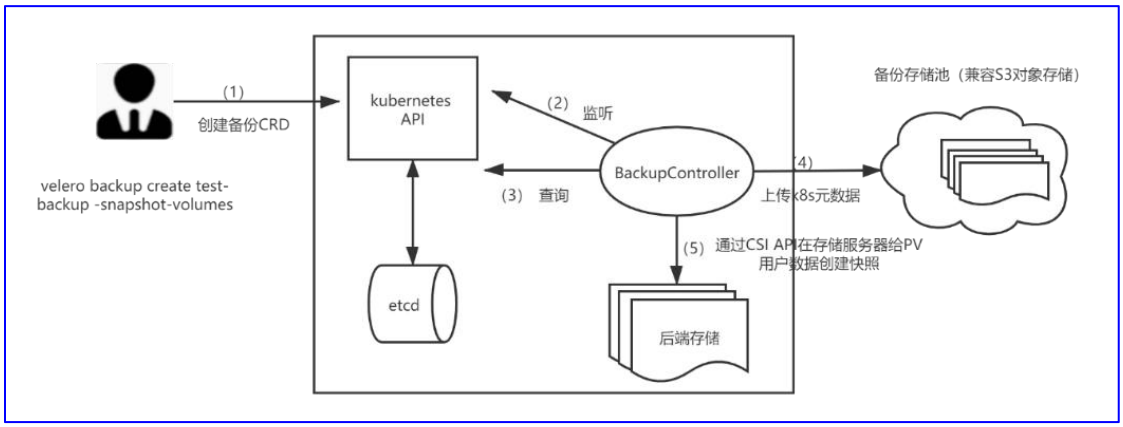
本平台的容器应用备份恢复能力是基于开源备份框架 **Velero** 和对象存储的技术路线实现，整体架构图如下图所示。能够基于存储快照能力和开源文件传输工具 **Restic**，实现存储快照级和文件级的容器持久卷备份。能够将应用元数据备份到对象存储，基于存储快照插件的备份将用户数据备份到存储系统，基于 **Restic** 文件传输插件的备份将用户数据备份到对象存储。



Velero+CSI 插件备份资源流程如下：

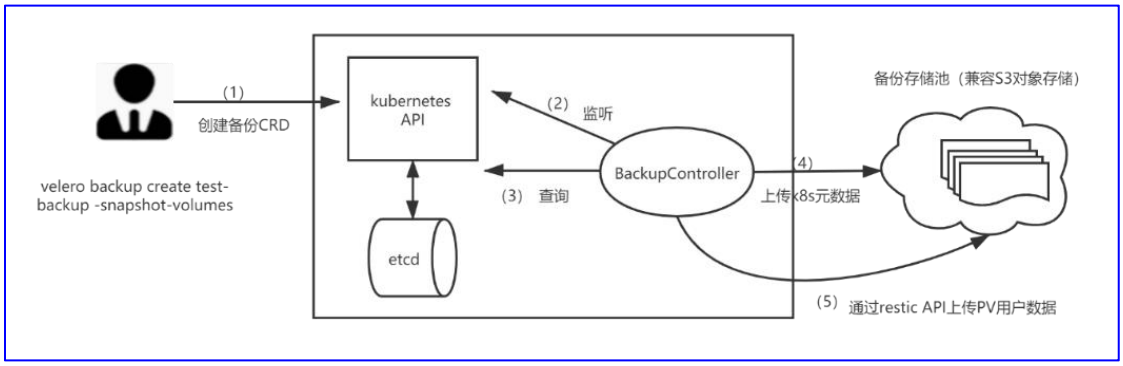
- 用户通过 **velero** 客户端工具发出资源备份命令，并创建备份 **CRD** 资源。
- 备份 **CRD** 资源控制器监听到备份 **CRD** 资源创建并进行校验，判断 **CRD** 资源合法性。

- 备份资源控制器通过 **k8s API** 查询需要备份的 **k8s** 元数据，并上传到备份存储池。
- 备份资源控制器调用 **CSI** 插件 **API** 在后端存储上给 **PV** 用户数据创建快照进行备份。



Velero+Restic 备份流程如下：

- 用户通过 **velero** 客户端工具发出资源备份命令，并创建备份 **CRD** 资源。
- 备份 **CRD** 资源控制器监听到备份 **CRD** 资源创建并进行校验，判断 **CRD** 资源合法性。
- 备份资源控制器通过 **k8s API** 查询需要备份的 **k8s** 元数据，并上传到备份存储池。
- 备份资源控制器通知 **restic** 需要备份的容器卷，**restic** 通过 **restic API** 将 **PV** 用户数据以文件的形式上传到备份存储池。



还原资源是备份资源的逆流程，还原资源的流程如下：

- 用户通过 **velero** 客户端工具发出资源还原命令，并创建还原 **CRD** 资源。
- 还原 **CRD** 资源控制器监听到还原 **CRD** 资源创建并进行校验，判断 **CRD** 资源合法性。
- 还原资源控制器从备份存储池获取 **k8s** 元数据，调用 **k8s API** 生成 **k8s** 集群资源。
- 如果是通过 **restic** 备份的 **PV** 用户数据，则还原控制器通过 **restic** 获取备份 **PV** 用户数据，并在容器组所在节点文件系统目录进行数据还原。
- 如果是通过 **CSI** 插件备份，则从后端存储获取 **PV** 快照，还原成容器组所需的 **PV** 用户数据。

4.5.8.3 约束

- 无。

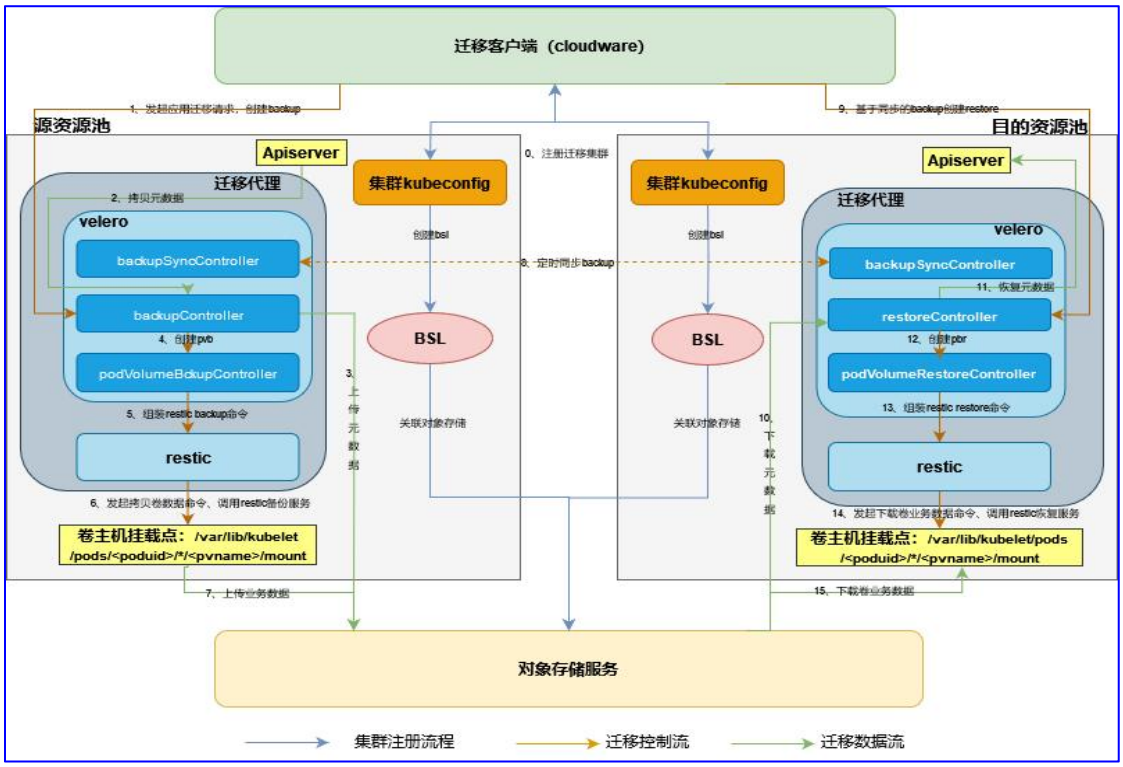
4.5.9 跨资源池迁移

4.5.9.1 功能描述

当生产环境需要跨大版本进行产品升级时，无法进行平滑升级，需要采用迁出、升级和迁回的手段进行，需要将老版本资源池中的应用迁移至新版本资源池。当多个资源池同时存在时，由于某个资源池负载过高影响当前资源池整体性能时，需要将负载高的资源池中某些应用迁移到负载较低的资源池中。在跨资源池迁移过程中，支持将源资源池中元数据（即 **K8S** 资源，如 **deployment** 等）和业务数据（容器卷数据）一致性的迁移到目的资源池，能够屏蔽跨版本间元数据差异，实现跨存储迁移业务数据。

4.5.9.2 技术原理

本平台的容器应用跨资源池迁移能力是基于开源备份框架 **velero+restic** 的技术路线实现，整体架构图如下图所示。容器应用迁移分为备份阶段和恢复阶段，备份阶段是基于 **velero** 将应用元数据和业务数据备份到平台或第三方对象存储，恢复阶段，将元数据和业务数据备份从对象存储拉取下载到目的资源池，并将应用恢复到目的资源池中。



跨资源池应用迁移流程如下：

- 将源、目的资源池注册到迁移客户端
- 迁移客户端发起迁移请求，并且创建 **backup** 资源
- **backupController** 根据 **backup** 配置的元数据名称以及命名空间从 **apiserver** 获取具体的元数据清单
- **backupController** 将元数据清单上传至对象存储

- **backupController** 根据备份的卷以及 **pod** 信息发起创建 **podVolumeBackup** 请求
- **podVolumeBackupController** 监听 **podVolumeBackup** 事件，并且创建 **podVolumeBackup**
- **podVolumeBackupController** 根据 **pvc** 的挂载点信息构建 **restic** 备份命令
- **restic** 将卷挂载点中的应用数据上传至对象存储
- **backupSyncController** 将 **backup** 从源资源池同步至目的资源池
- 迁移客户端根据同步的 **backup** 创建 **restore** 的请求
- **restoreController** 监听到 **restore** 从对象存储下载对应的元数据清单
- **restoreController** 根据元数据清单将元数据创建至目的资源池中
- **restoreController** 发起创建 **pvr** 的请求
- **pvrController** 监听到 **pvr** 构建 **restic** 恢复命令
- **restic** 关联卷主机挂载点
- 从对象存储服务下载应用数据恢复至卷主机挂载点

4.5.9.3 约束

- 源、目的资源池 **CNI** 插件一致。
- 源、目的资源池容器运行时一致。
- 源、目资源池存储卷使用的 **storageclass** 名称一致。
- 保证目的资源池可以访问源资源池镜像的镜像仓库。
- 只支持 **csi** 标准挂载点路径的存储以及本地存储卷。
- 迁移后，如果容器应用的镜像地址发生变化，需要手动修改

4.5.10 容器应用编排

4.5.10.1 功能描述

容器应用编排实现从容器应用的可视化编排、容器应用模板到应用商店的发布与下架以及通过应用模板创建应用实例的全周期的应用编排管理，用户可以通过拖拽的形式将需要的组件关联起来编排对应的蓝图，提高了编排文件编写的效率，解决了容器编排文件编写的繁琐性的问题，提高了用户上云的效率。

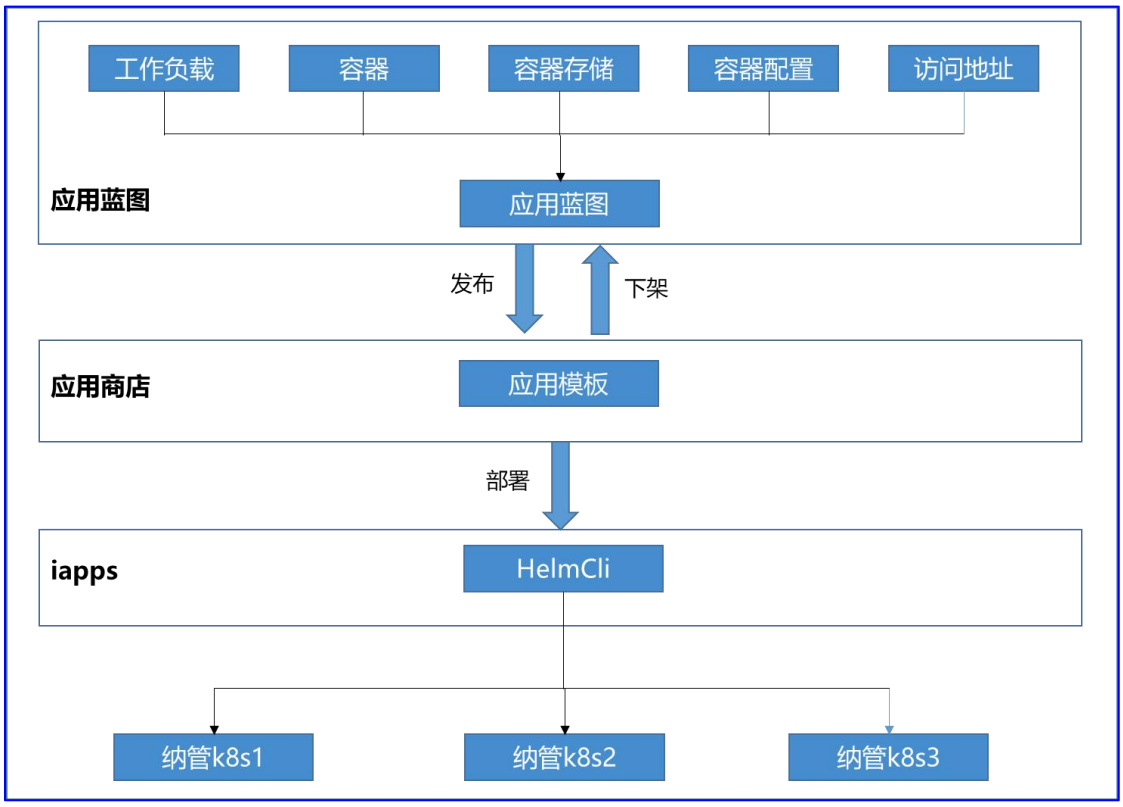
- 同时兼容 **Helm2**、**Helm3** 两种格式的 **Chart** 模板，能够自动识别并调用对应的执行命令。
- 容器编排支持对 **Kubernetes** 中各类资源，如：有状态负载、无状态负载、任务、定时任务、存储卷、普通配置、加密配置、服务等资源进行组合以及编排形成应用模板。
- 蓝图编排形成应用模板，支持应用模板发布到应用商店供用户一键式部署，支持应用商店应用模板的下架。
- 支持应用模板快速复制。

4.5.10.2 技术原理

应用蓝图服务通过蓝图管理器对应用蓝图实行全生命周期的管理，包括新建、复制、修改、删除等功能。

应用蓝图可拖拽的组件包括无状态负载、有状态负载、任务、定时任务、普通配置、加密配置、卷、应用访问地址、容器等组件。

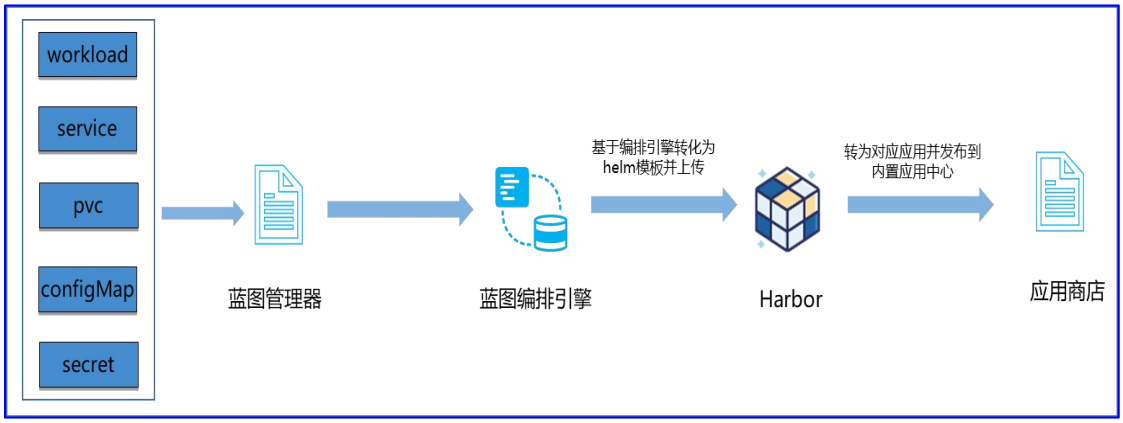
应用蓝图服务通过蓝图编排引擎将应用蓝图解析为 **helm** 模板并上传到 **Harbor** 进行持久化保存。



- 容器编排 **Helm** 模板生成：采用 **Helm3** 作为容器编排包管理工具。**Helm3** 是基于 **K8S** 原生 **api-server** 的应用编排。对纳管 **K8S** 集群无附加组件要求，容器平台对于应用部署时，采用 **K8S** 中的原生 **api-server**，在创建对应应用实例时，会基于纳管集群的鉴权信息调用其原生 **api-server** 接口创建对应资源，实现纳管集群与应用编排引擎解耦。
- 容器编排 **Helm** 模板生成整体流程如下：
- **UI** 拖拽式生成容器模板所包含 **Kubernetes** 资源，包括 **deployment** 等各 **workload**、**service**、**pvc** 等，以及各资源所涉及关键属性。
- 蓝图管理器将 **UI** 端选择的各资源进行整合调用应用蓝图服务。
- 蓝图编排引擎采用 **go** 模板引擎（**text/template** 库和 **html/template** 库）将前端应用模板中包含的各资源和属性转换为标准 **Helm** 模板。
- 应用蓝图服务调用 **Harbor** 接口将 **Helm** 模板上传到 **Harbor** 仓库进行

存储。

◆ 应用商店调用 **Harbor** 接口展示各容器应用模板，供用户一键式部署。



4.5.10.3 约束

容器编排服务服务需要容器服务正常并已存在对应的应用商店。

4.5.11 应用包管理

4.5.11.1 功能描述

应用包管理服务提供应用包的上传、修改、更新、列表展示等功能，并且支持将应用包直接部署成应用。

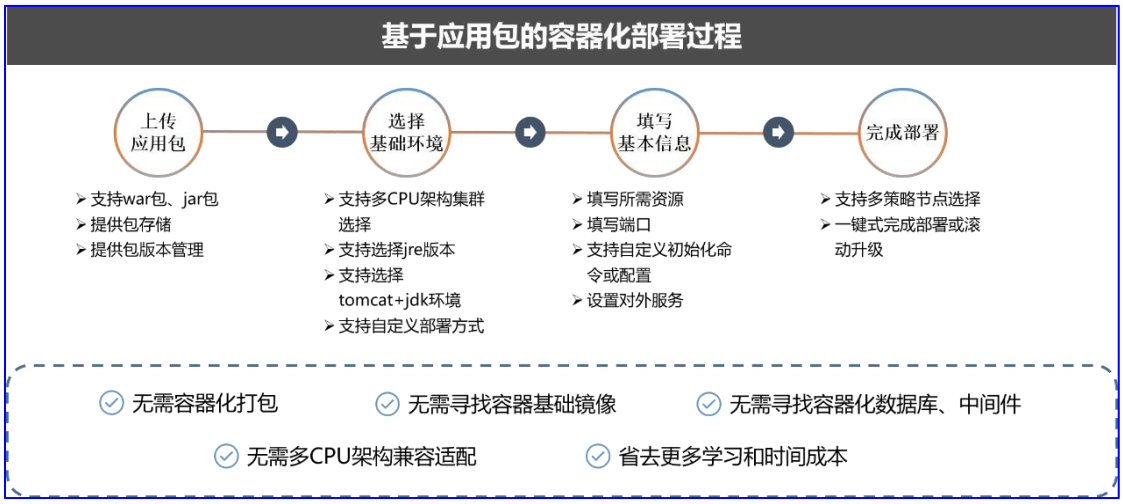
应用包管理服务提供可视化的包管理界面，用户可根据自身需求将其项目编译打包成 **Jar** 包或者 **War** 包上传到系统进行管理，并可直接部署成应用，进行后续的应用工作负载组件、访问地址或安全组设置、资源审计等一系列功能的整合。满足了用户在特殊场景下将现有的项目快速部署的需求，大大加速了企业整体容器化进程。

应用包管理功能包括：

应用包管理：上传、修改、更新、删除。应用包上传支持大文件应用包的分片上传。

应用包部署：支持设置应用信息、组件信息、资源审计、访问地址、安全组设置。

4.5.11.2 技术原理



容器资源池提供基于应用包的容器化部署方案。应用开发者只需提供 **War** 包、**Jar** 包，即可通过容器资源池实现包管理、多 **CPU** 架构下的一键式容器化部署与更新，免去了应用容器化改造的诸多复杂过程，助力企业实现传统应用快速上云。

4.5.11.3 约束

虚拟数据中心需要在容器集群上绑定资源后才可以上传应用包和部署应用。

4.5.12 容器资源池纳管

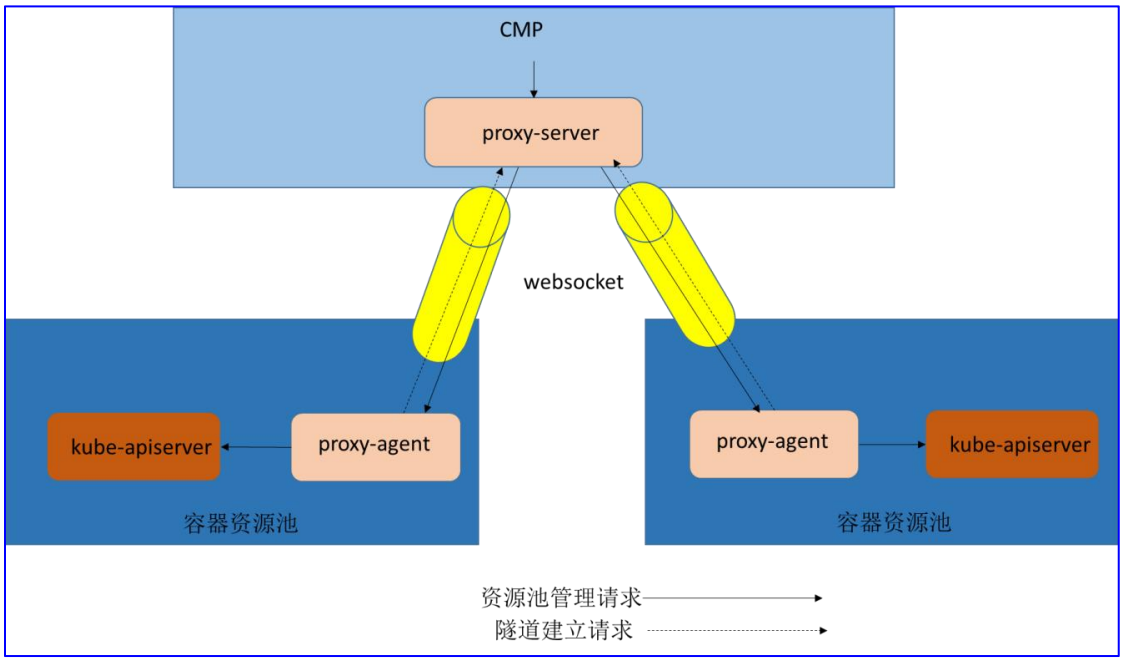
4.5.12.1 功能描述

在云平台中，容器资源池是一个 **K8S** 集群，需要纳管到 **CMP** 管理平台进行统一管理和使用，纳管的方式有两种，包括直连（也称为直接）纳管和代理纳管。如果 **CMP** 可以直接访问 **K8S** 集群 **kube-apiserver** 地址，则可以采用直连纳管。

如果 CMP 无法直接访问集群 kube-apiserver 地址，但是容器资源池可以访问 CMP 地址，则可以采用代理进行纳管。通过在 CMP 管理平台中暴露资源池纳管的代理服务地址，让资源池通过代理连接到 CMP。当容器资源池部署在虚拟化环境（CMP 管理节点无法直接访问虚拟机 IP 地址，但虚拟机可以通过 NAT 访问 CMP 地址）或者边缘环境（资源池处于一个局域网内，与 CMP 管理节点处于不同的局域网内）时，适用采用代理纳管。

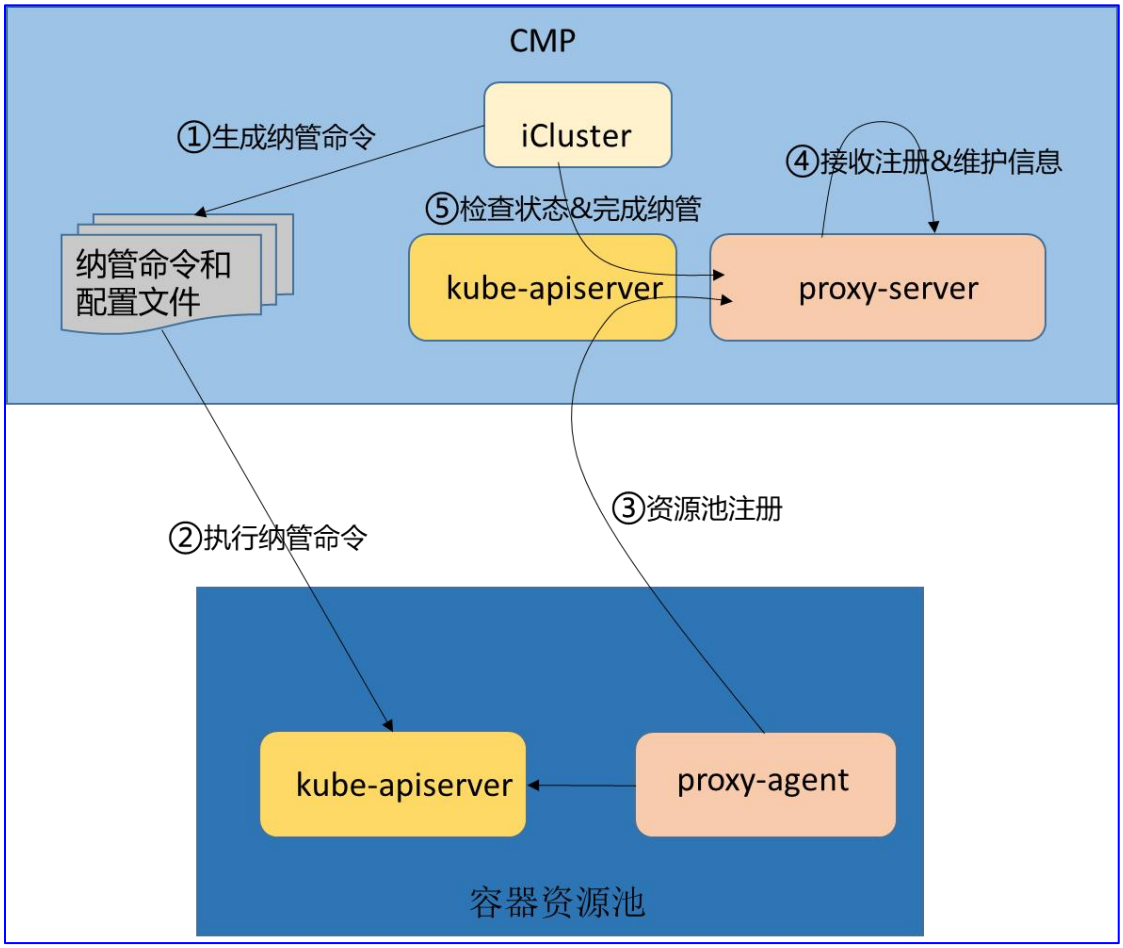
4.5.12.2 技术原理

通过代理导入资源池的核心技术原理为：通过代理在 CMP 和容器资源池之间建立一个网络通道，CMP 通过该通道发送管理请求到容器资源池，CMP 部署在 K8S 上，有时也称 CMP 为管理集群，容器资源池为工作集群。代理组件包括服务端 proxy-server 和客户端 proxy-agent 两部分，proxy-server 暴露资源池导入服务的端口，proxy-agent 主动发起访问该端口，并建立 websocket 网络通道，proxy-server 部署于 CMP 所在的节点内，proxy-agent 部署于容器资源池内，整体技术架构图如下图所示。每个容器资源池与 CMP 中 proxy-server 建立连接后，会在 proxy-server 内维护一个 session 池，并通过标识唯一标记一个容器资源池，在请求某个资源池的 kube-apiserver 时，可以根据标识转发到对应的 session，进而转发到对应资源池的 agent 内，agent 转发到 kube-apiserver。



◆ 容器资源池纳管流程

通过代理纳管容器资源池的核心流程如下图所示，具体的纳管流程包括：



1、生成纳管命令：通过容器资源池管理模块 **iCluster** 生成纳管命令以及相

关的纳管文件，纳管文件包括部署 **proxy-agent** 的 **YAML** 文件、接入服务地址和 **token** 配置文件，每个容器资源池都有自己的纳管文件。

2、执行纳管命令：获取步骤 **1** 中生成的纳管命令，到可以执行容器资源池 **kubectl** 命令的节点执行命令。

3、资源池注册：纳管命令执行后，会将 **proxy-agent** 部署到容器资源池中，**proxy-agent** 根据接入服务地址和 **token** 配置文件内容，主动向 **CMP** 中 **proxy-server** 注册。

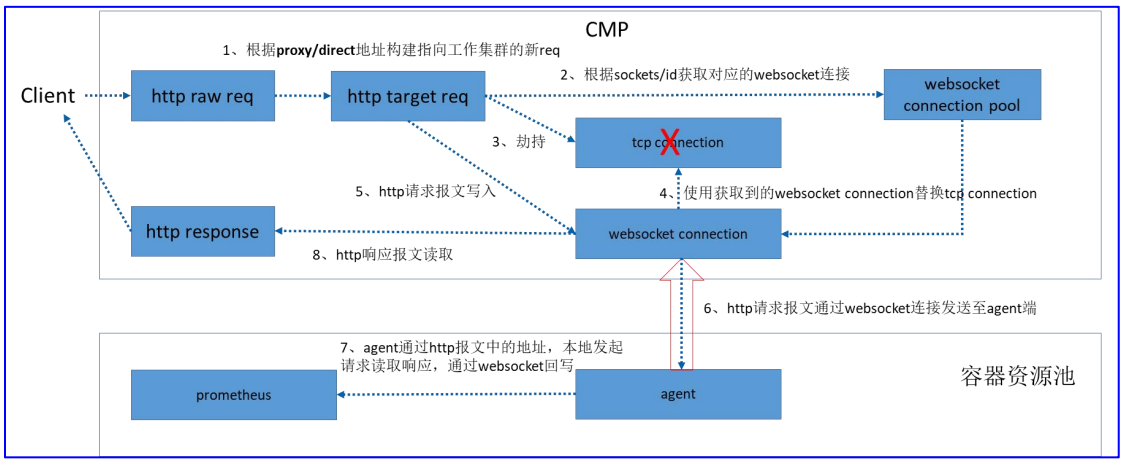
4、接收注册：**proxy-server** 对 **proxy-agent** 认证成功后，将其对应的容器资源池保存到 **session** 池中。

5、完成纳管：**iCluster** 模块检查 **proxy-server** 中 **session** 中容器资源池的状态，将接入成功的资源池 **kubeconfig** 信息保存到 **iCluster** 中，以后续容器资源池管理请求使用。

◆ 通过代理访问容器资源池流程

直连纳管的容器资源池在 **CMP** 中访问 **kube-apiserver** 的形式为 **IP:Port**，比如容器资源池 **IP** 地址为 **192.168.1.1**，**CMP** 的 **IP** 地址为 **172.16.1.1**，则 **CMP** 访问容器资源池的形式是直接访问 **IP:Port**，即 **https://192.168.1.1:6443**。通过代理纳管的容器资源池在 **CMP** 中访问其 **kube-apiserver** 的地址变为代理地址，通过 **CMP** 的 **IP** 地址进行代理，具体如 **https://172.16.1.1:8443/apis/proxies.clusternet.io/v1alpha1/sockets/32937fa2-ec7b-4d3b-9f19-44c574925244/proxy/direct**。后者代理的访问 **API** 是通过 **K8S** 的 **API** 聚合特性将通过代理访问的 **kube-apiserver** 注册到 **CMP** 中底座 **K8S** 的 **APIServer** 中。请求一个代理后的容器资源池的 **kube-apiserver** 地址的请求的通

信流程如下图所示。



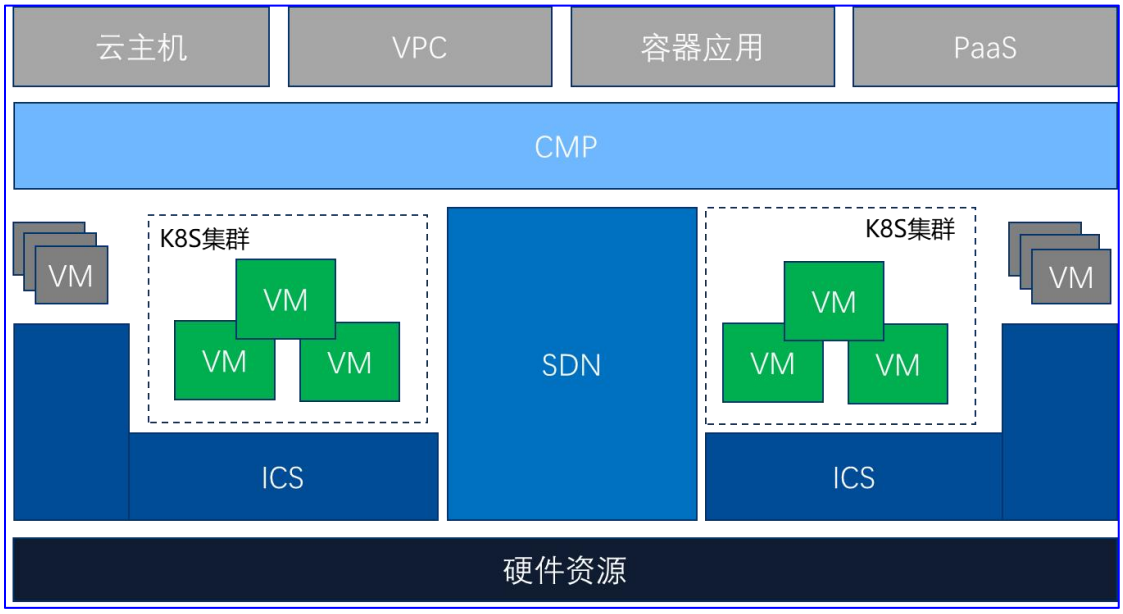
4.5.12.3 约束

无

4.5.13 轻量容器资源池

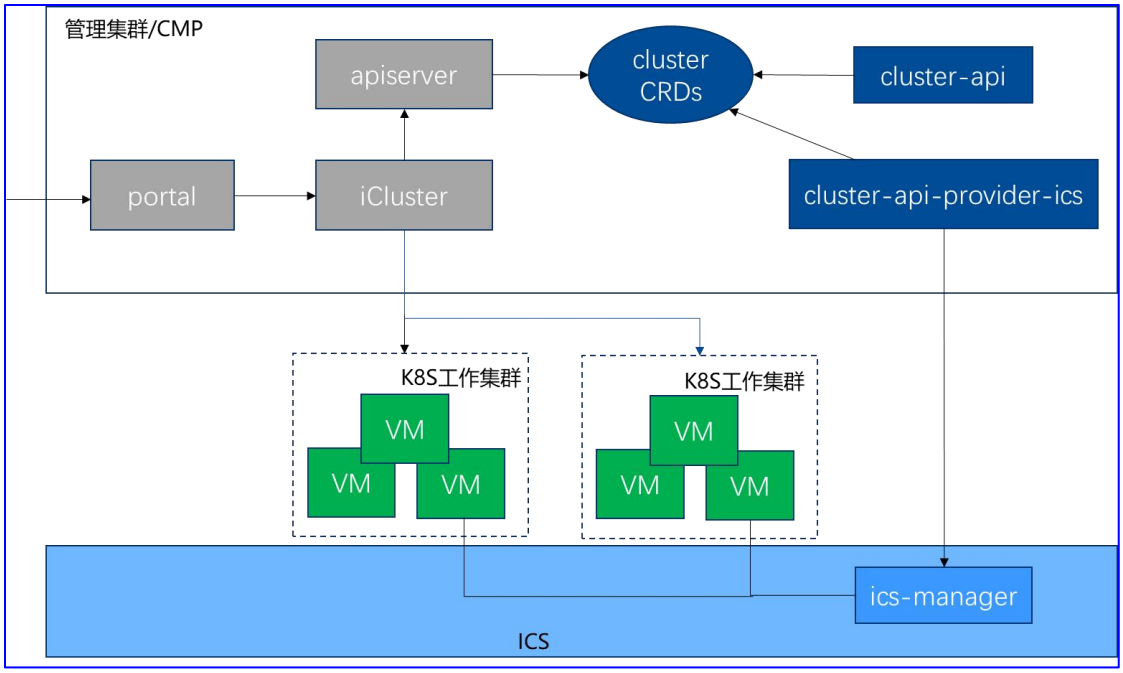
4.5.13.1 功能描述

轻量容器资源池为部署在虚拟化资源池虚拟机上的容器集群，使用虚拟化存储池提供数据持久化，利用 **SDN** 为容器集群内 **Pod** 和 **Service** 提供网络服务实现网络通信，达到虚拟机与容器统一 **VPC** 网络，通过 **CMP** 纳管进行统一管理。具体操作上，系统管理员纳管虚拟化资源池后，通过控制台选择虚拟化资源池创建容器集群，并自动将其纳管为容器资源池，并将资源池绑定到某个 **VDC** 的方式分配给租户独享使用。



4.5.13.2 技术原理

轻量容器资源池生命周期管理的整体技术架构如下图所示，基于自研组件 **iCluster** 和云原生多集群管理框架 **cluster-api** 实现。整体流程为用户通过 **portal** 发起集群创建请求，后端组件 **iCluster** 将创建请求规格参数转换为管理集群中 **CRDs**，**cluster-api** 和 **cluster-api-provider-ics** 组件监听 **CRDs** 在 **ICS** 中创建部署容器集群使用的云主机。云主机启动后，通过 **Cloudinit** 初始化脚本完成容器集群组件的部署，下面是一些关键组件的详细设计和描述。



- **portal**：多集群管理的前端 **UI** 组件，负责接受用户的集群管理请求，其中关键操作为集群创建流程，该组件需要与计算、存储和网络组件交互，查询虚拟化资源池和 **VPC** 等信息，以供用户创建集群时选择具体的规格参数。
- **iCluster**：多容器集群管理的核心后端组件，接受 **portal** 传递的请求，根据请求的具体规格参数创建集群。解析参数后，将参数转换为管理集群的 **CRDs**，包括 **Cluster**、**ControlPlane** 和 **MachineDeployment**。同时，**iCluster** 负责集群创建流程的监控，当集群创建完成或者失败后，更新集群的状态。
- **cluster-api**：多集群管理框架的核心模块，定义 **Cluster**、**ControlPlane** 和 **MachineDeployment**、**Machine** 等 **CRD** 以及对应的 **Controller**，通过监听集群、控制平面和计算节点对应的 **CRD** 资源，创建 **Machine** 资源，并分配角色和对应的配置信息 **BootstrapConfig**。
- **cluster-api-provider-ics**：按照 **cluster-api** 框架规则实现的 **IaaS** 层的

资源管理模块，定义 **ICSCluster**、**ICSMachine** 和 **ICSMachineTemplate** 等 **CRDs** 以及对应的 **Controller**。**Controller** 监听到 **Machine** 资源状态变动，创建 **ICSMachine** 资源，**controller** 监听到 **ICSMachine** 资源变动后，调用 **ics-manager** 创建云主机资源。虚拟机启动后，通过 **cloudinit** 调用 **BootstrapConfig** 脚本实现容器集群的组件启动并组成集群。

由于虚拟化环境中，处于安全隔离要求，云主机与 **CMP** 所在网络一般处于不同网段中，需要为资源池所在的虚拟机设置双网卡，即管理网和业务网，管理网为全局网络，需要提前配置与 **CMP** 能够通信，在容器集群创建完成后，为集群所在虚拟机的管理网卡设置静态路由，与 **CMP** 通信。通过 **CNI** 插件 **iveth** 实现 **Pod** 和 **Service** 与虚拟机的统一 **VPC** 网络，通过 **CSI** 插件 **ics-csi** 对接虚拟化存储池，实现容器应用数据持久化。

4.5.13.3 约束

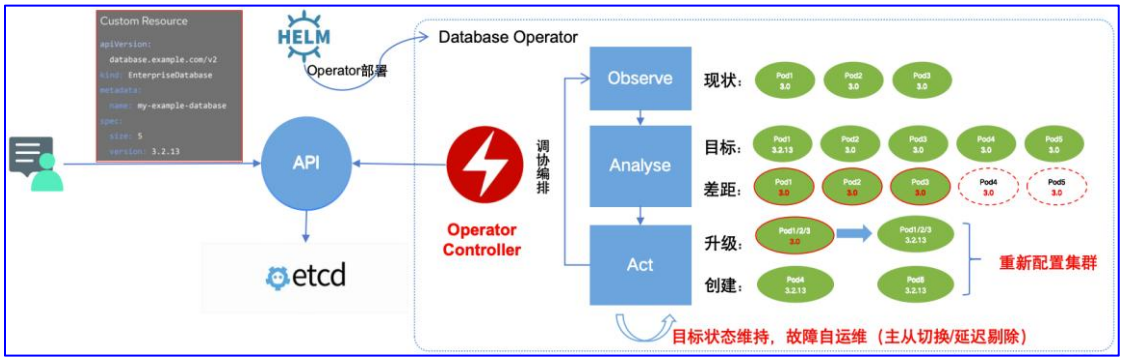
在创建资源池选择 **VPC** 网络模式时，仅支持选择开启 **DHCP** 的 **VPC** 子网，创建资源池时，虚拟机仅支持本地存储池和 **CFS** 存储池两种。

4.6 PaaS 服务

4.6.1 PaaS 技术架构

平台基于云原生 **Operator** 技术，为用户提供开箱即用的数据库/中间件全生命周期管理服务和自运维能力，支持通过平台界面便捷的进行服务的创建、扩缩容、重启、参数改配、备份恢复、对接块和本地存储、监控告警、日志查询等操作。

Operator 是一种以高效、自动化、可扩展的方式开发/构建 **K8s** 原生应用的设计模式和开发框架，主要由 **CRD+Crontroller** 组成，核心理念是通过扩展的 **K8s** 对象定义应用状态，由 **Controller** 替代运维工程师或 **SRE** 工程师来创建、配置和管理复杂的应用程序。使用者只需声明应用的期望状态，**Operator** 根据声明自动调谐应用到期望的状态，并通过持续监听应用运行状态的变化，维持目标状态以及故障自运维。



Operator 内部封装的管理逻辑的复杂程度各有不同，但对于大部分 **Operator** 可能包含的特定功能集来说,从成熟度上可以分为 **5** 个级别:基本安装、无缝升级、全生命周期、深度洞察、自运维。

4.6.2 MySql

4.6.2.1 功能描述

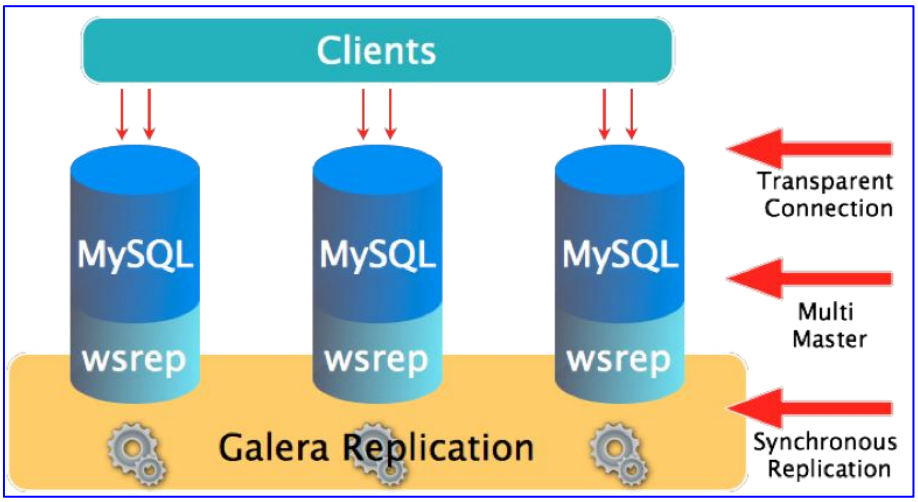
MySQL 是一个关系型数据库管理系统（**Relational Database Service**，简称 **RDS**），提供了关系型数据的存储能力，采用行存的形式进行数据存储，基于锁的机制可以保证较好的事务性，同时对于分布式的良好支持，可以在成本低廉的

同时实现大量数据的存储，适用于各类非海量数据，增、删、改、查频繁，事务性要求比较高的 **OLTP** 业务场景。

平台提供租户隔离的 **MySql** 数据库实例，允许用户创建属于自己的服务实例，并可对每个服务实例进行启停、配置修改和删除等管理操作。基于 **Galera** 协议支持多主集群的可视化创建、扩容、缩容、删除，支持全量备份、增量备份和基于时间点还原，支持一主多备、多主和读写分离，支持数据库集群参数模板管理（如 **binglog** 保存时效、执行队列线程数等），支持多节点读写数据强一致，可对接第三方存储。可以通过可视化界面查看数据库资源的使用情况及告警，支持连接数、连接错误数、活跃线程数、异常连接数、慢查询数等关键数据库指标的监控展示与告警。

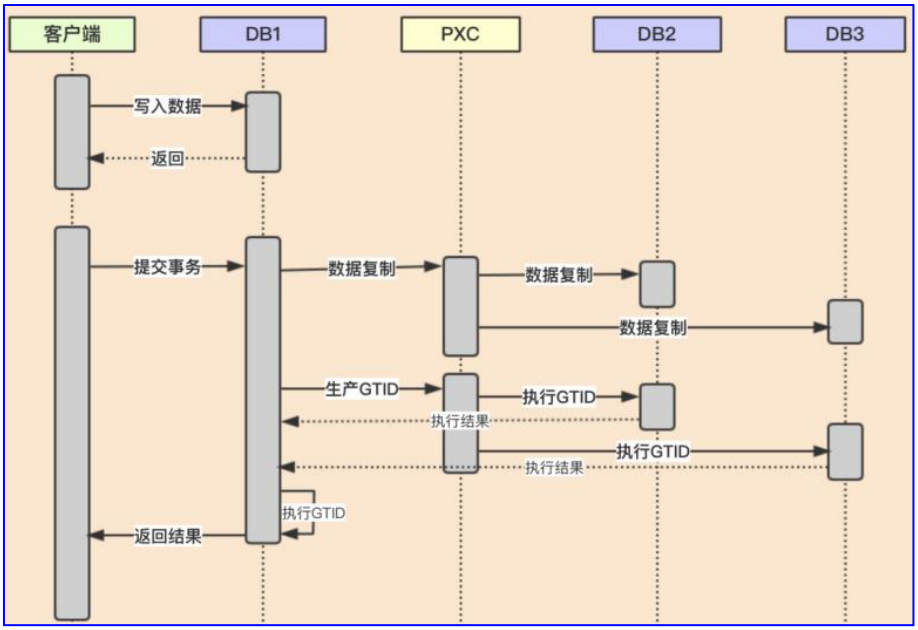
4.6.2.2 技术原理

PXC（**Percona XtraDB Cluster**）是一个开源的 **MySQL** 高可用解决方案。它将 **Percona server** 和 **XtraBackup** 与 **Galera** 库集成，以实现同步多主复制。相比传统的基于异步复制的主从高可用架构，**PXC** 是在存储引擎层实现了同步复制而非异步复制，所以其数据的一致性是相当高的，而且不同 **PXC MySql Server** 之间的关系是对等的，是一种多主架构。

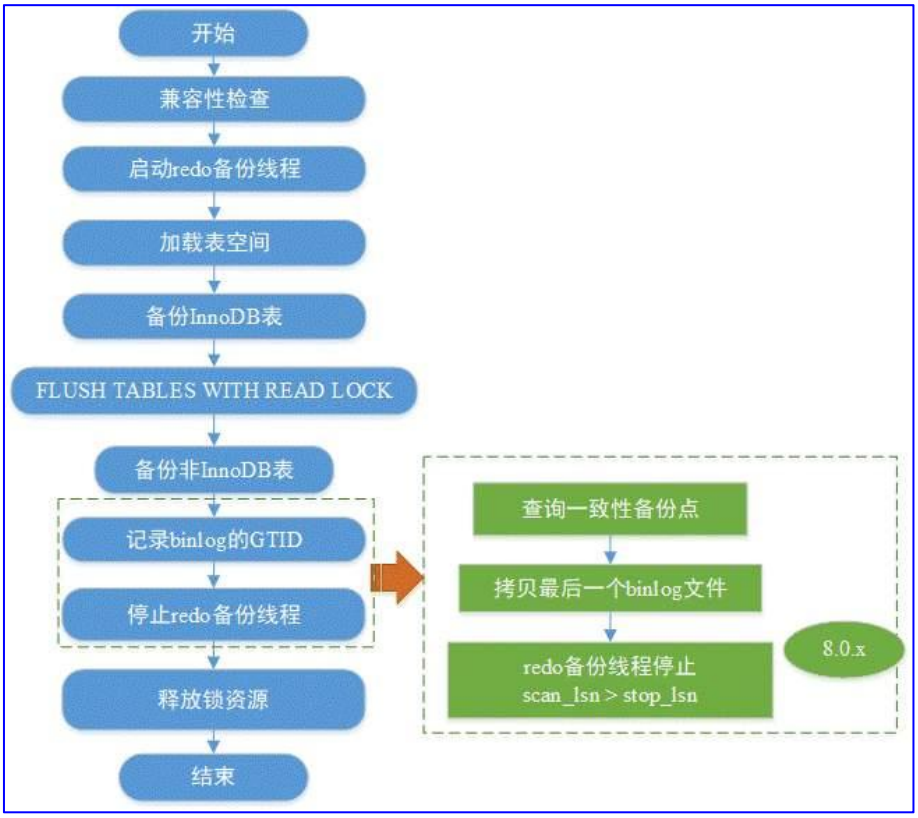


- 同步复制，主备无延迟
- 支持多主同时读写，保证数据一致性
- 集群中各节点保存全量数据
- 节点添加/删除，自动检测和配置
- 行级别并行复制
- 不需要写 **binlog**

