**自主式交通集成中台设计方案**

**--自主式交通端边云一体化智能操作系统关键技术研究与应用**

目录

[一、引言 3](#_Toc186101579)

[二、顶层设计 3](#_Toc186101580)

[三、技术架构 3](#_Toc186101581)

[3.1基础设施管理层 5](#_Toc186101582)

[3.2数据中台层 6](#_Toc186101583)

[3.3算法中台层 6](#_Toc186101584)

[3.4服务中台层 6](#_Toc186101585)

[3.5业务中台层 6](#_Toc186101586)

[3.6运营中台层 6](#_Toc186101587)

[3.7大规模分布式实时数据总线 7](#_Toc186101588)

[3.8 集成中台总控层 7](#_Toc186101589)

[四、技术框架和实现路径 7](#_Toc186101590)

[4.1 其整体部署架构如图 7](#_Toc186101591)

[4.1.1 不可变基础设施层 8](#_Toc186101592)

[4.1.2 按需组织的资源层 8](#_Toc186101593)

[4.1.3 专业化的服务层 9](#_Toc186101594)

[4.1.4 按需定制的业务层 9](#_Toc186101595)

[4.2 通过五种方式来实现五个层面统一 9](#_Toc186101596)

[4.2.1 功能丰富和快速实现的统一 9](#_Toc186101597)

[4.2.2 灵活开放和系统稳健的统一 9](#_Toc186101598)

[4.2.3 专业集成和方便使用的统一 10](#_Toc186101599)

[4.2.4 专家系统和人工智能的统一 10](#_Toc186101600)

[4.2.5 方便快捷和安全可控的统一 10](#_Toc186101601)

[4.3 各层实现概要设计 10](#_Toc186101602)

[4.3.1. 数据总线架构概述 10](#_Toc186101603)

[4.3.2 集成中台概要设计 20](#_Toc186101604)

[4.3.3. 五大中台设计 32](#_Toc186101605)

[4.3.4 基础设施层 33](#_Toc186101606)

[五、关键技术 33](#_Toc186101607)

[5.1基于云原生的自动化调度 33](#_Toc186101608)

[5.2基于凤凰架构的全域可观测体系 34](#_Toc186101609)

[5.3 LLM赋能的新一代智能交通感知、规划、决策能力 35](#_Toc186101610)

[六、实施路径 35](#_Toc186101611)

## 一、引言

面对新一轮科技革命和产业变革,交通运输行业正经历着数字化、网联化、智能化的深刻变革。传统的信息化"烟囱式"建设模式已难以满足当前智慧交通发展的需求,亟需构建一个全新的数字底座,以体系化思维、平台化架构、生态化模式实现各要素的深度融合,全面赋能行业数字化转型。

本技术方案提出了一种基于云原生的智能交通操作系统,融合了现代在云原生、微服务、服务网格等技术领域的深厚积淀,以在智能交通、自动驾驶、车路协同等场景的实践经验,并引入大语言模型(LLM)、凤凰架构等前沿理念,旨在打造面向未来的新一代交通基础设施云、边、端的智能操作系统平台，全栈协同致力于打造一个高可靠、高性能、高智能的分布式智慧交通操作系统。系统立足于全场景覆盖、全生命周期管理,秉承软硬协同、自主可控的理念构建统一技术架构,并采用中台化设计思想实现能力沉淀与复用,力求为智慧交通应用创新提供最坚实的数字根基。

## 二、顶层设计

系统以"端边云协同、软硬件一体、数智融合"为顶层设计原则，从接口、中间件到应用支持实现全栈自研、自主可控、安全可信，采用软硬分层解耦的微内核架构,既实现核心技术"换道超车",又能灵活适配多元算力平台。