当客户端发送指令到一个数据库时，数据库将数据修改后返回一个 **OK**，表示用户的请求已经被收到，但是事务并没有结束依旧可以撤销。当事务结束时用户发送一个 **commit**(提交)，服务器收到后会将数据的更新发送给其他的 **MySQLServer**，此时会开启一个全局性的事务 **ID** 给组内的其他服务器，其他服务器会查看是否有事务冲突，如果没有冲突就确认更改成功。



Xtrabackup 是生产级的 **MySQL** 数据库物理热备份工具，具有备份速度快、支持备份数据压缩、自动校验备份数据、支持流式输出、备份过程中几乎不影响业务等特点，是目前各个云厂商普遍使用的 **MySQL** 备份工具。其备份原理如下：



- 兼容性检查：**Xtrabackup** 支持 **MyISAM , InnoDB , CSV , MRG_MYISAM** 四种存储引擎的表，其他存储引擎的表不会备份；通

过查询 **tables**，若发现表的存储引擎不是上述四种引擎之一，打印 **warning, Xtrabackup** 不会备份该表。

- 启动 **redo** 后台备份线程：启动 **redo** 后台备份线程，从备份实例的最近一次 **checkpoint LSN** 的位置开始备份所有增量的 **redo log**，一直持续到备份结束。
- 加载所有的 **innodb** 表空间：打开并扫描所有 **innodb** 表的数据文件，检查所有表空间的第一个页面，初始化所有表的内存结构。
- 备份 **innodb** 表：遍历步骤 **3** 所构建的表的内存结构，备份每一个 **innodb** 表的数据文件，备份的过程中会检查每个页面的数据是否正确。
- 加备份锁 **FLUSH TABLES WITH READ LOCK (FTWRL)**：**FTWRL** 锁是 **MySQL** 实例级的读锁，加锁之后表的所有更新操作以及 **DDL** 都会堵塞。
- 备份非 **innodb** 表：因为在步骤 **5** 我们已经对实例加了读锁，因此，此时备份非 **innodb** 表是安全的，此时一定没有写业务。
- 记录 **binlog** 当前的 **GTID** 信息：请注意，此时我们仍持有全局读锁。这一步主要是方便我们使用该备份集快速地创建出备机。
- 停止 **redo** 备份线程，释放锁资源，备份结束。

4.6.2.3 约束

无

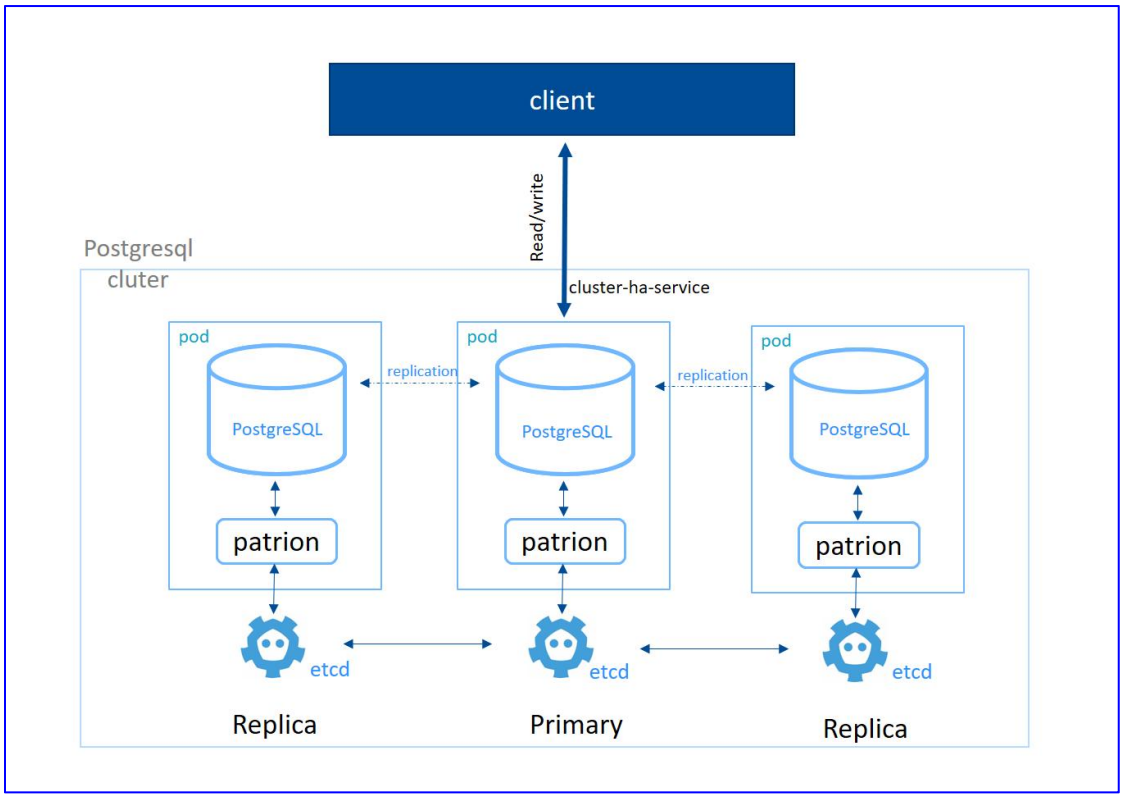
4.6.3 PostgreSQL

4.6.3.1 功能描述

PostgreSQL是一种强大的开源关系型数据库管理系统**(RDBMS)**,功能强大，支持丰富的数据类型，包括数组、**JSON**、**XML** 等复杂数据类型。提供完善的事务处理机制，确保数据的一致性和可靠性，扩展性高，可以通过自定义函数、存储过程和扩展插件来满足各种特定的业务需求，性能优异，具备优秀的索引机制和查询优化器，能够高效处理大规模数据

平台提供高可用 **PostgreSQL** 服务实例，采用一主多备的高可用架构，支持同步异步复制，支持自动故障切换。支持连接池管理能力，可以重用现有连接，降低数据库连接开销。支持数据库加密访问，保证数据安全。另外还集成了备份恢复等重要功能，减轻运维压力。

4.6.3.2 技术原理



Patroni 是一个功能全面、灵活易用的 **PostgreSQL** 高可用性解决方案，主持一主多备，支持同步异步复制，支持同步复制下备库故障时自动降级为异步复制，支持故障转移和主备切换，支持控制指定节点是否参与选主，是否参与负载均衡以及是否可以成为同步备机，支持多种常见 **DCS(Distributed Configuration Store)**存储元数据，包括 **etcd**，**ZooKeeper**，**Consul**，**Kubernetes**。

4.6.3.3 约束

无。

4.6.4 MongoDB

4.6.4.1 功能描述

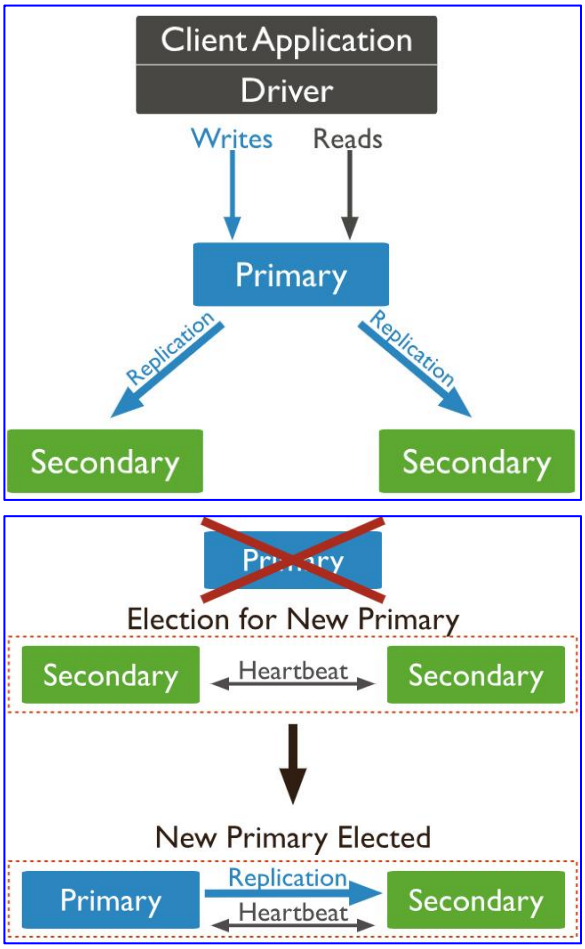
MongoDB 是一款高性能、高扩展的开源 **NoSQL** 文档数据库，主要用于存

储和管理大规模的非结构化数据，支持丰富的查询操作，**MongoDB** 几乎支持 **SQL** 中的大部分查询；支持索引，可以在任意属性上建立索引；支持聚合，可以使用 **MapReduce** 完成复杂的聚合任务；使用高效的二进制数据存储。

平台提供高可用 **MongoDB** 服务实例，采用一主多备的高可用架构，支持故障自动切换，备份恢复，监控运维等功能。

4.6.4.2 技术原理

平台提供 **MongoDB** 主从高可用集群，由一个主节点和多个从节点组成。主节点负责处理所有的写操作，而从节点则负责处理读取操作，这样可以分散负载并提高查询的响应速度。此外，主从集群还具备数据冗余备份的特点，从而确保数据的高可用性



- 高可用性：主节点负责处理写操作，多个从节点用于读取数据，
- 数据冗余：副本集中的多个节点上都有备份，防止数据丢失
- 故障恢复：主节点故障自动选举新主节点，确保服务持续可用
- 数据一致性：提供不同的读写关注级别，以平衡一致性和可用性
- 备份恢复：支持在线数据库备份，支持一键将数据库恢复到指定备份点

4.6.4.3 约束

无

4.6.5 Redis

4.6.5.1 功能描述

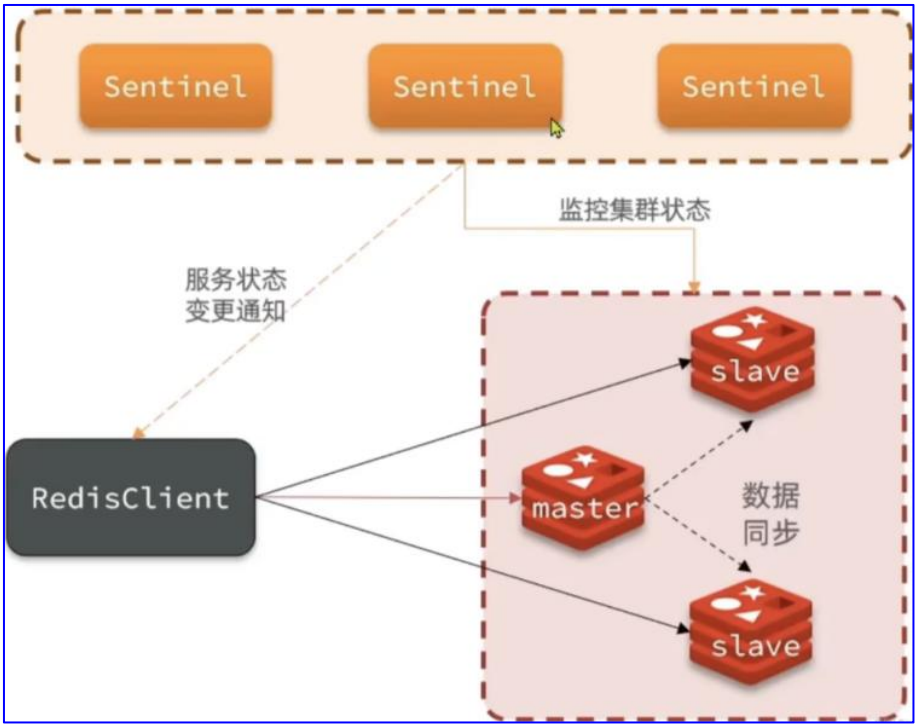
Redis 是一个高性能的 **key–value** 数据库。支持存储多种 **value** 类型，包括 **string**(字符串)、**list**(链表)、**set**(集合)、**zset**(**sorted set ––**有序集合)和 **hash**（哈希类型）。这些数据类型都支持 **push/pop**、**add/remove** 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，**redis** 支持各种不同方式的排序。为了保证效率，数据都是缓存在内存中。**Redis** 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 **master–slave**(主从)同步。

平台提供租户隔离的分布式缓存实例，允许用户创建属于自己的服务实例，并可对每个服务实例进行启停、配置修改和删除等管理操作。具备 **Redis** 哨兵集群参数的界面配置能力，支持实例的扩缩容、参数的预配置及自定义配置、用户密码自定义配置。可以通过可视化界面查看资源的使用情况及告警。支持集群关键指标（含平均响应请求时间、平均每秒处理请求总数、缓存命中率、连接总数

等）的监控展示与告警。

4.6.5.2 技术原理

Redis 提供了哨兵**(Sentinel)**机制来实现主从集群的自动故障恢复。哨兵的结构和作用如下**:**



- 监控**: Sentinel** 会不断检查 **master** 和 **slave** 实例是否按预期工作
- 自动故障恢复**:** 如果 **master** 故障**,Sentinel** 会将一个 **slave** 提升为 **master**。当故障实例恢复后也以新的 **master** 为主
- 通知**: Sentinel** 充当 **Redis** 客户端的服务发现来源，当集群发生故障转移时，会将最新信息推送给 **Redis** 的客户端

4.6.5.3 约束

无

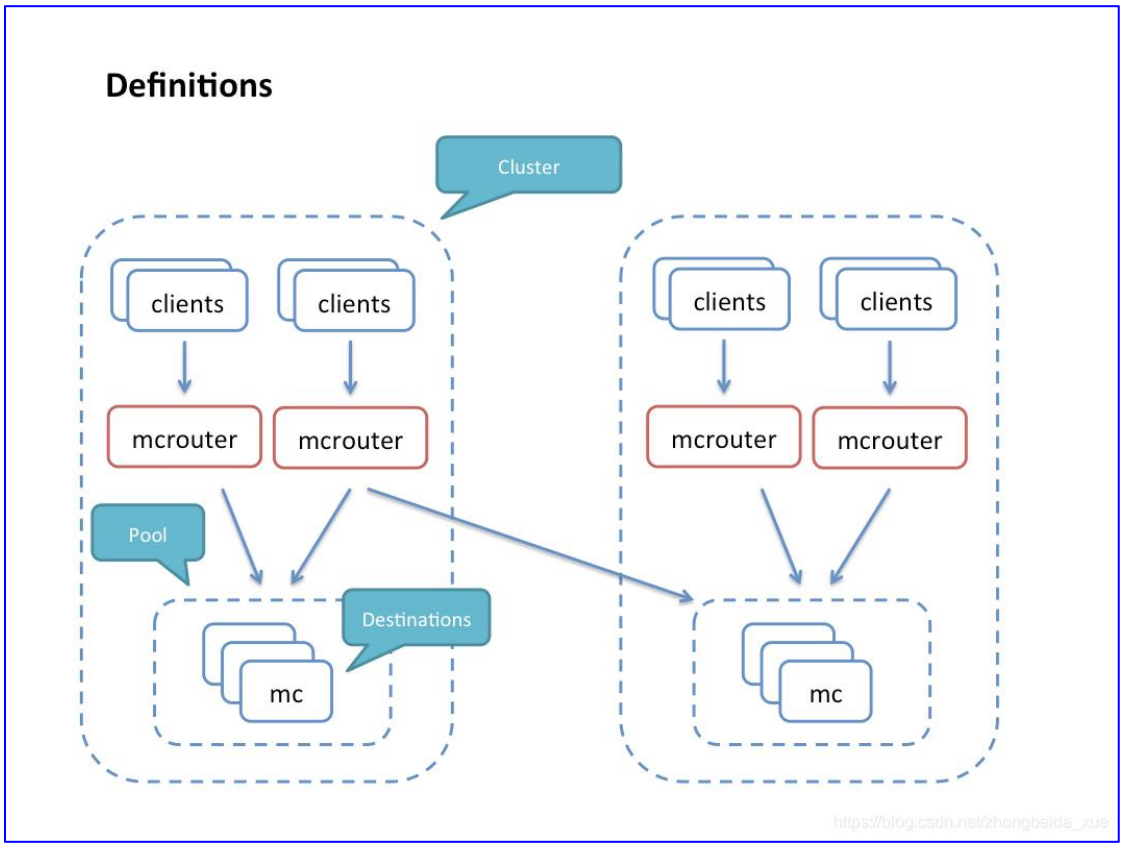
4.6.6 Memcache

4.6.6.1 功能描述

Memcache 是一个高性能的分布式的内存对象缓存系统，通过在内存里维护一个统一的巨大的 **hash** 表，它能够用来存储各种格式的数据，包括图像、视频、文件以及数据库检索的结果等。应用优先查询在内存中缓存数据和对象，来减少读取数据库的次数。从而分担数据库负的压力，提升 **Web** 等应用速度。平台提供高可用的 **Memcache** 集群生命周期管理能力。

4.6.6.2 技术原理

mcrouter 基于 **Memcached** 协议构建，为 **Memchache** 提供的高可用能力，在后端 **Memcached** 发生故障时仍能保持服务的连续性和稳定性。



◆ 连接池共享：客户端可以通过 **Mcrouter** 共享后端 **Memcached** 的连接

池，减少连接个数，提高效率

- 一致性哈希：**Mcrouter** 提供了一个非常有效的一致性哈希算法，允许给多个 **Memcached** 实例分配哈希值
- 热加载配置：**Mcrouter** 支持热加载配置文件，监控配置文件的修改，并自动重新加载和分析这些文件，根据新配置处理新请求
- 心跳检测和自动故障转移：**Mcrouter** 支持目的主机的心跳检测和自动故障转移，能够检测每个目的主机的心跳

4.6.6.3 约束

无

4.6.7 RabbitMQ

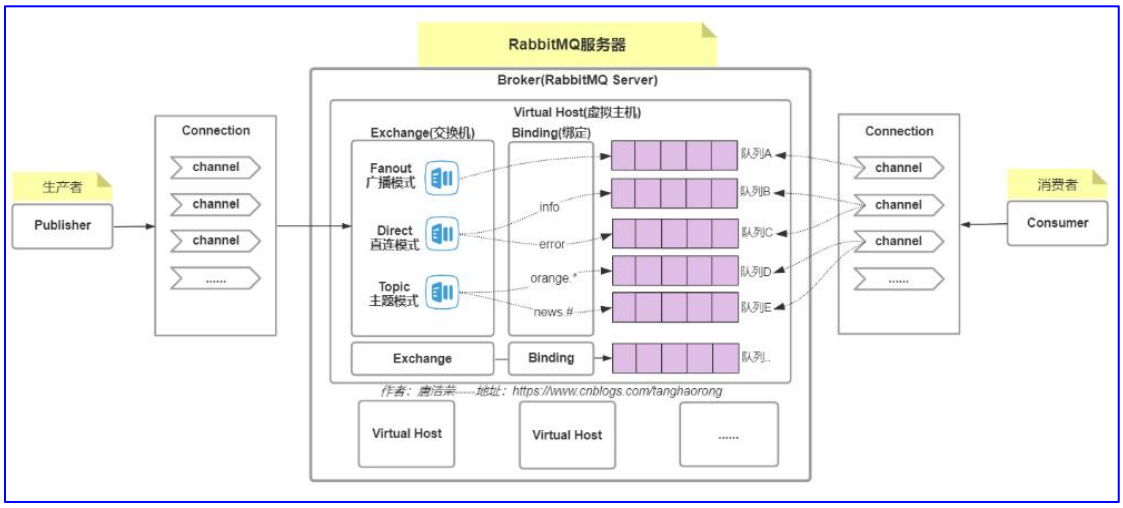
4.6.7.1 功能描述

RabbitMQ 是基于 **AMQP(Advanced Message Queue** 高级消息队列协议) 协议实现的消息队列，它是一种应用程序之间的通信方法，利用高效可靠的消息传递机制进行与平台无关的数据交流，并基于数据通信来进行分布式系统的集成。通过提供消息传递和消息排队模型，它可以在分布式环境下提供应用解耦、弹性伸缩、冗余存储、流量削峰、异步通信、数据同步等功能，其作为分布式系统架构中的一个重要组件，有着举足轻重的地位。

平台提供租户隔离的消息中间件实例，允许用户创建属于自己的服务实例，并可对每个服务实例进行启停、配置修改和删除等管理操作。无单点故障，支持水平扩展，支持 **TCP**、**HTTP** 等多种协议接入，支持最大通道数、队列策略等参数的预配置及自定义配置，支持用户名密码自定义配置。可以通过可视化界面查

看资源的使用情况及告警。支持集群关键指标（含连接数、队列数、生产者总数、消费者总数、通道总数等）的监控展示与告警。

4.6.7.2 技术原理



- **Publisher**：消息生产者，就是投递消息的程序，也是一个向交换机发布消息的客户端应用程序。生产者发送的消息一般包含两个部分：消息体和内容标签。
- **Consumer**：消息消费者，也就是接收消息的一方。消费者连接到 **RabbitMQ** 服务器，并订阅到队列上。消费消息时只消费消息体，丢弃标签。
- **Connection**：网络连接，比如一个 **TCP** 连接，用于连接到具体 **Broker**。
- **Channel**： 信道，**AMQP** 命令都是在信道中进行的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为建立和销毁 **TCP** 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 **TCP** 连接，一个 **TCP** 连接可以用多个信道。客户端可以建立多个 **channel**，每个 **channel** 表示一个会话任务。
- **Broker** 服务节点：表示消息队列服务器实体。一般情况下一个 **Broker**

可以看做一个 **RabbitMQ** 服务器。

- **Exchange**：交换器，接受生产者发送的消息，根据路由键将消息路由到绑定的队列上。
- **Queue**：消息队列，用来存放消息。一个消息可投入一个或多个队列，多个消费者可以订阅同一队列，这时队列中的消息会被平摊（轮询）给多个消费者进行处理。
- **Message**：消息，由消息体和标签组成。消息体是不透明的，而标签则由一系列的可选属性组成，这些属性包括 **routing–key**（路由键）、**priority**（相对于其他消息的优先权）、**delivery–mode**（指出该消息可能需要持久性存储）等。
- **Routing Key**：路由关键字，用于指定这个消息的路由规则，需要与交换器类型和绑定键(**Binding Key**)联合使用才能最终生效。
- **Binding**：绑定，通过绑定将交换器和队列关联起来，一般会指定一个 **BindingKey**，通过 **BindingKey**，交换器就知道将消息路由给哪个队列了。
- **Virtual host**：虚拟主机，用于逻辑隔离，表示一批独立的交换器、消息队列和相关对象。一个 **Virtual host** 可以有若干个 **Exchange** 和 **Queue**，同一个 **Virtual host** 不能有同名的 **Exchange** 或 **Queue**。最重要的是，其拥有独立的权限系统，可以做到 **vhost** 范围的用户控制。当然，从 **RabbitMQ** 的全局角度，**vhost** 可以作为不同权限隔离的手段。

4.6.7.3 约束

无

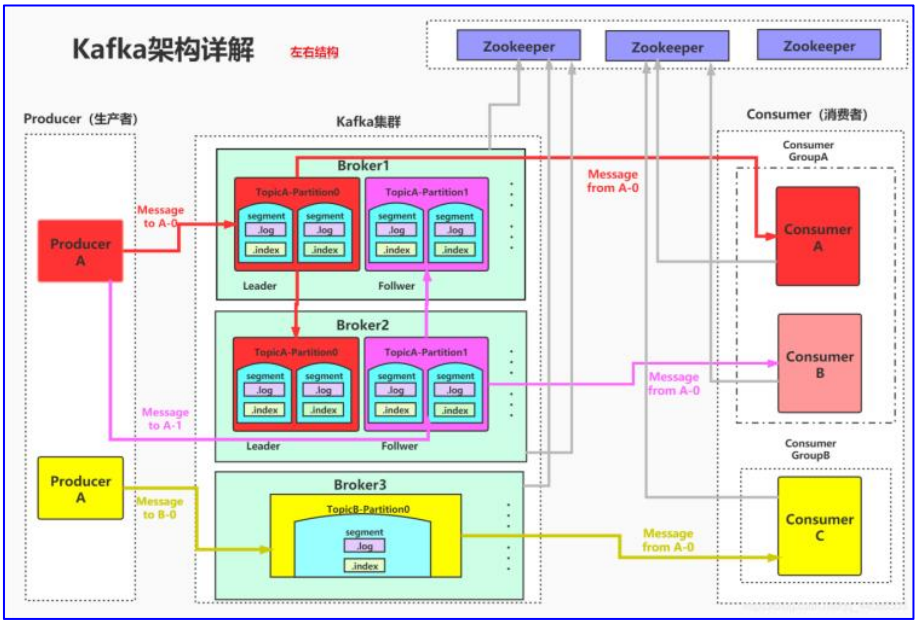
4.6.8 Kafka

4.6.8.1 功能描述

Kafka 是一种流式处理的高吞吐量的分布式发布订阅消息系统，具备快速、可扩展、可持久化的特点。它的最大的特性就是可以实时的处理大量数据以满足各种需求场景：比如基于 **hadoop** 的批处理系统、低延迟的实时系统、**storm/spark** 流式处理引擎。

平台提供租户隔离的流处理平台实例，允许用户创建属于自己的服务实例，并可对每个服务实例进行创建、查询、修改、删除、重启等管理操作。支持通过控制台操作集群后台，支持对 **Kafka** 启动参数、**jvm** 参数、存储容量的配置与修改，支持对 **topic** 的创建、删除、修改、查询。可以通过可视化界面查看资源的使用情况及告警。支持集群关键指标（含 **topic** 请求速率、活跃 **broker** 数量等）的监控展示与告警。

4.6.8.2 技术原理



- **Producer**：**Producer** 即生产者，消息的产生者，是消息的入口。

- **Broker**：**Broker** 是 **kafka** 实例，每个服务器上有一个或多个 **kafka** 的实例，我们姑且认为每个 **broker** 对应一台服务器。每个 **kafka** 集群内的 **broker** 都有一个不重复的编号,如图中的 **broker–0**、**broker–1** 等……
- **Topic**：消息的主题，可以理解为消息的分类，**kafka** 的数据就保存在 **topic**。在每个 **broker** 上都可以创建多个 **topic**。
- **Partition**：**Topic** 的分区，每个 **topic** 可以有多个分区，分区的作用是做负载，提高 **kafka** 的吞吐量。同一个 **topic** 在不同的分区的数据是不重复的，**partition** 的表现形式就是一个一个的文件夹!
- **Replication:** 每一个分区都有多个副本，副本的作用是做备胎。当主分区（**Leader**）故障的时候会选择一个备胎（**Follower**）上位，成为 **Leader**。在 **kafka** 中默认副本的最大数量是 **10** 个，且副本的数量不能大于 **Broker** 的数量，**follower** 和 **leader** 绝对是在不同的机器，同一机器对同一个分区也只可能存放一个副本（包括自己）。
- **Message**：每一条发送的消息主体。
- **Consumer**：消费者，即消息的消费方，是消息的出口。
- **Consumer Group**：我们可以将多个消费组组成一个消费者组，在 **kafka** 的设计中同一个分区的数据只能被消费者组中的某一个消费者消费。同一个消费者组的消费者可以消费同一个 **topic** 的不同分区的数据，这也是为了提高 **kafka** 的吞吐量。
- **Zookeeper**：**kafka** 集群依赖 **zookeeper** 来保存集群的的元信息，来保证系统的可用性。

4.6.8.3 约束

无

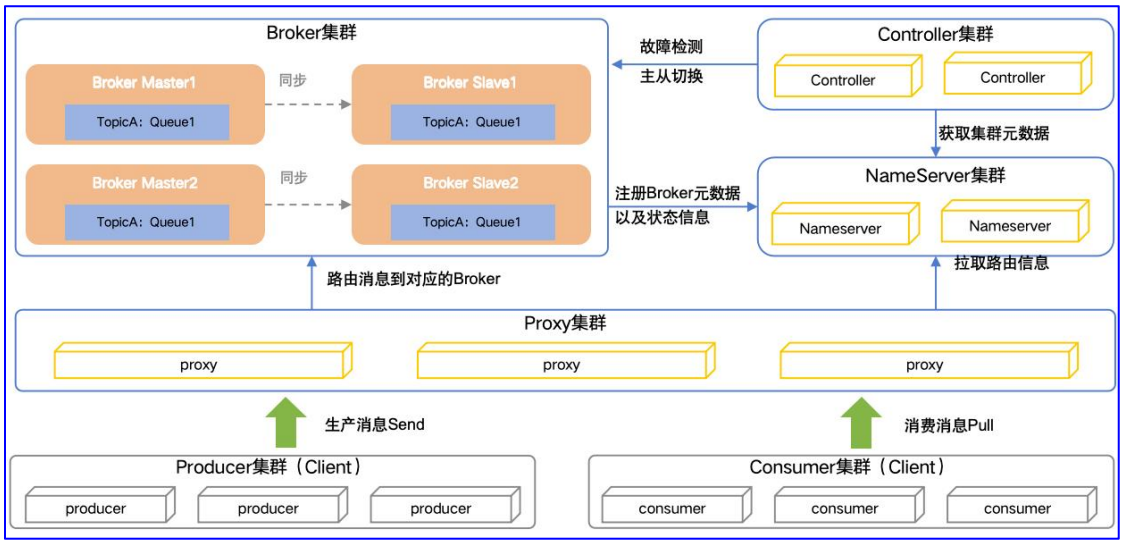
4.6.9 RocketMQ

4.6.9.1 功能描述

Apache RocketMQ 是一款低延迟、高并发、高可用、高可靠的分布式消息中间件。是云原生的“消息、事件、流”实时数据处理平台，支持顺序、延迟、定时、重投、死信、事务消息等功能，满足电商、金融等各类业务场景的需求，支持消息追踪、消息溯源、链路诊断、死信导出、监控告警等能力，保证业务正常运行，支持消息的流式处理和轻计算，以及事件驱动，拓宽了使用场景。

平台提供低延时高可用的分布式消息中间件实例，支持 **Broker** 高可用部署，具备 **RocketMQ** 集群参数的界面配置能力，支持实例的扩缩容，用户密码自定义配置。为保证 **Rocket** 服务的高可用，支持消息同步刷盘持久化，主备同步复制，**broker** 主备自动切换，无单点故障。支持 **broker** 横向扩展，满足大规模场景性能要求。支持集群关键指标（生产和消费消息速率、消息平均处理时延等）的监控展示与告警。

4.6.9.2 技术原理



- **Broker**：负责存储和转发消息，作为 **RocketMQ** 的核心组件。**Broker** 将消息存储在磁盘上，并根据需要将消息传递给消费者，支持主从架构来提升系统的容错能力和数据可靠性
- **NameServer**：提供轻量级的路由服务，存储生产者和消费者与 **Broker** 之间的路由信息，支持动态扩展以确保系统的高可用性
- **Controller**：在集群模式下，**Controller** 负责管理主从 **Broker** 的自动故障转移和高可用性
- **Proxy**：**5.0** 版本中新引入的组件，作为中间层处理生产者和消费者的请求，并将请求转发给 **Broker**。通过计算存储分离，提高资源利用率和弹性能力
- **Consle**：一个用于管理和监控消息队列 **Web** 控制台管理界面

4.6.9.3 约束

无。

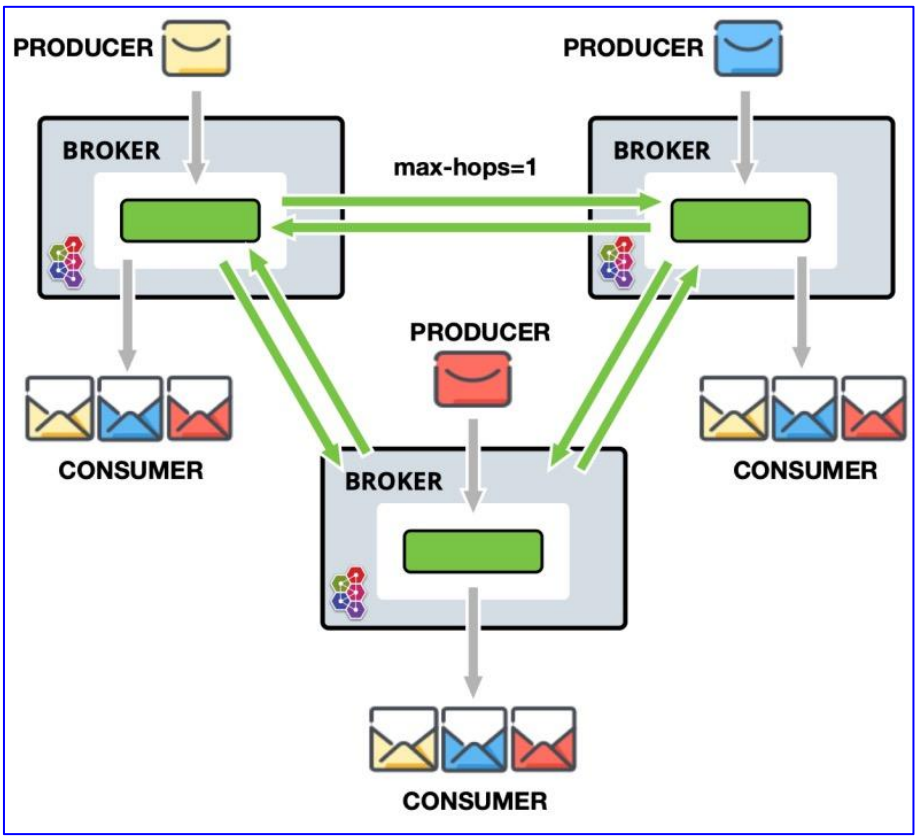
4.6.10 ActiveMQ

4.6.10.1 功能描述

Apache ActiveMQ 是最早的消息队列中间件之一，**java** 实现，功能丰富使用广泛，为应用程序提供高效的、可扩展的、稳定的和安全的企业级消息通信，缺少大规模验证，适用于中小型项目。

平台提供高可用的 **ActiveMQ** 实例服务，支持对称式集群，支持客户端和服务端的负载均衡，支持消息再分配，即当某个消费者下线时可以自动将消息路由给其他有匹配消费者的节点，兼容 **JMS** 规范，支持 **OpenWire, AMQP, MQTT, STOMP** 等通信协议，支持页缓存**(page caching)**和 **AIO(NIO** 的改进版**)**，使得它在高吞吐量和低延迟方面表现优异

4.6.10.2 技术原理



ActiveMQ 是 **Broker-Cluster** 模式，也就是多个 **ActiveMQ** 实例（**Brokers**）通过网络全连接形成集群。在这样的架构中，每个 **Broker** 都可以接收消息，并根据配置将消息转发到集群中的其他 **Broker**，实现消息的复制和负载均衡。

4.6.10.3 约束

无。

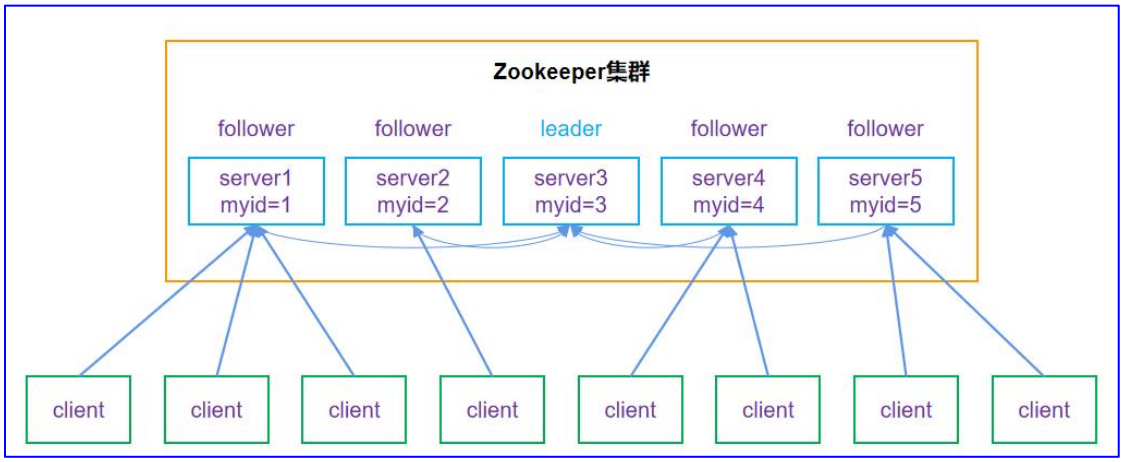
4.6.11 Zookeeper

4.6.11.1 功能描述

ZooKeeper 是一个分布式的应用程序协调服务，是 **Hadoop** 和 **Hbase** 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。**ZooKeeper** 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

平台提供租户隔离的流处理平台实例，允许用户创建属于自己的服务实例，并可对每个服务实例进行创建、查询、修改、删除、重启等能力，支持通过控制台操作集群后台，支持对 **Zookeeper** 启动参数、**jvm** 参数、存储容量的配置与修改。可以通过可视化界面查看资源的使用情况及告警。支持集群关键指标（含收发包速率、堆积请求数等）的监控展示与告警。

4.6.11.2 技术原理



- 集群：**Zookeeper** 是由一个领导者（**Leader**），多个跟随者（**Follower**）组成的集群。
- 高可用性：集群中只要有半数以上节点存活，**Zookeeper** 集群就能正常服务。
- 全局数据一致：每个 **Server** 保存一份相同的数据副本，**Client** 无论连接到哪个 **Server**，数据都是一致的。
- 更新请求顺序进行：来自同一个 **Client** 的更新请求按其发送顺序依次执行。
- 数据更新原子性：一次数据更新要么成功，要么失败。
- 实时性：在一定时间范围内，**Client** 能读到最新数据。

4.6.11.3 约束

无。

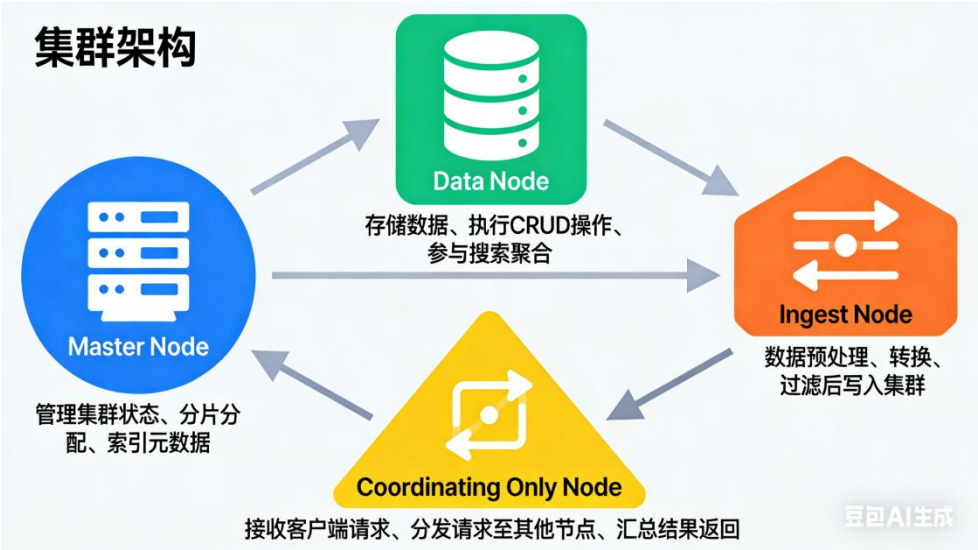
4.6.12 ElasticSearch

4.6.12.1 功能描述

Elasticsearch 是一个分布式、可扩展的实时搜索与分析引擎，广泛用于全文检索、日志分析、监控和数据可视化等场景。它支持结构化、非结构化及半结构化数据的快速存储、搜索和分析，并通过 **RESTful API** 提供简单易用的交互方式，具备高可用性和横向扩展能力，能够处理 **PB** 级数据，支持近乎实时的全文检索、复杂的查询过滤以及高效的数据聚合分析等功能。

平台提供租户隔离的 **Elasticsearch** 服务，允许用户创建属于自己的服务实例，并可对每个服务实例进行创建、查询、修改、删除、重启等能力，支持在线扩容 **CPU**、内存和存储，集成 **Kibana** 提供原生的使用方式，支持通过拓扑查看资源的使用情况及告警。

4.6.12.2 技术原理



Elasticsearch 集群可由一个或多个 **node** 组成，根据 **node** 的作用，可以分为如下的几种：

- **master**：管理整个 **cluster** 的设置及变化：创建，更新，删除 **index**；添加或删除 **node**；为 **node** 分配 **shard**。
- **data**：数据 **node**，负责存储数据分片，并执行数据相关的操作（**CRUD**、搜索、聚合）。
- **coordinating**：协调 **node**，不承担主节点或数据节点职责，仅接收客户端请求，将其路由到正确的数据节点，然后将各节点的结果聚合后返回给客户端。
- **ingest**：在索引前对文档进行预处理（如解析、转换），通过 **ingest pipeline** 实现，如解析日期、重命名字段等。

Elasticsearch 的分片与副本机制是其实现分布式、高可用和高性能的核心。分片是数据的基本分割单元，一个索引在创建时会被预先划分为多个分片。分片是一个功能完整的 **Lucene** 索引，可以独立存储在集群的任何节点上，通过分片，**Elasticsearch** 实现了数据的水平切分与分布式存储，从而突破了单机容量限制，并能将搜索请求并行分发到各个分片上执行，极大提升了处理海量数据的能力。而副本则是每个主分片的完整复制品，称为副本分片，当某个节点故障导致主分片丢失时，其对应的副本分片会自动提升为主分片，确保服务不中断和数据不丢失，保障高可用性；因为搜索和查询请求可以被负载均衡到所有主分片及其副本上执行，可以提升读取性能与吞吐量，有效利用集群资源。

4.6.12.3 约束

无。

4.7 人工智能服务

4.7.1 算力围栏

4.7.1.1 功能描述

算力围栏是一种将特定容器资源池的计算节点隔离池化的算力管控机制，最终形成专属算力池，保障 **AI** 推理业务算力需求与运行安全。大模型服务最终都运行在围栏里的节点上。

算力围栏模块提供了算力围栏全生命周期管理，支撑围栏的创建、修改、删除、查询以及算力节点的加入、移除及节点资源使用情况查看等一系列功能，管理员可根据环境的规划，指定特定的容器资源池作为模型围栏的主体，然后将包含 **GPU** 的节点加入到围栏里面。围栏通过算力隔离的方式保障里面的节点只运行大模型服务，从而实现与其他容器应用分节点隔离运行，保障了模型服务运行的稳定。

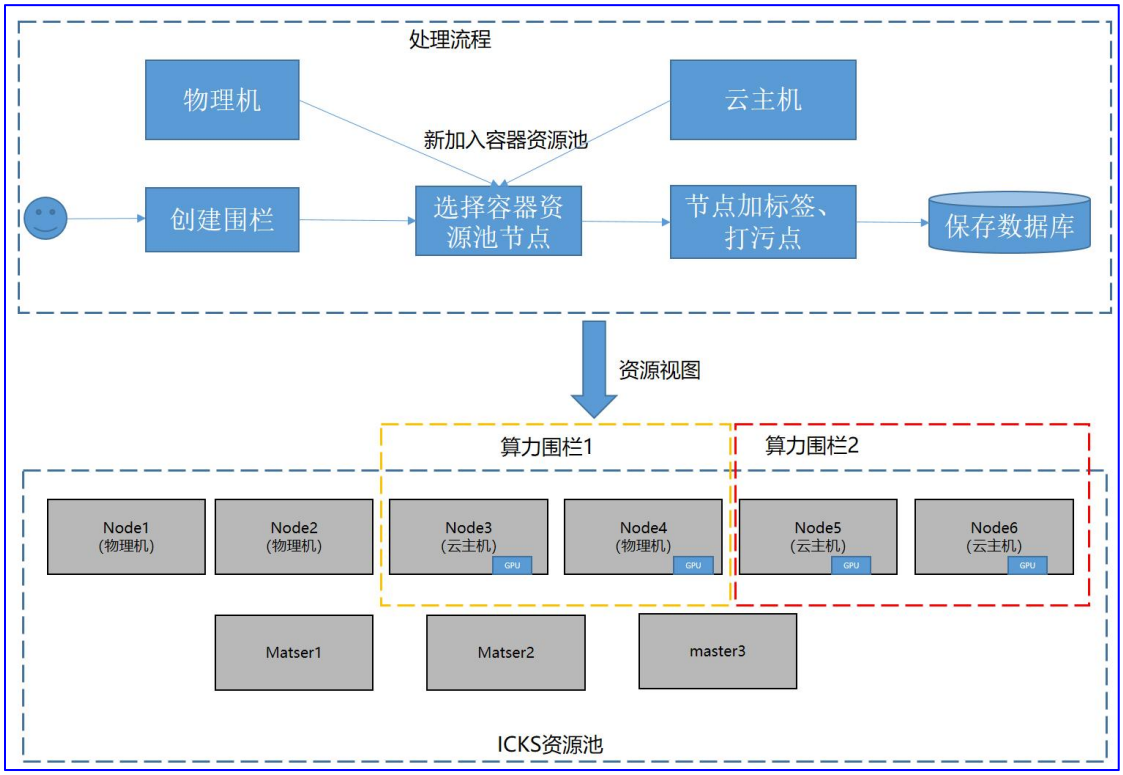
算力围栏管理服务功能包括：

- 围栏生命周期管理：增加、编辑、删除、查询
- 算力围栏全方位展示：基本信息、节点列表
- 节点管理：增加节点、移除节点、设为 **AI** 专享节点

4.7.1.2 技术原理

设计算力围栏模型，将算力围栏与容器资源池进行关联，最终将容器资源池的特定节点加入到围栏里面用来运行模型服务。模型围栏与容器资源池是多对一的关系，每个容器资源池对应多个围栏，但是每一个计算节点智能加入到一个围

栏面，产品即充分考虑客户资源利旧的场景又考虑了灵活扩展，如容器资源池的节点已经运行其他应用，加入围栏后可以通过疏散将应用尽量迁移到非围栏的其他节点运行，针对新的云主机和物理机，可以先加入容器资源池再添加到围栏里面。



- 创建算力围栏，并选择存在的容器资源池的某些节点加入围栏，一个容器资源池可以对应多个围栏，一个节点只能属于一个围栏；
- 对加入围栏的节点通过打标签的方式进行标识，通过打污点的方式实现资源隔离，不允许非模型服务的应用调度到节点上；
- 新物理机、云主机通过脚本方式加入到容器资源池内；
- 通过新增围栏节点的方式将新节点加入到围栏里。

4.7.1.3 约束

围栏里的云主机节点，**cpu** 模式必须是 **HOST_PASSTHROUGH**，否则运行

模型服务会报错；节点必须挂载文件存储存放模型文件的对应目录，否则无法将节点加入到围栏内。

4.7.2 模型仓库

4.7.2.1 功能描述

模型仓库将已经注册到平台上大模型进行管理，以供模型部署，仓库里面内置 **deepseek** 蒸馏的 **1.5B**、**7B**、**14B** 三个大语言模型，**bge-large-zh-v1.5** 嵌入模型，**bge-reranker-large** 重排序模型，**Qwen2.5-Omni-7B**、**Qwen2.5-VL-7B-Instruct** 两个多模态模型。

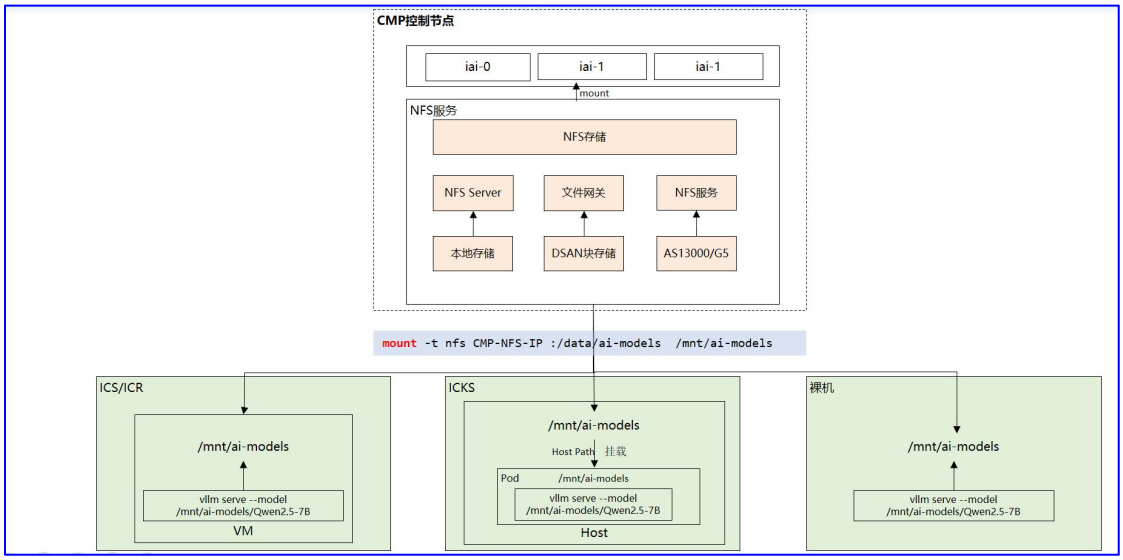
模型仓库模块提供了大模型全生命周期管理，支撑模型的注册、注销、分发、查询等一系列功能，管理员可先将其他的大语言模型先上传到 **NFS** 上，然后在模型仓库页面进行注册，注册时系统会校验模型文件在 **NFS** 的目录上是否存在，如果不存在，则注册失败。另外，支持为提升后续模型的部署速度，支持将模型文件从 **NFS** 上预热到主机节点的特定目录上，模型部署时直接从本地目录里加载模型文件，大幅度提升了部署效率。

模型仓库功能包括：

- 模型的生命周期管理：注册、注销、删除、修改、查询
- 模型预热：将模型从 **NFS** 提前加载到主机的目录
- 模型显存计算：根据模型的最大上下文长度、并发数测算部署模型所需要的显存大小
- 模型能力选择：对于多模态模型支持选择模型的能力，包括文本能力、语音能力、图片能力、视频能力等

4.7.2.2 技术原理

模型文件普遍很大，基于页面上传占用管理网带宽且耗时较长，综合考虑性能和可操作性，环境部署时，在容器资源池的主机节点上提前挂载 **NFS** 对应的目录，注册之前需要提前将模型文件存放到 **NFS** 的文件目录，模型注册时，需要校验 **NFS** 目录是否存在模型文件，若文件不存在，禁止注册，否则会导致模型注册后无法运行。具体过程如下图所示：



此外，通过 **k8s** 的 **Job** 实现模型文件从 **NFS** 目录到主机目录的分发，通过 **NFD** 技术，自动检测模型文件已经分发到哪些节点，模型运行时会优先调度到已经分发的节点上运行，有效提升模型加载速度。

4.7.2.3 约束

模型服务注册前，需要有对应的文件存储挂载到 **CMP** 节点，且已经上传模型文件到对应的目录下。

4.7.3 模型服务

4.7.3.1 功能描述

通过本地模型部署和外部模型服务接入两种方式，提供符合 **OPENAI** 规范的 **API** 提供给应用、智能体使用。针对本地模型部署，通过分配计算资源将大模型以容器的方式运行在模型围栏内的主机上；针对外部模型服务接入，通过将平台外部部署的私有模型、在线模型纳管到平台，实现统一管理。

模型服务提供了模型部署以及部署后模型服务的全生命周期管理，模型部署时系统自动选择最优的推理引擎，加载模型提供推理服务，针对 **GPU** 推理支持单机单卡、单机多卡和多机多卡。系统支持不同推理框架的快速适配，对于不支持的模型或推理快加通过后台配置的方式实现异构推理框架无代码修改的快速适配。已经部署或已接入的模型服务都提供 **OPENAI** 格式的 **api** 对外提供服务，外部调用时需要使用 **API–KEY** 进行鉴权。

模型服务功能包括：

- 本地模型服务实例管理：增加、编辑、删除、查询、启动、暂停，不是模型服务时支持单机部署及多机部署，支持将 **KVCache** 卸载到内存提升推理性能
- 模型服务体验：立即体验、**API** 示例
- 本地模型服务实例详情：基本信息、实例副本列表
- 外部模型管理：接入、编辑、移除、查询
- 外部模型体验：立即体验
- 关联应用：已经使用模型服务的知识库、应用和智能体列表
- 弹性扩缩容：根据 **LLM** 指标或资源使用率快速增加或减少模型服务的

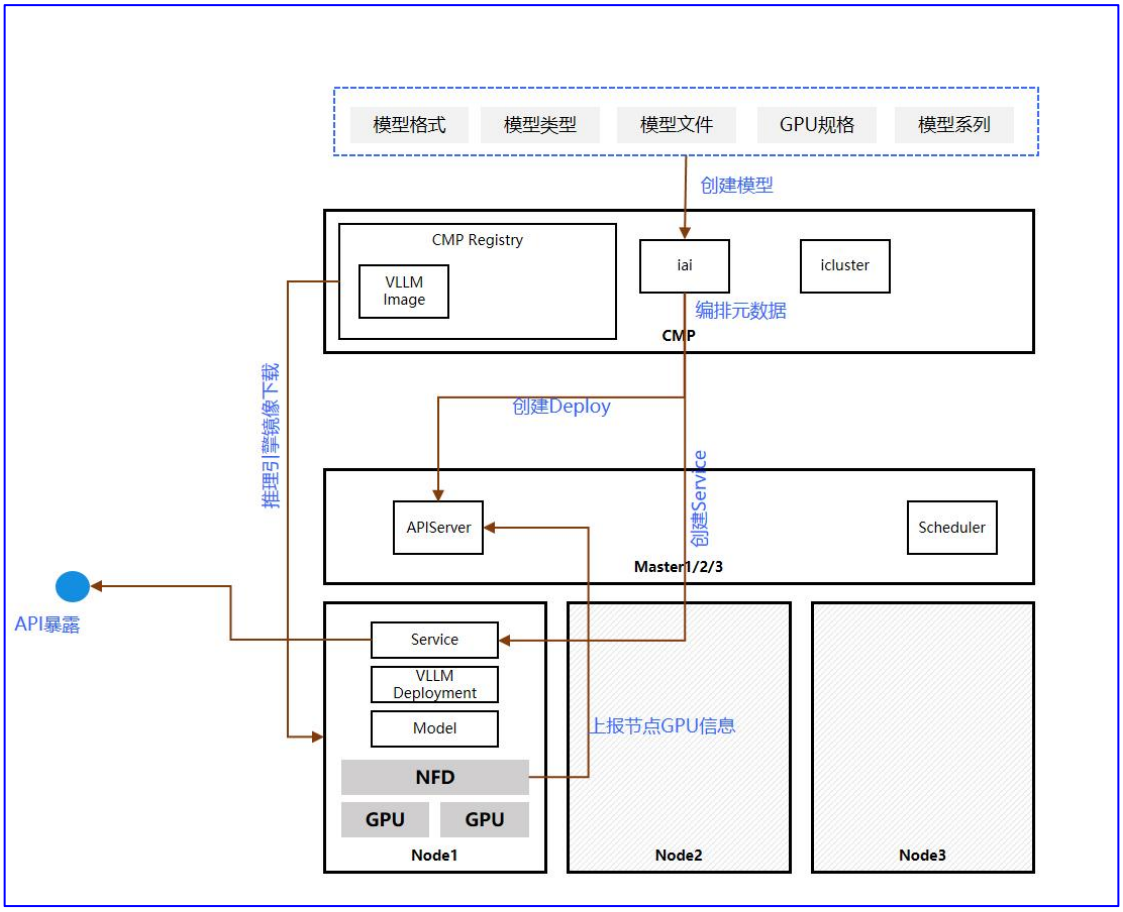
实例个数

- 模型扩展：支持通过后台配置模型与推理引擎的匹配关系，以快速适配新模型及不同推理引擎部署模型服务

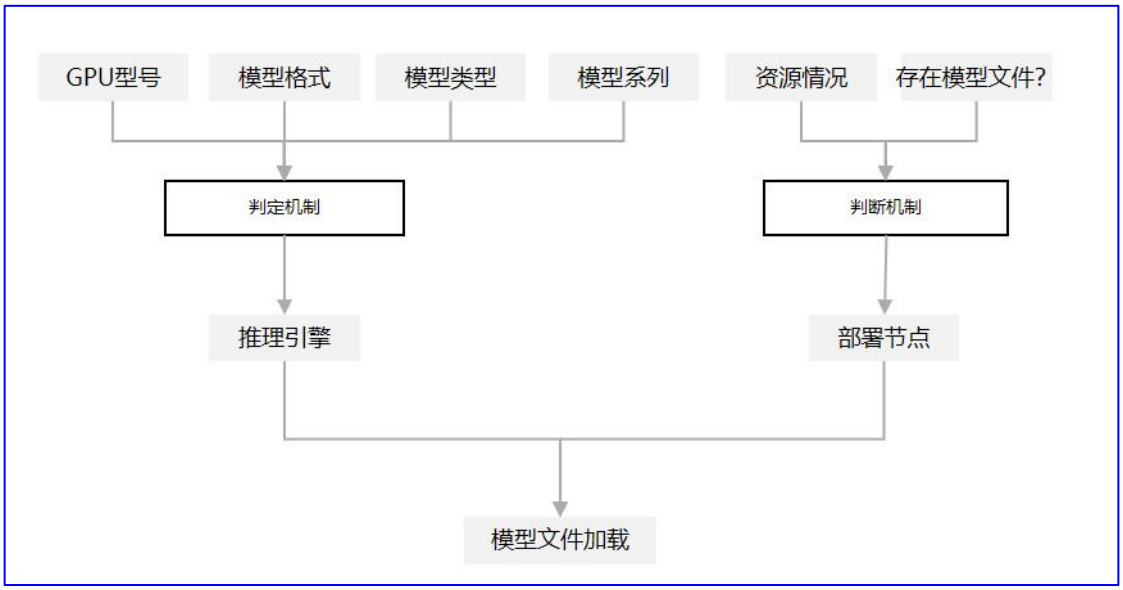
4.7.3.2 技术原理

基于 **Kubernetes NFD** 的节点特性发现，实现英伟达、沐曦等异构加速设备的自动发现和弹性扩展，为模型加载提供显存，通过为推理引擎适配不同加速设备的依赖库实现推理引擎对不同 **GPU** 的兼容，支撑模型在不同的加速设备上运行。

根据模型类型、模型格式、模型系列，自动选择最优的推理引擎，编排 **K8s** 的工作负载和网络资源的元数据进行创建，工作负载运行时自动加载模型文件，运行大模型服务，通过 **Service** 对外暴露 **OPENAI** 规范的 **API**，供外部调用。根据模型所需的显存大小和围栏内节点的配置，可进行单机单卡、单机多卡、多机多卡多种方式运行模型，整体流程如下图。

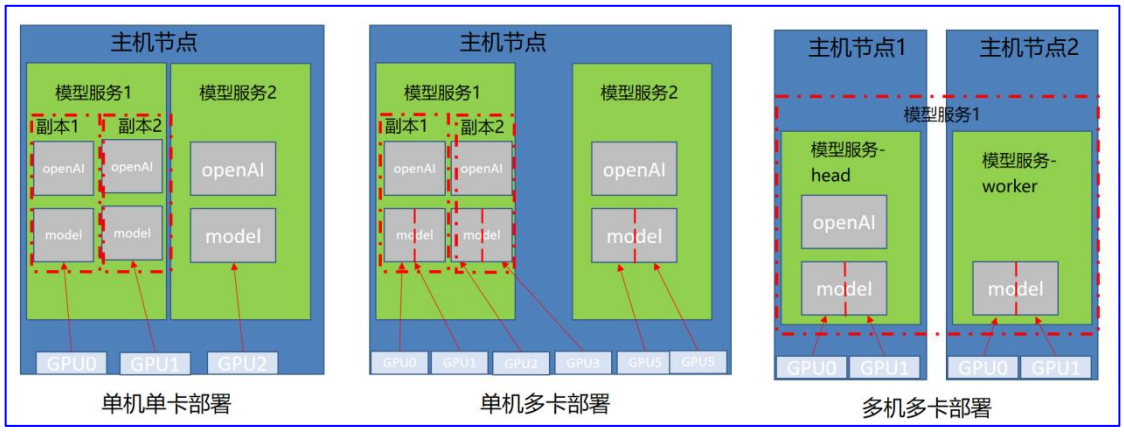


模型部署时根据模型的类型、系列、格式进行匹配得出最优的推理引擎，然后根据围栏内的节点的资源和是否有预热的模型文件进行节点选择，选出最优节点运行推理引擎加载大模型。如下图所示。

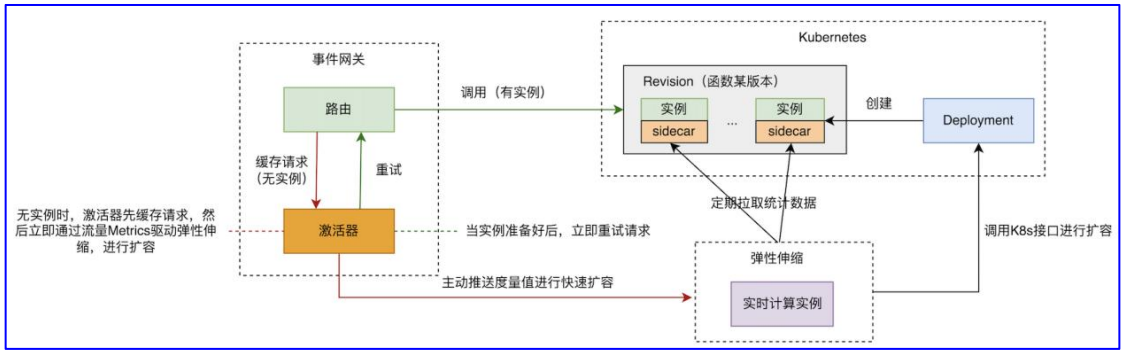


为支持不同型号的 GPU 卡，根据模型大小和 GPU 卡的显存大小，决定 GPU

卡的数量，需要多张 **GPU** 时推理引擎根据模型文件的分片，分别加载到多个 **GPU**，推理时多卡共同计算推理结果，整合后输出结果。如下图。



可以为弹性服务配置弹性伸缩，当模型性能下降或资源使用量过高时，模型服务进行副本的增加或减少。弹性伸缩由控制平面组件与数据平面组件，并完全基于 **Kubernetes** 进行开发，采用完整的云原生设计来确保系统的可扩展性、可靠性以及资源效率。充分利用 **Kubernetes** 的现有功能，包括自定义资源 **(CRD)**、控制器机制以及动态服务发现等，为大规模 **LLM** 推理服务提供了稳健的基础设施，控制平面组件主要负责管理模型元数据注册、自动扩缩容、模型适配器注册，并执行各种策略。数据平面组件则提供可配置的请求派发、调度与推理服务能力，实现灵活且高性能的模型推理执行。



4.7.3.3 约束

针对外部模型服务的接入，只允许接入提供符合 **OPENAI** 的 **api** 的模型服

务接入。

4.7.4 模型评估

模型评估是指对实际运行中的大模型推理服务进行多维度评估，主要包括模型性能评估和质量评估，模型性能评估用于衡量某个模型在特定资源下的运行效率，质量评估用于衡量模型输出结果的正确性、相关性，该版本实现了模型性能评估。

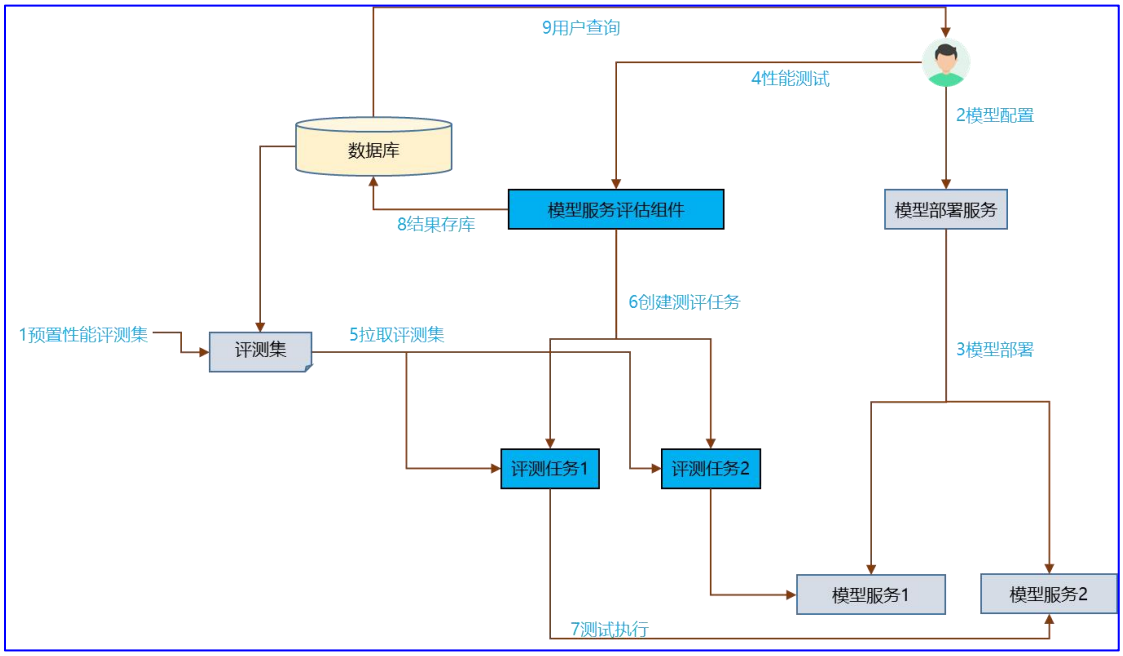
4.7.4.1 性能评估

4.7.4.1.1 功能描述

模型性能评测通过配套轻量化的一键式实测工具，支持自动化测试、多并发负载模拟，并以可视化图表形式展示模型性能数据。当模型服务部署完成后，一键进行推理性能自动化评测，提供详细的性能指标数据，评测指标包括 **TTFT**（**Time toFirst Token**）首 **token** 延迟、**TBT**（**Time between Tokens**）**Token** 间延迟、**TPS**（**Tokens perSecond**）每秒 **token** 数、**RPM**（**Request per Minute**）每分钟请求数，同时展示每个指标测试期间样本数的百分位数，以展示系统的稳定性。通过模型性能评估，为模型选型、部署优化及资源分配提供科学依据，提高决策效率。

4.7.4.1.2 技术原理

模型性能评测通过性能基准测试工具实测得出性能数据，整个流程支持自动化测试、多并发负载模拟，并通过数据可视化技术以图表形式展示，其核心工作流程如下图所示。



- 模型性能评估与输入内容无关，只与输入输出长度有关，故可使用任意数据集，系统中预置一首英文诗歌作为评测集；
- 用户选择某个已经注册的模型文件进行部署，通过手动配置其所需要的 **CPU**、内存、**GPU** 等硬件规格，根据推荐配置或实际业务需要进行选择；
- 模型部署服务根据所选模型文件和硬件规格进行模型部署任务的下发；
- 用户发起模型性能评测，通过模型部署服务获取待测模型服务的访问地址、**API Key** 等信息，并传递到模型评估组件中；
- 模型评估组件从数据库中获取预置的性能评测数据集，随机从中获取指定长度的数据集，如 **550 token**，将其传递给评测任务；
- 评测任务设计为 **K8s** 中的 **Job** 资源，针对每个模型服务创建单独的评测任务，创建任务时将数据集、模型服务信息以及并发数等信息传递给评测任务；
- 评测任务根据注入的模型服务信息，以及内置的评测数据集，调用性能

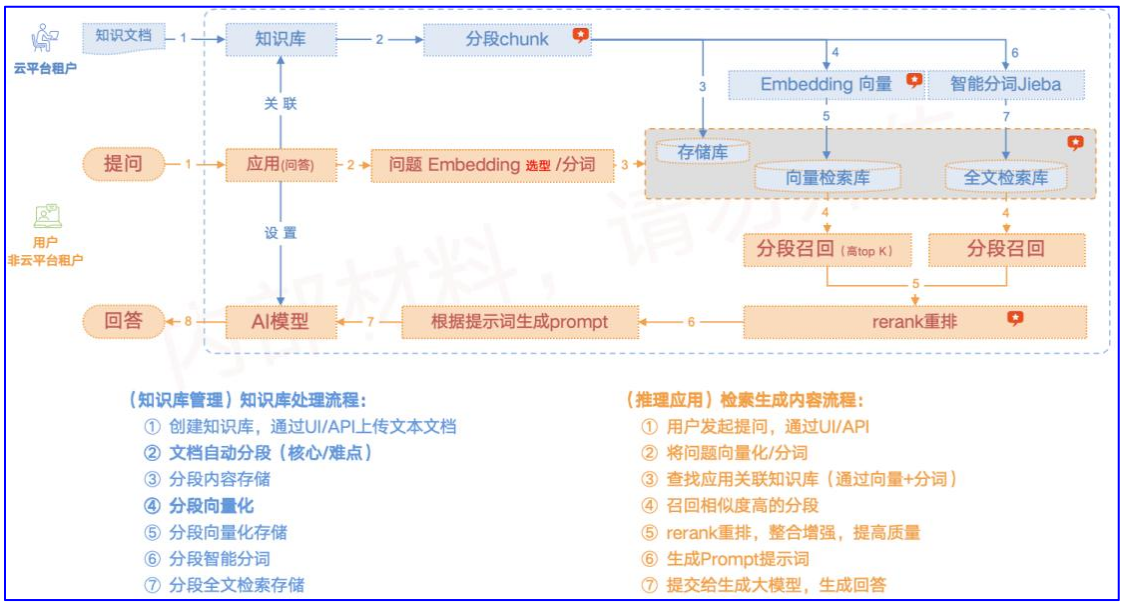
评测工具进行实际测试，工具对模型服务发起请求调用直到测试过程结束，并将测试进度和结果保存到 **Job** 的注解中；

- 模型服务评估组件将 **Job** 执行状态和执行完成注解中的评测结果，保存到数据库中，展示到 **UI** 中；
- 用户通过 **UI** 查看模型服务性能评测结果。

4.7.4.1.3 约束

- 当对某个模型服务进行性能评估时，任何对模型服务的调用会影响评估结果。

4.7.5 知识库&推理应用



4.7.5.1 知识库

4.7.5.1.1 功能描述

知识库（**Knowledge**）是一系列文档（**Documents**）的集合，一个文档内可能包含多组内容分段（**Chunks**），知识库可以被整体集成至一个应用中作为检

索上下文使用。平台提供高效、安全的私域知识管理服务，通过本地化部署实现数据不出域，避免将敏感信息上传至第三方平台；保障企业私有数据的安全性；灵活加载垂直领域数据（如医疗指南、法律条文），解决通用模型的专业盲区问题；支持动态接入最新数据，突破大模型训练数据的时间限制，可直接检索最新文档生成答案，而非依赖过时信息；支持 **Word**、**Excel**、**PDF**、**MarkDown**、**HTML** 等文本文档自动分段 **chunking**，方便索引和搜索，快速找到最相关的片段，支持向量检索、文本检索、混合检索，支持通过 **Rerank** 向量模型重排以及调整权重，提升文档检索召回的准确性。

功能包括：

- 知识库管理：知识库创建、删除，设置向量/嵌入模型，设置检索类型
- 文档管理：文档上传、启用、禁用、分段设置
- 分段管理：分段和关键字新增、修改、删除，分段启用、禁用、删除
- 数据采集：从 **NFS**、**S3** 对象存储定时同步文档自动更新知识库、立即执行
- 命中测试：设置检索类型进行命中测试，命中测试记录

4.7.5.1.2 知识自动分段技术原理

通过智能化的文本分块策略，优化大语言模型对知识库内容的检索效率与回答精准性。

- 多格式适配：支持对 **Word**、**Excel**、**MarkDown**、**TXT**、**PDF**、**HTML** 等多种文件格式解析，通过内置解析器提取文本内容后再执行分段
- 分段标识符：默认以换行符**\n** 为分隔符，将文档按段落拆分，支持自定

义拆分规则（如按句号、井号等拆分），适应不同文本结构

- 动态分块：当单段落超过预设长度时，系统自动递归拆分，将按字符数二次分割为多个子块
- 重叠机制：相邻分块保留部分重叠内容（默认 **50 Tokens**，支持设置），避免关键信息被切断，确保上下文连贯性
- 实时预览：配置后可直接查看分块效果及字符数，支持动态调整参数直至符合需求

4.7.5.1.3 约束

无

4.7.5.2 知识库数据采集

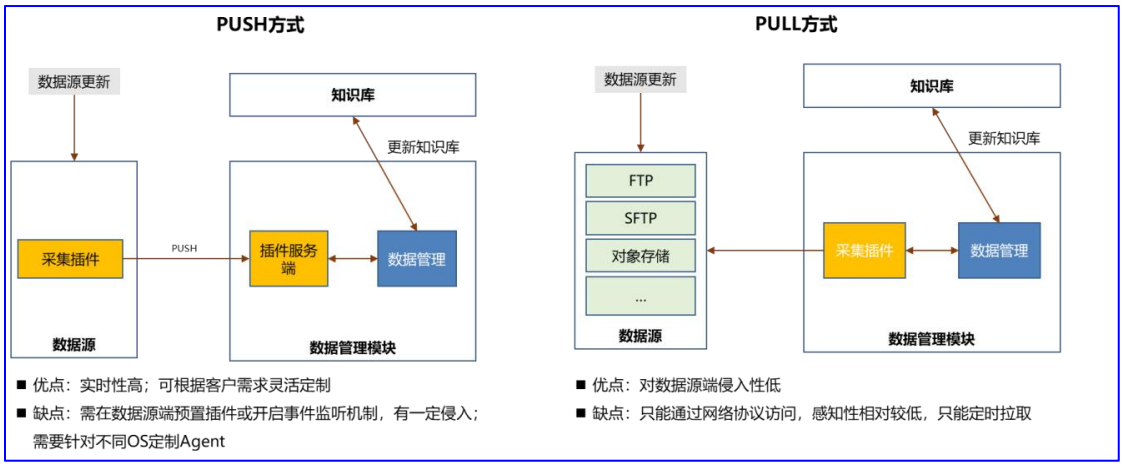
4.7.5.2.1 功能描述

平台提供知识库自动更新能力，可以将用户应用系统的业务数据，定时、增量同步到平台知识库，提升知识库更新效率、减少人工维护成本。

目前支持源的业务数据存储类型为 **NFS**、对象存储，用户业务系统将知识库存放在相应位置后，平台会定时获取相应知识文件，并自动传输到平台知识库中。

支持的文件类型：**TXT**、**MARKDOWN**、**MD**、**MDX**、**PDF**、**HTML**、**XLS**、**XLSX**、**DOCX**、**CSV**，

4.7.5.2.2 技术原理



平台支持 2 种同步机制：**PUSH**、**PULL**。**PUSH** 机制即数据源端主动发送数据到平台中，如对象存储支持的事件机制；**PULL** 即平台预置定时任务，定时到数据源端获取知识文件，通过时间戳比对等方式获取增量文件，同步到平台知识库中。

以 **NFS** 为例，说明数据采集流程：

平台配置数据源端 **NFS** 服务器目录、同步周期、是否访问子目录等信息

任务启动时，平台会自动扫描上述目录，并根据时间戳判断所需更新文件

拉取文件到平台，并调研知识库上传文件接口，上传知识库，完成同步

平台会记录同步的文件列表，以便用户查询。

4.7.5.2.3 约束

支持 **NFS**、对象存储

4.7.5.3 推理应用

4.7.5.3.1 功能描述

平台通过本地知识库结合大语言模型构建推理应用，提供基于 **RAG** 的智能

问答服务，通过检索本地知识库约束生成内容，减少模型“虚构事实”的概率，进一步提升答案的精确性；仅需更新知识库内容，无需重新训练大模型，节省算力与时间成本；支持多模型调试预览、生成提示词、设置开场白、显示引用和归属等。

功能包括：

- 推理应用管理：推理应用创建、编辑、删除
- 推理应用编排：提示词、开场白、知识库选择、引用和归属、调试预览
- 摘要：会话 **UI** 启用/禁用、**API** 启用/禁用、密钥管理、应用指标监控
- **API** 查看和示例
- 会话历史查询
- 检索模式：全文检索、向量检索、混合检索

4.7.5.3.2 技术原理

平台支持全文检索、向量检索以及混合检索：

1.全文检索：通过关键字匹配进行，利用 **TF-IDF**、**BM25** 等技术，根据确切的关键字匹配搜索文档，已知确切术语的精确查询，快速又有效。

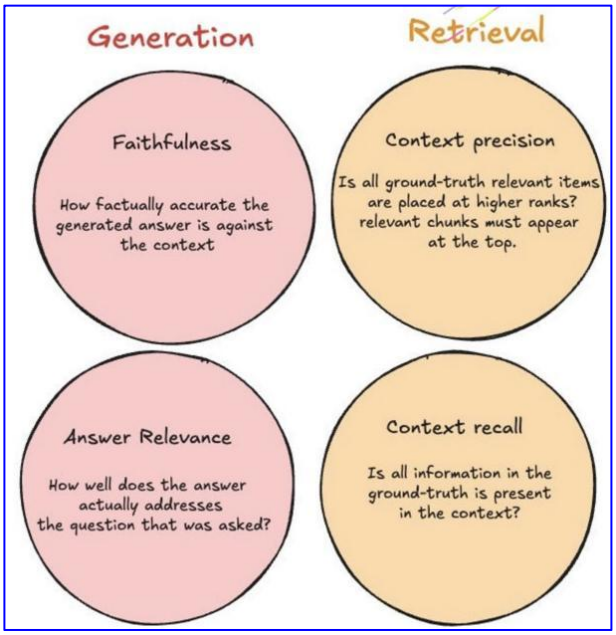
2.向量检索：利用嵌入（向量表示）来理解查询和文档的上下文和含义，这使得即使查询中不存在确切的关键字，也能检索到相关信息。如：通过余弦相似度或欧几里得距离等指标来衡量查询和文档向量之间的相似性。

3.混合检索：结合不同的搜索技术，以提高从数据库或知识库中检索信息的准确性和效率。在检索过程中，同时使用多种检索方式，将多种检索方式进行融

合，得到最终的结果，弥补单一检索的不足，提高检索的准确性和效率，主要利用传统的基于关键字的全文检索和基于语义的向量检索。

混合检索优点：

- 提高召回率和精确度：通过将关键字匹配与语义理解相结合，混合搜索增加了检索到相关文档的可能性，单独使用这两种方法可能会遗漏这些文档
- 处理同义词和上下文：密集检索有助于理解同义词和上下文，确保相关文档不会因词汇不匹配而被忽视
- 效率：稀疏检索通常更快，可以快速缩小文档集的范围，而密集检索则可以优化此集以确保语义相关性
- 鲁棒性：混合搜索对不同类型的查询更具鲁棒性，无论它们是精确的基于关键字的查询，还是更抽象且依赖于上下文的查询



4.7.5.3.3 约束

无

4.7.6 智能体

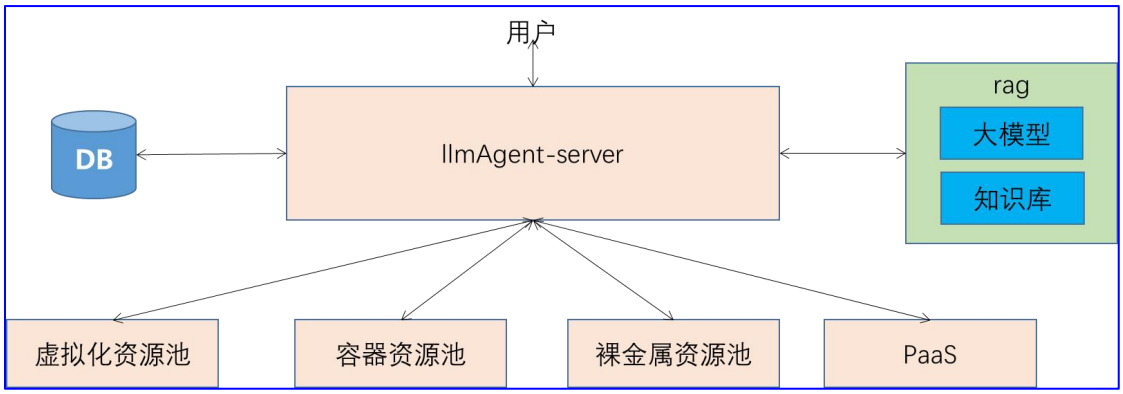
4.7.6.1 功能描述

智能体服务是基于虚拟机、裸金属、容器、**PaaS** 资源结合大模型、知识库等服务，为满足用户快速构建智能体应用而设计，提供智能体应用的构建、开机、关机、资源监控、导入、导出等全生命周期管理。

支持的资源类型：云主机、容器应用、大模型、知识库、推理应用、**mysql**、**postgreSQL**、**Redis**、**Memcached**、**MongoDB**、**RabbitMQ**、**ActiveMQ**、**RocketMQ**、**Kafka**、**ZooKeeper**。

支持一键启停智能体资源及资源监控。

4.7.6.2 技术原理



功能&流程设计：

- 用户选择部署智能体所需资源类型、资源池、推理应用（包括了大模型、知识库）等信息，其中推理应用（包括了大模型、知识库）选择已有，用户提交后调用后台接口，自动化创建虚机、裸机、容器、**paas** 资源，

本质是一个资源编排

- 可以直接配置大模型和知识库，也可以配置推理应用。
- 用户选择提交后，平台自动调用相应模块接口创建对应的资源，包括云主机模块创建云主机、调用容器模块接口创建容器应用等
- 智能体模块记录智能体及资源关联关系到数据库，用于统一管理。

4.7.7 AI 网关

4.7.7.1 功能描述

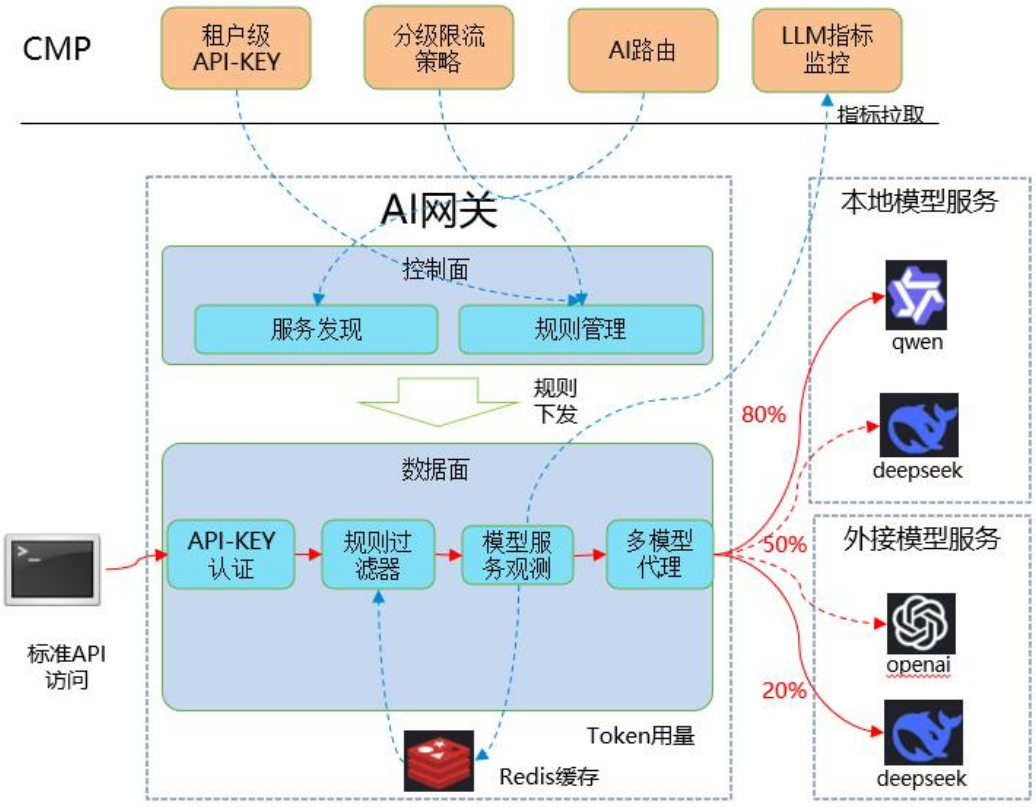
AI 网关是微服务网关和 **AI** 的结合体，使用 **AI** 网关的目的是大模型服务进行统一的代理，暴露统一的 **API** 接口给知识库、**AI** 应用、智能体使用，实现大模型服务的统一入口和鉴权，是连接大模型与业务逻辑的关键枢纽。

AI 网关的功能包括：

- **AI** 网关部署：通过插件中心，实现 **AI** 网关的部署、卸载操作
- **AI** 路由创建：创建 **AI** 路由作为通过 **AI** 网关访问大模型的入口，可配置域名、**API–KEY** 及限流策略、模型服务及流量比例等信息
- **AI** 路由管理：对 **AI** 路由进行修改、列表查询、删除等生命周期管理操作
- **AI** 路由详情：支持 **AI** 路由的基本信息、代理的模型服务以及使用示例查看，支持查看调用总 **token** 数、输出/输出 **token** 数、请求次数，可以精确到某个 **API–KEY** 或某个模型
- **API–KEY** 管理：支持 **API–KEY** 的创建、修改、删除、列表查询等生命周期管理，支持设置 **API–KEY** 的过期时间

- **API–KEY** 详情：支持查看 **API–KEY** 的基本信息，支持查看调用总 **token** 数、输出/输出 **token** 数、请求次数，可以精确到某个路由下面的某个模型
- 限流策略管理：支持限流策略的创建、修改、删除、列表查询等生命周期管理，支持根据首 **Token** 延迟、**Token** 间延迟、生成阶段的平均吞吐量设置不同级别的限流规则，支持设置限流规则生效时的状态码和返回内容。
- 限流策略详情：支持查看基本信息和限流规则。

4.7.7.2 技术原理



AI 网关包含数据面和控制面，控制面通过多种形式进行服务的动态发现对接云上部署的模型服务和外部接入的模型服务，支持丰富的路由规则语义将 **k8s**

的 **crd** 资源转化成网关路由规则，通过 **XDS** 协议实时将路由规则下发到数据面，实现规则的在线配置和热更新；数据面基于 **envoy** 高性能高并发网络代理进行路由策略的执行和请求转发，支持通过插件、过滤器或自定义模块注入 **AI** 流量处理功能。当请求通过网关时，首先针对 **API-KEY** 进行权限校验，通过后规则过滤器拦截请求，根据限流规则执行对应的限流策略，党

当 **AI** 应用访问大模型时，请求首先到达网关，经过身份认证和基本校验后，请求被规则过滤器拦截，过滤器根据限流规则执行对应的限流策略，达到限流条件的请求直接返回，未达到由网关将请求路由到后端大模型服务进行处理，并将结果返回客户端，整个架构串联起大模型请求全流程。

4.8 软件开发服务

4.8.1 代码源及流水线

4.8.1.1 功能描述

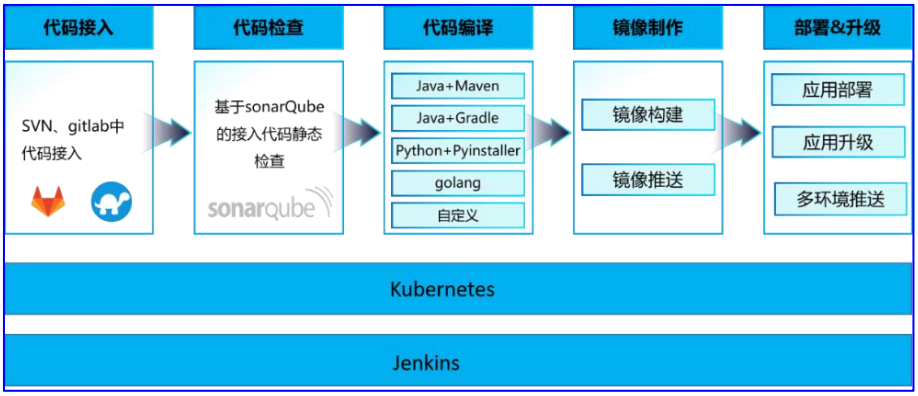
平台代码源及流水线服务实现基于 **jenkins** 的 **CI/CD**，集成从源码管理到流水线任务的可视化自定义配置。代码源支持 **GitLab** 和 **SVN**，实现代码的统一接入；流水线支持三种个性化创建方式：自定义/模板创建/克隆流水线创建，任务类型主要分为：构建、代码扫描、发布和部署，支持用户自定义任务执行时间和任务内容，完成一键式 **CI/CD** 流程化持续集成、持续交付、持续部署。

代码源及流水线服务是对 **jenkins** 流水线功能的最大程度的包装，可视化实现了 **CI/CD** 的全流程，同时增加代码源管理机制。

代码源及流水线特性功能包括：

- 支持 **GitLab/SVN** 等多种形式的代码源管理，实现代码的统一接入，接入协议支持 **http** 和 **https** 两种方式。
- 支持流水线立即、定时执行，提供从代码接入、代码扫描、代码编译到镜像发布、部署的可视化流程化管理。
- 提供多个流水线最佳实践模板，基于模板可以快速构建高效、敏捷的 **CI/CD** 流水线。
- 提供流水线任务的灵活自定义，满足用户多样化需求；立即执行选择代码分支和标签，一次任务，多次使用。
- 支持基于代码源的多运行环境的一次创建，持续集成、持续交付、持续部署，同时可以设置定时任务保证自动化构建。
- 提供流水线任务克隆创建，支持不同源码但是过程相似流水线任务的一键创建。
- 提供通过导入 **jenkinsfile** 的方式创建流水线，将云外的持续构建脚本快速迁移到 **InCloud OS** 提供的软件开发服务内

4.8.1.2 技术原理



- 从代码源的管理开始，用户接入需要管理的代码项目。
- 流水线中提供了代码检查，代码编译，镜像制作到部署&升级的任务可

视化自定义。

- 流水线执行实际上是触发部署在 **K8S** 集群的 **jenkins** 后台任务，在执行记录中可以实时查看任务进度以及最终流水线执行结果。

4.8.1.3 约束

所选的虚拟数据中心对应的集群必须有 **jenkins** 集群。

4.8.2 微服务注册中心

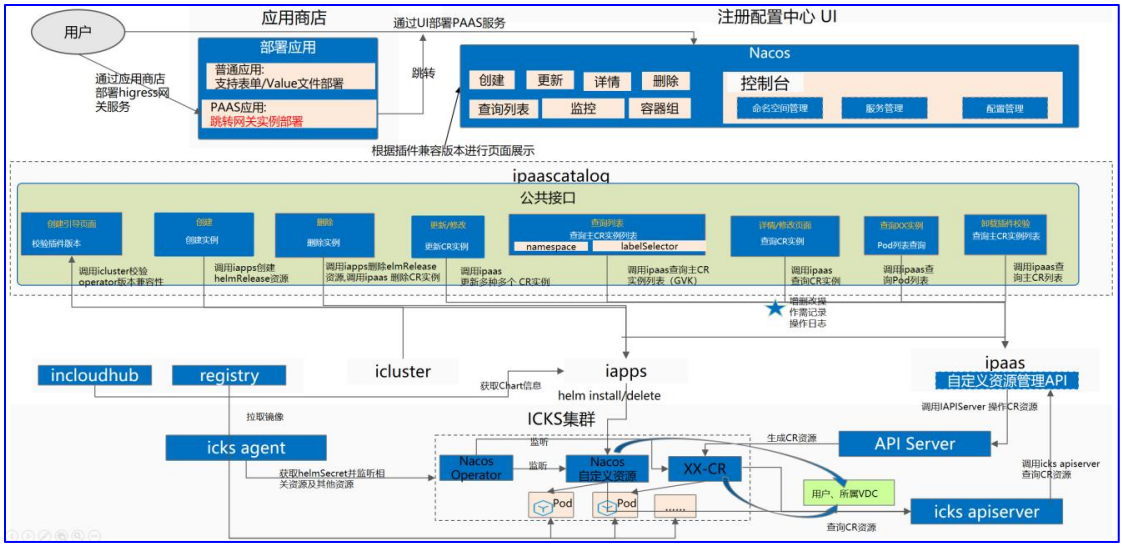
4.8.2.1 功能描述

InCloud OS 的注册配置中心主要是通过 **Nocas** 对微服务应用提供注册和配置中心，具体包括对创建 **Nacos** 实例的规格进行管理，对 **Nacos** 实例进行增、删、改、查、详情查看、监控，并提供控制台对每个实例内的命名空间、服务和配置进行管理。提供 **Nacos** 原生控制台，方便熟悉 **Nacos** 操作的用户进行操作。

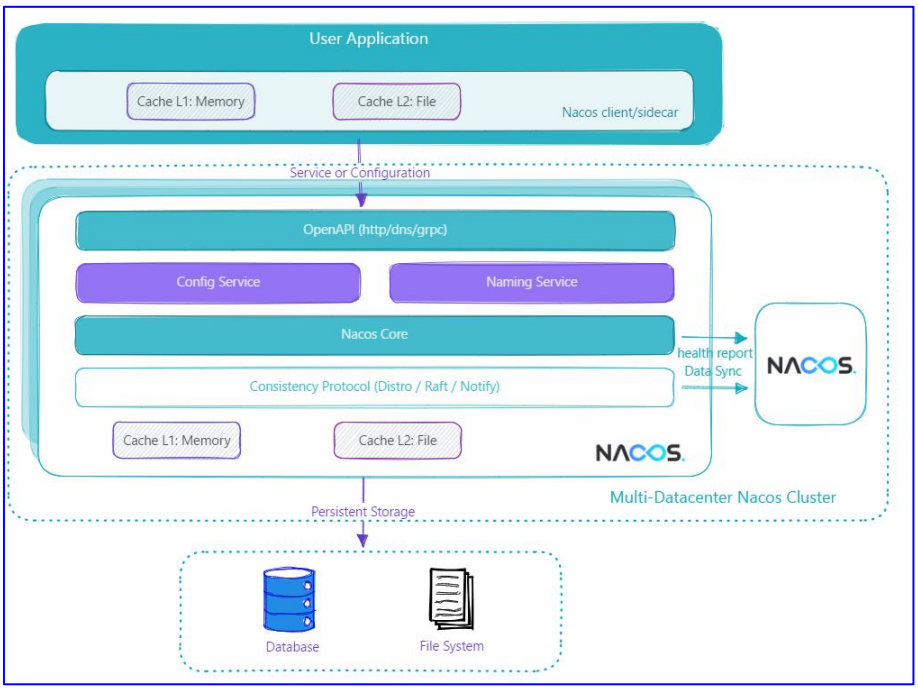
4.8.2.2 技术原理

复用 **InCloud OS** 的 **PaaS** 管理方式，要先通过插件部署 **Nacos** 的 **Operator** 服务，初始化自定义资源和 **Controller**，然后可以对注册配置中心实例进行全生命周期管理和自运维

通过平台页面对 **Nacos Operator** 的自定义资源实例进行增、删、改的管理，**Controller** 实时监听到资源实例的变化后，对新增事件，触发承载 **Nacos** 服务的 **pod** 创建逻辑；对于修改事件，触发承载 **Nacos** 服务的 **pod** 的配置和变量的修改逻辑处理；对于删除事件，触发 **Nacos** 实例的删除操作。客户不再需要 **Nacos** 服务时，卸载 **Operator** 并清除在平台里注册的自定义资源，具体如下图所示。



Nacos 是 **Dynamic Naming and Configuration Service** 的首字母简称，一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。提供了一组简单易用的特性集，快速实现动态服务发现、服务配置、服务元数据及流量管理通过配置管理，关键特性主要包括：**1**）服务发现和服务健康监测，支持基于 **DNS** 和基于 **RPC** 的服务发现。服务提供者使用原生 **SDK**、**OpenAPI** 注册 **Service** 后，对服务的实时的健康检查，阻止向不健康的主机或服务实例发送请求服务，消费者可以使用 **HTTP&API** 查找和发现服务并进行调用；**2**）动态配置服务，以中心化、外部化和动态化的方式管理所有环境的应用配置和服务配置，消除了配置变更时重新部署应用和服务的需要；**3**）服务及其元数据管理，从微服务平台建设的视角管理数据中心的所有服务及元数据，包括管理服务的描述、生命周期、服务的静态依赖分析、服务的健康状态、服务的流量管理、路由及安全策略、服务的 **SLA** 以及最首要的 **metrics** 统计数据。



4.8.2.3 功能约束

Nacos 需要 **mysql** 数据库持久化数据，创建 **Nacos** 实例前需要有可用的 **mysql** 来进行对接

4.8.3 微服务网关

4.8.3.1 功能描述

InCloud OS 的微服务网关基于 **gatewayapi** 标准，将后端服务接口注册到网关实现 **API** 生命周期管理、**API** 监控、**API** 流量策略配置和执行，实现北向流量的统一管控。

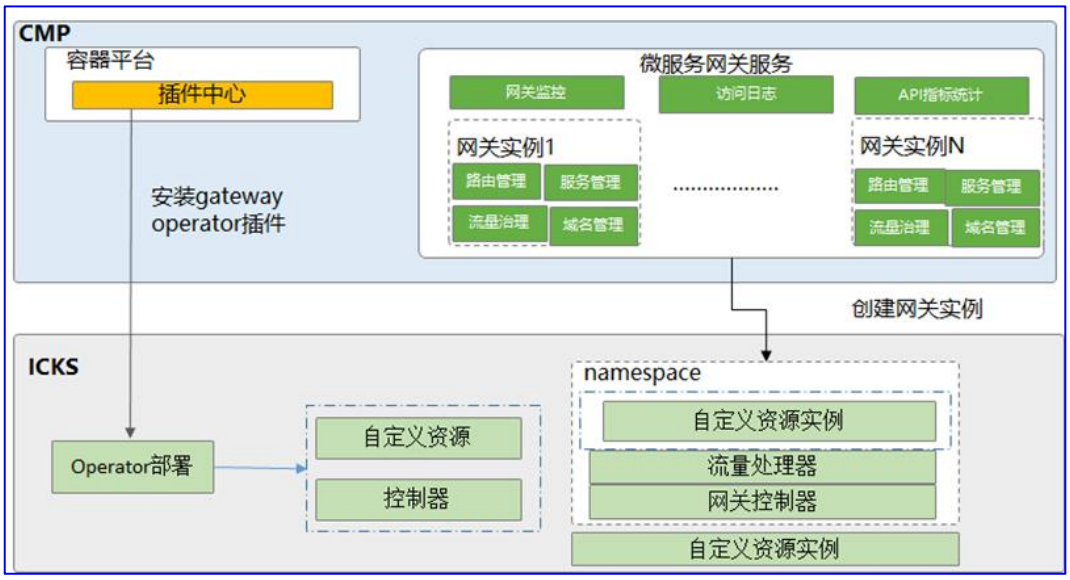
微服务网关的功能包括：

- 规格创建：为微服务网关实例创建定义统一的规格，包括部署架构、副本数、每个副本及控制器的 **CPU** 和内存大小、部署策略、参数配置等信息。

- 规格管理：对微服务网关的规格进行修改、删除、列表查看等生命周期管理
- 规格详情查看：查看规格的基本信息、实例和控制器的资源配置、部署策略、参数配置信息
- 实例创建：选择不同的规格创建租户级的网关实例，用来进行 **API** 接口及路由策略的管理
- 实例管理：对微服务网关实例进行修改、删除、列表查看等生命周期管理
- 实例详情查看：查看网关实例的基本信息、访问方式、架构拓扑、配置参数等信息
- 提供网关控制台对网关的资源进行统一的管理：**1)**、**API** 的创建、修改、删除、上线、下线、查询等生命周期管理；**2**）、**API** 的策略配置，包括路径改写、请求头重写、限流、访问黑白名单、基础认证等；**3**）、服务列表的查询；**4**）、服务来源的增、删、改、查等生命周期管理；**5**）、域名的增、删、改、查等生命周期管理；**6**）证书增、删、改、查等生命周期管理。