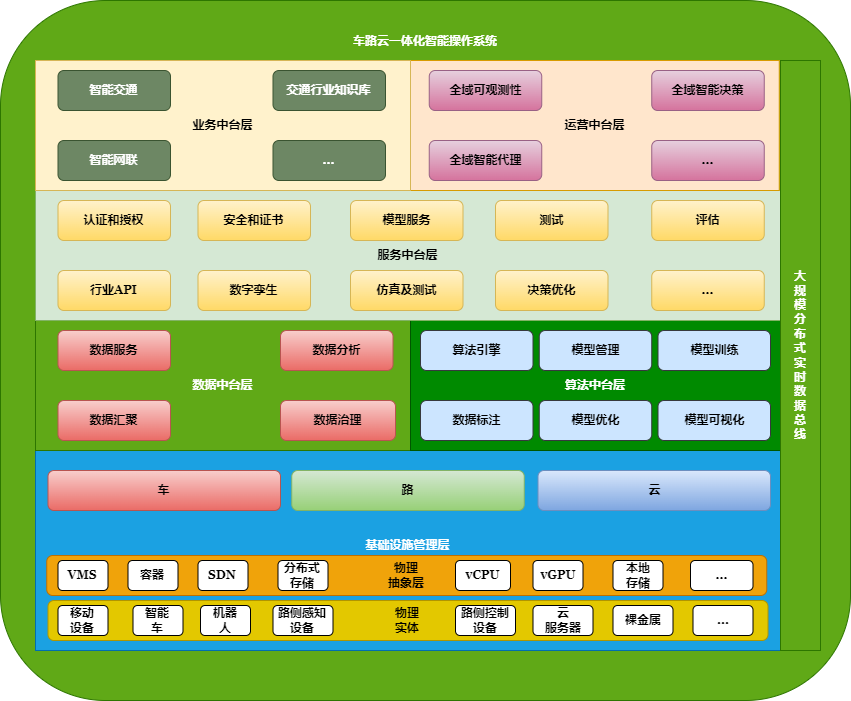
端侧系统：聚焦车路协同计算与通信单元，采用自主可控的车规级计算平台,支持多传感器融合感知、多模态人机交互、实时决策控制等全栈能力开发和应用。

边侧系统：面向复杂交通场景,构建具备自治、协同、进化能力的自主式交通边缘云，采用高可靠、低时延、高带宽的新一代通信技术,布局MEC边缘云节点,下沉核心业务和关键数据,实现毫秒级实时处理和智能响应，通过服务网格、区块链等云原生技术实现多层级、多粒度的边缘协同,提升任务处理效率。

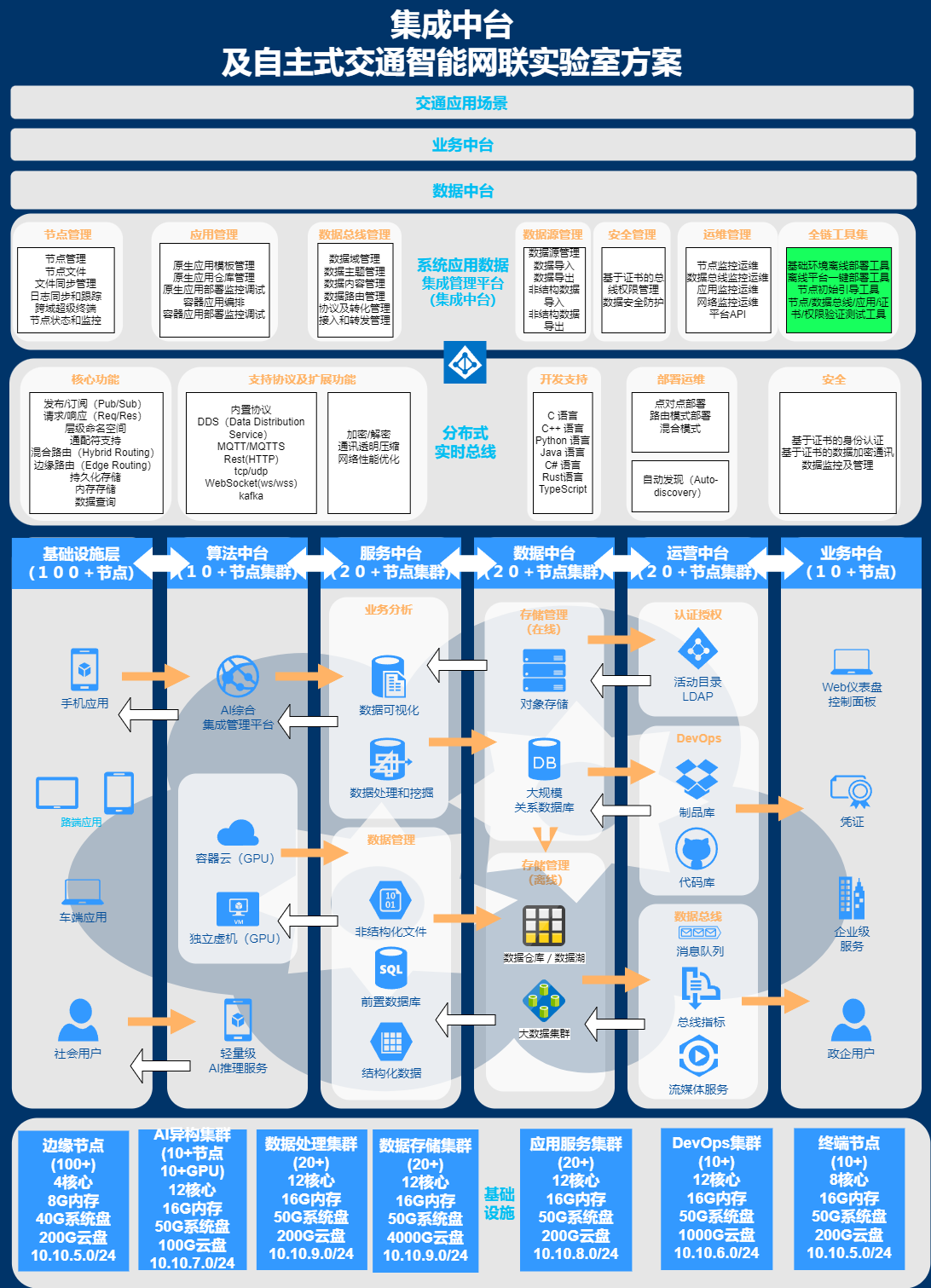
云端系统：汇聚海量异构资源,发挥云计算规模效应和聚合效应,打造智慧交通的数字大脑，并与端、边实现计算协同，采用"分而治之、统一协同"的多云架构,通过软件定义、虚拟化容器等实现资源配置自动化、业务部署敏捷化、运维管理智能化，运用大数据分析、知识图谱、深度学习等人工智能技术,沉淀智慧交通领域知识,实现数据资产化。