4.8.3.2 技术原理

复用 **InCloud OS** 的 **PaaS** 管理方式，先通过插件部署微服务网关的 **Operator** 服务，初始化自定义资源和 **Controller**，然后可以对注册配置中心实例进行全生命周期管理和自运维。



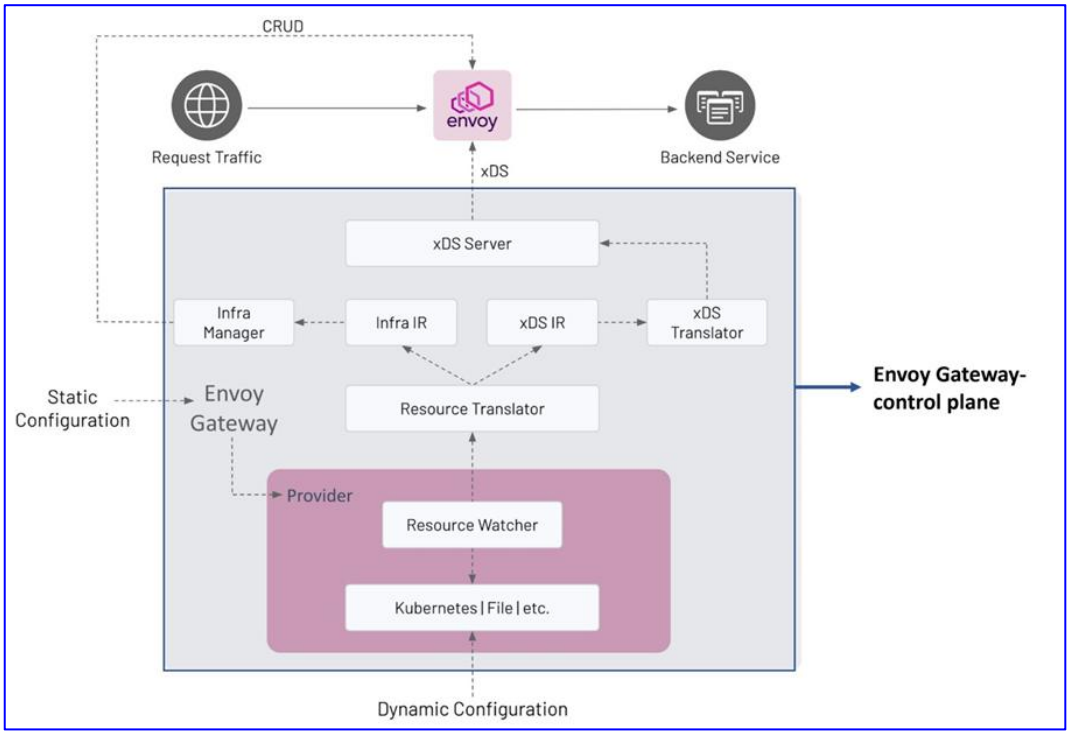
1. 微服务网关插件部署时，部署网关服务控制器并初始化自定义资源。

2. 创建网关实例时，通过创建对应的网关资源实例，网关服务控制器 **list/watch** 实例后创建网关实例控制面和数据面组件。

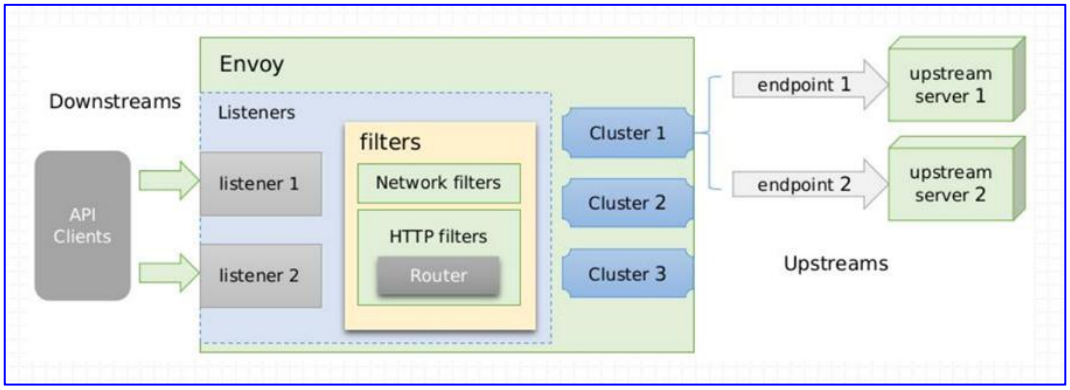
3. 在网关实例下，进行路由、服务、域名管理，完成 **api** 的发布。

4. 在路由下设置流量策略，流量访问时由网关数据面执行流量策略并将流量路由到 **api** 的实际后端服务。

微服务网关基于 **Ingress/Gateway API** 和 **Envoy** 扩展的 **API**，用于处理入口，分为控制面和数据面。



在控制面，使用 **XDS** 进行服务发现。为 **Envoy** 代理和 **Ingress** 提供动态配置更新和配置管理，**TLS** 证书管理，并进行数据解析、转化后下发到数据面组件。



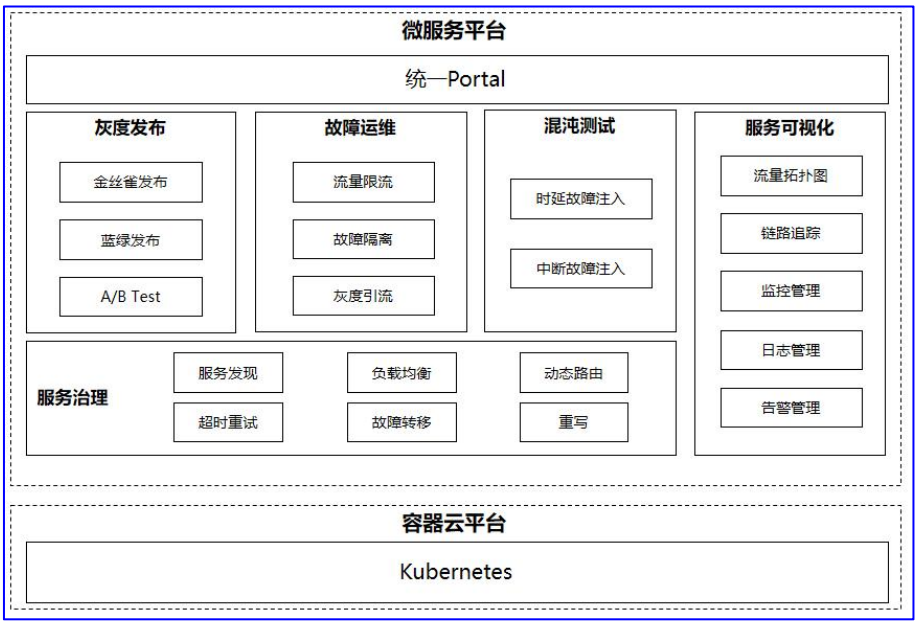
在数据面，流量经过网关（**envoy**）时，监听器识别到对应的端口后，经过过滤器执行限流、请求头重写、鉴权、熔断等流量策略后，执行路由转发，若请求不满足下一步处理要求，则直接返回响应，若需要进一步处理则路由到后端的服务进行业务处理后返回响应。

4.8.3.3 功能约束

无

4.8.4 服务网格

服务网格通过云原生技术帮助用户快速实现易接入、易运维的微服务解决方案，用户可以轻松部署、更新、测试以及治理其微服务应用。通过服务代理自动注入和代码无侵入设计，开发人员无需关心繁杂的微服务架构管理，通过简单配置即可将应用接入微服务平台，实现应用整个生命周期全方位的可视化监控与治理。



服务网格的特性功能包括：

- 服务治理：图形化应用拓扑，实时的展示应用的流量监控、异常请求、超长时延响应、服务健康状态等，并支持基于拓扑图对服务进行负载均衡、并发控制、熔断隔离、故障注入等流量治理规则配置，实现真正意义上的实时、可视化的微服务流量治理。
- 灰度发布:支持金丝雀发布，可以实现基于流量百分比或请求内容的灰

度发布策略，并支持动态的修改灰度流量策略，实现流量平滑的迁移到新版本。

- 服务可视化——为服务提供全方位的自动化链路跟踪、资源性能监控和日志管理，可视化应用整个生命周期内的运行状态，保障端到端的性能和可靠性。
- 故障运维——针对故障应用，微服务平台提供便捷的 **web** 界面操作，可以实时对应用进行自定义配置的流量限流、熔断隔离操作，保证整个应用系统的稳定性，防止故障雪崩。同时微服务平台还支持在不影响线上业务系统的前提下，将生产环境的访问流量灰度引流到其他环境进行系统故障重现与处理。
- 混沌测试——微服务平台支持对应用的单个服务或全局进行故障注入操作，可以实现时延故障注入和中断故障注入，从而测试整个系统的健壮性，达到混沌测试的效果。

4.8.4.1 灰度发布

4.8.4.1.1 功能描述

灰度发布是指在产品的新版本与旧版本间平滑过渡的一种发布方式，即让一部分用户继续用旧产品特性，一部分用户开始用新产品特性，如果用户对新产品特性没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到新产品上面来。

灰度发布支持多种微服务灰度发布方式：包括金丝雀发布、蓝绿、**A/B Test**、流量镜像。可以实现应用的平滑升级，保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。

金丝雀发布

- 支持 **UI** 选择任意开启服务治理的服务中的某一个组件进行金丝雀发布任务。
- 支持两种根据流量比例、根据请求内容两种策略方式进行金丝雀发布。
- 支持 **UI** 界面实时监控：随时动态调整金丝雀发布策略，实现 **UI** 界面一键流量接管、一键下线旧版本。
- 支持灰度策略实时动态修改：在灰度发布过程中，可以根据需求实时动态的修改灰度发布策略。

蓝绿发布

- 支持 **UI** 选择任意开启服务治理的服务的某一个组件进行蓝绿发布任务。
- 支持 **UI** 界面实时监控，实现 **UI** 界面一键流量接管、一键下线旧版本。

A/B Test

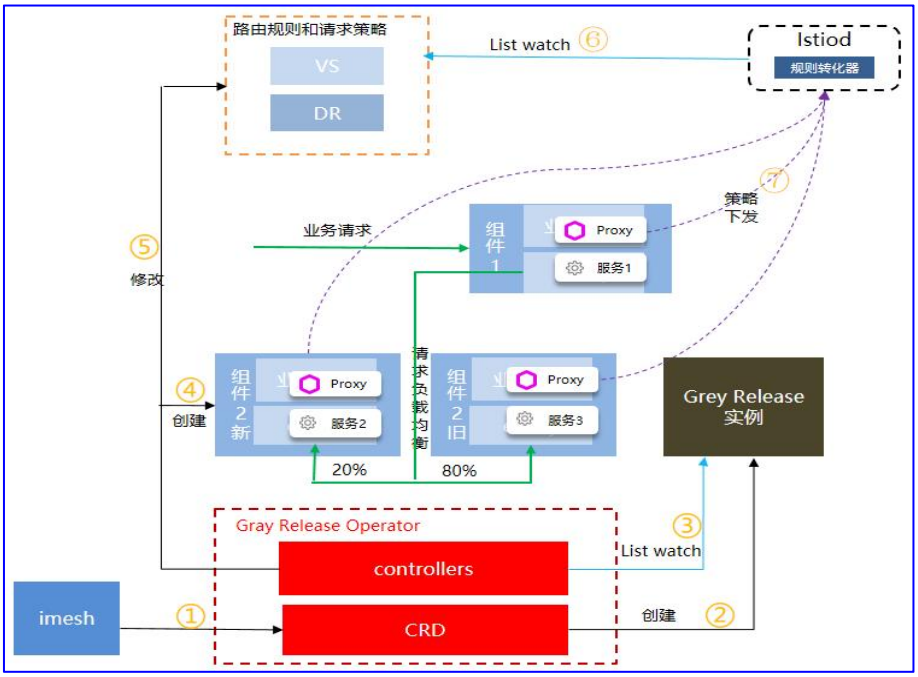
- 支持同一服务的应用组件多版本同时进行灰度发布。
- 支持两种根据流量比例、根据请求内容两种策略方式进行 **A/B Test** 发布。
- 支持 **UI** 界面实时监控 **A/B Test** 各组件版本的实时运行状态。
- 支持 **UI** 界面一键完成 **A/B Test** 灰度测试任务。

流量镜像

- 支持自定义流量比例，将旧版本组件的流量按照配置比例镜像复制到新版本组件。

- 支持 UI 界面实时监控新旧组件运行状态，并支持动态调整流量比例。
- 支持 UI 界面一键停止流量镜像任务。

4.8.4.1.2 技术原理



- 针对业务升级标准流程，自定义包含工作负载信息、规则类型、发布类型、虚拟服务、状态机等特征的发布模型。
- 利用路由规则和请求策略下发技术，将模型实例中的规则和虚拟服务下发到下游工作负载的代理容器，更新 **listener** 配置，流量经 **7** 层路由转到指定版本，完成新、旧版本的流量切换。

4.8.4.1.3 约束

- 灰度发布的服务需要开启微服务治理。
- 服务中的应用组件为非运行状态时，因服务无法访问所以不能进行灰度发布。
- 灰度发布完成后，才可继续灰度发布。

4.8.4.2 流量治理

4.8.4.2.1 功能描述

微服务系统在设计之初就需要考虑容错机制，需要引入流量治理的架构，当系统负载过高时，可以确保核心业务不被破坏，以及当某个服务出错时，把影响范围降低到最小，避免整个系统不可用。

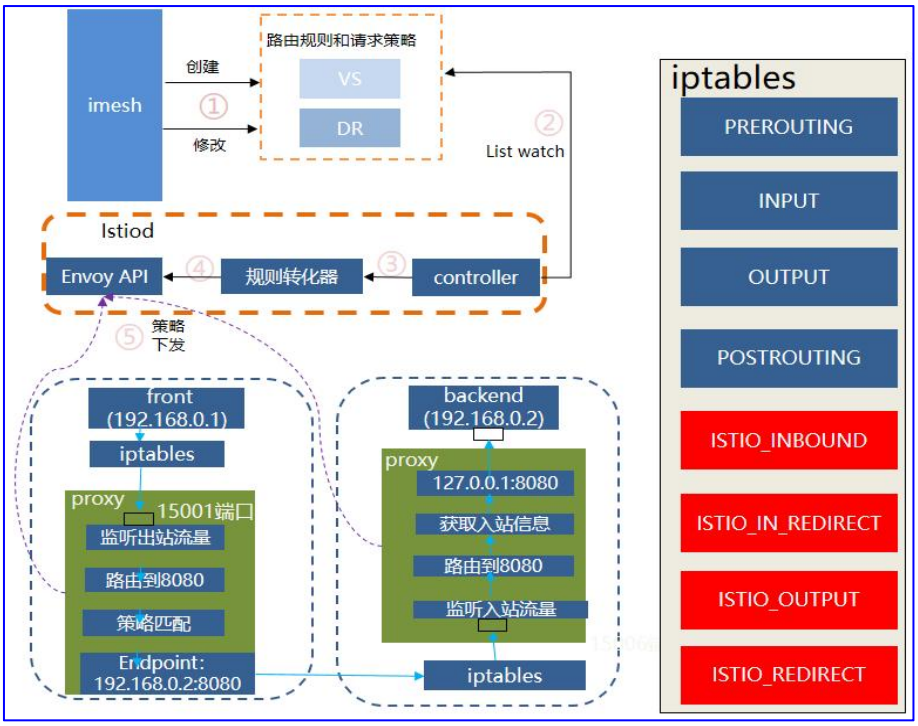
流量治理支持多种负载均衡路由策略配置，支持对访问请求进行不同维度的流量控制，可以根据客户的需求对微服务的流量分发进行动态的配置与修改。

流量治理支持图形化应用拓扑展示，实时的展现应用的流量监控、异常请求、超长时延响应、服务健康状态等，并支持基于拓扑图对服务进行负载均衡、并发控制、熔断隔离、故障注入等流量治理规则配置，实现真正意义上的实时、可视化的微服务流量治理。

流量治理的功能特性包括：

- 支持跨应用空间的微服务流量拓扑展示：可以很直观的展示微服务跨应用空间的 **pod** 级别链路调用情况，包括请求时延、每秒请求数、故障次数等详细信息。支持 **UI** 界面直接查看应用服务的流量详细信息：包括服务请求统计信息、请求错误信息等。
- 支持 **UI** 界面直接对应用的某个服务进行流量治理操作：包括负载均衡策略、超时、重试、服务的限流与熔断等。
- 支持 **UI** 界面直接对应用的某个服务进行流量安全配置：支持配置组件间 **mTLS** 策略，同时支持按端口配置 **TLS** 策略
- 支持多协议、多策略的服务流量管控能力，支持应用级别批量或全局服务流量管控配置。

4.8.4.2.2 技术原理



- 控制面基于 **list/watch** 技术监听策略变化，完成策略整合及转化。
- 数据面通过消息订阅机制动态感知策略变化，通过 **gRPC** 到控制面获取策略并更新 **listener**、**filter**、**cluster**、**route** 四个配置，依据配置实现流量治理。
- 利用 **K8S** 的 **initContianer** 技术实现对 **Pod** 的 **iptables** 设置，将自定义的规则链插入到 **Pod** 的 **iptables**，存储到 **NAT** 表中。
- 流量经过 **iptables**，根据自定义链的规则将流量重定向到代理容器；整个过程与业务容器完全解耦，实现无感知的流量劫持。

4.8.4.2.3 约束

- 流量治理的前提是服务已经开启微服务治理。
- 流量治理作用在访问地址上，因此需要当前服务下必须存在访问地址。

- 流量治理目前只作用于 **TCP** 和 **HTTP** 类型的访问，对于 **UDP** 类型的访问地址，流量治理不会生效。
- 直接通过固定 **IP** 或者是外接负载均衡方式访问的服务，被访问服务的流量治理不会生效，需要添加并使用域名访问地址。

4.8.4.3 服务追踪

4.8.4.3.1 功能描述

服务追踪为分布式应用的开发者提供了完整的调用链路还原、调用请求量统计、链路拓扑、应用依赖分析等工具，可以帮助开发者快速分析和诊断分布式应用架构下的性能瓶颈，提高微服务时代下的开发诊断效率。

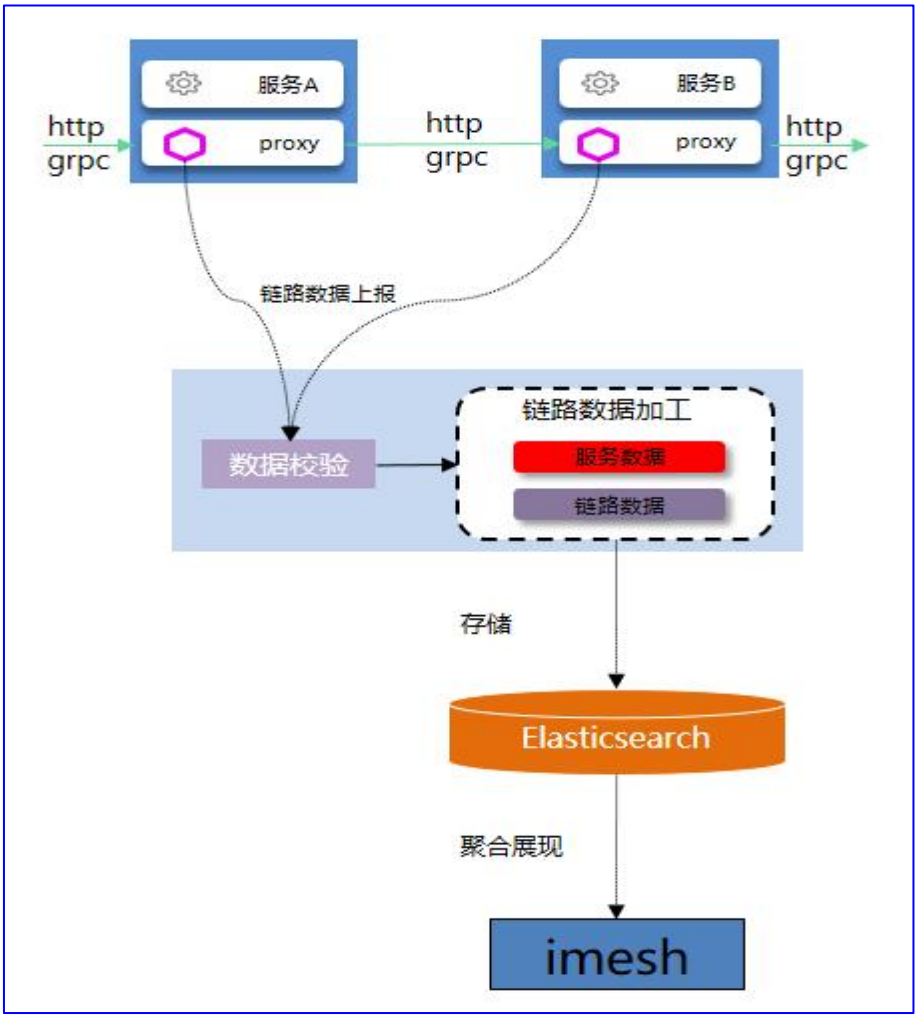
服务追踪兼容 **Opentracing** 规范，支持 **jaeger**、**zipkin** 等链路追踪与展示工具，支持应用级别链路请求最高耗时、错误请求数、总请求数概览展示，支持服务之间链路追踪信息的可视化展示与详细查询。

调用链追踪特性功能包括：

- 支持应用列表查看整个应用的链路详情信息：包括请求耗时、最长耗时、请求错误数，可以直观的展示应用的概览情况。
- 支持查看应用组件下的链路列表：支持耗时排序，最近时间排序，链路段数排序。
- 支持查看调用链瀑布图、时序图：可以直观的看出单次调用的所途经服务的上下游关系、耗时、是否成功等信息。
- 支持查看单个调用段详情：包括请求 **id**、链路段 **id**、**http** 方法、**http** 协议、**http** 状态、**http** 请求路径、容器组 **ip**、容器组名称、请求大小、

返回大小、响应标识、数据流向、客户端等。

4.8.4.3.2 技术原理



- 基于 **open-tracing** 规范，构建出链路追踪的概念模型与数据模型。
- 基于 **tcp** 的解包、封包技术，代理容器对劫持到的数据包解包添加链路模型数据后重新封包，实现链路数据的收集与传递，做到代码无侵入。
- 将上报数据抽象为服务和链路两类索引保存到 **elasticsearch** ，并对数据进行聚合展现。

4.8.4.3.3 约束

- 需要链路追踪服务的应用需要接入微服务治理。

- 网络请求类型必须是 **http** 服务，并且应用服务在发起请求时，默认传递 **http** 的 **header** 内容。
- 目前链路追踪只作用于网络请求层，无法对方法内部过程做链路追踪。

4.8.4.4 方法级链路追踪

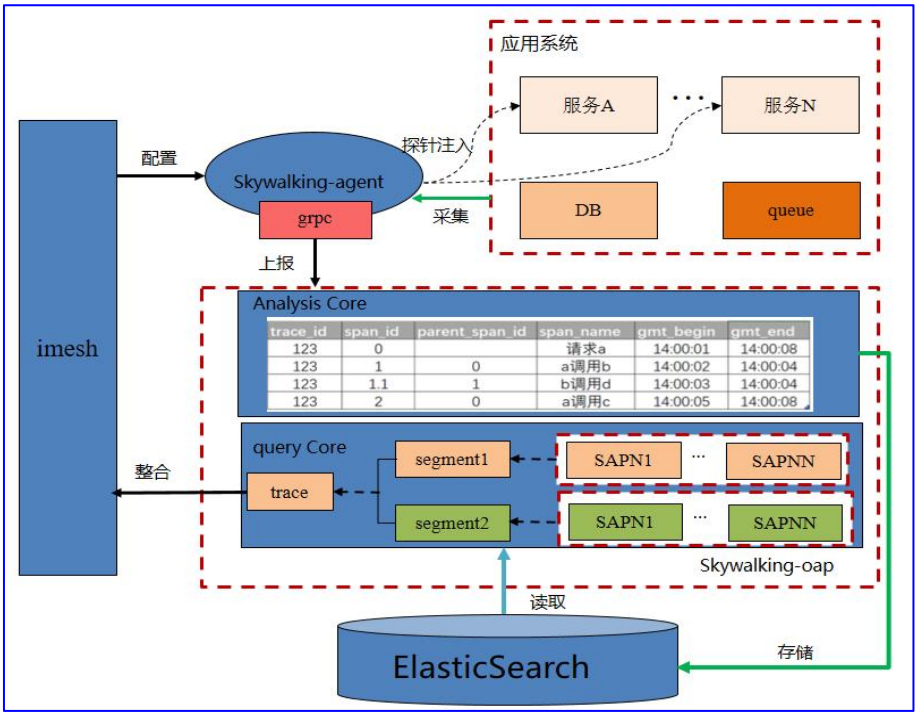
4.8.4.4.1 功能描述

平台基于 **skywalking** 架构，支持方法级链路追踪功能，可以实时展示应用请求途径的各组件内部自定义方法的执行情况，满足用户更细粒度问题故障及性能瓶颈定位的需求。

方法级链追踪特性功能包括：

- 支持多种语言方法级链路追踪信息的收集与展示，包括 **java**、**php** 等常用语言。
- 支持 **Java**、**Nodejs** 服务启动时自动注入 **skywalking agent** 的方式实现方法级链路追踪，对业务无感知。
- 支持应用随时开启或关闭方法级链路追踪。

4.8.4.4.2 技术原理



- 通过共享文件系统，实现采集探针的无侵入式注入。
- 对于 **java** 应用，通过字节码增强技术；对于 **node.js** 应用，在程序入口处添加拦截代码，通过动态加载机制，实现两种应用的代码埋点。
- 通过可观测性分析平台对 **agent** 上报的数据进行整合，基于 **open-tracing** 模型实现服务的方法级别的全链路追踪。

4.8.4.4.3 约束

- 基于 **java** 的方法级链路追踪，自动注入支持 **JDK1.8** 及以上版本，**tomcat7** 及以上版本，**jetty9** 及以上版本。
- 基于 **Node.js** 的方法级链路追踪，自动注入支持 **nodejs v10.x.x/v12.x.x/v14.x.x** 版本。

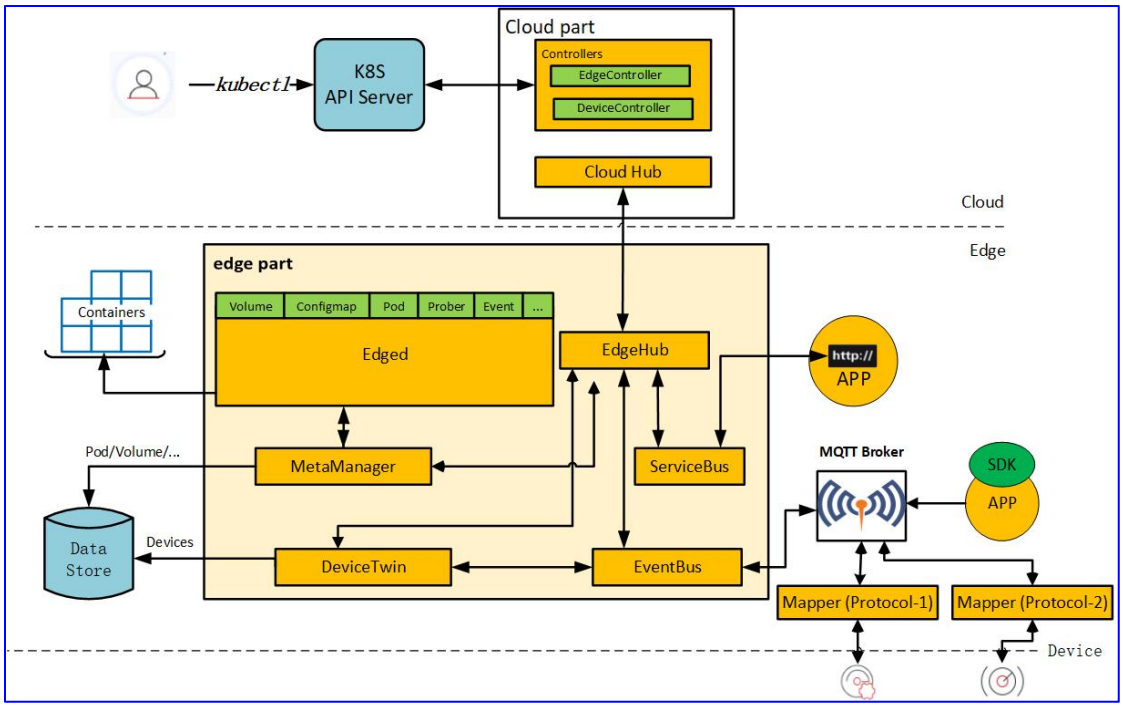
4.9 边缘服务

4.9.1 功能描述

边缘服务将云数据中心的云计算能力延伸到边缘侧，提供边缘资源和边缘应用的管理能力，满足边缘应用对低时延、高带宽和合规性等要求，具体功能包括边缘资源统一管理和调度、边缘自治、边缘轻量化以及边缘应用跨域统一分发等，与云服务一起实现云边协同。

4.9.2 技术原理

边缘服务基于容器和 **Kubernetes** 技术架构实现，构建一个云边协同资源池，管理节点部署在云端，将计算节点扩展到边缘侧，支持云边不稳定网络下的消息同步，实现边缘计算节点统一接入和管理，将容器应用部署到边缘节点中。



该云边协同框架主要由云–边两部分构成，主要功能模块包括：

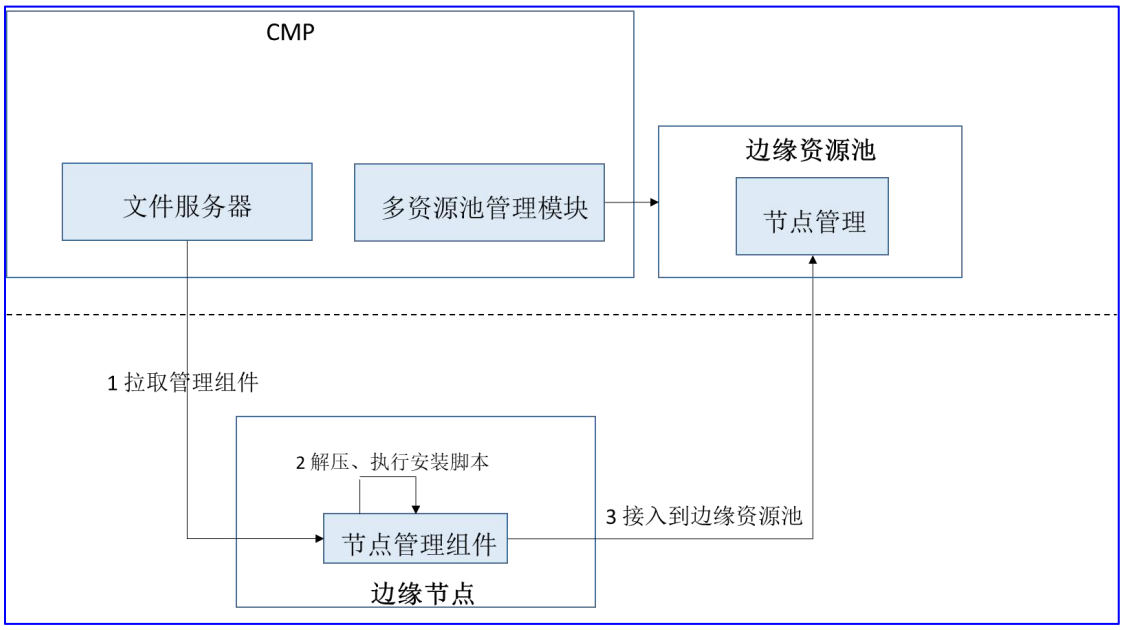
- 云端控制器：包括两个扩展的 **Kubernetes** 控制器 **EdgeController** 和

DeviceController，EdgeController 负责管理边缘节点和 Pod 元数据的下发，DeviceController 负责管理边缘端设备，确保设备信息、设备状态的云边同步；

- 云边通信隧道：由 CloudHub 与 EdgeHub 两个组件建立，Cloudhub 是部署在云端的一个 Web Socket 服务端，负责监听云端变化，将消息发送到边端的 Edgehub，Edgehub 是部署在边端的 Web Socket 客户端，负责将云端消息同步到边缘其他组件，同时将其他组件的消息同步到云端；
- 边缘自治：由 MetaManager 和 DataStore 组成，MetaManager 为 kube-apiserver 代理，承担当前边缘节点 Apiserver 的责任，负责 Edged 与 Edgehub 的交互，同时将数据缓存到 DataStore 中，在与云端断开时能够继续管理本地的 Edged 组件；
- 组件轻量化：Edged 是运行在边缘节点的管理组件，一方面通过通信隧道接入到云端，另一方面管理本地的容器化应用，通过 kubelet 裁减实现管理组件轻量化；

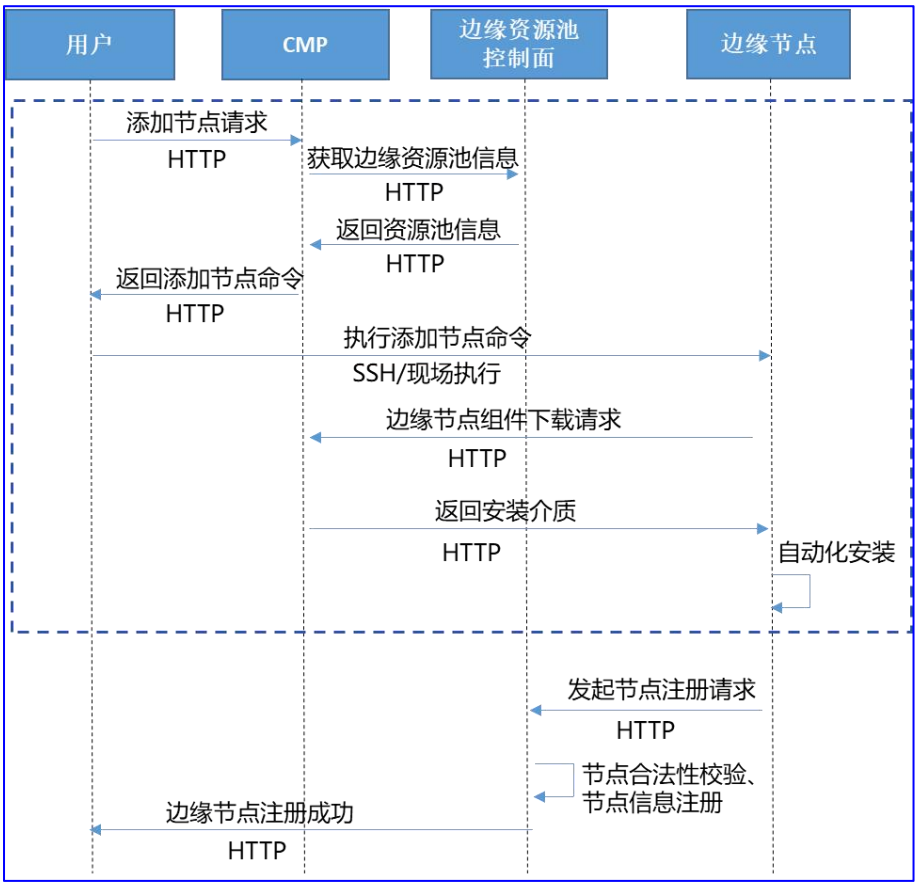
4.9.2.1 边缘节点统一接入

边缘节点接入的前置条件是，需要将部署完成的 K8S 集群以边缘资源池的形式接入到统一管理平台 CMP 中，在云端生成边缘节点的接入命令，然后在边缘节点执行，实现一键接入，整体的技术架构如下图所示。



边缘节点执行接入命令后的关键原理如下：

- 执行命令后，会首先到云端文件服务器拉安装包，主要为边缘节点的管理组件以及相关的安装脚本；
- 获取到安装包后，解压后执行安装脚本，启动边缘节点的管理组件；
- 管理组件启动后，自动将该节点加入到边缘资源池中，成为 **K8S** 集群的一个 **Node** 节点，边缘节点注册的详细流程如下图所示；

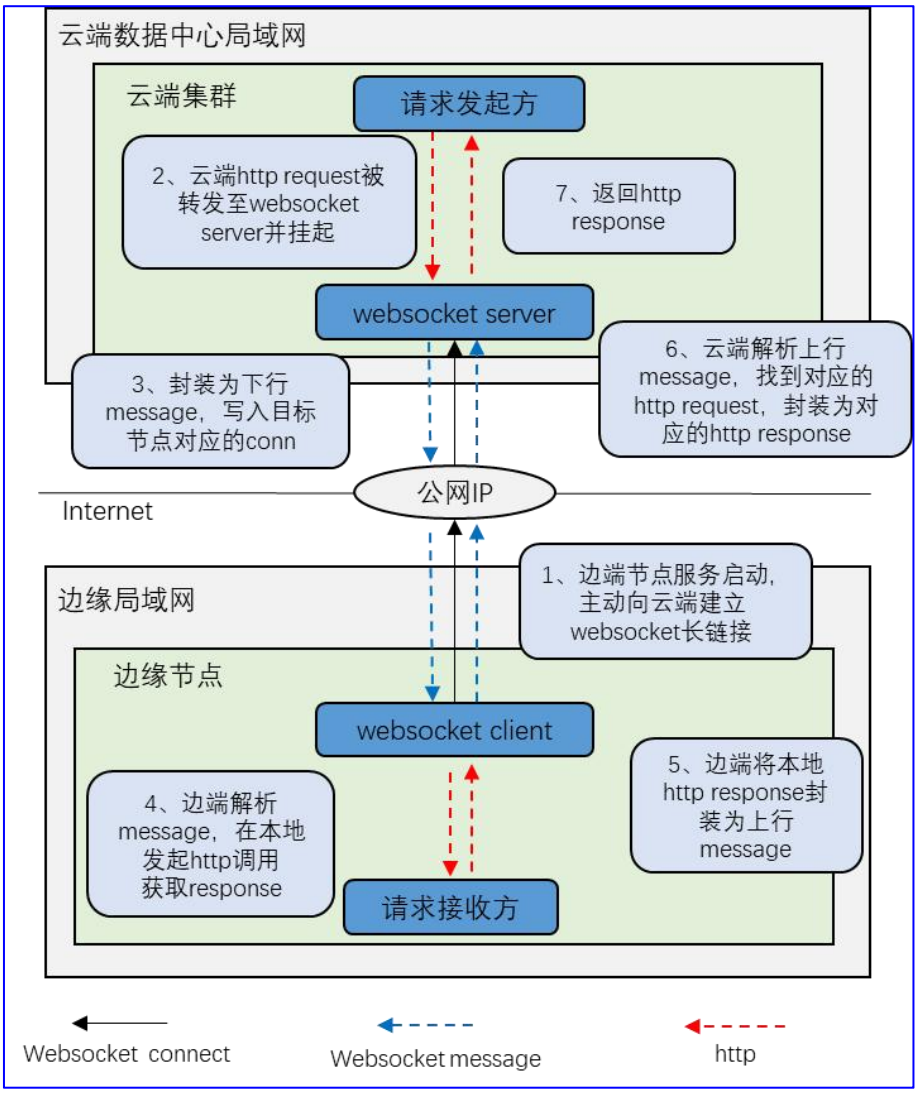


- 基于 **token**、接入个数以及设备序列号的关联认证机制，实现了一个接入命令达到批量边缘节点接入的效果。

4.9.2.2 云边网络隧道

云边单向网络通信隧道基于 **websocket** 通信协议实现，边缘节点通过单向网络主动连接云端控制节点后，实现双方的双向通信。每个边缘节点服务启动时 **proxy-agent** 主动向云端 **proxy-server** 建立两个 **websocket** 长链接通道，云端维护边缘节点名和长链接通道的对应关系，通过元数据通道进行该节点所关注的 **K8S** 对象生命周期变化、状态变化等消息的主动下发，通过运维通道进行如 **logs/exec** 等运维指令的主动下发。

云边隧道通信技术的原理如下图所示，主要包括以下几个方面：



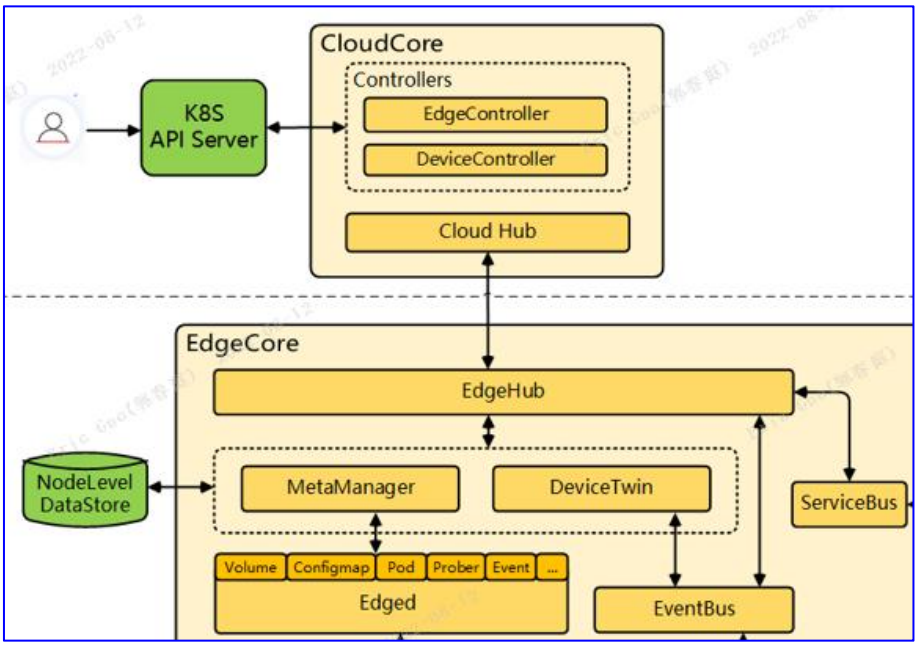
- 云端：云端配置路由拦截规则，将指向边端的 **http** 请求拦截到云端的 **websocket server** 上；同时，云端的 **controller** 监听到资源变动后，主动调用云端 **tunnel-server** 接口，下发资源同步请求。
- 边缘节点：实现 **websocket client**，主动与 **websocket server** 建立长连接，将自身信息上报给云端。同时实现 **message** 解析逻辑，从云端下发的 **message** 中解析出目标地址、目标协议，在本地发起调用。
- 控制和运维流管理：云边控制流和运维流通过两个长链接通道实现，技术原理一致，由于长链接隧道是顺序读写，这样可以防止双方流量互相影响，控制流包括云端对象的创建、变更等消息下发到边端、边端资源

状态变化上报到云端，运维流包括云端主动执行的 **exec**、**metrics**、**logs** 等 **k8s-apiserver** 主动向边端发起调用的流量；

4.9.2.3 边缘自治

原生 **K8S** 中，管理组件 **Kube-Apiserver** 与计算节点组件 **Kubelet** 直接通信，如果 **Kubelet** 无法通信后，则会报告该 **Kubelet** 对应的 **Node** 为 **NotReady** 状态。为了解决断网但节点还在继续运行 **Pod** 被驱逐的问题，需要在边缘节点实现自治组件，能够在无法连接到 **Kube-apiserver** 的时候，采用本地代理进行管理 **Kubelet**。

边缘自治包括两个核心设计，一个是 **Apiserver** 代理，一个是本地数据缓存，如下图所示。



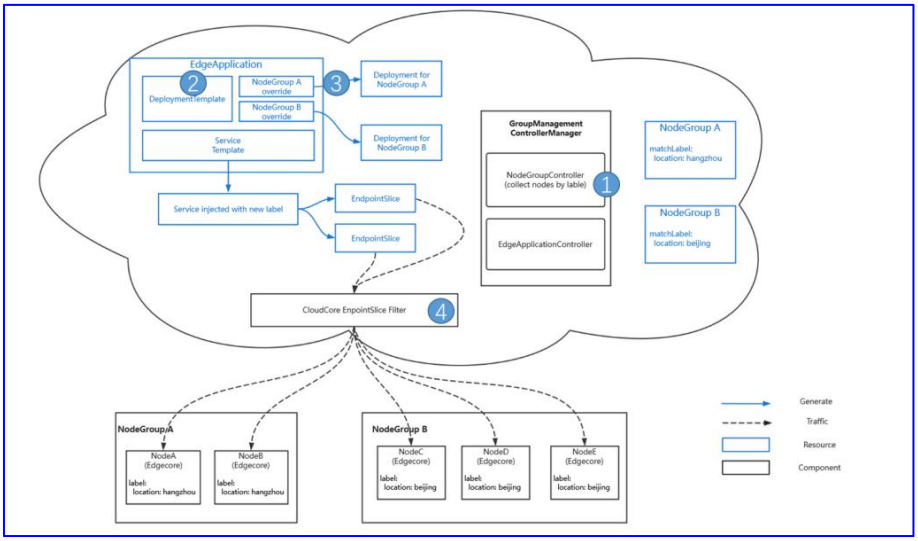
- **Apiserver** 代理：云边之间的请求交互，全部由 **Apiserver** 代理进行转发，云边网络可达时，转发给云端处理，云边断网时，请求本地元数据进行响应，而请求又包括上行请求和下行请求：
- 上行请求：**edged** 管理的 **node**、**pod** 的状态信息，它先报到 **MetaManager**

这边，**MetaManager** 在传到 **EdgeHub**，经过 **edgehub** 把数据同步到云上。这样就实现了 **node**、**pod** 状态的上报。当断网重启后，**edged** 从本地数据库中拿到相应数据进行服务恢复。

- 下行请求：**CloudHub** 下发事件到 **EdgeHub**，如果是 **pod** 的元数据，**Edged** 就推给 **metamanager** 以及 **Edged** 去拉起相应容器，同时将相应的数据保存在数据库中，如果是设备信息，就推给 **DeviceTwin** 通过 **EventBus** 与端设备交互。
- 本地元数据缓存：采用 **Sqlite** 轻量级的数据库作为元数据的本地缓存后端，每次的更新请求都会存储到缓存数据库中。

4.9.2.4 跨域应用分发

在边缘场景下，某些容器应用需要部署到多个地域中，原始做法是需要重复分发，这样做的问题是，后期升级维护复杂度高。为了解决该问题，基于节点池和自定义部署模型实现跨地域应用统一分发，技术架构图如下所示。



- 通过实现自定义资源 **NodeGroup** 对边缘进行池化管理，将边缘集群中位于同一地域的节点纳管到统一的节点池中，通过节点池的维度对边缘

节点进行统一的管理与运维。

- 基于边缘集群中的节点池进行应用的分发以及统一生命周期管理，通过实现自定义资源 **EdgeApplication** 实现差异化的 **Deployment** 分发，即对不同地域的应用进行统一的分发创建、更新以及删除等全生命周期管理操作，并可以在分发时为不同地域的应用指定相应的镜像、副本数等以满足不同地域的差异化配置。
- 边缘节点访问 **Service** 时，只会访问本地域内的 **Pod** 副本，实现 **service** 的地域内闭环访问。云端向边缘节点更新 **service** 时，只更新该边缘节点所属 **nodegroup** 中的节点上的 **endpoints**。

4.9.3 约束

- 边缘应用当前只支持本地存储卷进行应用的持久化。

4.10 容灾

云平台容灾服务整体上包括三个关键的方面：云管理容灾、虚拟化容灾和容器应用容灾。

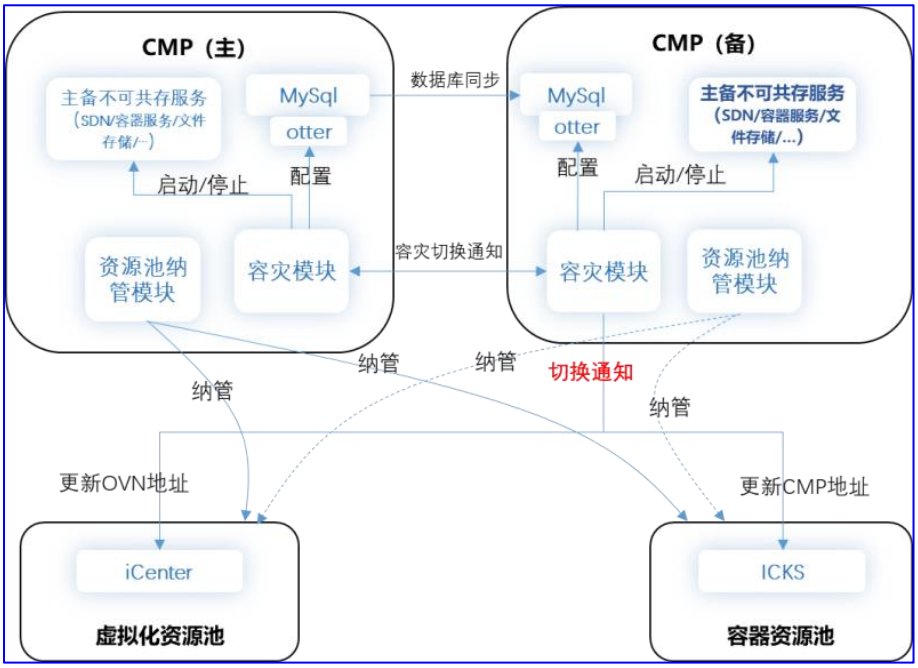
4.10.1 CMP 容灾

4.10.1.1 功能描述

CMP 主备容灾，即在生生产中心和灾备中心各部署一套云管平台，通过数据复制技术，将生产环境数据复制到灾备环境，保证两侧数据一致。生产环境 **CMP** 承载正常的业务操作，备 **CMP** 正常情况下不启用，上面不进行任务操作。当生产环境 **CMP** 由于各种原因（病毒、硬件故障、断电、火灾、地震等）发生

故障时，备 CMP 启用，接管资源管理，对外提供服务。

4.10.1.2 技术原理



CMP 主备容灾的技术架构如上图所示，涉及的关键技术主要包括如下：

- 数据库同步：通过数据库同步工具 otter 来实时增量同步主备 MySql 实例间的数据，云管元数据不丢失；
- 多纳管冲突：为了加快切换速度，备 CMP 以热备的形式存在，但某些服务会有定时自动请求发送到资源池，造成多纳管冲突，故需要开启容灾时，将备 CMP 中部分冲突服务停止；
- 切换自动感知：资源池依赖 CMP 中的服务，如 SDN、容器服务等，当 CMP 切换后，能够自动通知所有资源池进行更新，资源池收到消息后，进行地址信息更新，包括容器镜像库、OVN 地址等；

CMP 容灾主备切换（模块）由前端 UI 界面和后台容灾切换服务两部分组成。后台服务提供 Rest Api 给前端页面调用。用户通过页面完成容灾的环境配置，开启主到备数据同步，主到备切换，备到主切换等操作，主要步骤如下：

- 容灾配置：配置主备容灾环境和解除容灾，主要进行配置和清除主、备 CMP 系统的环境信息以及主、备标签设置、启停数据的同步。
- 主备同步：主要包括全量、增量同步数据库数据以及停止不同同时存在的服务。采用数据库的导出、传输和目标环境导入，实现数据库数据的初始全量同步；通过 otter 实现数据库数据的增量实时同步；在备 CMP 中停止与主 CMP 同时存在冲突的服务，如容器、软件定义层等。
- 主备切换：将主 CMP 设置为禁用状态、备 CMP 设置为启动状态，启动备中停止的相关服务，通知裸机服务、SDN 等模块环境发生切换。
- 主备回切：将主 CMP 设置为启动状态、备 CMP 设置为禁用状态，全量同步数据库数据到主，并开启 otter 进行实时同步。启动主 CMP 中停止的相关服务，停止备 CMP 中相关服务，通知裸机服务、SDN 等模块环境发生切换。

4.10.1.3 约束

- 备环境开放数据库外部访问端口和 SSH 端口。
- 主备网络带宽：>业务平均写 IO 带宽。

4.10.2 虚拟化容灾

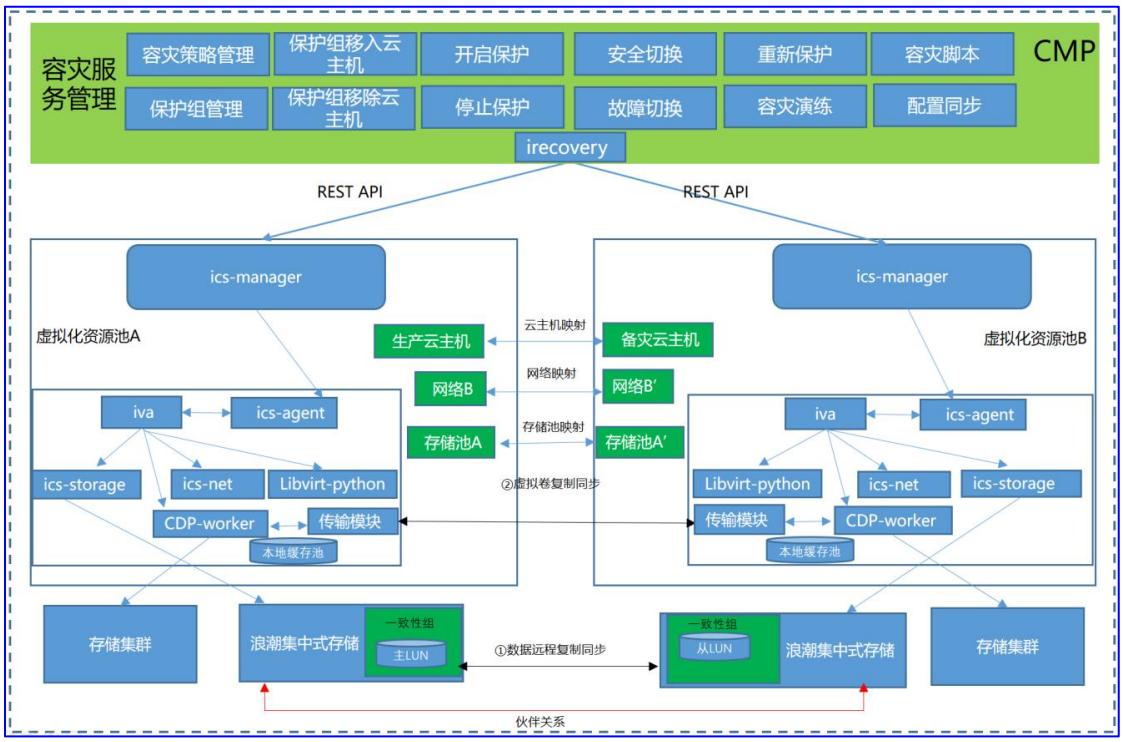
4.10.2.1 功能描述

云管理平台同时管理两个资源池：一个生产资源池，一个灾备资源池。两个资源池建议在同城，在同城内可以保证业务数据同步和网络通信。主资源池承载全部业务，备资源池承载核心业务的备份，业务运行在主资源池，在主资源池故障时，在备资源池能够快速恢复已保护的业务，并对外提供服务，提供两种容灾

模式：阵列复制模式和虚拟卷复制模式，其中阵列复制模式支持存储的同步复制（**RPO=0**）或异步复制（**RPO** 约等于零）；虚拟卷复制模式下 **RPO** 可根据网络带宽情况自定义设置，最小 **30s**，最大 **24** 小时。

在阵列复制模式下，主资源池和备资源池的存储系统通过波分设备或者 **IP** 专线进行数据的远程复制；在虚拟卷复制模式下，主资源池和备资源池间通过独立于管理网络的二层容灾网络（要求时延小于 **50ms**）进行数据的远程复制。云管理平台通过三层网络管理主和备资源池的资源，客户通过互联网或者企业网访问业务系统，并能够基于负载均衡进行故障切换后继续访问。

4.10.2.2 技术原理



云主机容灾管理架构图如上图所示，用户通过 **CMP** 控制台管理两套虚拟化资源池，容灾服务管理模块发送请求到各自虚拟化资源池的 **ics–manager** 服务，并将请求参数整合处理后发送到 **ics–agent**，然后转发到 **iva**，接着通过 **ics–storage** 实现对存储集群资源的管理操作，通过 **ics–net** 实现对网络资源的管理，通过

libvirt-python 实现对云主机资源的管理，通过 **CDP-worker** 和传输模块完成虚拟卷 **IO** 复制传输。容灾服务关键技术包含：

- 云管理层编排

云管理平台提供云主机的异地灾备能力，依靠底层存储的远程复制能力，分别将两个存储分布在两个虚拟化资源池，通过云管理平台统一管理两个虚拟化资源池，并且管理生产资源池和备灾资源池的容灾资源，核心资源包括：

- 容灾策略管理 支持将主和备资源池的公共参数抽象为一个模板，例如容灾模式、主可用区、主资源池、备可用区、备资源池和设置容灾网络带宽，目的是减少创建保护组时重复输入参数。
- 保护组管理 保护组是容灾保护最重要的资源，创建保护组需要指定容灾策略、主备存储池映射和主备网络映射。
- 保护组详情展示：支持展示基本信息、容灾云主机统计（总量，并按照开机、关机和其它状态统计）、保护组传输速率历史曲线、目的存储池（使用率和读写速率）、源存储池（使用率和读写速率）。
- 容灾统一视图：支持云平台统一视角展示保护组统计（总数，并按照空闲、未保护、保护中、故障切换和其它状态统计）、容灾云主机统计（总量，并按照开机、关机和其它状态统计）、容灾存储资源使用率（容量使用率、读写速率）、容灾网络带宽使用率。
- 容灾云主机管理 支持在保护组内加入待保护的云主机，在阵列复制模式下：需要将云硬盘从普通存储池迁移到容灾存储池，并在备资源池中创建备云主机；从保护组中移除待保护的云主机，需要将云硬盘从容灾存储池迁移到普通存储池中，并在备资源池中删除备云主机。在虚拟卷

复制模式下：待保护的云主机需要来自于已添加到保护组内的存储映射上的云主机，并在备资源池中创建备云主机；从保护组中移除待保护的云主机，会在备资源池中删除备云主机。

- 容灾云主机启动优先级管理：支持为加入保护组内的云主机设置启动优先级，实现在安全切换、故障切换时，占位云主机能够根据优先级顺序进行云主机的启动，保证高优先级云主机快速恢复业务，低优先级云主机延迟启动，保证故障后用户业务的有序恢复。
- 开启保护 在保护组中加入待保护的云主机后，需要开启保护，实现生产资源池数据同步或异步地复制到备资源池，并支持校验存储池的连通性。
- 停止保护 在不需要将生产资源池的数据复制到备资源池时，需要停止保护，实现中断数据的复制，并支持校验存储池的连通性。
- 安全切换 针对计划内的业务切换，可以通过安全切换功能，实现业务在生产资源池和备资源池间业务互切，并支持校验存储池的连通性。
- 故障切换 针对计划外的生产资源池故障，可以通过故障切换功能，在备资源池拉起云主机，继续对外提供业务，并支持校验存储池的连通性。
- 重新保护 针对已进行故障切换的保护组，生产资源池已经恢复后，此时云主机尚未回切，继续在备中心允许，在执行主备回切前，需要先进行重保护，实现在备资源池产生的增量数据同步到生产资源池，并支持校验存储池的连通性。
- 保护组统一管理

建议将同一业务的生产卷加入到同一个保护组管理，主要是保证云平台侧具

有业务关系的一组云主机所附属的云硬盘之间的协同数据一致性。比较典型的是数据库的数据卷和日志卷，一个事务操作往往需要同时更新数据卷和日志卷，此时如果只单一保护其中任何一个卷，或者保护时间不统一，就会造成整体数据库的数据不一致。同时通过一致性组，可实现单个云主机或一组云主机在主备或故障切换时的操作一致性，避免因为主备或者故障切换时间不一致导致的数据差异使业务不可用。

◆ 阵列复制模式技术

基于阵列复制模式方案有两种方式：同步复制方式和异步复制方式。

同步复制方式：可以做到主/备中心磁盘阵列同步地进行数据更新，应用系统的 **I/O** 写入主磁盘阵列后(写入 **Cache** 中)，主磁盘阵列将利用自身的机制（如浪潮集中式存储的 **InRemoteCopy**）同时将写 **I/O** 写入后备磁盘阵列，后备磁盘阵列确认后，主中心磁盘阵列才返回应用的写操作完成信息。采用同步方式，使得后备磁盘阵列中的数据总是与生产系统数据同步，因此当生产数据中心发生灾难事件时，不会造成数据丢失。为避免对生产系统性能的影响，同步方式通常在近距离范围内（**FC** 连接通常是 **100KM** 范围内，实际用户部署多在 **35KM** 左右）。

异步复制方式：在应用系统的 **I/O** 写入主磁盘阵列后(写入 **Cache** 中)，主磁盘阵列立即返回给主机应用系统“写完成”信息，主机应用可以继续进行读、写 **I/O** 操作。同时，主中心磁盘阵列将利用自身的机制（如浪潮集中式存的 **InRemoteCopy**）将写 **I/O** 写入后备磁盘阵列，实现数据保护。而采用异步方式应用程序不必等待远程更新的完成，因此远程数据备份的性能的影响通常较小，并且备份磁盘的距离和生产磁盘间的距离理论上没有限制（可以通过 **IP** 连接来实现数据的异步复制）。

- 虚拟卷复制容灾模式技术

虚拟卷复制容灾模式是通过 **CDP**（持续数据保护）技术实现，实现原理为云主机在执行写 **IO** 时，系统自动捕获云主机的 **IO** 信息，将写 **IO** 信息封装为 **IO** 日志缓存文件，主资源池系统自动将 **IO** 日志缓存文件传输到备资源池，备资源池将 **IO** 日志合入到本地的云硬盘中，从而实现异地资源池之间的数据同步。可通过 **RPO**（**Recovery Point Objective**）配置项设置资源池之间的同步频率，系统自动将一个 **RPO** 周期内的云主机 **IO** 封装为一个 **IO** 日志缓存文件发送到异地资源池。

4.10.2.3 约束

- 阵列复制模式

❖ 1.1 同步复制组网要求：

- 容灾网络距离： **<100km**，光纤网络
- 回路延时：**<3ms**
- 网络带宽：>业务峰值写 **IO** 带宽

❖ 1.2 异步复制组网要求

- 容灾网络距离： 无限制
- 回路延时：**1Gbps IP**，**<80ms**；**10Gbps IP**，**<10ms**
- 网络带宽：>业务平均写 **IO** 带宽
- 阵列复制模式

❖ 2.1 容灾网络带宽规划：

- 数据中心互联线路
- ①**DWDM+**裸光纤：高性能，低时延，成本高

- ②**MPLS VPN** 专线：高性价比，配置简单
- 时延选型
- 无距离限制,推荐使用裸光纤或者运营商专线作为容灾站点之间的数据传输线路，时延小于 **50ms**，丢包率小于千分之一
- 带宽选型

容灾网络带宽应至少保证不低于所有容灾云主机的平均数据增量速率,即容灾云主机数量*每云主机 **RPO** 周期内平均变化数据量（单位 **MB**）***8/RPO** 周期（单位秒）

- 带宽计算公式
- 容灾网络带宽具体评估公式：**Z=k*(SUM*8/(PRO－SUM/Y))**，在 **RPO**（单位秒）周期内云主机业务产生的总增量数据是 **SUM MB**，缓存存储池最大稳定写入速率为 **Y MB/s**，**k** 系数是服务响应时间和网络带宽损耗，取固定值 **1.2**，容灾网络总带宽为 **Z Mbps**。请注意，评估公式中的 **RPO–SUM/Y** 如果为负值，说明增量数据落缓存盘速率过慢，落盘占用的时间超过了 **RPO** 时间，需要更换更高性能的缓存盘或者调大 **RPO** 的值。

❖ 2.2 缓存存储池规划

- 单个云主机 **IO** 日志最大使用空间计算公式：平均写 **IO** 速度***3600**（**1** 小时秒数）*保留时间（小时数）**/1024=GB**。例如，业务系统平均写 **IO** 速度为 **10MB/s**，保留时间为 **1** 天，则 **IO** 日志最大使用空间为：**10*3600*24/1024=844GB**。

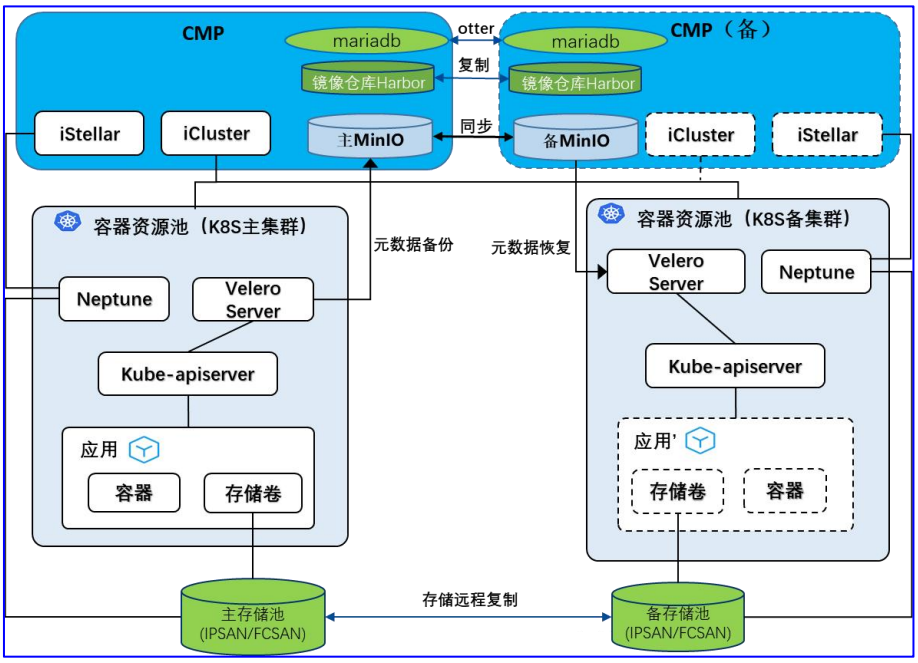
4.10.3 容器应用容灾

4.10.3.1 功能描述

当业务系统在资源池升级、硬件故障等计划内和计划外等场景时，如何快速恢复客户业务系统，并能够保留业务数据，是用户急迫的关键需求。本平台的容器应用跨资源池容灾功能，能够将容器应用元数据进行备份，同时对存储卷数据进行跨集群近实时复制，在需要时，能够通过主备切换/故障切换快速恢复业务。同时，为了满足不同的业务场景，支持只备份元数据，也可同时备份两者。

4.10.3.2 技术原理

本平台的容器应用跨资源池容灾备份能力是基于开源备份框架 **velero** 和具备远程复制能力的浪潮集中式存储的技术路线实现，整体架构图如下图所示。容灾备份是基于 **velero** 将应用元数据备份到系统自带对象存储 **MinIO** 中，容灾恢复(主备切换/故障切换)时，将元数据备份从对象存储拉取下载到备资源池，并将应用恢复到备资源池中。基于浪潮 **IPSAN/FCSAN** 存储的远程复制能力，实现应用存储卷的跨资源池容灾能力



跨资源池元数据备份流程如下：

- **cmp** 调用容器主资源池 **k8s API**，查询应用需要备份的元数据的资源类型和名称，并调用容器主资源池 **API** 创建备份 **backup CRD** 资源。
- 主资源池 **velero** 的备份 **CRD** 资源控制器监听到 **backup CRD** 资源创建并进行校验，判断 **backup CRD** 合法性。
- 备份资源控制器通过 **k8s API** 查询 **backup CRD** 中备份元数据的 **yaml** 信息，打包上传到对象存储 **MinIO**。
- 对象存储 **MinIO** 通过桶复制能力，将主中心备份元数据实时同步到备中心 **MinIO** 对象存储系统中。

跨资源池元数据恢复流程如下：

- **cmp** 调用容器备资源池 **k8s API**，以备份 **backup CRD** 为基础，创建恢复 **restore CRD** 资源。
- 备资源池 **velero** 的 **restore CRD** 资源控制器监听到 **restore CRD** 资源创建并进行校验，判断 **CRD** 资源合法性。

- **restore CRD** 资源控制器根据 **restore CRD** 中的元数据信息，从对象存储下载备份元数据的 **yaml** 文件，并调用备资源池 **k8s API** 恢复应用到备资源池

跨资源池存储卷复制流程如下：

- 主备存储池提前建立伙伴配对关系。
- 调用主存储池接口，创建一致性组，状态为空，数据复制方向为主到备。
- 主资源新建存储卷，为主卷；备资源池建立与主卷对应的备存储卷，主备卷的的所有参数均相同。
- 主备卷间建立远程复制关系，并将远程复制关系加入一致性组。
- 一致性组开启同步，组内所有远程复制关系的主备卷间开启远程复制，一致性组状态为非一致性同步。
- 待一致性组状态由非一致性同步变为一致性同步后，主备卷数据同步完毕，后续的同步为增量同步。

4.10.3.3 约束

- 主备资源池 **kubernetes** 版本需要相同。
- 主备资源池网络双栈配置需保持一致。
- 主备存储池需提前配置伙伴关系。
- 元数据备份周期默认 **40** 分钟，即可能丢失最长 **40** 分钟内应用元数据更改。

4.11 多云管理

4.11.1.1 功能描述

多云环境资源统一服务交付及生命周期管理是一个重要的应用场景，多云管理的功能可以实现不同类型的虚拟化平台统一的纳管、云主机统一管理。云管平台通过实现多云的统一接入，可以统一多个层面维度的管理，对多云资源按照组织行政、业务等多维度进行统计分析展示，帮助全局了解多云的整体使用情况，从而有效降低管理的复杂度和压力。

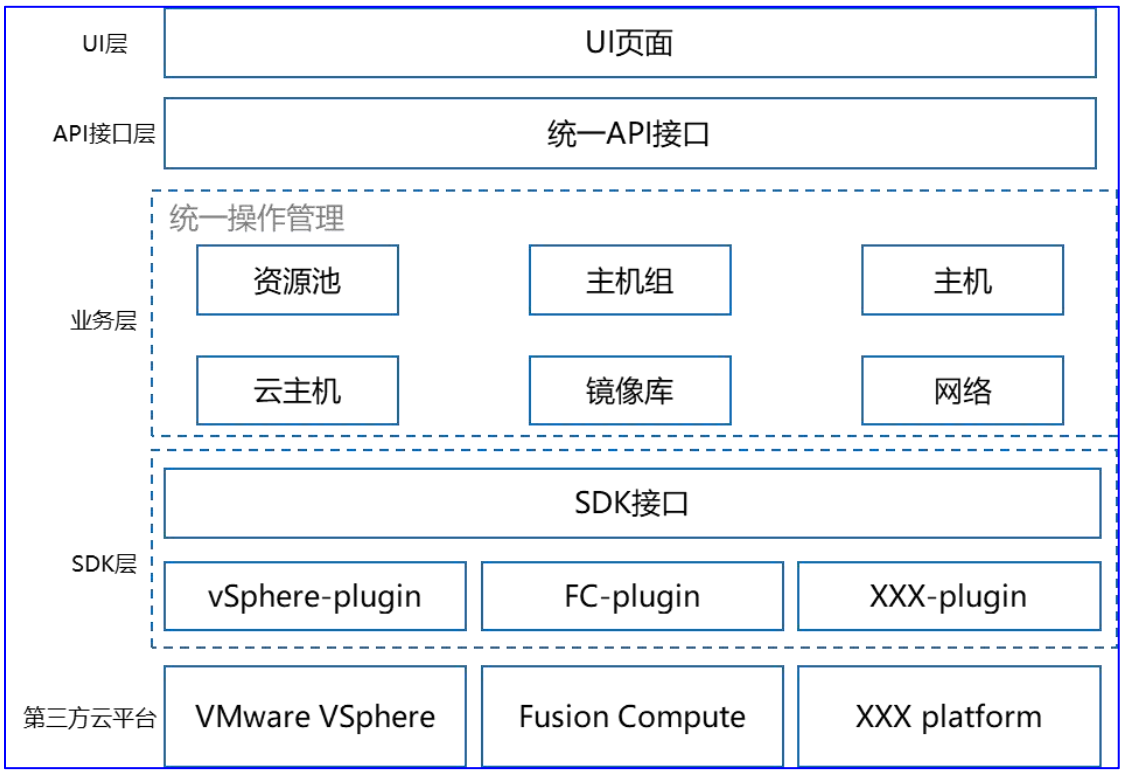
4.11.1.2 技术原理

多云管理整体架构采用插件化的方式，定义统一的南向接口来实现统一的界面功能，不同的虚拟化产品以插件的形式接入，实现统一南向接口的功能。此架构灵活性、拓展性很高。

基于分层设计思想，解耦形成 **UI** 层、**API** 接口层、业务层、**SDK** 层，降低系统设计复杂度；**API** 层对外提供统一接口，兼容不同云厂商，业务层对资源池、主机组、主机、云主机等资源统一接入、统一管理、统一操作，**SDK** 层向下适配底层各云厂商的接口差异

基于可扩展插件模式的标准 **SDK** 层，在 **SDK** 层统一封装标准接口，在 **plugin** 插件中具体实现各厂商的功能差异，能够以新增 **plugin** 的方式实现对其他类型厂商的快速简单的水平扩展支持；

提供标准的统一 **API** 接口作为北向接口交互，将多云管理作为整体对外提供服务，方便后续基于外部虚拟机部署服务、统一资源池模型的快速对接



4.11.1.3 约束

无

4.12 跨云迁移

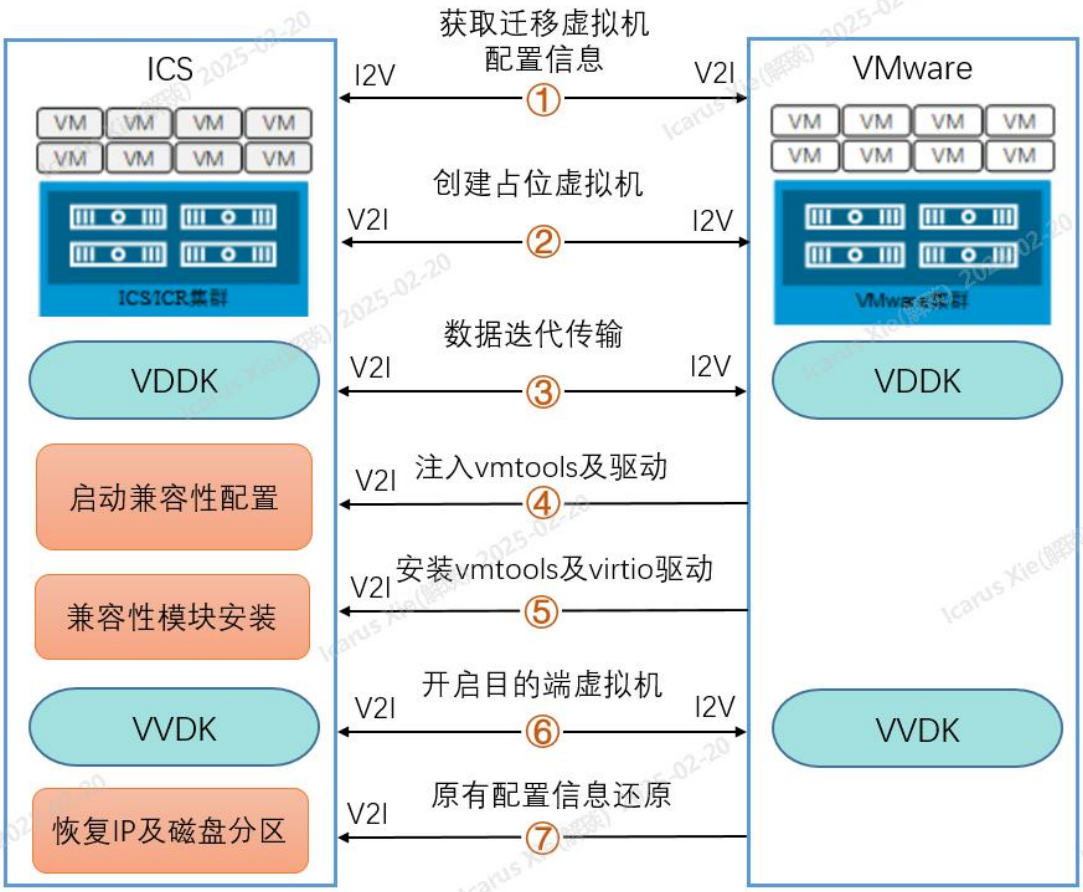
4.12.1 功能描述

多厂商、异构云平台的统一管控是多云管理的基础，云平台可实现云主机在 **VMware** 和 **InCloud Sphere** 资源池之间的在线相互迁移，用户无需担忧迁移过程业务中断，亦无需购买额外的第三方迁移服务；且从 **VMware** 至 **InCloud Rail** 资源池支持批量迁移，无需用户重复执行迁移操作，还可以在迁移前根据当前硬件环境，通过修改迁移过程中每次迁移的数据块大小，优化不同硬件环境下的迁移速率，并且在数据迁移完毕后，自动修复虚拟机 **IP** 并注入虚拟磁盘驱动以及 **ICS vmtools**，实现无人值守的迁移效果。

InCloud Sphere 资源池支持云主机导出 **vmdk** 格式，导出的 **vmdk** 文件可以直接导入到 **VMware** 平台，方便用户将云主机从 **InCloud Sphere** 迁移到 **VMware** 平台，同时也支持导入 **vmdk** 格式的云主机。

4.12.2 技术原理

下图为在线迁移磁盘数据原理图，其中 **V2I** 表示虚拟机从 **VMware** 平台迁移至 **InCloud Sphere** 平台；**I2V** 表示虚拟机从 **InCloud Sphere** 平台迁移至 **VMware** 平台。



4.12.2.1 VMware 云主机在线迁移至 InCloud Sphere

在 **InCloud Sphere** 上创建一个新空白磁盘，将 **VMware** 虚拟机通过快照复制的方式，不断把虚拟机数据同步到 **InCloud Sphere** 平台，当脏数据(脏数据是

指 VMware 平台虚拟机运行过程中不断修改、但是还未复制到 InCloud Sphere 平台虚拟机的磁盘数据)足够小时，暂停 VMware 虚拟机，并将最后一次增量数据同步到 InCloud Sphere 平台，最后在 VMware 平台关闭虚拟机，在 InCloud Sphere 平台开启虚拟机；若存在多任务并发迁移，则根据虚拟机 ID 批量进行上述操作，迁移过程中按照目的虚拟机 ID 不同来区分不同的迁移任务及迁移数据量的评估，多个迁移任务共享管理网带宽且互不干扰。迁移数据完成后，InCloud Sphere 平台会自动将磁盘驱动、vmtools 等工具注入虚拟机内部，待虚拟机启动后自动修复由于网卡驱动变化导致 IP 丢失的问题，并支持系统盘使用 virtio 接口，以实现性能最优化。

VMware 虚拟机在线迁移至 InCloud Sphere 可划分为以下步骤：

(1) InCloud Sphere 获取待迁移虚拟机的配置信息；

(2) 将运行于 VMware 平台上的虚拟机的虚拟磁盘复制到 InCloud Sphere 存储池的空白磁盘；

(3) VMware 平台虚拟机持续创建磁盘快照，并调用 VMware 接口将快照数据迁移到 InCloud Sphere 存储池，迁移过程中 InCloud Sphere 记录 VMware 平台虚拟机由于数据写入产生的磁盘脏数据；

(4) 当脏数据小于指定阈值时，暂停 VMware 平台虚拟机，剩余脏数据全部复制到 InCloud Sphere 环境；

(5) 关闭 VMware 平台虚拟机；

(6) InCloud Sphere 平台执行兼容性配置，将 vmtools 以及磁盘驱动自动注入虚拟机内部，完成对网卡和磁盘的驱动支持；

(7) 在 InCloud Sphere 平台开启虚拟机，完成用户业务切换。

4.12.2.2 InCloud Sphere 虚拟机在线迁移至 VMware

在 VMware 上创建一个占位虚拟机，将 InCloud Sphere 虚拟机通过快照复制的方式基于 VMware 的虚拟机磁盘开发库 vddk 和 InCloud Sphere 的虚拟机磁盘开发库 vvdk，把虚拟机数据不断同步到 VMware 平台，当脏数据足够小时，暂停 InCloud Sphere 虚拟机，并将最后一次增量数据同步到 VMware 平台，最后在 InCloud Sphere 平台关闭虚拟机，在 VMware 平台开启虚拟机。

InCloud Sphere 虚拟机在线迁移至 VMware 可划分为以下步骤：

(1) InCloud Sphere 获取待迁移虚拟机的配置信息，在 VMware 创建同样配置的虚拟机；

(2) 将运行于 InCloud Sphere 平台上的虚拟机的虚拟磁盘复制到 VMware 占位虚拟机对应的空白磁盘；

(3) InCloud Sphere 平台虚拟机持续创建磁盘快照，并调用 VMware 的 vddk 将快照数据迁移到 VMware 存储池，迁移过程中 InCloud Sphere 记录迁移虚拟机由于数据写入产生的磁盘脏数据；

(4) 当脏数据小于指定阈值时，暂停 InCloud Sphere 平台虚拟机，剩余脏数据全部复制到 VMware 环境；

(5) 关闭 InCloud Sphere 平台虚拟机；

(6) 在 VMware 平台开启虚拟机，完成用户业务切换。

4.12.2.3 ICS VM 导出 vmdk

InCloud Sphere 虚拟机导出 vmdk 格式技术原理：使用 qemu-img convert 命令生成 vmdk 格式磁盘文件，遵循 ovf（开发虚拟化格式）要求生成配置文件。

InCloud Sphere 虚拟机导出 vmdk 格式可划分为以下步骤：

(1) InCloud Sphere 获取虚拟机导出格式、导出位置，支持镜像库、远程服务器以及 Windows 系统共享目录；

(2) 按照导出格式执行命令，生成 vmdk 格式的磁盘文件；

(3) 根据 InCloud Sphere 虚拟机的配置信息生成 ovf 配置文件；

(4) 将磁盘文件和配置文件打包，导出到指定位置；

(5) 删除导出过程中产生的临时文件。

4.12.2.4 ICS VM 导入 vmdk

InCloud Sphere 导入 vmdk 格式虚拟机的技术原理：使用 ics-storage volume copy 命令拷贝 vmdk 磁盘数据，在新虚拟机中完整还原 vmdk 数据。

基于 vmdk 创建虚拟机可以划分以下步骤：

(1) 首先是预检测资源创建情况，如存储池状态、容量大小是否满足，申请相关资源，以待导入 vmdk。

(2) 整理虚拟机相关配置， 预创建虚拟机，包括磁盘（此时无数据）。

(3) Copy 磁盘镜像卷，命令：ics-storage volume copy。

(4) 等待所有磁盘 copy 完毕，完成虚拟机的相关配置落库操作，虚拟机创建完成。

4.12.3 约束

无。

4.13 服务目录

4.13.1 功能描述

服务目录为用户提供一系列开箱即用的服务，包括云内生安全服务、云安全服务、高性能计算服务、人工智能服务、线下服务、数据库服务、服务镜像服务等。为方便管理员运维，支持在同一个界面中提供管理员视角和租户视角管理虚拟机资源，管理员视角支持虚拟机克隆、迁移、快照、磁盘多副本、转换磁盘格式、时钟同步、引导顺序、完整性校验、崩溃恢复策略、初始化 **hostname** 等运维功能；租户视角提供云主机开关机、重启、纳管、分配/回收、快照、克隆、虚拟机代理软件、监控等使用功能。

为租户提供自助服务界面，展示租户可使用的服务列表，支持服务的申请审批。

系统管理员可以在服务目录中进行服务的上线和下线，服务上线后，租户可申请服务并使用。

已具备的服务列表如下：

类别	名称	版本	服务提供商	功能简要说明	服务能力
云内生安全	云内生安全服务	V6		面向云主机、容器、裸金属等云内资产提供统一安全防护，包含资产清点、网络安全、应用安全、病毒防护、容器风控、安全审计等内生安全能力。	1、服务配置、发布、下架 2、服务开通 3、服务使用 4、操作日志记录 5、开启、关闭服务白名单，开启后可向白名单中添加和删除虚拟数据中心

类别	名称	版本	服务提供商	功能简要说明	服务能力
云安全	云数据库审计	V6.0	奇安信	云数据库审计基于 DPI+DFI 技术的数据库审计系统，对来自应用系统客户端和 DBA 对数据库的访问行为进行全面审计，不仅针对 SQL 语句，还可以对 FTP、TELNET、POP3、SMTP、NFS 等网络协议进行审计。审计系统能详细记录查询、删除、增加、修改等行为及操作结果，对危险操作还可以实时预警、及时阻止，从而达到保护数据库的良好效果。	1、服务发布、下架 2、实例管理：创建、编辑、删除、开机、关机 3、实例访问：免密单点登录 4、监控告警：查看节点监控信息 5、查看详情，包括摘要、规格列表、白名单、操作日志 6、开启、关闭服务白名单，开启后可向白名单中添加和删除虚拟数据中心 7、规格变更：CPU、内存、系统盘、数据盘（仅日志审计、数据库设计、堡垒机支持挂载数据盘 8、业务网卡变更：新增、删除、变更 IP 地址、变更网卡安全组、浮动 IP 绑定/解绑（仅使用互联模式 vpc 网络网元支持浮动 IP） 9、网元进行规格变更、业务网卡变更
	云日志审计	V6.328	奇安信	云日志审计收集处理来自云环境中的安全系统、操作系统、业务中间件等信息化组件所产生的日志信息，通过自身高性能运算解析，标准化、统一化存储管理，同时向用户提供搜索、查询、安全事件告警等功能，辅助用户获悉全网整体安全运行态势，满足各类合规需要。	

类别	名称	版本	服务提供商	功能简要说明	服务能力
	云漏洞扫描	v6.0.9	奇安信	云漏洞扫描通过智能主机服务发现、智能化爬虫和 SQL 注入状态检测等技术，可以通过智能遍历规则库和多种扫描选项组合的手段，深入准确的检测出系统和网站中存在的漏洞和弱点。并可以根据扫描结果，提供测试用例来辅助验证漏洞的准确性，同时提供整改方法和建议，帮助管理员修补漏洞，全面提升整体安全	需要关机 10、 云防火墙仅支持互联 vpc 网络模型，需要在 vpc 下创建安全服务引流服务将 vpc 内虚拟机流量引入云防火墙实例进行流量防护
	云防火墙	V6.1.13	奇安信	云防火墙在提供复杂环境组网、扫描攻击防护和虚拟系统等功能的基础上，深度集成了漏洞防护、间谍软件防护、失陷服务器检测等高级安全防护功能，快速构建基于威胁情报、态势感知、智能协同、安全可视化等新一代技术的安全防护解决方案。	

类别	名称	版本	服务提供商	功能简要说明	服务能力
	云 WAF	V8.1.6	奇安信	云 WAF 通过多种机制的分析检测，能够有效的阻断攻击，保证 Web 应用合法流量的正常传输，这对于保障业务系统的运行连续性和完整性有着极为重要的意义。同时，针对当前的热点问题，如 SQL 注入攻击、网页篡改、网页挂马等，能够按照安全事件发生的时序考虑问题，优化最佳安全–成本平衡点，有效降低安全风险。	
	云 VPN	V6.1.13	奇安信	云 VPN 可以通过 IPSEC VPN 或者 SSL VPN 的加密通道的方式实现企业数据中心、分支总部或移动办公终端与云平台之间建立安全可靠的连接。通过灵活便捷、按需定制的网络 连接，为企业提供专业的 NAT，Qos，DHCP，企业 VPN 等功能。	

类别	名称	版本	服务提供商	功能简要说明	服务能力
	云堡垒机	V3.4.42	奇安信	云堡垒机是连接云资源的安全管理工具，对运维过程事前规划、事中控制和事后审计。事前规划即用户身份管理，自定义用户访问资源的权限，托管设备密码并定期改密；事中控制即操作过程中可对特殊命令精准管控，防止越权操作，操作过程实时监控；事后审计即历史会话记录精确查询、操作指令精准定位、操作视频回放，及综合审计报表。	
	云主机安全	V8.0.3	奇安信	云主机安全集成了防恶意软件、webshell 扫描、应用程序控制、完整性监控、漏洞管理、安全基线、防暴力破解、防火墙、威胁情报、DDoS 防护、入侵防御、网络可视化及管理等多个安全模块，以确保应用及数据安全。除此之外，系统提供可视化的安全管理，通过对海量访问日志数据的分析，找到异常行为、定位未知的安全威胁，帮助用户快速制定应对的安全策略	

类别	名称	版本	服务提供商	功能简要说明	服务能力
高性能计算	HPC 高性能计算服务	V5.4		高性能计算管理平台是一个高度可扩展的集群管理和作业调度系统，可以为上万节点高性能计算集群提供稳定、高效、易用的 HPC 管理平台。	1、服务配置、发布、下架 2、服务开通 3、服务使用 4、操作日志记录
数据库	旗舰数据库	V3.14		基于云原生技术研发的多数据库应用平台，提供标准的、统一的、可扩展的数据库平台级服务。	1、服务配置、发布、下架 2、服务开通 3、服务使用 4、操作日志记录
人工智能	人工智能平台	V5.0		面向大模型全流程业务以及传统 AI 研发部署场景的一体化 AI 平台。能够整合计算资源、数据资源、研发部署环境和流程支持能力，实现计算资源统一分配调度、数据集中管理与加速、流程化模型训练部署，支持 AI 能力快速构建，具有敏捷高效的特性。	1、服务配置、发布、下架 2、服务开通 3、服务使用 4、操作日志记录

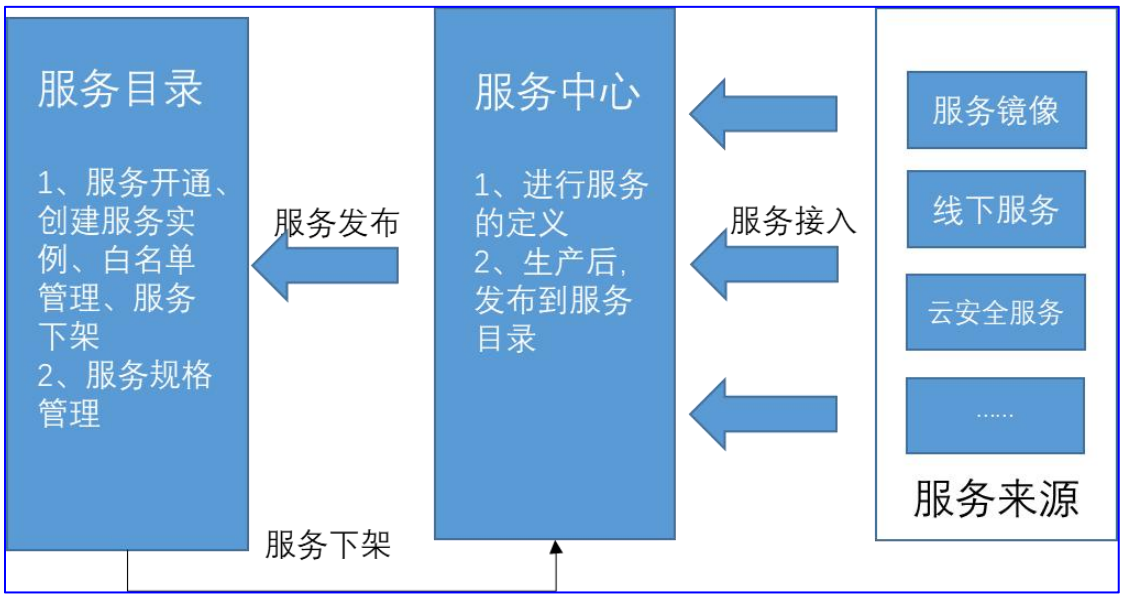
类别	名称	版本	服务提供商	功能简要说明	服务能力
租户 DNS 服务	NamedManager	V1.9.0		NamedManager 是一个基于 web 的 DNS 管理系统，用于添加、调整和删除 DNS 的 zones/record 数据，支持 Bind 作为后端 DNS 服务，并提供 IPV4/IPV6 管理功能。	1、服务发布、下架 2、实例管理：创建、编辑、删除、开机、关机 3、监控告警：查看节点监控信息 4、查看详情，包括摘要、规格列表、白名单、操作日志 5、开启、关闭服务白名单，开启后可向白名单中添加和删除虚拟数据中心 6、规格变更：CPU、内存、 7、业务网卡变更：新增、删除、变更 IP 地址、变更网卡安全组、浮动 IP 绑定/解绑（仅使用互联模式 vpc 网络网元支持浮动 IP）

4.13.2 技术原理

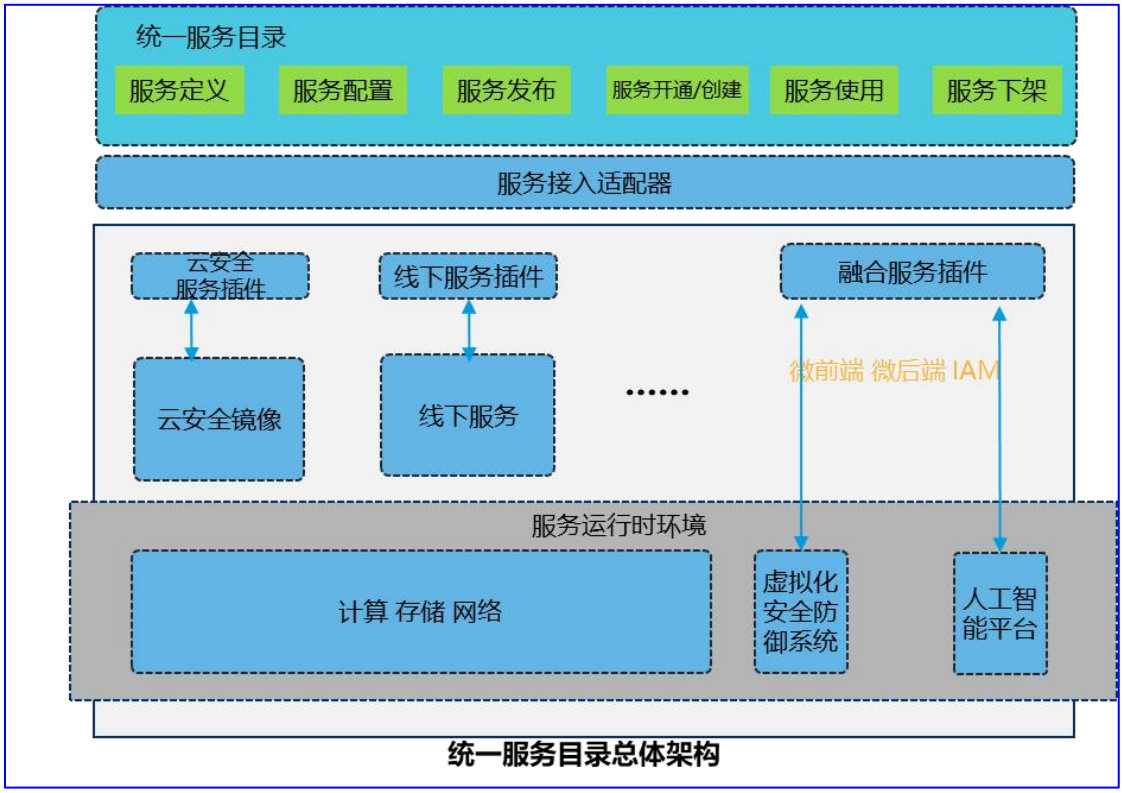
根据 OSB API 标准，服务目录由以下概念组成：

- 服务（Service）：一种对租户提供内容的概念包装。
- 服务实例（Service Instance）：承载服务的一组虚拟机或者其他的任何实体均可（比如，服务的一个租户或者一个业务流程）。
- 服务规格（Service Plan）：描述了服务实例的规格，例如服务实例的

配置大小等。



服务中心通过插件式设计，将各种来源的资源（服务镜像服务/云内生安全服务/线下服务等）统一生产出符合 **OSB** 规范定义的服务概念（服务/服务规格/服务实例），在服务目录中进行统一的呈现和使用。



4.13.3 约束

无

4.14 云内生安全服务

4.14.1 功能描述

浪潮云海云内生安全服务，提供基于无代理的架构的云内生安全功能，具备安全中心、资产管理、网络安全、应用安全、病毒防护、行为安全、容器风控、安全审计、配置管理以及系统审计等功能，实现了防御、检测、响应和预测等安全功能的一体化。为云服务器的系统、应用和数据提供了全面的安全保护。

云内生安全防护采用了低资源消耗、高效率的无代理技术架构，因此无需在虚拟机内部署的任何安全代理组件，也能高效地保护文件和网络安全。无代理架构技术显著提升了系统的安全和性能，还确保了用户业务系统的持续稳定性，为用户提供了可信赖的云内生虚拟化安全解决方案。

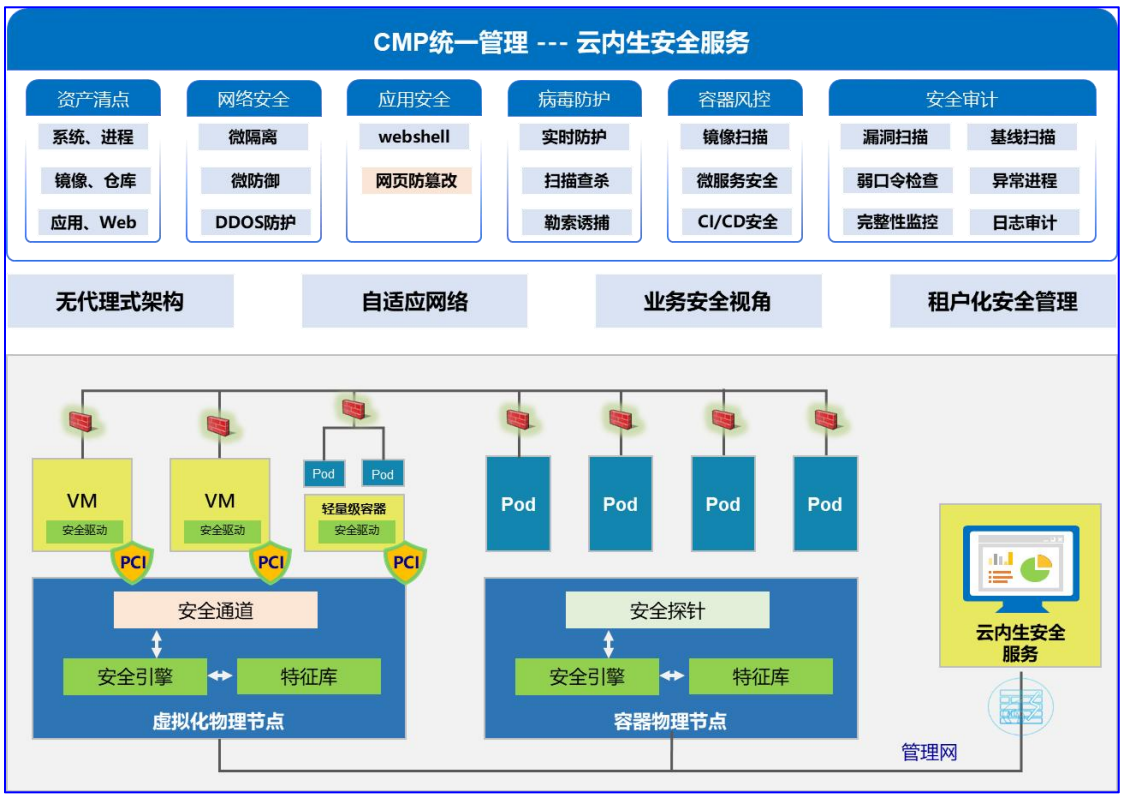
4.14.2 技术原理

安全管理中心：在每个数据中心内部部署管理中心模块，可以虚拟机、物理机、容器资源进行统一安全管理，配置每个虚拟机或物理终端的安全策略。管理中心接收安全组件上传的安全事件和网络流量日志，通过多维度、细粒度的大数据分析，并以可视化的形式展现给用户，从而帮助用户对已知威胁进行溯源，并对未知威胁进行预警。管理中心支持对无代理和有代理进行混合部署，同时管理无代理和有代理资产。

安全组件：在云环境中，安全组件安装在数据中心每个计算节点、物理服务

器上，接收管理中心配置的安全策略，对云主机进行文件、网络和系统的安全防护，并将安全事件及行为日志上传到管理中心进行分析。

云内生安全总体架构如下：



无代理架构，不侵入业务

- 安全组件下沉部署，不侵入业务主机或容器；
- 支撑虚拟机、容器、轻量级容器资源；

无需业务网络

- 适应 **overlay** 网络，不需要改造 **VPC** 通信
- 统一管理网通信

多架构支持

- 支持 **X86、C86、ARM**

- 兼容 KOS、Windows、Centos、RedHat、Ubuntu、麒麟、统信等

面向云内提供全面基础防护

- 东西向网络安全
- 漏洞检测与防护
- 容器安全检测与防护
- 病毒防护、勒索防护

4.14.3 约束

无

4.15 高性能计算服务

4.15.1 功能描述

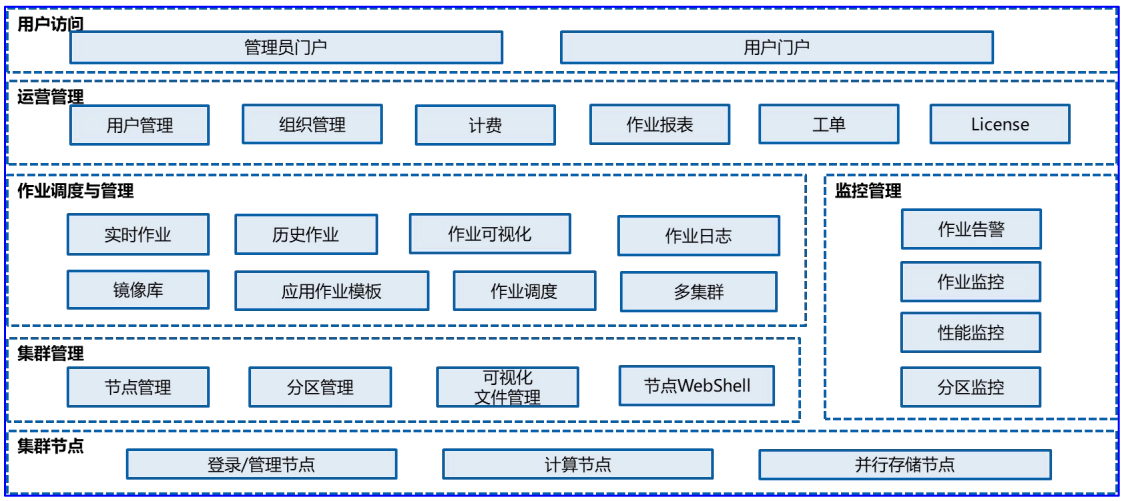
通过微前端的方式集成 **ClusterEngine**，整合高性能硬件资源和业务资源，实现计算资源统一分配和调度、全面的集群监控、作业的多样化提交和管理、计费管理的精准灵活，为 **HPC** 开发构建稳定、高效、易用的全栈式高性能计算管理平台。

云上 **HPC** 核心功能如下：

- 支持 **HPC** 高性能计算服务的配置、上架、下架
- 支持管理员或租户开通或申请开通 **HPC** 服务，开通时可以为租户设定配额，开通后，**ClusterEngine** 平台同步创建组织及配额，并进行映射
- 支持高性能计算服务的改配或退订
- 支持高性能计算服务相关的流程申请与审批

- 支持 **ClusterEngine** 以微前端的方式接入云平台，管理员或租户通过 **IAM** 映射的方式，登录到超算平台的微前端，使用超算平台提供的资源
- 支持高性能平台监控数据在云平台上统一展示
- 支持高性能计算平台资源池的管理
- **ClusterEngine** 接入云平台 **IAM**，实现用户的映射及统一。当云平台用户删除时，**CE** 平台进行相应的操作（校验或清理）。
- 提供面向高性能计算资源的作业管理、集群管理（分区管理、节点管理、文件管理、集群 **Shell**）、集群监控（资源、性能、作业、报表）、计费管理、数据统计等功能。

HPC 集群系统功能如下，集成云后，用户门户等功能通过微前端方式，深度集成到云平台上。

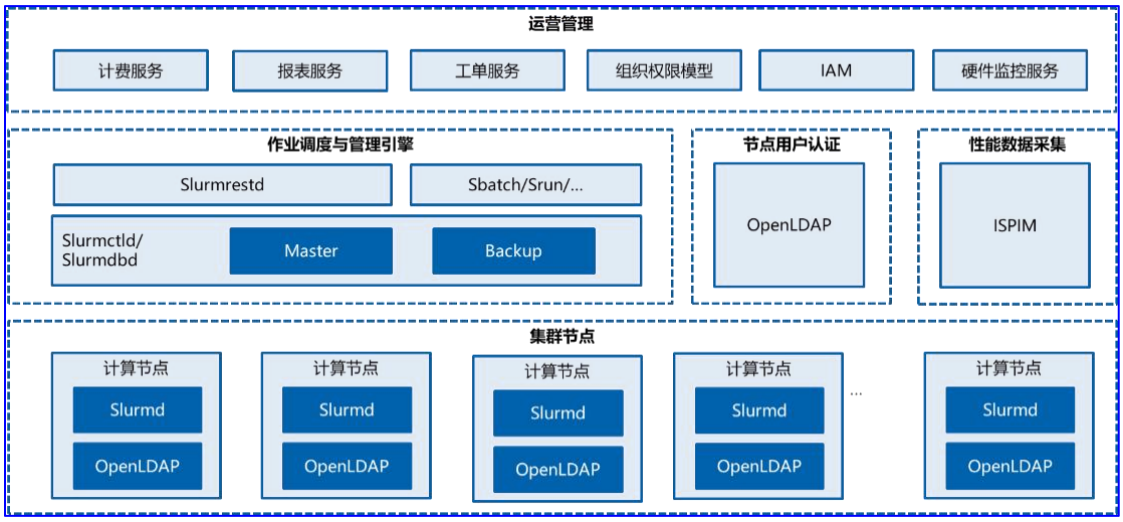


4.15.2 技术原理

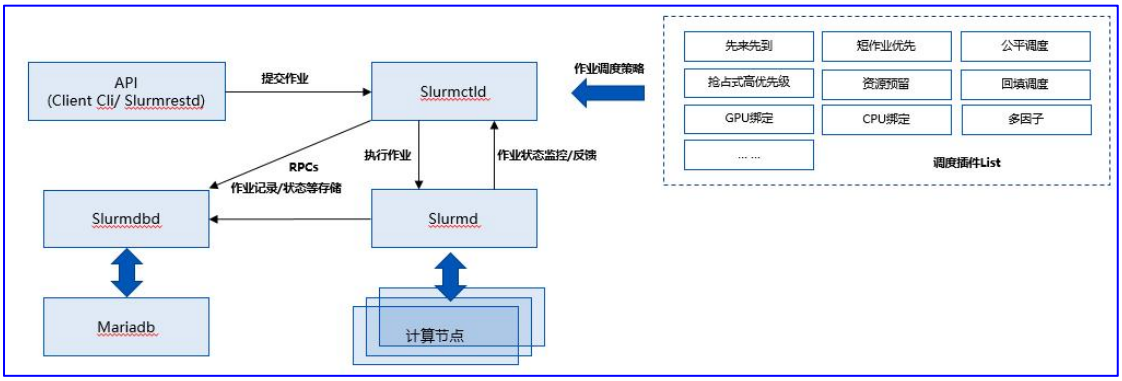
4.15.2.1 HPC 集群技术原理

InCloudOS 集成了 **Cluster Engine** 提供高性能计算服务。**Cluster Engine** 采用开源 **Slurm** 作为作业调度引擎，采用 **ISPIM** 作为集群硬件运维和性能信息采

集引擎，上层运营管理采用平台化的用户认证、组织模型、权限模型、计费模型以及报表模型，并通过为前端的方式接入云平台，与云平台统一门户、统一入口、统一资源组织模型、统一监控。



Slurm（**Simple Linux Utility for Resource Management**）是一个高度可扩展的，支持现代多线程多核调度的资源调度器，其高度可定义热插拔模块能够无缝对接于各类工作流，网络体系结构，排队策略以及调度策略。目前 **Slurm** 已广泛应用于全球超级计算机。

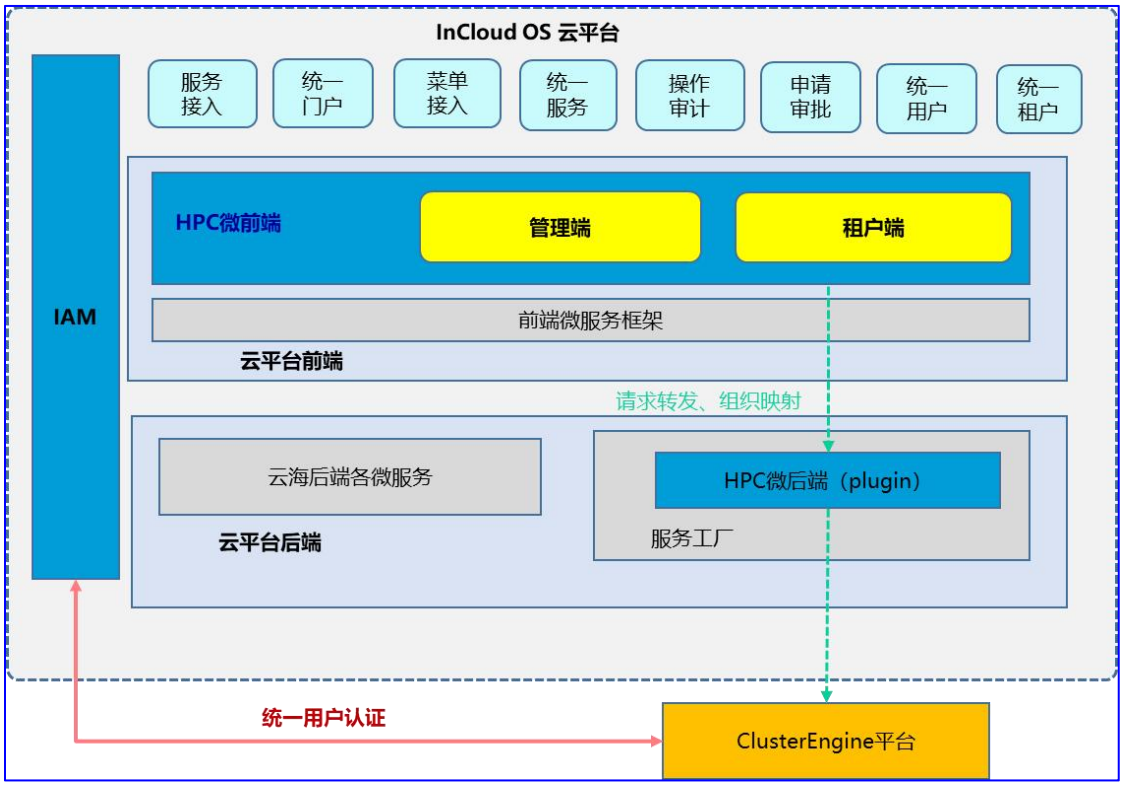


- **Slurmctld**：属于控制核心，负责定义和执行调度策略，协调 **Slurm** 活动，包括作业排队、监控节点状态以及将资源分配给作业 。
- **Slurmd**：位于计算节点，负责作业的启停和监控，并返回作业状态数据给 **Slurmctld** 。

- **Slurmdbd**：主要负责集群、用户和作业信息的存储，如集群状态信息、作业历史数据等。

4.15.2.2 HPC 云服务原理

HPC 平台对接采用微前端+微后端+**IAM** 总体架构，实现服务接入的深度融合。



微前端：由 **HPC** 系统的前端代码进行适当的裁剪，保留业务功能，并进行框架适配，接入云海 **OS** 前端微服务框架，提供统一入口。

微后端：微后端以插件形式接入云海 **OS** 后端服务工厂模块，实现租户体系打通、请求转发等功能。

IAM：基于 **OIDC** 标准协议实现数智平台与云平台的用户体系打通，采用用户映射的方式接入，支持与其原有用户体系并存。

云平台与高性能计算平台组织模型映射模型如下：

	云海 OS	HPC	对接方案
组织	VDC	组织	vdc 开通服务，接口交互创建 HPC 组织，设置配额
用户	超级管理员/域管理员	超级管理员	IAM 对接自动创建用户，完成用户和角色映射。
	VDC 管理员	组织管理员	
	VDC 用户	普通用户	
配额	VDC	组织	vdc 开通服务，调用接口设置/查询展示配额，HPC 平台完成配额校验

4.15.3 约束

当前版本支持通过微前端方式共享接入 ClusterEngine 集群服务，暂不支持弹性集群服务。

4.16 算力测算工具

4.16.1 功能描述

算力预测工具主要用于评估和量化主机硬件（尤其是 CPU、GPU 等处理器）的计算性能，帮助用户进行规划，具体包含如下几个方面：

- 算力预测：工具支持计算指定硬件配置下的算力值，可以比较不同硬件或不同配置下的主机性能的差异。
- 硬件规划：在购买新硬件或升级现有硬件时，算力测算工具可以帮助用

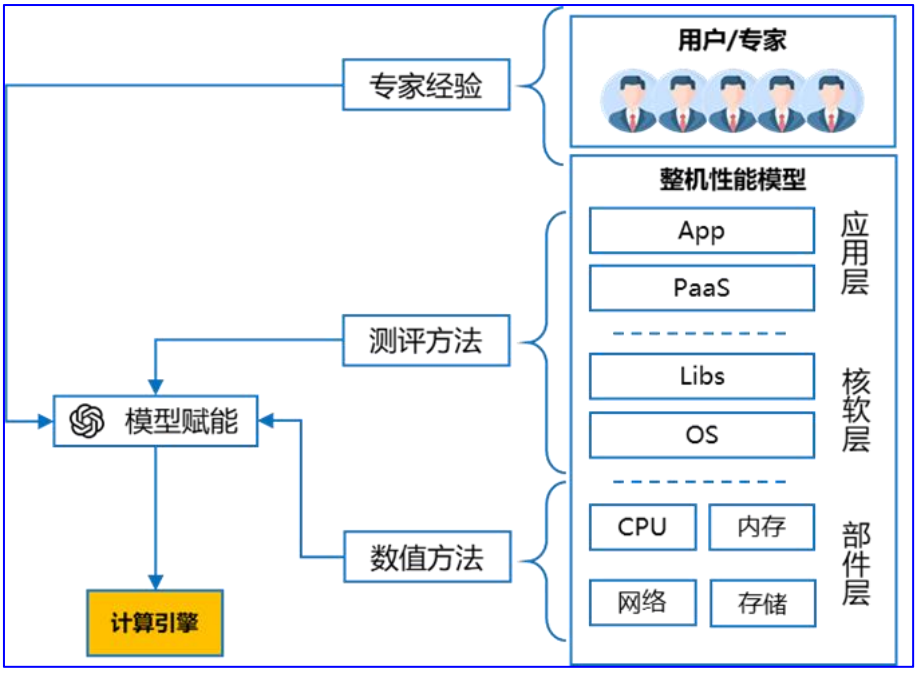
户进行更精细的规划，特别在架构方面。

- 资源优化：算力测算工具可以协助用户更有效地分配和优化资源使用，作为应用资源规划、应用迁移的参考依据。

4.16.2 技术原理

算力测算工具包含智能计算引擎和计算工作流两个主要的组件。

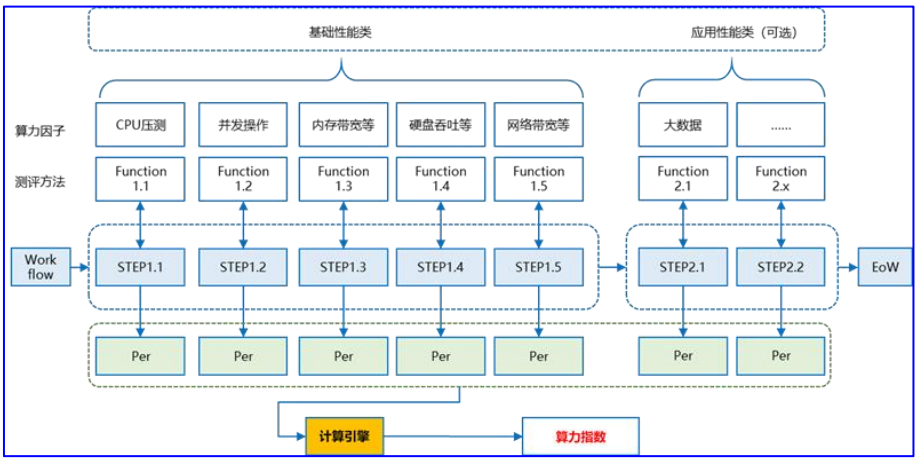
智能计算引擎主要架构如下：



其设计要点如下：

- 首先实现了对于整机性能的建模，通过将整机抽象为部件层、核软层以及应用层，与规格算力、有效算力及业务算力产生层次性对应
- 采用数值的方法对部件层代表的规格算力进行预测
- 同时采用基于测评的方法对有效算力及业务算力进行测算
- 结合专家经验以及智能模型，形成系统最核心的计算引擎

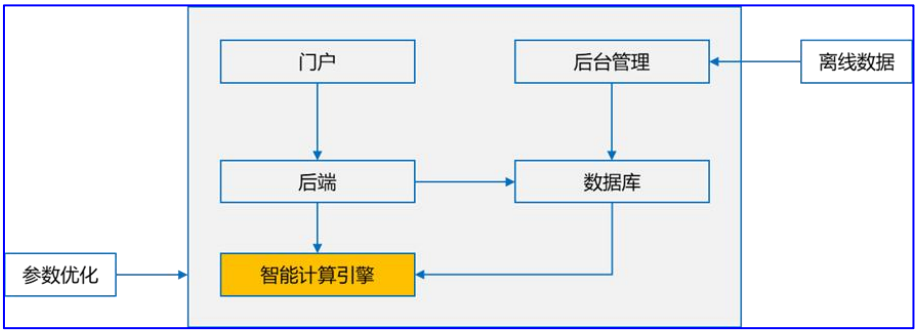
计算工具流整体架构如下：



算力测算的核心是工作流，将不同的算力因子与测评方法进行整合，并将结果输入到智能计算引擎，从而实现算力的计算。

工作流的最小工作单元是算力因子，不同算力因子代表不同的性能评估维度，通过与之相对应的测评方法形成分项的算力指数 **Per**，所有因子的算力指数计算完成后，输入预先训练好的智能计算引擎，最终得到算力指数。

系统整体架构如下：



系统采用前后端分离架构，前端采用基于 **Express** 框架，后端采用 **Spring Cloud** 微服务架构，并且采用了前端技术平台及云管技术平台加速了系统开发及迭代效率。

4.16.3 约束

无。

4.17 线下服务

4.17.1 功能描述

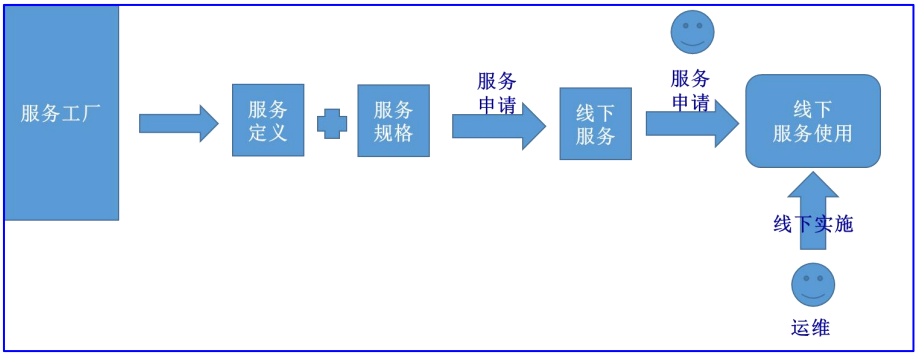
线下服务（**Offline Service**）是管理线下产品的服务。针对一些不能在系统线上自动编排发放的服务，提供在系统线上申请、审批，线下开通或执行的功能。

线下服务的实施，与系统没有任何关系，系统只是提供了一个统一申请、审批、结果反馈，管理线下资源的统一平台，具体实施在线下进行。线下服务相比线上服务具有资源类型丰富、资源参数可自定义、提供线下任务下发等特点。

4.17.2 技术原理

线下服务定义流程如下：

- 由服务工厂定义出线下产品。
- 定义线下产品的服务规格。
- 发布对应线下服务。
- 租户端申请对应线下服务。
- 服务实际实施由运维人员线下实施交付。



4.17.3 约束

4.18 运营服务

4.18.1 统一用户认证（IAM）

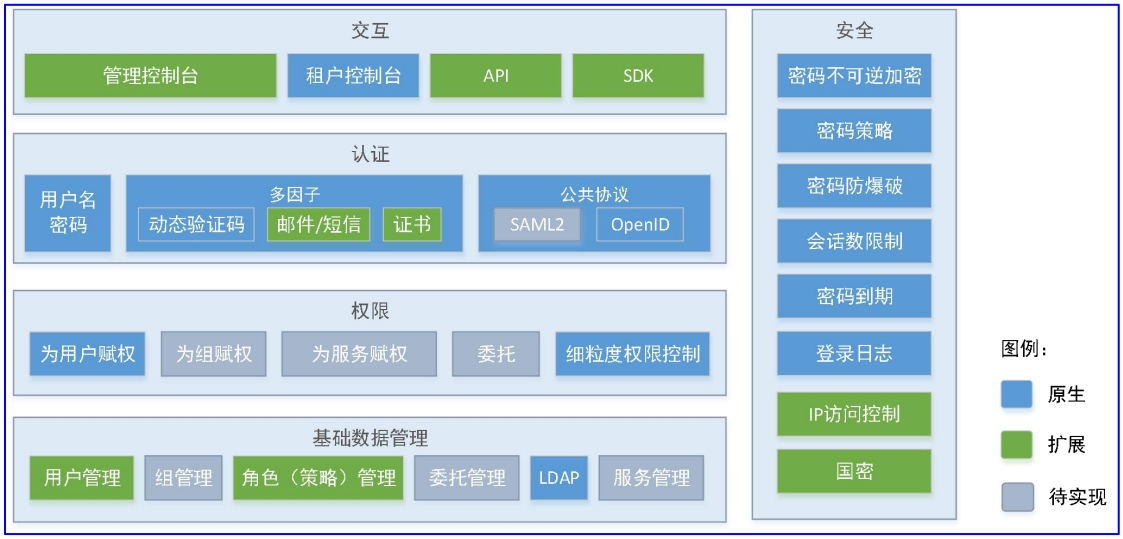
4.18.1.1 功能描述

平台中引入了 **IAM**（**Identify and Access Management**，身份识别与访问管理系统）体系，使用标准化的认证体系替换自研体系。

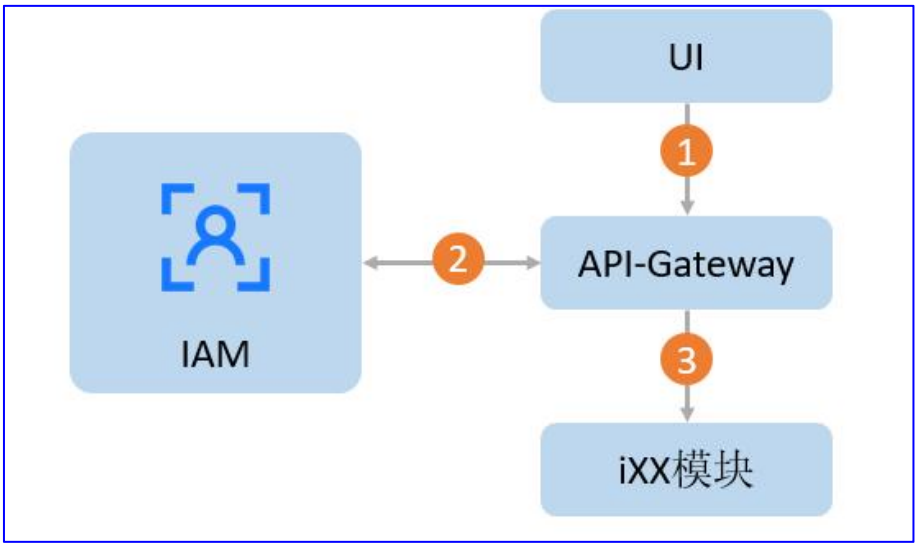
4.18.1.2 技术原理

平台中的统一用户认证体系基于开源的 **KeyCloak**。**KeyCloak** 依托于 **Redhat**，是 **RedHat SSO** 的上游项目。作为一套开源的、独立的认证授权服务器，**KeyCloak** 主要基于 **OAuth2.0** 协议和 **JWT**（**Json Web Token**）规范实现，提供了多种语言的集成，例如 **Louketo**、**Spring Boot** 插件、**NodeJS** 插件等，不同语言都可以通过插件适配 **KeyCloak**，从而提供了快速集成、快速适配上线的能力，便于其他系统对接接入。

基于 **KeyCloak** 提供的原生的功能进行了二次开发和扩展，如下



目前基于 **KeyCloak** 的 **IAM** 体系交互流程如下：



- **UI** 访问后台需要经过 **API** 网关。
- 访问 **API** 网关时，**API** 网关通过 **IAM** 验证认证信息是否正确，如果正确，则继续。
- 请求路由到 **iXX** 模块，完成具体的功能执行。

基于标准化的对接协议，支持与第三方 **IDP**（身份提供者）快速接入，例如支持与 **CAS** 协议的 **IDP** 进行对接，实现快速接入，避免非标准方式对接的开发工作。

支持用户组管理，支持将用户添加至用户组，并关联角色，用户组内的用户

均具有该角色权限。

4.18.1.3 约束

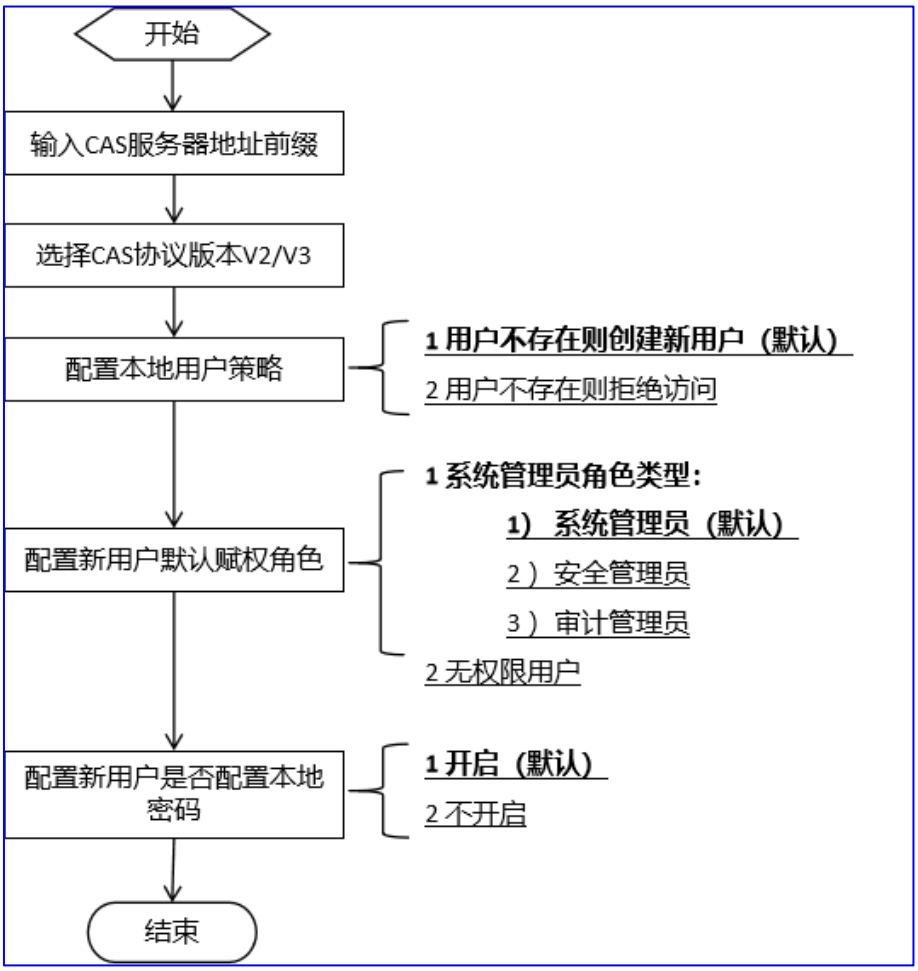
无。

4.18.2 IAM 对接 CAS IDP

4.18.2.1 功能描述

IAM 支持对接标准 **CAS** 协议类型的外部 **IDP**，实现单点登录，支持的版本为标准协议版本 **V2** 或 **V3**。

以下是系统管理员增加第三方 **IDP** 认证对接的操作流程：



- 系统管理员输入 **CAS** 服务器地址前缀，并选择协议版本（**V2/V3**）

- 配置本地用户策略，支持以下两种：
- 用户不存在则新建用户（默认）：如果本地 **IAM** 中不存在某账号信息，会先在 **IAM** 中自动创建一个同名的本地账号（如果已有同名账号，则直接使用）。

说明：如果选择该策略，则不再需要对接外部的 **LDAP/AD** 用户系统 。

- 用户不存在则拒绝访问：如果本地 **IAM** 中不存在某账号信息，则拒绝该用户登录系统。在实际对接场景中，一般都会先通过 **LDAP/AD** 对接，把外部用户系统中的用户账号同步到 **IAM**。
- 如果本地用户策略为上述第 **1**）种方式，这里需要配置新用户默认赋权角色，支持以下两种：
- 系统管理员角色类型（默认），如果是三权分立部署模式，可选择系统管理员（默认）、安全管理员和审计管理员角色。
- 无权限用户
- 配置本地用户密码策略是否开启(针对上述本地用户策略为不存在则新建场景，默认开启）：
- 如果不开启本地用户密码策略，不影响用户账号的登入，但是在某些需要用到本地密码的操作场景下，需要系统管理员通过重置密码的方式先为该账号设置下本地密码。
- 如果开启了本地用户密码策略，在用户账号通过 **CAS** 服务器认证后，系统会要求用户先设置一个本地密码存储到 **IAM**。
- 经过上述 **1**、**2**、**3**、**4** 步成功添加 **CAS** 服务器后，可对该 **CAS** 服务器设置是否自动跳转，默认：不自动跳转。

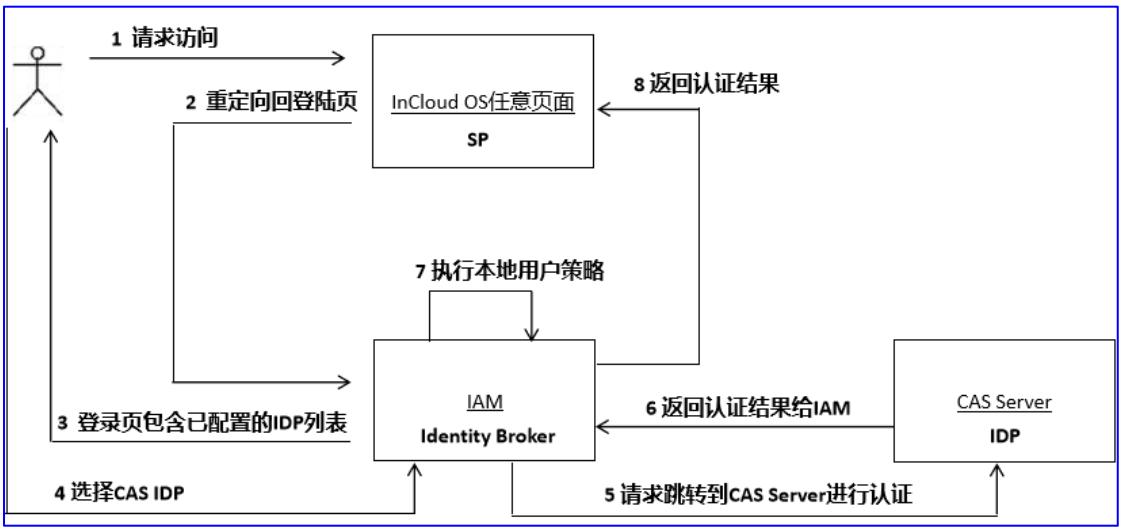
- 如果不设置自动跳转，在浏览器中输入 **InCloud OS** 的访问地址后，需要先选择 **CAS** 方式登录，才触发登录流程。
- 如果设置了自动跳转，在浏览器中输入了 **InCloud OS** 的访问地址并回车后，可自动跳转到 **CAS** 服务器进行登录流程。

说明：对接 **CAS IDP** 后，对现有的退出登录、会话机制的影响

- 退出登录后，不会级联退出 **CAS IDP**，适用于单点登录场景
- 不影响现有的会话机制，原有的多会话机制、会话超时机制等同样生效。

4.18.2.2 技术原理

IAM 的 **IDP** 机制允许将 **IAM** 集成到用户现有的身份验证系统中，从而实现单点登录（**SSO**）功能。通过配置 **IDP**，**IAM** 可以与这些系统进行交互，以验证用户的身份并获取用户信息。以下是 **IAM** 对接外部 **IDP** 后的请求认证过程：



- 用户访问系统任意页面
- 系统自动重定向回登陆页面
- 登录页面上包含已配置的 **CAS IDP** 列表（可以允许多个同时存在）
- 用户选择要进行认证的 **CAS IDP**

- IAM 自动跳转到 CAS Server
- CAS Server 认证通过后，返回认证结果给 IAM
- IAM 执行配置的本地用户策略
- IAM 返回认证结果给用户

4.18.2.3 约束

无。

4.18.3 双因子认证

4.18.3.1 功能描述

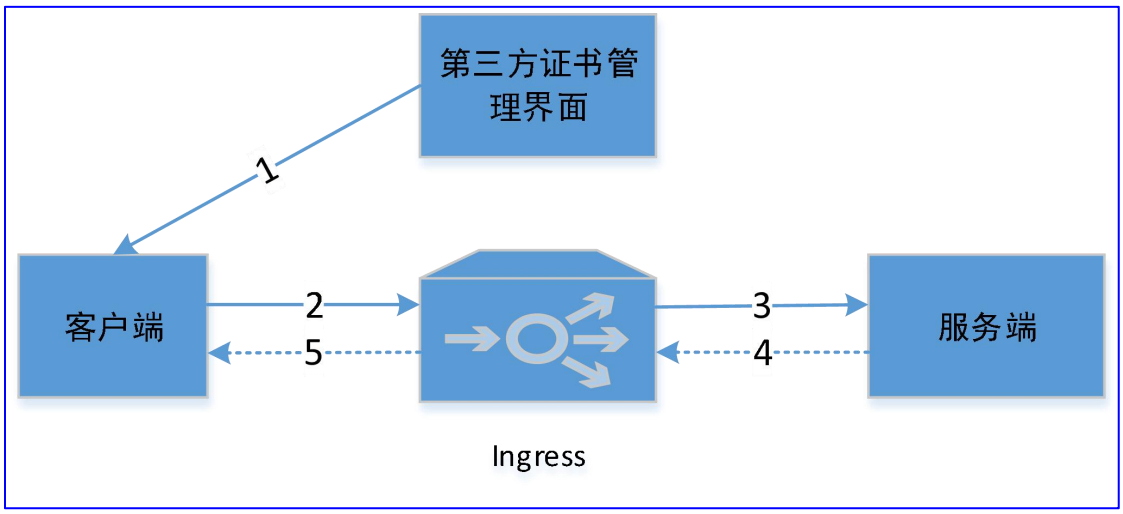
双因子认证服务，是一种基于证书安全认证技术，除了常规的用户名和密码的认证之后，增加了一种基于证书的身份认证方式，该方式的认证使得系统的安全性更高。当系统开启了双因子认证之后，即使用户的用户名密码被盗取了，没有证书仍然无法登录系统，这样将大大提高系统的安全性。

用户若要访问系统，应该首先获取证书。系统管理员作为系统的证书管理员，需要线下给用户颁发证书，该证书由用户自行妥善保管。颁发的证书形式包含两种：一种是电子版的数字证书，管理员线下给到用户后，用户需要自行保存，并安装到客户端；另一种是 Ukey 形式，管理员通过 Ukey 厂商提供的工具将证书写入到 Ukey 中，交由用户使用，用户妥善保管 Ukey。后续用户访问系统时，浏览器自动读取客户端的数字证书（不区分是否装在 Ukey 中）用于登录系统。

4.18.3.2 技术原理

双因子认证功能，主要包括证书管理和证书认证。证书管理主要是负责证书

的颁发、删除，证书认证是负值对证书的可信认证。



- 系统管理员线下颁发证书给对应的用户。
- 用户访问云平台时，浏览器自动携带证书访问 **Ingress**。
- **Ingress** 进行证书的双向认证，验证证书可信性。
- 证书认证成功，客户端与服务端建立安全通信，否则请求被拒绝。

4.18.3.3 约束

- 使用该功能的前提条件是当前系统开启了双因子认证功能。
- 当前系统开启了双因子认证服务之后，所有用户都需要安装证书之后才能登录。
- 用户使用 **UKey** 登录，当前仅支持海泰方圆国密 **UKey**，并且需要使用红莲花浏览器登录，同时需要在服务端进行配置。
-

4.18.4 多区域（Region）管理

4.18.4.1 功能描述

客户上云建云过程中，由于建设周期和地域的不同，往往会同时存在不同版本、不同地域的资源池。传统的建设方案为建设一套集中式的云管平台，纳管所有资源池，这种架构会产生若干问题：

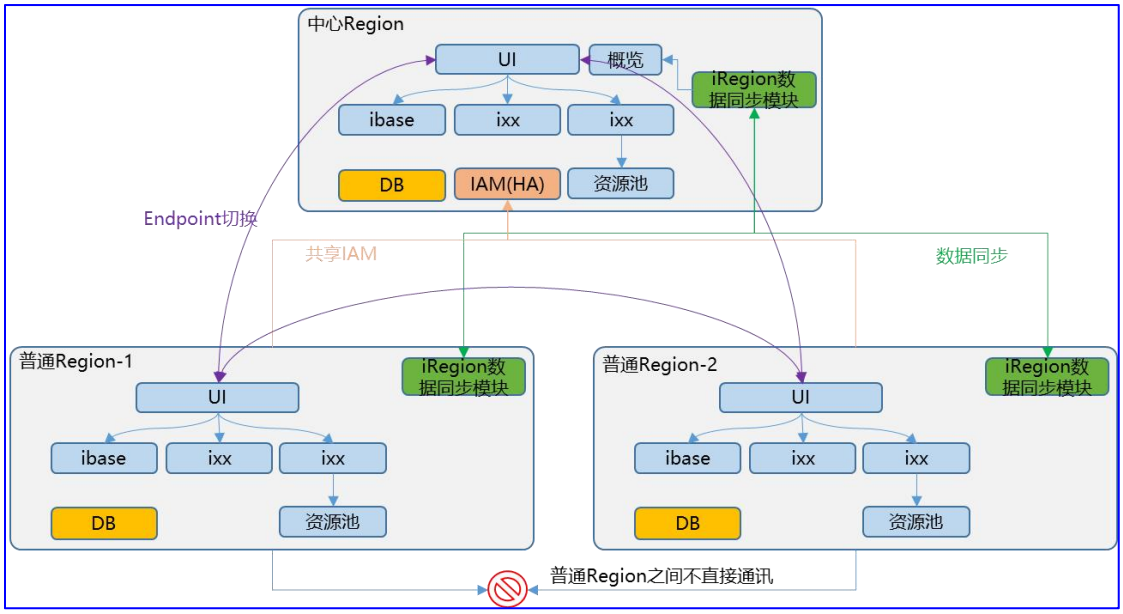
- 云管平台需要兼容多个版本的底层资源池，要求前后端进行复杂的逻辑判断，以精准匹配不同版本资源池的功能差异，随着兼容版本的增多，系统复杂度大幅提升 。
- 由于资源池地域分布广泛，采用集中式云管平台会不可避免的导致网络链路较长的跨地域访问，在距离远、网络不稳定的情况下，可能产生较高的访问延时甚至访问中断，降低使用体验。

针对这种场景，云海 **OS** 设计了基于中心 **IAM** 的多区域管理方案，同地域内保障前端界面/后端组件/资源池版本的匹配，基于中心 **IAM** 实现各地域的统一认证及切换管理，可以充分满足客户不同地域、分期建云及统一管理的需求。

在多区域管理模式下，中心区域的系统管理员可以创建跨区域的虚拟数据中心及子虚拟数据中心并分配用户，各本地区域的系统管理员可以在各自区域内对该虚拟数据中心绑定资源池、配置配额。

说明：如果 **InCloud OS V6.8.1** 作为普通区域接入时，系统不支持从中心区域创建跨区域的虚拟数据中心及子虚拟数据中心并分配用户。

4.18.4.2 技术原理



多区域之间使用基于 **IAM** 的共享用户体系，通过切换 **Portal** 的方式对不同区域的资源池进行具体的管理，并支持中心区域的系统管理员可以创建跨区域的虚拟数据中心及子虚拟数据中心并分配用户，满足客户多地域建设不同版本云计算资源池的诉求。

基于多区域管理方案，可实现多版本组成多 **Region**，以新版本作为中心 **Region**，通过切换 **Portal** 的方式，切换到老版本上对资源进行使用和管理。

方案采用了如下关键技术：

1，基于中心 **IAM** 的统一鉴权，各区域间实现统一用户体系，统一鉴权；

2，基于消息队列联邦技术，设计了消息的远距离投递通道，实现资源数据的汇总展示；

3，基于实时同步的本地 **IAM** 副本，解决网络中断情况下本地访问的问题。

4.18.4.3 约束

无。

4.18.5 费用与成本

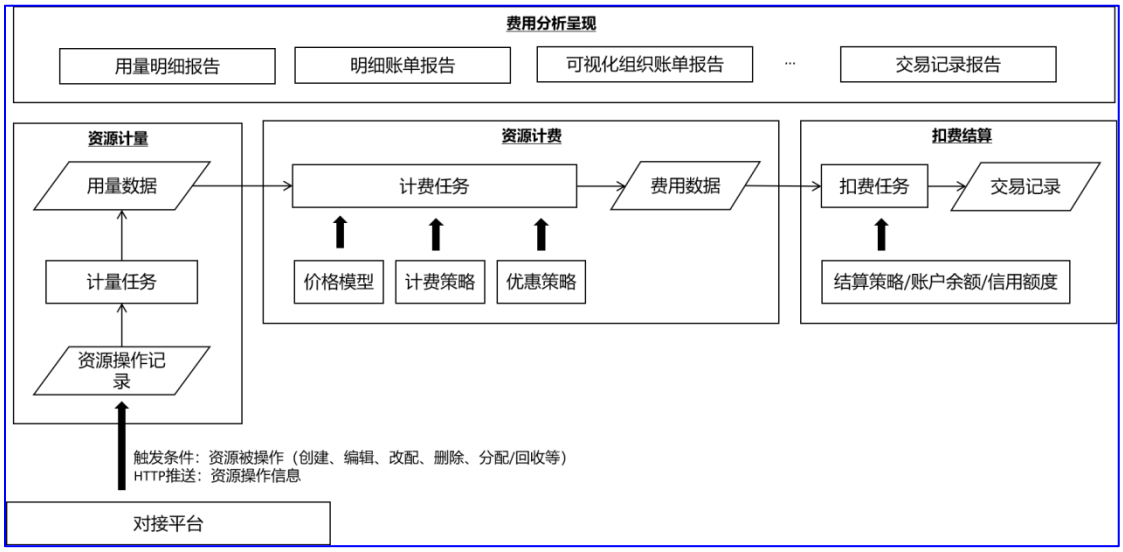
4.18.5.1 功能描述

费用与成本模块通过对私有云资源使用情况的精确计量和费用核算，帮助各组织了解其内部各部门、各业务单元对资源的需求和使用效率，为资源的合理分配和优化提供依据。

- 将资源的使用数据抽象为用量数据，再结合每种资源类型的特点，配合相应的计费方式和价格模型，进一步将资源用量数据提炼为更直观的能体现成本效应的费用数据
- 自动对费用数据进行对比分析，形成组织维度的可视化组织账单，为组织决策提供量化的参考
- 可查看各组织最近一周、月度、季度、年度及任何自定义时间范围内的总费用分析，包括了环同比变化、趋势情况、费用分布，并可导出为本地报告
- 可查看最近一周、月度、季度、年度及任何自定义时间范围内的详尽的资源费用账单、用量明细账单并可导出为本地报告
- 抽象账户概念，提供组织账户和用户账户两种账户类型，覆盖组织整体结算和用户个人结算两种业务场景
- 提供对账户充值、账户结算扣费、配置账户信用额度、查看账户交易明细及优惠策略等功能，满足组织自身可能需要提供付费服务交易的业务场景
- 提供配置化的价格模型，包括计费开关、计费方式以及计费项基础价格配置

4.18.5.2 技术原理

费用与成本模块目前由资源计量、资源计费、扣费结算以及费用分析呈现四部分组成。



- 对接平台推送要计费的资源操作信息到资源计量引擎
- 资源计量任务依据入库的资源操作信息生成资源用量数据
- 资源计费任务依据资源用量数据以及配置的价格模型、计费策略以及优惠策略形成资源费用数据
- 资源扣费引擎依据资源费用数据以及配置的结算策略等进行账户的扣费结算，形成结算交易记录
- 费用分析引擎依据资源用量数据、资源费用数据、结算交易数据形成一系列费用呈现报告，包括可视化的组织账单报告等

4.18.5.3 约束

无

4.18.6 多级业务流程自动化

4.18.6.1 功能描述

在实际生产场景中，很多客户基于企业或组织自身情况，并不都是由管理员统一创建资源并下发给用户使用，而是需要用户通过资源的申请审批流程，一层层的审批通过，最后才能收到按需申请的相关资源。

系统内配置了多个业务类型，例如申请创建云主机、申请创建虚拟数据中心、申请开通生态服务等，可以基于具体的业务创建新的流程，使业务与流程绑定在一起，用户进行业务操作时，根据流程的引擎进行发送流转，进行层层审批。用户提交业务订单后，相应的审批节点可以接受到邮件提醒，审批者可登录平台进行相关的业务审批。

支持根据用户现有工作流程自定义资源的申请审批流程，可动态添加审批流程用户节点，可以选择系统任意人员作为审批人，支持多人同时审批生效、任一审批生效的策略，可添加不低于 **10** 个审批节点，每个节点不低于 **10** 个审批人，审批人范围为系统内所有有权限的用户。

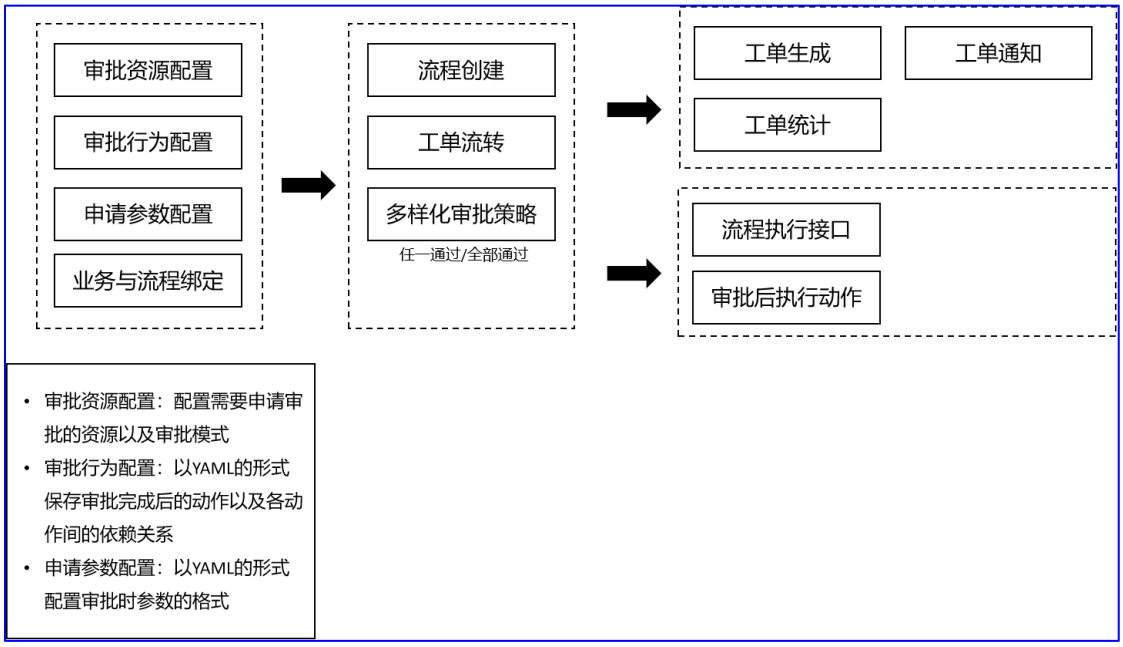
支持将资源申请与流程动态绑定，可支持多种不同资源共用同一流程。

4.18.6.2 技术原理

在流程管理模块可以创建流程，进而将流程制定给某个或多个业务类型，完成指定后，可以针对该类型的业务发起申请审批，进而形成工单。工单审批完成后，流程模块会调用各业务模块的代码进行资源创建。

云资源的申请需要通过管理员审批方可使用，平台支持多级审批流程，可以根据用户实际组织进行自定义；审批环节支持邮件通知相应责任人；支持管理员

进行通过、打回和关闭等基本审批操作。



4.18.6.3 约束

无。

4.18.7 报表

4.18.7.1 功能描述

云平台提供了各种类型的报表：资源报表、费用报表、业务报表、日志报表，具体有以下类型：

报表大类	报表子类	功能说明
资源报表	虚拟化资源池主机	实时报表、日报、周报、月报、季报、年报
	虚拟化资源池云主机	实时报表、日报、周报、月报、季报、年报
	裸金属服务器	实时报表

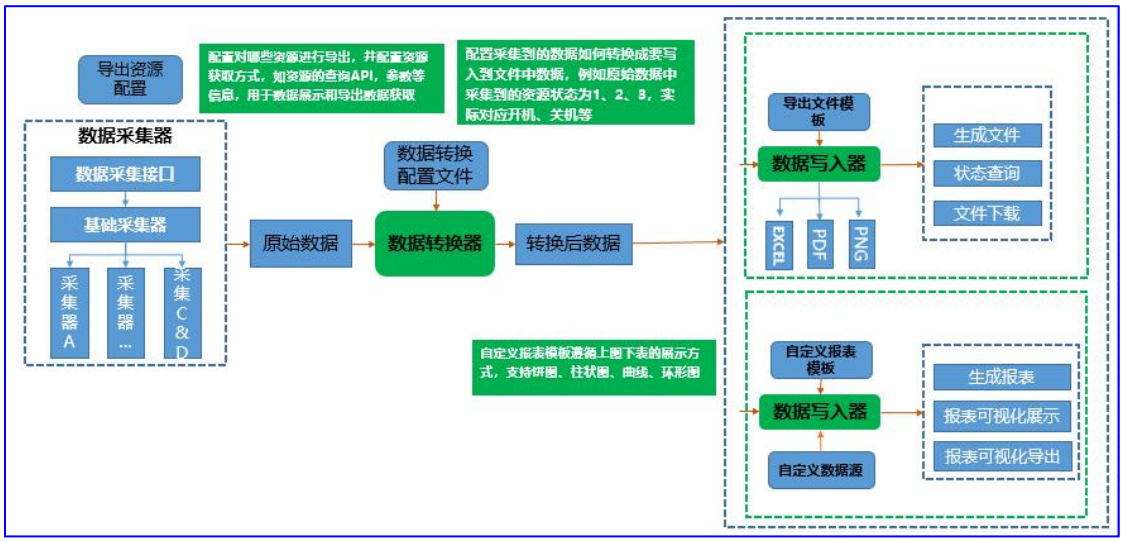
	容器资源池主机	实时报表
	容器资源池有状态负载	实时报表
	容器资源池无状态负载	实时报表
费用报表	组织账单	日报、周报、月报、季报、年报
	用户账单	日报、周报、月报、季报、年报
	资源账单	日报、周报、月报、季报、年报
	资源用量	日报、周报、月报、季报、年报
业务报表	–	组织工单统计、业务类型工单统计
日志报表	–	操作日志统计

系统可以支持将报表导出为本地报告存档，具体为：

- 导出方式：支持立即导出、定时导出及周期性导出
- 导出格式：支持 **Excel**、**PDF** 或 **PNG** 格式
- 导出范围：支持导出全部或导出当前页
- 支持将导出的报表文件发送给指定用户邮箱

4.18.7.2 技术原理

系统中配置对哪些资源进行数据导出，并对资源进行分类，当用户进行数据导出时，系统会根据系统中配置的资源采集方法进行数据获取，进而将数据展示到界面。



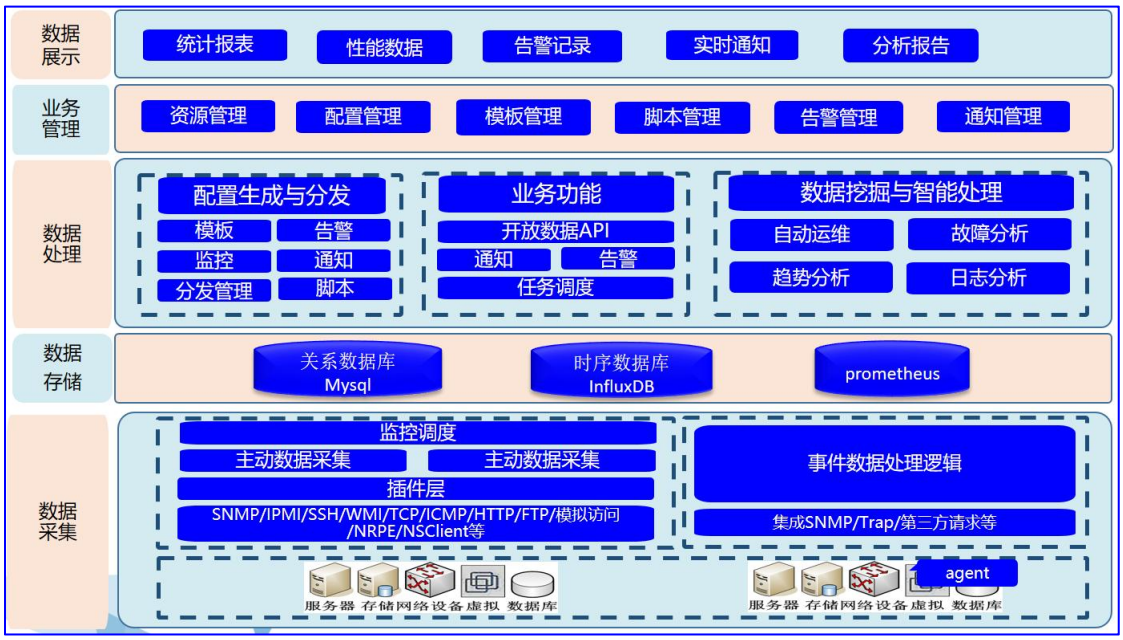
4.18.7.3 约束

无。

4.19 云监控

随着客户系统和业务规模的快速提升，对如何高效、快速及灵活的实现对系统和业务进行监控和告警的需求越来越迫切。一方面要保证服务的高可用、高性能、可扩展，另一面要定制灵活、用户友好，提高异常处理效率，降低损失。

监控告警系统提供大规模系统的监控、告警、通知和数据分析等服务，具有高可用、高性能、精细化、可扩展、智能化和用户友好等特点，帮助用户打造稳定可靠、定制灵活、准实时的监控告警服务。



监控告警系统提供精细化功能服务，包括数据展示、业务管理、数据处理、数据存储和数据采集等，及时准确地监控整个系统运行情况，提高系统异常处理的响应速度，提升运维效率，降低系统维护成本，有效保证用户业务的高可靠性。

- 数据展示包括大屏展示、性能数据展示、告警记录、实时通知和分析报表等。
- 业务管理包括资源管理、配置管理、模板管理、脚本管理、告警管理、通知管理，拥有良好的用户操作界面和业务自动化的处理方式。
- 数据处理包括支持采集、告警、通知配置生成和分发；告警、通知、数据接口和任务调度的业务功能；数据挖掘和智能处理。其中告警通知采用常规模式、智能模式通过设置告警阀值合理解决问题；智能处理中引入人工智能和机器学习算法，支持异常检测、定位、分析、预警和故障发现、止损、修复、归并等处理流程。
- 数据存储采用时序数据库实现数据快速存储，采用数据流分层处理的方式加快数据处理速度，采用分布式缓存、消息队列等技术解决大数据并

发传输。

- 数据采集包括带内、集中采集、代理采集等多种方式，拥有异构设备的数据采集能力，采用插件的方式，实现采集、告警、分析的可扩展性；支持批量上传采集端数据以缓解数据存储的压力。

4.19.1 大规模监控

4.19.1.1 功能描述

监控系统支持丰富的采集插件，可以采集到集群中关键数据，具体包括系统本身（宿主机、核心服务）、虚拟化资源池（宿主机、云主机、存储）、容器资源池（宿主机、容器、容器组、应用、核心服务、平台事件）、**PaaS** 服务、操作系统等，以及动态阈值检测、趋势预测等智能运维功能。具体如下：

- 操作系统层的监控告警，包括操作系统级别的资源利用情况，包括 **CPU**、内存和磁盘等的状态和性能等
- 云平台层的监控告警，包括平台组件、平台服务的资源消耗、平台服务状态、平台服务性能等。
- **PaaS** 层的监控告警，包括 **PaaS** 层中间件的资源消耗、中间件本身业务的监控告警等
- 应用程序、操作系统和其他服务的日志集中索引，实现实时的多维度检索和告警，快速发现问题和定位问题。
- 智能运维功能，提供动态阈值检测、趋势预测等智能运维功能

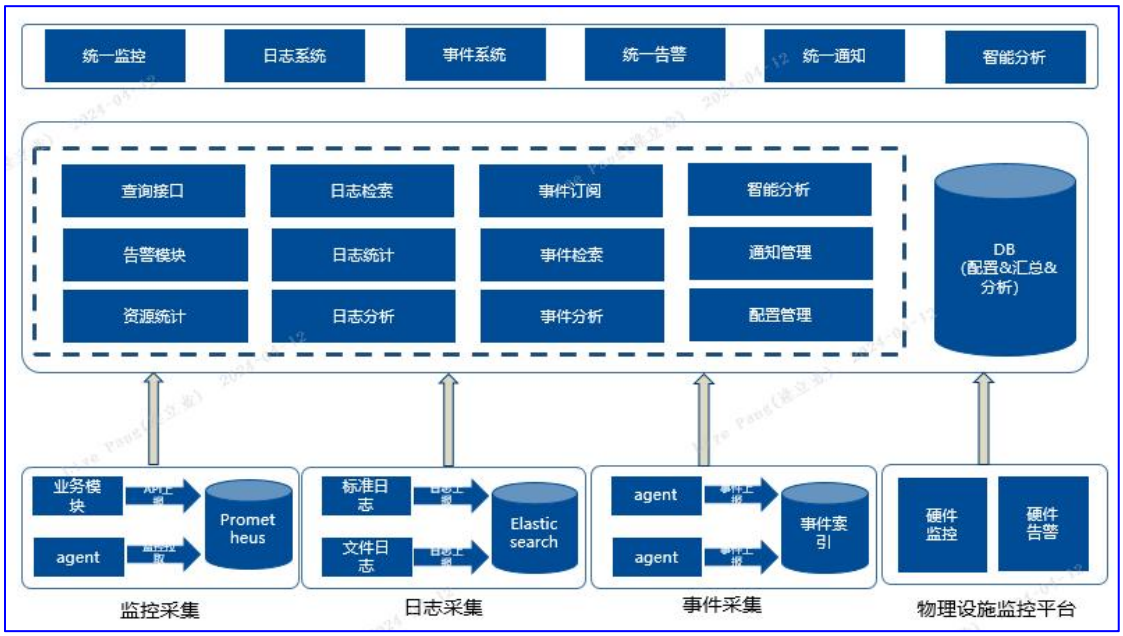
此外，集成物理设施监控平台支持面向行业数据中心的硬件智能运维管理，具备资源管理、故障监控、性能监控、能耗管理、报表统计、拓扑展示等功能，

可以实现服务器、机柜、刀箱、一体机、边缘设备、网络设备、安全设备、存储等设备的统一管理。

总之，平台通过提供全面、统一、多维度的管理监控子系统，可以管理监控数据中心的海量异构资源，包括服务器、存储、网络设备等硬件资源，各种操作系统、数据库、虚拟资源等软件资源；并且能够及时发现故障产生的告警，有效提高故障处理响应速度，降低运维成本。

4.19.1.2 技术原理

平台统一监控告警架构将所有业务系统中所涉及的网络资源、硬件资源、软件资源、云上资源、云上应用等纳入云监控，实现统一管理、统一规范、统一处理、统一展现、统一用户登录、统一权限控制。



监控采集：采用分布式时序数据库和资源粒度指标的汇聚，按需拉取，降低并发，统一执行告警任务实现告警通知

日志采集：采用分布式采集客户端，实时发现日志和采集日志，通过分词后的倒排索引从而实现海量日志的快速检索。

事件采集：通过事件发射器和事件消息队列，将 **Kubernetes** 事件集中索引，供定时任务的告警和通知。

物理设施监控：采用微服务显示到云平台页面的方式实现监控的详情展示，基于告警收集的方式实现告警通知的统一发送。

监控资源混合采集

平台监控集群支持上千种数据采集，系统设计目标支持百万级别的监控项和告警项，可以为用户实现精细化、可扩展的监控告警服务。监控数据采集可以支持集中采集、代理采集以及事件采集三种采集模式，可以实现异构设备的数据采集。

在集中采集模式下，监控集群会主动读取目标服务的监控数据并进行存储，可以实现监控采集的横向扩展，支持大规模的采集服务和采集数据种类；在代理采集模式下，可以由部署于目标服务器的代理服务程序采集数据并主动上报到监控集群。代理采集方式主要用于核心监控指标的采集；在事件处理采集模式下，监控集群会接收来自目标服务器主动上报的事件数据，主要是实时性要求非常高的事件数据。此类数据会立即处理，这种数据处理方式可以提升监控和告警的实时性。事件采集模式，主要是为客户提供了一种可自定义的监控数据采集器，用户只要按照一定的规则约束，定义并实现自己所需的数据采集器即可。比如，数据的采集内容、数据的上报格式、数据的上报地址等等。

此外，在数据采集过程中还支持用户自定义采集项。包括，采集周期、采集逻辑、数据处理方式等，从而支持更多种类的数据采集。

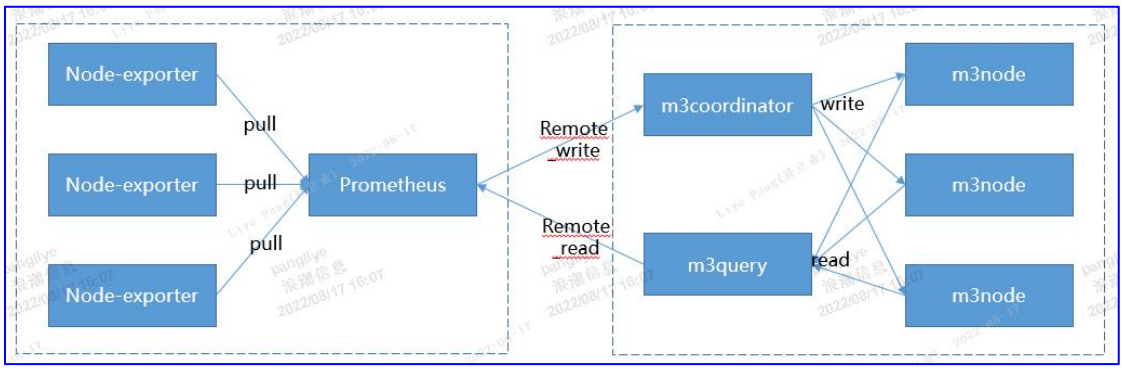
监控数据快速传输

平台监控集群采集到的数据分为性能数据和时间数据。其中，性能数据对海量数据处理能力要求非常高，时间数据对实时性要求非常高，因此采用如下技术实现对监控数据快速传输的支撑。

采用数据库集群实现数据与其他系统模块的交互，有利于提升系统数据吞吐和稳定性。定制数据传输策略。其中性能数据采用周期性批量上传方式，减少与数据频繁交互造成的性能损失。时间数据采用实时接收和处理方式，提高数据处理的实时性。采用数据流分层和消息队列方式提高数据处理速度和并发度，从而实现数据从产出到展现的传输速度。

监控数据的高效存储和处理

采用分布 **Promet**。



采集端优化：设置随机抖动时间来控制采集器采集前的休眠时间，从而降低网络瞬时流量；加入缓存机制，通过设置对采集数据量进行缓存后批量写入的方式，降低采集器的并发请求，从而降低存储端的并发操作，提高监控的性能。

存储优化：基于 **LSM** 的存储引擎，增大数据库并发量和吞吐量；加大快照容量，提高吞吐量；将内存索引优化为内存磁盘混合索引，降低存储内存消耗关闭非必须的日志，减少磁盘写入，提高数据存储速度。

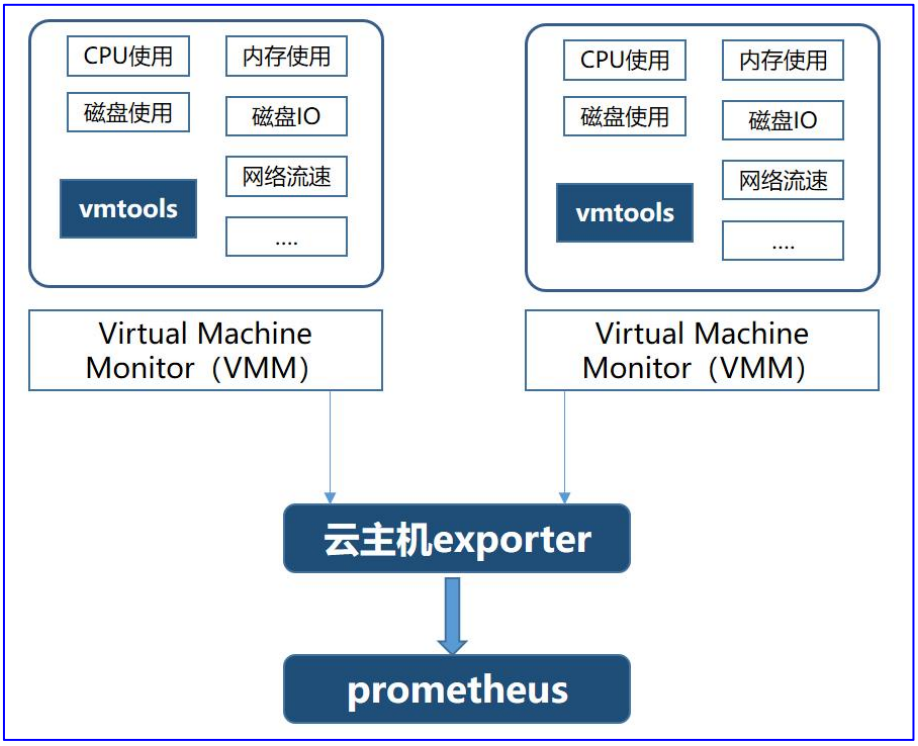
告警系统也采用集群化解决方案，将告警项均衡分配在多个告警服务实例中，

每个告警服务实例中的监控项互斥，确保每个告警项仅处理一次。此方案一方面实现了告警服务的横向扩展，另一方面实现了告警服务灾备处理，提高了云平台告警服务的高可靠性。

云主机监控数据采集

云平台云主机所在网络是业务网，监控系统部署在云平台宿主机上，所在网络属于管理网。服务器虚拟化的虚拟化软件层称为虚拟机监控器 **(Virtual Machine Monitor, VMM)**，也称为 **Hypervisor**，常见虚拟机监控器分为三类：

- 宿主模式：在 **VMM** 之下还有一层宿主操作系统，并不直接运作在裸机上。由于 **Guest OS** 对硬件的访问必须经过宿主操作系统，因而带来了额外的性能开销，但可充分利用宿主操作系统提供的设备驱动和底层服务来进行内存管理、进程调度和资源管理等。
- 独立监控器模式：**VMM** 直接运作在裸机上，管理和使用底层硬件资源，**Guest OS** 对硬件资源的访问都要通过 **VMM** 来完成，作为底层硬件的直接操作者，**VMM** 拥有硬件的驱动程序。
- 混合模式：为宿主模式和独立监控器模式的综合，**VMM** 直接运行在裸机上，但是驱动程序需要由 **Service OS** 提供，**Service OS** 为运行在 **VMM** 上的一台特殊虚拟机。



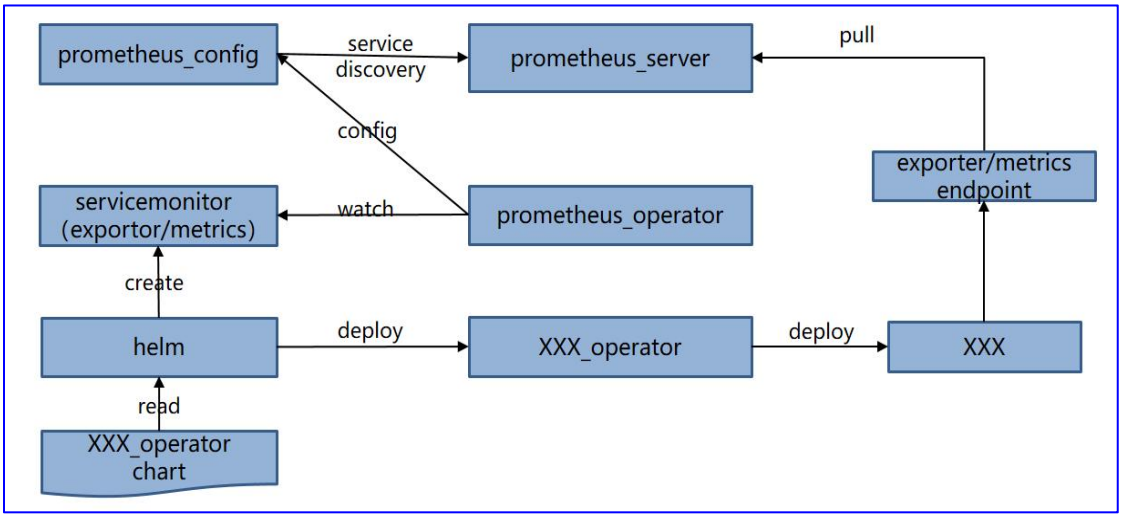
当 **VMM** 采集到所需的数据后，通过网络连接传输到云主机 **exporter** 组件。**Exporter** 在这里扮演了一个桥梁的角色，它负责接收来自客户端的数据，并以 **Prometheus** 可以理解的方式暴露这些数据。**Prometheus** 会定期（根据配置的时间间隔）主动去拉取 **exporter** 上暴露的数据。拉取到的数据会被 **Prometheus** 存储在其内置的时间序列数据库中，供后续的查询、分析和告警使用。

PAAS 监控

基于 **prometheus** 实现 **PaaS** 监控，不同中间件类型采用不同的采集器，具备不同的指标。各指标以 **key–value** 形式存入 **TSDB** 后，通过 **ipluto** 提供统一查询接口，包括列表， 实时指标，性能曲线， 告警设置， 告警列表， 通知设置，通知列表等。

按需部署采集器，若中间件提供 **metrics** 接口，则不需部署该中间件的 **exportor**，**ServiceMonitor** 中配置该 **metrics** 的 **endponit**，采用该方式的中间件为 **etcd**、**coreDNS**；若中间件不提供 **metrics** 接口，则需部署该种中间件的 **exporter**，

ServiceMonitor 中配置该 **exporter** 的 **endponit**，采用该方式的中间件为 **mariadb**、**redis**、**elasticsearch**、**kafka**、**rabbitmq**、**ES**。



4.19.1.3 约束

无

4.19.2 日志系统

4.19.2.1 功能描述

随着客户系统和业务规模的快速提升，对如何高效、快速及灵活的实现对系统和业务日志进行管理、检索和分析的需求越来越迫切。当系统发生故障或出现性能瓶颈时，需要通过分析各种日志来定位故障原因，找出导致系统性能瓶颈。

平台通过提供全面、多维度的日志系统，可以管理容器集群的海量日志数据，包括操作系统日志、核心服务日志（如 **Etcd** 集群）、容器组件日志、应用日志，在此基础上支持主机日志分析、组件日志分析，有效提高故障处理响应速度，节省运维成本。

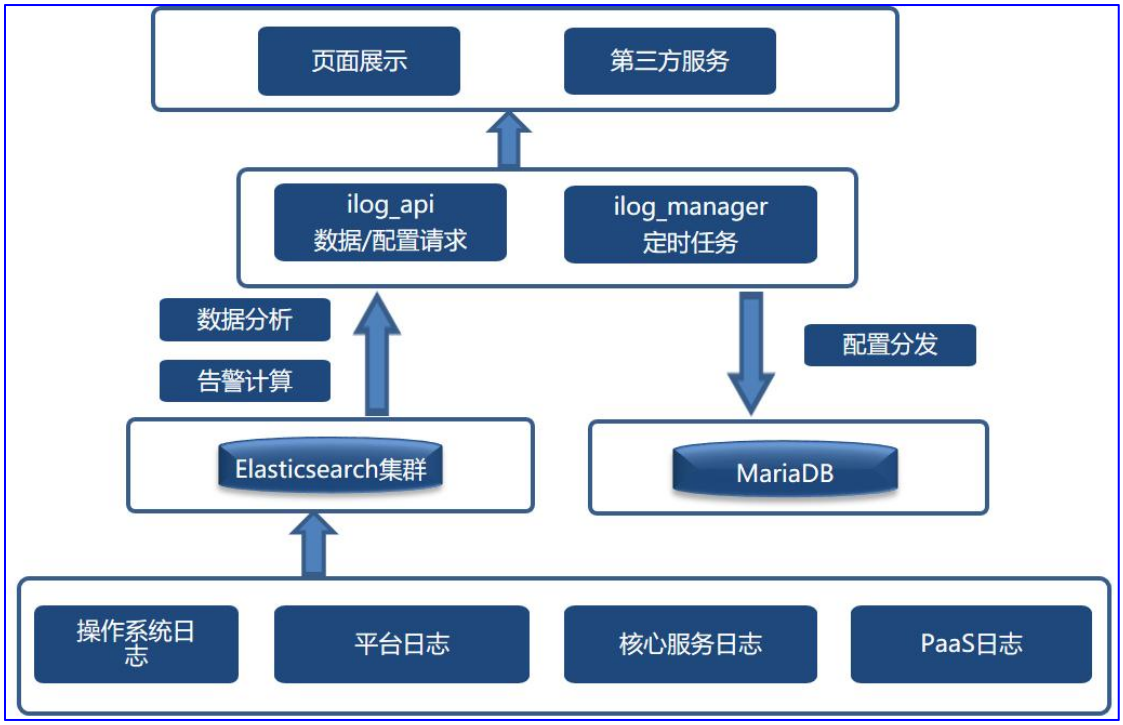
同时，日志系统也具备日志上报功能。在云平台的日志采集客户端通过配置

目标 **syslog** 服务器的主机和端口将格式化后的日志发送到指定位置。

4.19.2.2 技术原理

大规模日志索引

日志系统提供大规模系统的日志采集、日志存储、日志检索和日志分析等功能，具有高可用、高性能、可扩展、和用户友好等特点，帮助用户打造稳定可靠日志系统服务，提高问题定位效率。



日志系统包括日志检索、日志分析、日志数据接口服务、日志存储和日志采集等，提高系统异常处理的响应速度，提升运维效率，降低系统维护成本，可以使用户专注核心创新。

- 数据分析包括支持主机日志分析、组件日志分析、主要组件调用错误、调用错误等核心分析功能。
- 日志数据接口服务处理日志系统管理，为 **UI** 前端和包括监控系统在内的第三方系统提供数据查询服务。

- 日志存储采用企业级全文搜索引擎 **ElasticSearch** 集群，具有分布式、高扩展、高实时的特征，支持大数据量的检索和分析。
- 数据采集支持操作系统日志、核心服务日志（**Etcd** 等）、容器平台日志；支持定制化采集规则；支持自定义上报周期。

日志存储采用优化后的企业级全文搜索引擎 **ElasticSearch** 集群，支持大数据量的检索和分析，平台日志存储具有以下特性：

- 高可扩展性：作为大型分布式集群容易实现集群扩容，也可实现单机形式的轻量级搜索引擎。
- 功能丰富：提供了全文检索、同义词处理、相关度排名、复杂数据分析、海量数据的近实时处理等功能。
- 数据分片：一方面提高数据检索、分析效率，另一方面提升数据量存储能力。
- 高可用：提供副本机制，即使在某些服务器宕机后，集群仍能正常工作，提高数据可用性。
- 平台对日志数据多维度解析：日志数据支持时间维度（精确到毫秒）、主机维度（云物理机）、级别维度（**Info**、**Warning**、**Error** 等）、工作负载维度、容器组维度、容器维度等。
- 日志异常检测
- 随着云平台规模的不断变大，其日志数据量庞大，错误分散，当出现日志时，特别是影响系统运行的致命错误日志时，很难第一时间被发现，因此平台提供日志异常检测功能，用户可采用基于规则和 **AC** 自动机实现云平台错误日志的自动发现。

- 检测的主要流程如下：
- 日志指标数据采集：从各个数据源收集日志，并整理成合适的数据格式，将采集到的日志存储到 **ES** 集群。
- 设置时间窗和定时任务，定时处理 **ES** 中最新的日志数据。
- 专家规则检测：利用专家知识总结常见异常日志，并汇总为正则表达式集合，并行检测最新日志中的异常日志。
- **AC** 自动机异常词检测：利用专家知识总结常见异常词，并使用 **AC** 自动机进行检测和打分，识别包含异常词过多的日志。
- 告警生成及上下文数据存储：在使用正则检测和 **AC** 自动机异常词检测之后，生成对应日志数据的告警信息，并将其日志上下文存储于合适的数据库中。
- 日志数据量大，因此采用定时扫描 **elasticsearch** 索引的方式，实现异常串的检测，检测的主要流程及原理如下：
- **Elasticsearch** 首先创建倒排索引，倒排索引实现快速检索的基石。通过分词器将每个文档的内容域拆分成单独的词（称为词条或 **Term**），创建一个包含所有不重复词条的排序列表，然后列出每个词条出现在哪个文档。这种结构由文档中所有不重复词的列表构成，对于其中每个词都有一个文档列表与之关联。这种由属性值来确定记录的位置的结构就是倒排索引。
- 当对 **elasticsearch** 进行异常串检索时，对检索字段使用建立索引时相同的分析器进行分析，产生 **Token** 列表；根据查询语句的语法规则转换成一棵语法树，并查找符合语法树的文档；对匹配到的文档列表进行相

关性评分，评分策略一般使用 **TF/IDF**；根据评分结果进行排序后返回。

4.19.2.3 约束

无

4.19.3 容器事件系统

4.19.3.1 功能描述

随着云平台的规模不断扩大及第三方平台的接入，代理定时采集不是全部适用。平台具备基于事件的告警通知功能，应用场景包括：

1. 容器平台的业务模块，可以识别到很多关键事件，这些事件需要接入事件系统，进行统一告警和通知。

2. 容器平台本身的 **kubenetes** 集群会产生 **event**，从这些 **event** 中可以识别出一些关键事件，用来体现系统或者资源的运行状况，以便产生告警或者通知。同时，对所有的 **event** 提供检索功能。

4.19.3.2 技术原理

事件系统中存在事件订阅，事件订阅包含事件的一些基本参数，主要用来展示关键告警事件和发送通知的。每种事件订阅可以设置不同的通知策略（包括通知联系人、通知方式和通知策略）。只有平台上报的事件，没有业务模块上报的事件，会初始化一条总的事件订阅（不可删除，只能修改描述、提示信息和通知策略）。并且支持事件订阅的添加、删除和修改。

对于添加的事件订阅，可能有重合的地方，即一条事件可以匹配到多条事件订阅，这种情况下，是会按照匹配的多条事件订阅进行通知的。

事件的数量是相对较多的，当事件积累到一定数量时，可能对系统组件产生影响，需要设定数据清理策略。在运维端，资源池的参数设置中进行设置。

4.19.3.3 约束

无

4.19.4 趋势预测

4.19.4.1 功能描述

云平台往往需要根据业务的诉求对未来做出合理的规划，包括扩容的预测，缩容的预测等。以往基本采用人工预测的方式，但运维人员的能力可能存在差异，使得结果不尽如人意。云平台的性能数据多为平稳时间序列或可转换为平稳时间序列，可以进行预测。但另一方面单一的预测算法，对一种数据预测效果可能很好，对另一种可能很差。趋势预测包含对云平台的重点指标与对单一资源指标进行曲线拟合，实现了时间序列数据的预测，从而实现了云平台重点指标提前异常预警的效果。

4.19.4.2 技术原理

平台提供三种预测模型对指标数进行曲线拟合，包含三次指数平滑，鲁棒局部加权回归时间序列分解及差分整合移动平均自回归算法。

指数平滑算法是通过计算指数平滑值，配合一定的时间序列预测模型对现象

的未来进行预测。其原理是任一期的指数平滑值都是本期实际观察值与前一期指数平滑值的加权平均。其中三次指数平滑算法则是对数据进行了三次指数平滑运算，在二次指数平滑的基础上保留了季节性信息，适用于预测带有季节性的时间序列。

鲁棒局部加权回归时间序列分解算法是通过将时间序列分解为趋势项、周期项与残差项，并配合指数平滑算法对时间序列进行预测。其原理是使用局部多项式回归拟合的方式，提取时间序列的趋势项与周期项，并去除时间序列中的残差项。之后使用指数平滑算法对去除后残差后的时间序列进行预测，适用于带有季节性与趋势性的时间序列。

差分整合移动平均自回归算法是通过捕捉时间序列内部的线性关系进行建模，从而对时间序列进行预测。其模型建立在时间序列为平稳时间序列的基础上，因此时间序列的平稳性是建模前的重要前提，一般使用单位根检验的方式对时间序列的稳定性进行检验，若时间序列不满足平稳性，通过一些操作使序列变得平稳，然后使用差分整合移动平均自回归算法对序列进行建模预测，适用与具有时序平稳性的序列。

根据云平台实际的数据趋势，季节性及平稳性，对算法的实际应用进行了参数调优。通过参数调整，将定性分析和定量研究结合，修正趋势相关值，提高了数据预测的精度。其中调优项主要体现在以下两个方面：

- 采样数据取值，采集数据的相隔时间太短，则容易降低预测结果的精度，在实际调参中，发现整点的数据作为开始数据进行预测可以取到更好的效果。
- 预测模型参数，当周期参数固定时，模型参数不适合参数值时，也容易

导致预测结果的精度过低，通过对数据进行分析和对模型内部参数的自适应调整，从而提高预测精度。

4.19.4.3 约束

无

4.19.5 动态阈值异常检查

4.19.5.1 功能描述

传统云平台的性能监控都采用阈值告警，即运维人员根据经验设置告警阈值，当监控数据达到此阈值时，产生告警。但实际的应用过程中发现，阈值设置太依赖于经验，阈值太高，漏掉的告警过多，质量隐患难以发现；阈值太低，告警太多往往引发告警风暴，干扰运维人员的判断。另外，对于一些性能数据抖动的情况，阈值告警也无法检测，产生漏报。平台监控将动态阈值检测作为阈值告警的补充，很好的解决了上述问题。

4.19.5.2 技术原理

动态阈值检测是不需要设置阈值，通过算法模型对指标进行检测的方法。云平台性能数据为单维度时间序列数据，在实际应用中发现，若采用无监督模型检测，模型训练人工参与少，但往往准确率不高；若采用有监督模型检测，模型需

要大量训练数据，模型训练复杂、时间较长。为避免上述缺点，动态阈值检测采用无监督模型和有监督模型联合检测的方式，如下图所示。为了对数据进行更全面的检测，无监督模型检测采用基于 **3σ** 原则的统计算法、多项式回归算法和孤立森林算法实时构建模型；有监督模型检测默认提取统计、拟合、分类等特征，采用 **xgboost** 算法构建模型，同时支持手动训练 **LSTM** 模型。当有监督学习模型为收敛模型时，采用有监督模型检测结果，否则，采用无监督学习模型的检测结果。另外，在正常运行的云平台中，异常点占比较少，训练数据的选取和人为标注的工作量及其庞大，针对此也加入动态阈值模型检测异常群体，供人工标注。

动态阈值异常检测既发挥了无监督模型构建速度快、人工参与少和有监督模型特别是神经网络模型准确性高的优点，同时克服了无监督模型准确性不高和有监督模型训练复杂、时间长等缺点，取得了很好的效果。

4.19.5.3 约束

无

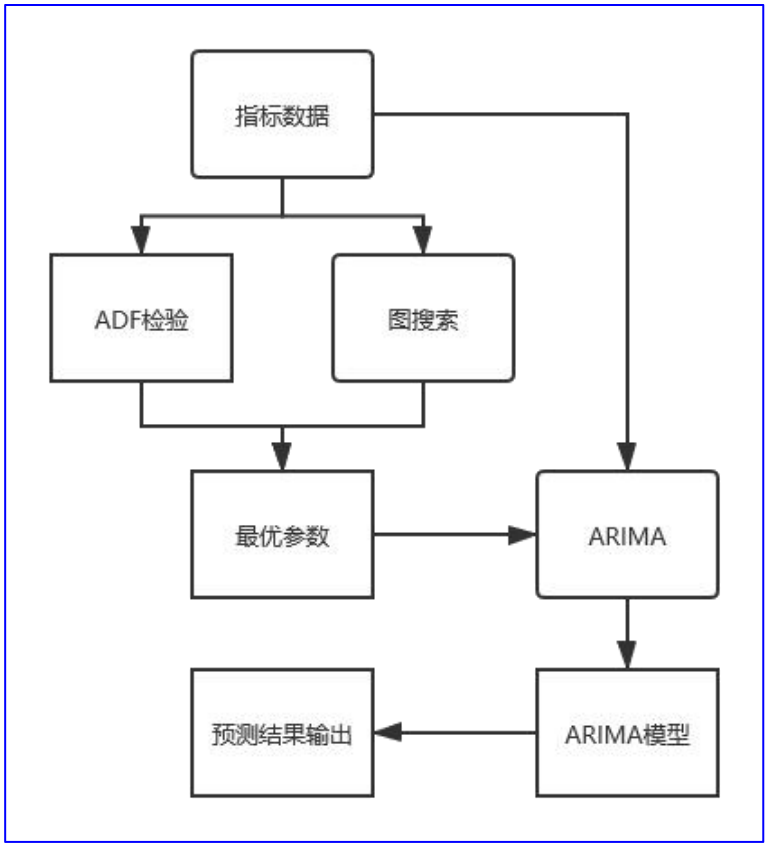
4.19.6 资源画像

4.19.6.1 功能描述

如何资源规格一直以来都是困扰应用运维人员的关键问题，过高的资源规格会导致大量的资源浪费，而过低的规格又会为应用带来潜在的稳定性风险。**InCloudOS** 为负载提供了资源画像的能力，实现容器粒度的资源规格推荐，可以有效降低为 **Pod** 配置 **Request** 和 **Limit** 的复杂度。同时也可为物理资源提供画像，防止资源过度使用，为用户进行扩容、迁移负载等操作提供依据。

4.19.6.2 技术原理

资源画像的难点是对资源使用率的预测。针对此功能，**InCloudOS** 产品中集成了三次指数平滑算法、**ARIMA** 算法、**STL** 算法构建模型，以 **smape** 作为评价标准。此处的资源画像，考虑到模型训练复杂性和数据量的限制，会采用短时序数据的预测进行刻画，以 **ARIMA** 算法构建模型为例。



资源的使用趋势预测，核心步骤如下：

- 判断样本点中数据点是否皆为正常。将数据进行切片，以 **80%**的数据用于训练，**20%**的数据用作测试。

1. 利用 ADF 检验对数据平稳性进行判断，判定差分阶数。

2. 利用图搜索的方法来进行判定 ARIMA 模型的另外两个参数。判定方法为使用训练数据训练 ARIMA 预测模型，参数设置为可能的参数范围。利用训练出的模型进行预测，预测数据与测试数据对比，以 BIC 作为依据，寻找 BIC 最小的参数组。

3. 以图搜索算法调优出的参数作为最终参数，将所有数据作为模型的训练数据输入，得出最终的 ARIMA 模型。

4. 利用训练出的模型计算出最终的预测值。

4.19.6.3 约束

无

4.20 巡检工具

4.20.1 功能描述

Incloud OS 巡检运维工具 **InCloud Doctor,** 在 **Windows** 系统下运行，主要用于 **InCloud OS** 系统的智能化巡检、云资源的状态巡检（包括虚拟化、容器、裸机和云管平台，支持多资源池），并提供信息收集（如资源规模、平台日志等）、调整修复等功能于一体的智能工具集。主要功能包括：

系统管理：界面运行交付、无需安装，直接运行；支持历史版本、支持多种模式部署方式；集成巡检、修复、查询等一体化智能工具

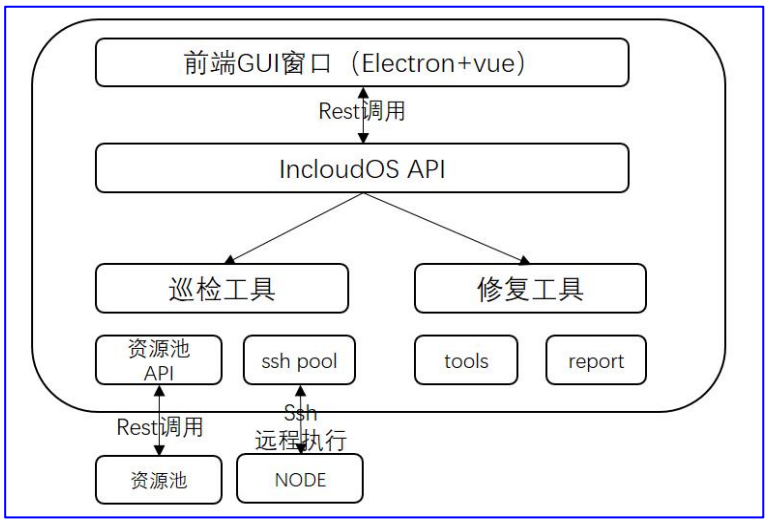
巡检工具：针对 **InCloudOS** 平台的硬件健康状态检测场景，包括总体运行资源及配置检测、计算资源检测、存储资源监测，并且能够智能化的针对异常问题提供修复建议，并能导出巡检报告。

巡检智能体：通过整合监控、日志和事件等多维数据，深入分析系统异常、资源负载和使用趋势，进而提供科学的瓶颈分析和扩缩容建议，确保系统始终处于最佳运行状态。

4.20.2 技术原理

InCloud Doctor 作为 **Incloud OS** 巡检运维工具，通过 **WEB** 对外提供页面服务，通过 **restful** 接口进行命令交互，封装 **python** 工具实现无需安装即运行效果，通过 **InCluodOS API**、**SSH pool**、**tools** 对于系统信息调用和后台命令执行，

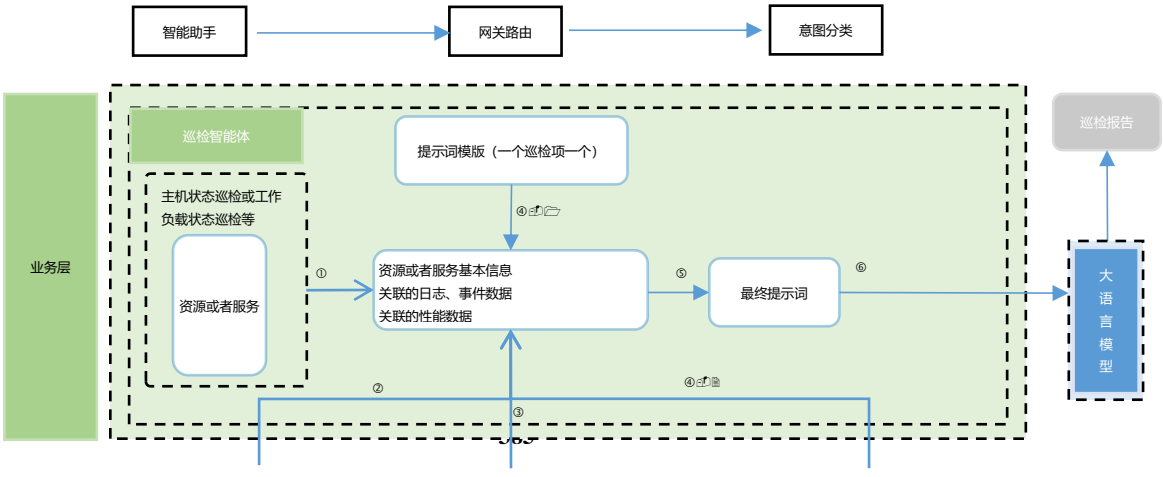
从而实现一体化巡检、修复、查询等工具。

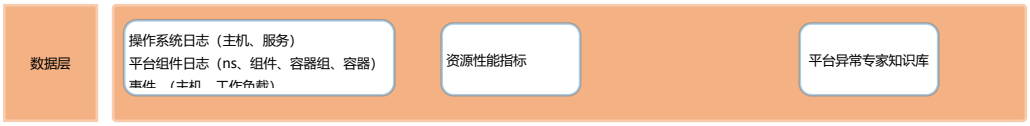


InCloud Doctor 前端采用 **Electron + vue** 技术架构，复用平台技术栈，降低开发难度同时提高产品的兼容性；后台采用 **rest** 调用 **IncloudOS** 接口，研发人员无需熟悉，直接上手；通过 **rest** 调用和 **ssh** 命令执行访问远端设备，对系统实现零侵入，提高整个工具的易用性和兼容性。

主要的技术点如下：

- 定期巡检云平台总体运行资源及配置，充分保障业务安全稳定运行。
- 可按照需求自定义检测项，精准快速检测所关注问题。
- 智能化的针对异常问题提供修复建议，辅助客户快速高效解决问题。
- 导出 **word** 版本巡检报告，检查修订后可直接用于汇报巡检结果。
- 在巡检工具无法解决问题时通过日志收集功能，导出日志进行分析。





巡检智能体分为 AI 助手层、应用层、数据层三层。AI 助手层：负责接收用户自然语言输入，进行意图识别与请求路由，是系统与用户交互的智能入口。业务逻辑层：协调巡检任务执行，融合多源数据并构建大模型提示词，是实现智能分析与决策的核心引擎。数据层：提供底层支撑数据，包括日志、性能指标和专家知识，为上层推理提供全面、实时的上下文信息。

4.20.3 约束

无

4.21 自动化运维

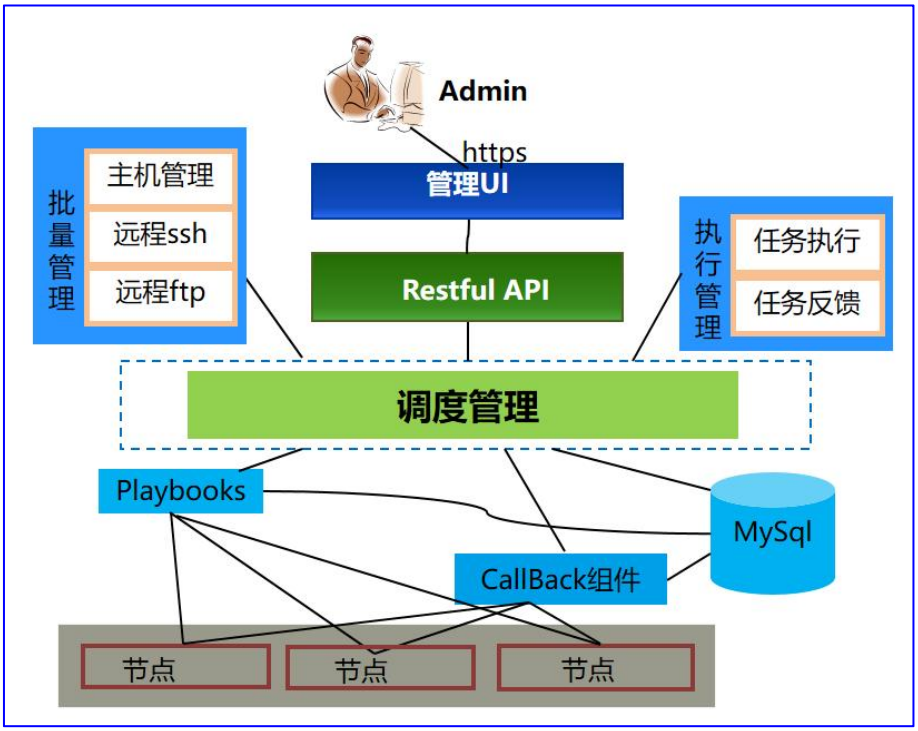
4.21.1 功能描述

自动化运维是通过将日常 IT 运维中大量的重复性工作由过去的手工执行转为自动化操作，从而减少乃至消除运维中的延迟，实现“零延时”的 IT 运维。

InCloud OS 自动化支持用户自定义常见格式的运维脚本（例如 shell、python 等），通过运维平台将运维脚本批量下发到指定虚拟机、裸金属、计算/管理节点并执行，简化重复性大的运维管理工作。也支持将不同的脚本按照一定的逻辑、流程编排成一个复杂的运维任务，管理员可以选择定时执行、周期执行或者手动执行等多种方式调度这些运维任务，以完成更复杂的运维动作。

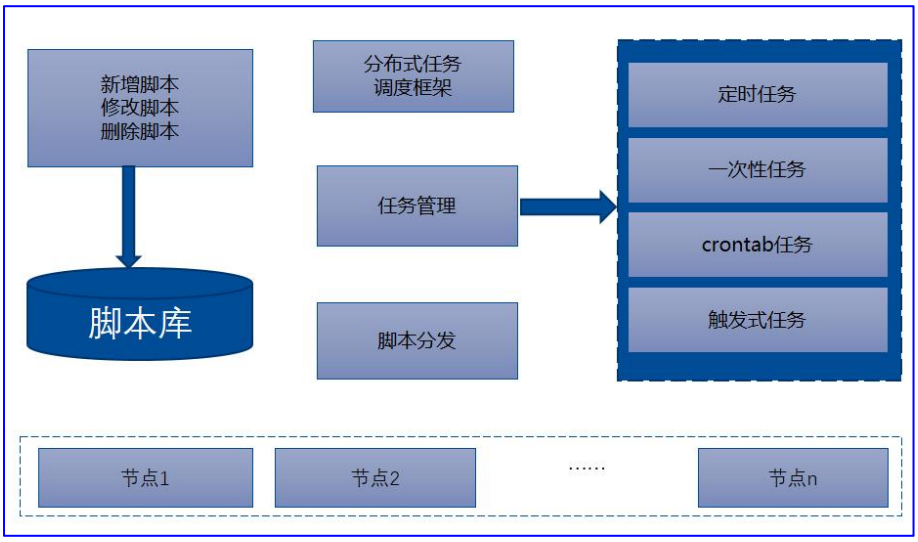
4.21.2 技术原理

基于 **Xterm.js** 开源前端组件，开发了一个页面客户端，直接执行 **SSH** 服务和 **FTP** 服务，保证通信安全，无需开启额外的端口和启动额外的程序。这个客户端模拟 **Xshell** 和 **FTP** 的功能，使应用程序在浏览器中为用户提供功能齐全的终端。通过使用加密的 **SSH** 和 **FTP** 协议，客户端确保数据传输的安全性，同时实现了安全的身份验证机制，确保只有授权用户才能访问远程服务器。客户端还使用了 **HTTPS** 协议，确保客户端和服务器之间的通信是加密的。



InCloud OS 作业管理系统的技术原理是基于以下几个关键组件：脚本解释器与执行引擎、任务编排引擎、任务触发机制和任务管理。脚本解释器与执行引擎负责解释和执行两种脚本语言（**shell** 和 **python**），并通过 **ftp** 将脚本任务分发到多个节点，通过 **ssh** 提供手动测试。任务编排引擎支持将多个脚本任务按照预定义逻辑和流程编排成一个复杂任务的机制，实现任务的自动化执行。任务触

发机制支持定时任务、一次性任务、**crontab** 任务、触发式等任务触发方式，确保任务能够按照预定义的时机执行。任务管理功能基于任务实时进度和执行日志查看，支持作业继续、终止和删除等管理，支持作业日志的查看，确保运维人员能够全面地管理和监控作业。同时，系统自动记录每条指令的执行结果，提供实时进度监控和多角度作业展示。通过这些技术原理，高效的作业管理系统能够提供一个高效、灵活、可靠的作业管理平台，满足云平台运维的需求。



4.21.3 约束

无

4.22 可视化安装部署

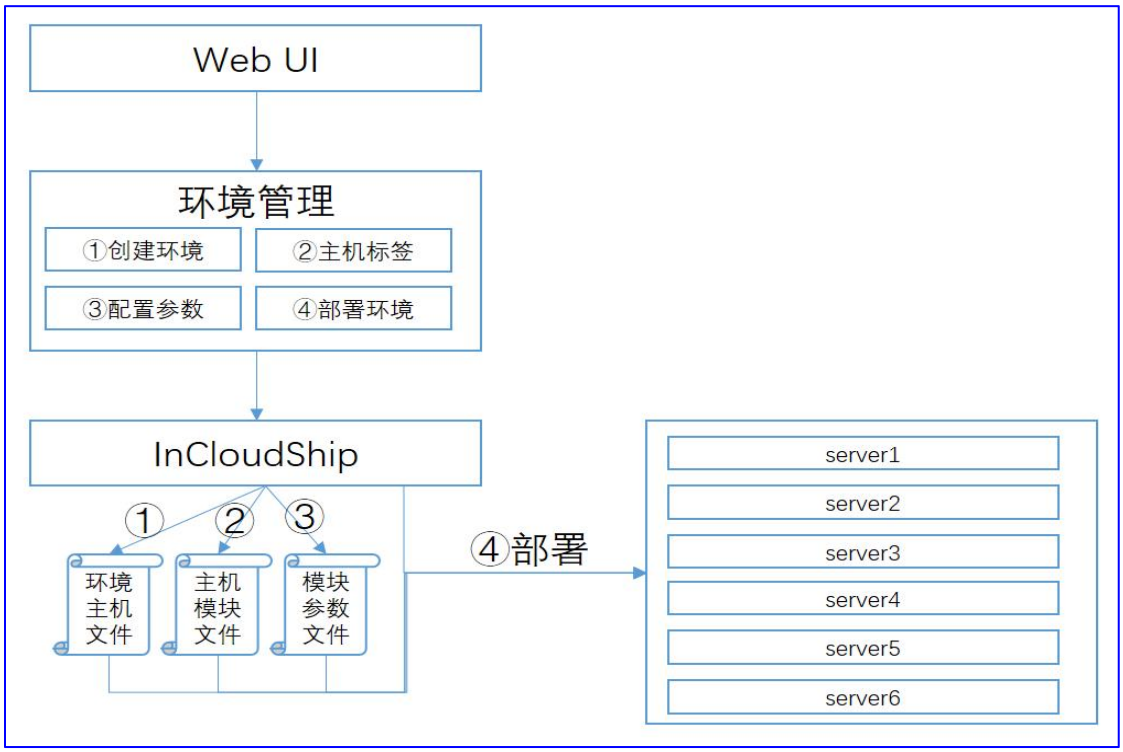
4.22.1 功能描述

可视化安装部署提供一套清晰易用的图形部署操作界面，简化部署参数配置，协助交付人员降低交付复杂度，提升交付效率。可视化安装部署实现了云管平台、容器资源池、裸金属资源池的部署，具体功能包括环境的创建、主机的选择、标签的设置、配置参数的输入、环境的编辑、环境的部署、环境的扩容、缩容等基

础部署功能。

4.22.2 技术原理

可视化安装部署工具包括 **UI** 层、业务逻辑层、部署接口层、硬件层。**UI** 层为交付人员提供操作接口，业务逻辑层负责把 **UI** 操作进行业务处理，把 **UI** 上的操作转化成部署云平台组件所必须的准备工作，通过 **InCloudShip** 接口层生成环境主机文件、主机模块文件、模块参数文件。环境主机文件说明了环境是云管平台、容器资源池还是裸金属资源池以及部署某个环境所用的主机列表，主机模块文件根据主机标签的管理定义了某个主机上将要部署哪些具体的模块，模块参数文件定义了这些模块分发到主机上时的参数配置。通过以上业务逻辑处理完成硬件上承载的环境部署。



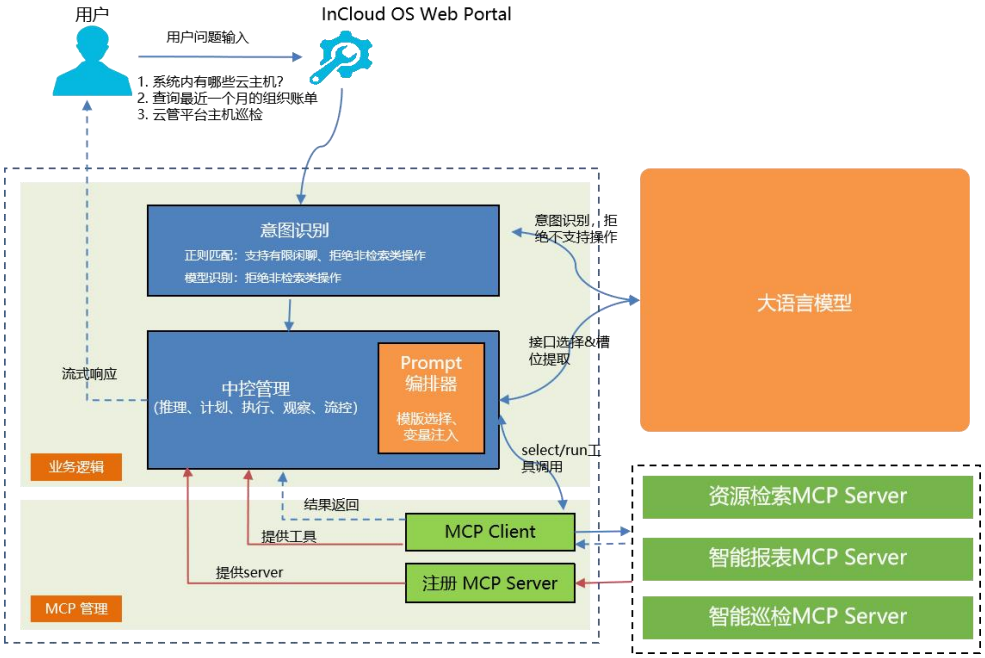
4.23 智能助手

4.23.1 功能描述

智能助手支持通过自然语言对话的方式，实现对系统中的云主机、云硬盘等云资源的检索、组织最近月度账单及资源用量等报表的生成以及对主机等系统工作负载的自动巡检。

4.23.2 技术原理

智能助手基于标准 **MCP** 协议实现了资源检索、智能报表及智能巡检 **MCP Server** 及相应的客户端。



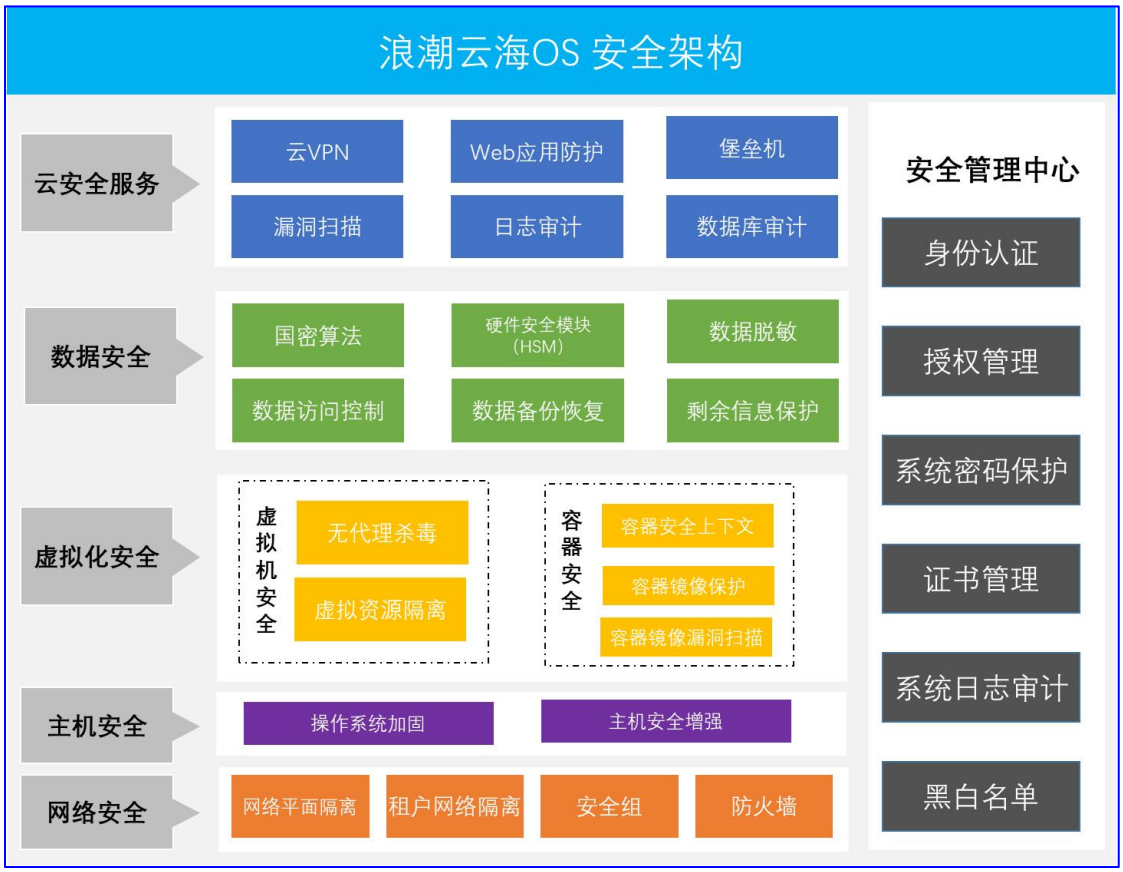
4.23.3 约束

无

5 安全可靠设计

5.1 多维度安全体系设计

浪潮云海 OS 采用“一个中心、多重防护”的安全策略，由安全管理中心统一调度管理，实现对云平台网络层、主机层、虚拟化层、数据层的安全防护，并提供面向多租户的云安全服务，构建起多层纵深防御的安全防护体系。



❖ 网络安全

1. 网络平面隔离：通过对 BMC 带外管理网络、管理网络、业务网络、存储网络的安全区域划分和隔离，阻止网络攻击在云平台不同网络平面间的扩散，将可能受到的攻击的影响最小化；

2. 租户网络隔离：基于 **VPC** 构建逻辑隔离的、由用户自主配置和管理的虚拟网络环境，提升用户资源的安全性；
3. 安全组：对进出虚拟机端口的网络报文进行安全过滤，为具有相同安全保护需求的云主机提供安全访问控制策略；
4. 防火墙：对进出路由器的网络报文进行安全过滤，为路由器所连接子网提供安全访问控制策略。

❖ 主机安全

1. 操作系统加固：通过最小化服务、用户权限管理等措施实现对操作系统文件、网络连接、进程等资源的有效保护，提高操作系统整体安全性和抗攻击能力；
2. 主机安全增强：集成浪潮 EDR 产品，采用内核级主动防御技术、应用软件自动化识别和白名单控制技术，实现主机安全增强。

❖ 虚拟化安全

1. 虚拟机安全：

- 虚拟资源隔离：不同用户的虚拟机独立运行、互不干扰，保障租户间的计算、网络和存储等资源不可见，避免恶意攻击和数据窃取；
- 无代理杀毒：支持 **Windows** 和 **Linux** 操作系统无代理底层防病毒能力，不需要在宿主机和虚拟机中部署安全防护代理。

2. 容器安全

- 容器安全上下文：支持容器安全上下文配置，包括是否以特权模式运行、文件系统读写权限、运行容器进程的用户/用户组；
- 容器镜像保护：提供镜像数字签名验证功能，确保容器镜像未被篡改。支持上传时启用镜像签名功能，可在镜像详情查看开启状态，在应用容器启动时验证签名信息；
- 容器镜像漏洞扫描：基于漏洞数据库对镜像版本进行安全扫描，用户可以快速的扫描容器镜像版本，以及查看镜像存在的漏洞详情。

3. 数据安全

(1) 国密算法：云管平台支持用户使用携带国密证书的 **USBKey+**用户名口令进行身份鉴别，鉴别方式符合《信息技术 安全技术 实体鉴别》（**GB/T 15843**）中规定的鉴别方式；云管平台用户可使用支持国密传输通道的浏览器进行登录，通过自主可控的密码技术，保护数据传输安全和服务器身份可信；支持使用 **SM2**、**SM3**、**SM4** 国密算法保护数据的机密性和完整性。

(2) 硬件安全模块（**HSM**）：通过对接第三方硬件密码设备，采用 **SM4** 算法对云平台的关键信息进行加密（包括用户姓名、邮箱、电话、密码等）后存储，采用 **HMAC SM3** 算法对云平台的关键信息（包括用户名、邮箱、手机号、用户权限、云主机信息、镜像元数据、操作日志等）进行完整性基准值计算，密钥及加密过程、完整性计算均由硬件加密机完成。

(3) 数据脱敏：对个人数据，如姓名、电话、邮箱等敏感信息通过脱敏规

则进行数据转换，实现敏感隐私数据的可靠保护。

(4) 数据访问控制：通过对文件、目录、数据库的权限管理以限制用户对某些信息项的访问；

(5) 数据备份恢复：支持云主机备份与快照、云硬盘备份与快照，并支持通过备份、快照恢复云主机或云硬盘；

(6) 剩余信息保护：支持云硬盘安全删除、内存空间删除等方式保证虚拟机所使用的内存和存储空间重新分配前得到完全清除。

❖ 云安全服务

1. **IPsec VPN**：为远端用户和 **VPC** 之间建立一条安全加密的通信隧道，使远端用户通过 **IPsec VPN** 直接访问 **VPC** 中的业务资源；
2. Web 应用防护：提供 **Web** 应用防火墙服务，防止 **Web** 应用免受各种常见攻击，比如 **SQL** 注入，跨站脚本漏洞（**XSS**）等；
3. 堡垒机：提供运维安全管控服务，包括帐号管理、认证管理、权限管理、审计管理、自动化运维等功能，解决系统帐号复用、运维权限混乱、运维过程不透明等 **IT** 运维难题；
4. 漏洞扫描：提供漏洞扫描服务，帮助用户高效、全方位的检测出业务环境中的各类脆弱风险，并提供专业、有效的安全分析和修补建议；
5. 日志审计：提供日志审计服务，有效地管理网络资产输出的各种日志数据，提供多维度的分析工具，方便用户对日志进行全面、高效的运维，并帮助用户发现日志中蕴含的各种安全风险。
6. 数据库审计：通过对业务人员、运维人员、研发人员等访问数据库的行

为进行解析、分析、记录、汇报，从而帮助用户进行事前风险评估，事中行为实时监控、违规操作行为及时响应告警，事后合规报告、事故追踪溯源，同时加强内、外部数据库操作行为监管、促进数据安全的正常运营。

7. 下一代防火墙：主要用于 **VPC** 边界 **3–7** 层的安全管控，保护 **VPC** 内虚拟机不被非授权访问，提供 **IP/URL** 访问控制、精细化应用控制、威胁防护、失陷检测、**APT** 联动防御、网络安全可视等安全能力，为云用户 **VPC** 边界提供更高级的边界安全能力。

❖ 安全管理中心

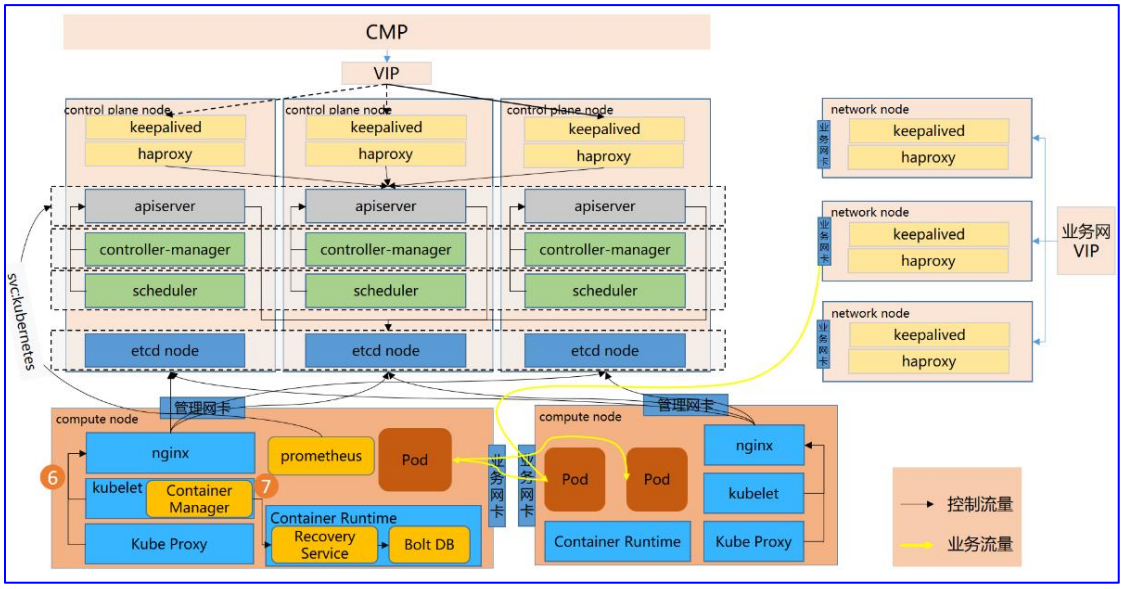
1. 身份认证：支持本地用户认证和域用户认证两类认证方式，支持双因子认证；
2. 权限管理：支持基于角色的访问控制模型（**RBAC**），将管理员按职责划分成不同角色并赋予最小权限，并支持用户自定义角色以及权限分域功能；
3. 账号密码保护：采用安全通道和密码技术实现用户账号密码信息的安全传输和加密存储，支持密码复杂度校验、登录失败锁定、会话管理等多种密码安全策略；
4. 证书管理：用户可以将自己的证书上传到云平台，证书可用于负载均衡数据加密场景和其他加密场景，证书内容将以加密形式保存于后端；
5. 系统日志审计：支持系统操作日志与运行日志的记录和查看，仅允许审计管理员或授权用户查看和管理系统日志，避免产生抵赖风险；

6. 黑白名单：支持配置访问系统的 IP 地址黑名单和白名单，限制非法访问的 IP 地址和地址段。

5.2 可靠性设计

5.2.1 容器资源池可靠性

容器集群采用副本机制维护集群组件的高可用，控制平面至少由 3 台服务器组成，其上运行 ICKS 的控制组件，避免单点组件，架构如下图所示。



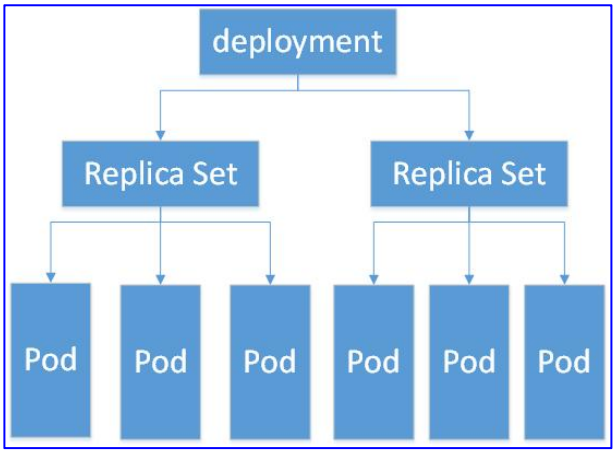
- ICKS 集群内 loadblancer 高可用：使用 ingress-nginx-controller 实现，ingress-nginx-controller 运行在 worker node 节点，通过负载控制平面 kube-apiserver 满足 ICKS 可靠性，控制平面 kube-apiserver 内部地址配置在 nginx 中接受负载请求，nginx 将流量分配给目标列表中所有运行状况良好的控制平面节点。
- 外部请求到 ICKS 的 loadblancer 高可用：使用静态 pod 方式部署的 haproxy、keepalived 实现，haproxy 分别运行在 Master 节点服务器上组成主备集群，通过 keepalived 组件提供 vip，vip 所在的节点为 api

的请求入口，请求被负载到后端状态正常的 **apiserver** 组件上。当 **keepalived** 所在节点的 **haproxy** 出现问题时，**keepalived** 将把 **vip** 切换到其他节点。控制平面节点 **kube-apiserver** 放置在 **haproxy TCP** 后面接受负载请求。负载均衡器将流量分配给目标列表中所有运行状况良好的控制平面节点 **kube-apiserver**。

- **kube-apiserver** 高可用：在 **Node** 节点上使用 **nginx** 代理对多个 **Master** 做负载均衡。
- **kube-controller-manager** 与 **kube-scheduler** 高可用：这两项服务是 **Master** 节点的一部分，他们的高可用相对容易，仅需要运行多份实例即可。这些实例会通过向 **apiserver** 中的 **Endpoint** 加锁的方式来进行 **leader election**，当目前拿到 **leader** 的实例无法正常工作时，别的实例会拿到锁，变为新的 **leader**。
- **etcd** 高可用：使用独立的 **etcd** 集群，使用 **3** 台或者 **5** 台服务器只运行 **etcd**，独立维护和升级。这个 **etcd** 集群将作为基石用于构建整个集群。采用这项策略的主要动机是 **etcd** 集群的节点增减都需要显式的通知集群，保证 **etcd** 集群节点稳定可以更方便的用程序完成集群滚动升级，减轻维护负担。
- **Worker node** 高可靠：**Worker node** 是 **ICKS** 集群中为 **pod** 提供运行环境的资源池，其上的 **pod** 都由控制平面控制，控制平面中 **etcd** 维护者所有 **ICKS** 资源的元数据、状态等信息，**controller-manager** 通过 **list/watch** 机制查看 **etcd** 里存储的数据来决定 **pod** 的生命周期与调度。当某个 **worker node** 出现异常状况后，其上的 **pod** 资源会得到合理的处

理，始终维持着组件可靠运行所需的 **pod** 个数。因为 **ICM** 以 **workload pod** 的形式部署在 **ICKS** 之上，所以 **ICM** 组件的可靠性受益于成熟的 **ICKS** 高可靠设计，加之 **ICM** 自身的高可靠设计，使得 **ICM** 的运行更加稳定可靠。

- 业务 **POD** 高可靠：使用 **Deployment** 来创建 **Replica Set**。**Replica Set** 在后台创建 **pod**。确保 **Kubernetes** 中有指定数量的 **pod** 在运行。如果少于指定数量的 **pod**，会创建新的 **pod**，反之则会删除掉多余的以保证 **pod** 数量不变。



- **pod** 健康检测：当 **pod** 不健康，运行出错或者无法提供服务时，由于某种原因挂掉或崩溃。**ReplicaSet** 会立刻用这个 **Pod** 的模版新启一个 **Pod** 来替代它。
- 弹性伸缩：在业务高峰或者低峰期的时候，可以动态的调整 **pod** 的数量来提高资源的利用率。平台定时自动监控平台 **pod** 的整体资源使用情况，并基于预设的策略做到自动伸缩 **pod** 数量。
- 滚动升级：滚动升级为一种平滑的升级方式，通过逐步替换的策略，保证整体系统的稳定，在初始化升级的时候就可以及时发现和解决问题，避免问题不断扩大。

- 回滚：当升级 **pod** 镜像或者相关参数的时候发现问题，可以回滚，回滚到上一个稳定的版本或者指定的版本。每一次对 **Deployment** 的操作，都能保存下来，以便为以后回退使用。
- 云内容器资源池高可用：云内资源池为一键创建在 **ICS** 资源池云主机上的容器资源池，该版本针对云内类型的容器资源池，在资源池管理和 **Node** 节点访问 **Apiserver** 方面进行了高可用优化：

资源池管理：优化前，**CMP** 通过第一个 **Master** 云主机节点绑定的浮动 **IP** 来管理资源池,存在单点故障,故障后需要手动切换浮动 **IP** 到其他 **Master** 节点；优化后，**CMP** 创建云内资源池后，资源池自动通过代理向 **CMP** 进行注册和纳管，在资源池 **Master** 节点和 **CMP** 间形成高可用的管理通信隧道，资源池单 **Master** 故障，不影响整个资源池的管理。

Node 节点访问 **Apiserver**：优化前，容器资源池的 **Node** 节点通过第一个 **Master** 节点的浮动 **IP** 访问 **Apiserver**，此 **Master** 或者浮动 **IP** 故障后，**Node** 节点无法与 **Master** 节点进行通信，导致整个集群不可用；优化后，**Node** 节点通过访问本地负载均衡来进行 **Apiserver** 的代理，负载均衡配置三个 **Master** 节点的固定 **IP** 即可，达到既不依赖浮动 **IP**，也不存在单点故障。

5.2.2 虚拟化资源池可靠性

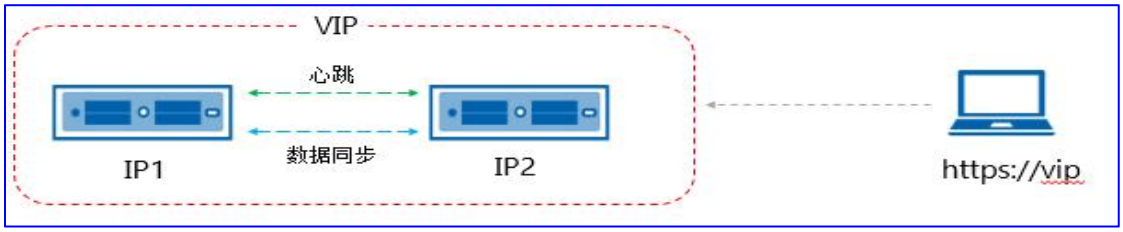
虚拟化资源池管理节点采用双机热备的架构，保障平台的可靠性。为消除管理节点单点故障，虚拟化资源池提供管理节点双机热备的部署方式，通过浮动 **IP** 访问管理平台。两台 **iCenter** 管理节点实时互备数据，其中一台 **iCenter** 节点作为整个系统的管理节点，当主管理节点宕机时另一台 **iCenter** 节点会迅速接管整套系统以保障系统整体的持续运行。

虚拟化资源池通过添加多个仲裁 **IP**、多条心跳链路、配置仲裁盘等方式提高双机热备的稳定性。

仲裁 **IP**：用于检测 **iCenter** 节点本身的网络是否正常，建议配置三个或者三个以上。

心跳链路：用于 **iCenter** 节点信息的交互，建议配置两条或者两条以上。

仲裁盘：用于判断 **iCenter** 节点数据是否正常，只可配置一个。

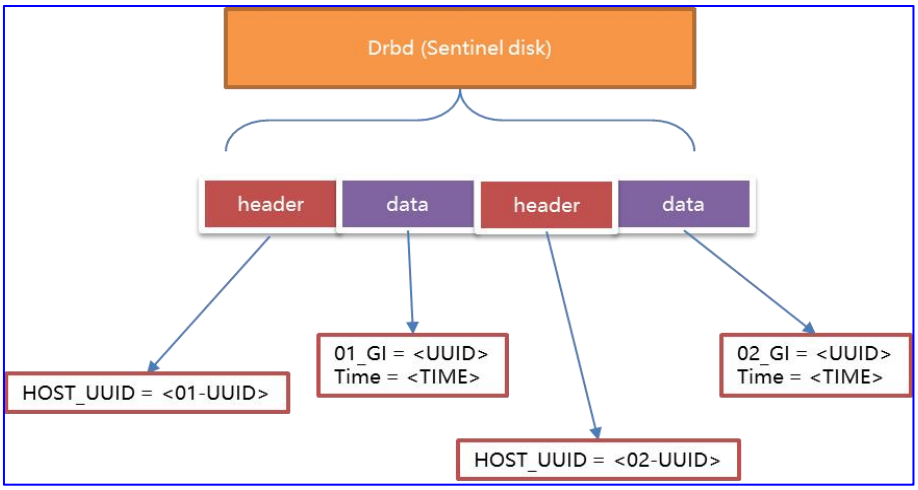


双机热备

虚拟化资源池通过 **Heartbeat** 机制实现了心跳监测和资源接管调度，心跳监测可以通过网络链路和串口进行，而且支持冗余链路，主备管理节点之间相互发送报文以告诉对方自己当前的状态，备管理节点如果在指定的时间内未收到主管理节点发送的报文，则备管理节点就会判定对方失效，进而启动系统接管服务。

虚拟化资源池基于 **DRBD** 块设备存储技术主管理节点的数据在更动后自动复制到备管理节点，实现了主备节点间数据块的实时完整复制以达到双机数据的一致性。

Heartbeat 通过读写 **DRBD** 仲裁盘中的 **GI** 值以判断主备节点的数据是否一致。**DRBD** 仲裁盘是一个共享的磁盘，主备节点通过定时服务不停地向该共享盘中写入 **DRBD** 的 **GI** 信息。如图所示，**GI** 信息是 **DRBD** 数据状态的一种签名，如果主备节点的 **DRBD** 数据状态是同步的，则 **GI** 信息值是一样的，反之则不一样。

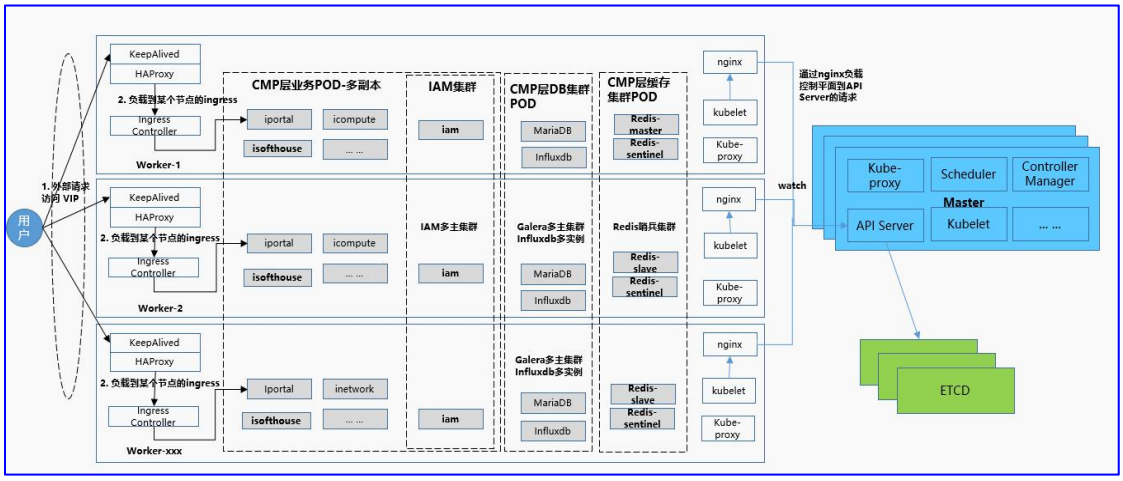


当主备管理节点同时意外关机，且其中一台管理节点因硬件故障无法开机时，另一台可以正常开机的管理节点需要通过读取仲裁盘中的 **DI** 信息以判断其是否可以正常启动管理服务。如果该管理节点的 **GI** 信息包含对端的 **GI** 信息，则该节点可以单独启动管理服务，否则禁止启动管理服务。

5.2.3 云管平台可靠性

5.2.3.1 总体介绍

云管平台采用多副本机制维护服务组件的高可用，避免组件单节点故障。具体架构图如下：



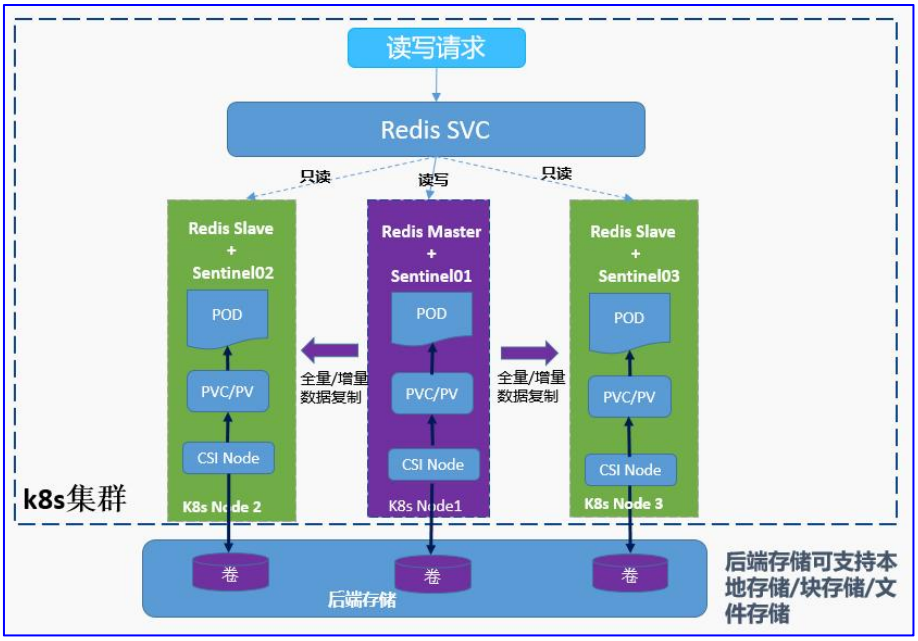
云平台的功能模块高可用设计包括下面几个方面：

- 数据库中间件高可用：**Redis** 高可用通过一主二从三 **Redis** 哨兵部署模式实现 **Redis** 服务的高可用，三哨兵保证了哨兵节点的高可用，主从数据复制、主从选举机制，保证了 **Master** 节点宕机后，会重新选举另一个 **Master** 节点承接写服务，保证业务的连续性和高可用。
- 服务组件高可用：基于 **K8S+SpringBoot**、**SpringCloud** 的架构模式，**CMP** 的每个服务组件多副本部署，在一个节点因为故障停止服务时，请求会自动分配到其他可用节点，并且 **K8S** 会维护节点的副本数，保证系统中每个微服务的可用性。**IAM** 组件由于存在多副本缓存共享的问题，在基于多副本部署的基础上，引入了多主缓存集群来实现 **IAM** 的高可用。多个副本的 **IAM** 具备平等地位，多个节点可同时提供请求服务，避免单节点故障后，影响用户正常的访问系统。

5.2.3.2 数据库中间件高可用

系统使用 **Redis** 作为内存数据库、分布式锁，通过哨兵模式部署实现 **Redis** 服务的高可用。系统采用一主二从三哨兵的部署模式，主从读写分离，三哨兵保证了哨兵节点的高可用，通过主从数据复制保证主从数据的一致性，通过主从选举机制，确保 **Master** 节点宕机后，可以重新选举 **Master** 节点承载写操作。从而保证了业务联系性，实现了 **Redis** 服务的高可用。

产品架构



功能特性

- 主从复制：主从基于 **socket** 连接，实现主从数据的准实时同步，可实现全量/增量的数据同步方式。
- 主从选举：哨兵(**Sentinel**)会实时的监控 **redis** 服务节点状态，当 **Master** 所在节点宕机后，哨兵会选举某个 **slave** 节点作为 **Master** 节点，继续提供服务
- 读写分离：**Redis** 客户端使用 **Redission**，支持三种读写模式
- **master** 模式：**master** 读写，**slave** 只做备份
- **master–slave**：**master** 读写，**slave** 读。
- **slave** 模式：**master** 写，**slave** 读
- 支持三种随机，权重，轮询负载方式，可自定义负载算法

5.2.3.3 微服务组件高可用

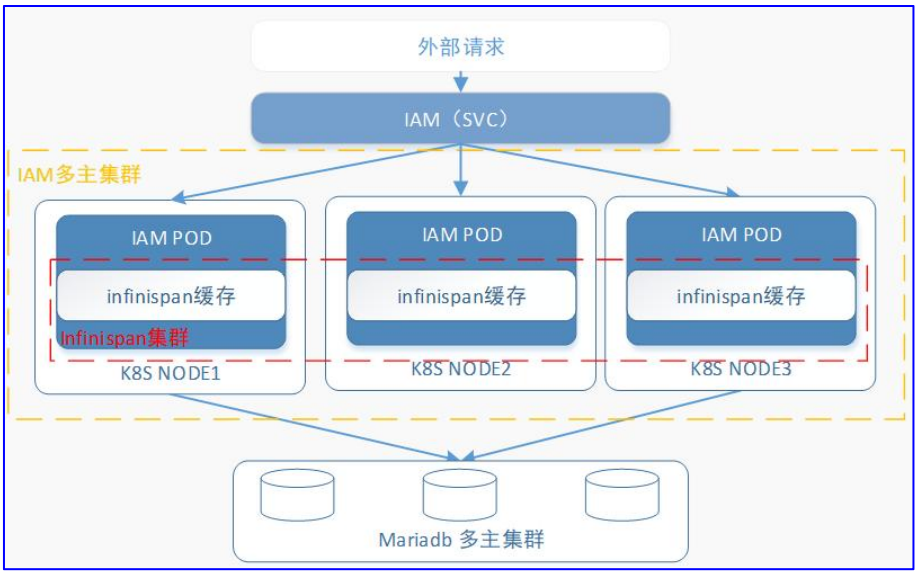
平台采用微服务架构的设计理念，基于 **K8S+SpringBoot**、**SpringCloud** 的架构模式，每个服务节点可分布式独立部署。平台微服务架构可实现多服务节点

间的服务的负载均衡、服务路由、集群容错等机制，使系统具有很强的连通性，健壮性，伸缩性和扩展性。

平台采用全方位的高可用设计，平台中的每个微服务都可以多副本部署，在一个节点因为故障停止服务时，请求会自动分配到其他可用节点，并且 **K8S** 会维护节点的副本数，保证系统中每个微服务的可用性。系统内置健康检查机制，定时扫描各个服务是否健康，一旦某个服务鉴定为非健康状态，那么该服务会重启部署，保证节点的高可用。

- 多副本：系统的各个组件以 **pod** 集群的形式运行在 **K8S** 容器环境下，**K8S** 可以配置每个 **pod** 的副本数量，当某个 **pod** 由于某种原因宕掉时，**K8S** 会自动重启一个新的 **pod**,保持正在运行的 **pod** 数量和其副本数量一致。
- **VIP**：**K8S** 的 **Service VIP** 机制为同一个组件的 **pod** 提供了统一的对外访问 **IP**，实现 **pod** 服务集群内的访问负载均衡。
- 服务熔断：**Hystrix** 实现熔断器确保某个服务故障时系统依然可用。
- 服务间调用:各模块服务独立部署于各自 **pod** 集群,模块服务间用 **Feign** 实现内部调用，**UI** 与网关、网关与各个服务模块之间通过 **REST API** 的方式调用。

IAM 相比较 **CMP** 其他的组件中有些特殊,存在各副本之间的用户缓存数据共享的问题，在基于多副本部署的基础上，引入了 **infinispan** 多主缓存集群来实现 **IAM** 的高可用。每个 **IAM** 节点具备平等地位，各副本节点同时支持认证和用户等信息的读写操作,解决了单节点宕机后服务短暂不可用和认证状态丢失的问题，实现多节点 **IAM** 的高可用能力。



- 部署 **infinispan** 集群，通过 **infinispan** 缓存用户认证信息等关键数据，数据持久化层面使用 **Mariadb** 的高可用能力，解决了宕机后服务短暂不可用和认证状态丢失的问题，实现多节点 **IAM** 的高可用能力。

5.3 商用密码应用安全性评估设计

InCloud OS 需要通过《商用密码应用安全性评估量化评估规则》（后面简称密评），

按照专业、合规、经济的原则，对接专业密码领域生态厂商（三未信安、海泰方圆等），形成云平台的密码合规方案，满足行业客户安全要求，助力行业客户满足密码安全相关要求。

涉及到对接第三方加密机（三未信安）进行加解密、**HMAC** 运算,使用第三方 **ukey**、国密浏览器进行身份鉴别、传输机密性等，包含 **4** 个方面的需求：

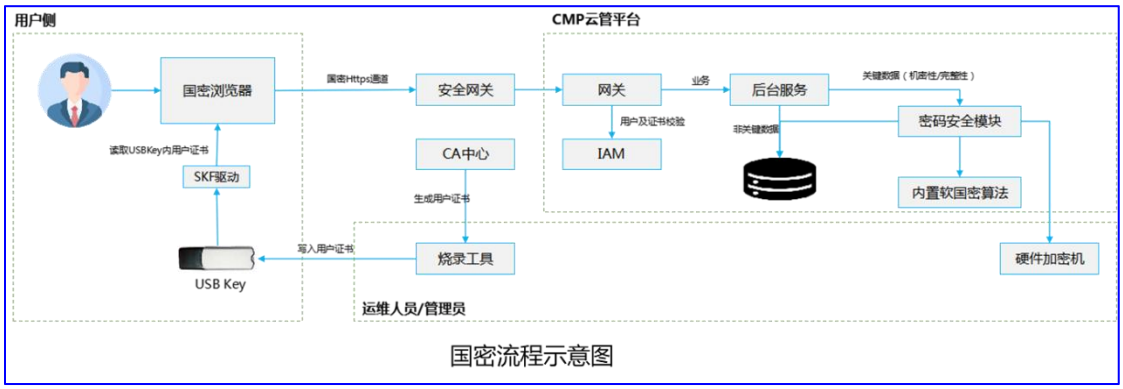
1.传输机密性：云管平台用户使用支持国密传输通道的安全浏览器、**SSL VPN** 安全网关及智能秘钥 **USBKey** 在网络层面共同完成，通过自主可控的密码技术，保护数据传输安全和服务器身份可信（兼容红莲花浏览器、三未信安

IPSec/SSL VPN 综合安全网关、海泰方圆 USBKey）。

2. 身份鉴别：云管平台用户使用携带国密证书的 **USBKey+**用户名口令进行身份鉴别，鉴别方式符合《信息技术 安全技术 实体鉴别》（**GB/T 15843**）中规定的鉴别方式（兼容海泰方圆 **USBKey**，同时需要使用红莲花国密浏览器）。

3. 数据机密性：通过对接第三方硬件密码设备，采用 **SM4** 算法对云平台的关键信息进行加密（包括用户姓名、邮箱、电话、密码等）后存储，密钥及加密过程由硬件加密机完成，从硬件层面保障数据安全性。（兼容三未信安密码机）。

4. 数据完整性：通过对接第三方硬件密码设备，采用 **HMAC SM3** 算法对云平台的关键信息（包括用户名、邮箱、手机号、用户权限、云主机信息、操作日志等）进行完整性基准值计算，完整性计算由硬件加密机完成，从硬件层面保障数据完整性。



国密流程示意图

机密性**&**完整性数据说明

机密性保护数据

关键数据描述	数据类型	安全需求	算法
账号、口令	原始数据	机密性	**SM4**
用户个人信息(姓名、邮箱、电话)	原始数据	机密性	**SM4**

完整性保护数据

关键数据描述	数据类型	安全需求	算法
账号、口令	原始数据	完整性	**HMAC–SM3**
用户个人信息(姓名、邮箱、电话)	原始数据	完整性	**HMAC–SM3**
用户访问权限信息	原始数据	完整性	**HMAC–SM3**
记录系统操作日志等信息，操作时间、账号、**IP**、操作对象、操作结果、日志级别、详细信息	原始数据	完整性	**HMAC–SM3**
虚拟机配置数据（**id**、名称、所有者、生命周期、**CPU**、内存）、规格数据（**id**、名称、大小、上传时间、**MD5**值）	原始数据	完整性	**HMAC–SM3**

6 产品核心组件清单

类别	组件名称	备注
CMP 子系统架构（含软件定义层）	**ibase/IAM**	认证鉴权服务组件
	icompute	计算服务组件
	inetwork	网络服务组件
	istorage	存储服务组件
	icluster	容器集群服务组件
	ipaas	**Paas** 服务组件
	ipluto	监控服务组件
	isync/otter/irecovery	容灾服务组件
	neutron	软件定义网络组件
	ironic	裸机服务组件
	manila	文件服务组件
虚拟化子系统	**iCenter**	虚拟化管理服务组件
	iNode	虚拟化计算服务组件
容器子系统	**kube-apiserver**	容器资源 **api** 服务组件
	kube-scheduler	容器资源调度服务组件
	kube-controller-manager	容器资源控制器管理组件

	etcd	容器资源数据存储组件
	kube-proxy	容器网络服务组件
	kubelet	容器计算服务组件
	ingress-controller	容器资源池对外提供服务组件

7 缩略语

中文全称	英文缩写	英文全称
应用程序编程接口	API	Application Programming Interface
中央处理器	CPU	Central Processing Unit
动态主机配置协议	DHCP	Dynamic Host Configuration Protocol
域名系统	DNS	Domain Name System
分布式虚拟路由	DVR	Distributed Virtual Router
光纤通道	FC	Fiber Channel
简单文件传输协议	TFTP	Trivial File Transfer Protocol
图形处理器	GPU	Graphics Processing Unit
高可靠性	HA	High Availability
超文本传输协议	HTTP	Hypertext Transfer Protocol
基础设施即服务	IaaS	Infrastructure as a Service
互联网数据中心	IDC	Internet Data Center
输入输出	I/O	Input and Output
每秒读写（I/O）操作次数	IOPS	Input/Output Operations Per Second
网络互连协议	IP	Internet Protocol
智能平台管理接口	IPMI	Intelligent Platform Management Interface
互联网传输协议安全性	IPSec	Internet Protocol Security

中文全称	英文缩写	英文全称
因特网小型计算机系统接口	iSCSI	Internet Small Computer Systems Interface
基于内核的云主机	KVM	Kernel-based Virtual Machine
模型-视图-视图模型	MVVM	Model-View-ViewModel
网络附属存储	NAS	Network Attached Storage
网络地址转换	NAT	Network Address Translation
网络文件系统	NFS	Network File System
操作系统	OS	Operating System
平台即服务	PaaS	Platform as a Service
预启动执行环境	PXE	Preboot Execute Environment
独立冗余磁盘阵列	RAID	Redundant Array of Independent Disks
服务质量	QoS	Quality of Service
内存	RAM	Random Access Memory
基于角色的访问控制	RBAC	Role-Based Access Control
表示状态转移	REST	Representational State Transfer
软件即服务	SaaS	Software as a Service
存储区域网络	SAN	Storage Area Network
服务器信息块	SMB	Server Message Block
软件定义网络	SDN	Software Defined Network
源地址转换	SNAT	Source Network Address Translation
传输控制协议	TCP	Transmission Control Protocol

中文全称	英文缩写	英文全称
用户数据报协议	UDP	User Datagram Protocol
用户界面	UI	User Interface
通用串行总线	USB	Universal Serial Bus
虚拟图形处理单元	vGPU	Virtual Graphics Processing Unit
虚拟 IP	VIP	Virtual Internet Protocol
虚拟局域网	VLAN	Virtual Local Area Network
云主机	VM	Universal Serial Bus
云主机监控器	VMM	Virtual Machine Monitor
虚拟路由冗余协议	VRRP	Virtual Router Redundancy Protocol
可扩展虚拟局域网	VXLAN	Virtual Extensible Local Area Network
虚拟私有云	VPC	Virtual Private Cloud
虚拟私有网络	VPN	Virtual Private Network
扩展标记语言	XML	Extensible Markup Language
复原时间目标	RTO	Recovery Time Objective
复原点目标	RPO	Recovery Point Objective
基于角色的访问控制	RBAC	Role-Based Access Control
浪潮云海云操作系统	InCloud OS	InCloud OS
硬件安全模块	HSM	Hardware Security Module