## 三、技术架构

系统遵循云原生架构设计原则,采用"6+1+1"的中台化架构,自下而上分为基础设施管理层、数据中台层、算法中台层、服务中台层、业务中台层，以及贯穿其中的运营中台层和分布式数据总线，最后由集成中台层进行统一的抽象和管理。。



结合现有科研的条件下，我们构建了集成中台落地方案化方案



### 3.1基础设施管理层

整合了底层计算机操作系统能力，面向支撑的计算平台硬件设备和关联的智能网联硬件设备，通过软件实现对各种资源(计算，网络，存储等)、负载的统一定义，支撑上层数据、算法、服务和业务的统一管理和协同计算。

同时通过嵌入智能代理，实现底层数据贯通和智能代理动作。

### 3.2数据中台层

数据汇聚：利用分布式实时数据总线,对多源异构数据“一点接入、多点分发”；

数据治理：数据全生命周期管理,确保数据按需访问、合规使用、质量可信；

数据服务：数据虚拟化、数据地图、API网关等形成统一数据视图和访问入口；

数据分析：分布式计算引擎、在线机器学习等实现海量异构数据的高效分析。

### 3.3算法中台层

算法引擎：预置高频通用算法如路径规划、ETA预测等,支持低代码开发和部署；

模型管理：自动化机器学习流水线,支持模型全生命周期（数据标注，模型训练，模型优化，模型可视化，模型部署等）的管理与优化；同时支持大语言模型的微调，多模态，RAG，代理等开发，部署等。

服务化：将算法转化为可复用的微服务单元,实现灵活编排和弹性伸缩；

持续学习：利用增强学习、迁移学习等使模型不断自我迭代,提高预测泛化能力。

### 3.4服务中台层

数字孪生：融合BIM、GIS、IoT等数据,构建全维度、全周期的道路交通孪生体；

仿真测试：提供实时/快速仿真环境,加速智能驾驶、交通组织等场景测试验证；

决策优化：整合路况预测、信号优化、交通诱导等服务,实现区域交通协同管控；

行业API：沉淀标准化的交通行业API,如路网、车流、信号灯等,促进数据流通。

### 3.5业务中台层

面向城市交通管理、公路运营、交通执法等领域构建行业解决方案

提供如智能网联车车载系统、自适应信号控制系统等行业专属应用套件

支持基于通用和基础的微服务、低代码的业务模块灵活组装,实现应用的快速开发和发布

共享各类行业模型、知识、专家经验等,形成行业知识体系和最佳实践库

### 3.6运营中台层

全栈性能观测：采集系统状态数据，实现系统的全局实时监控；

故障智能诊断：利用LLM、因果推断等技术快速根因定位,实现秒级故障自愈；

资源智能管理：智能预测资源需求变化,实现基础设施的弹性扩缩和提前预置。

### 3.7大规模分布式实时数据总线

基于车端大规模数据吞吐和低延迟通讯技术，构建全域贯通的数据总线，并被集成于操作系统的各个层面。

主要特征：

高吞吐,低延迟，插件化，配置化，兼容TCP,UDP,MQTT,KAFKA,WebSocket,RPC等传统通讯方式。

丰富的QoS支持，支持持久化，自动路库，动态数据类型，支持常用协议的转译。

### 3.8 集成中台总控层

集成中台将以上6+1和基础设施再次进行按用户的视角进行重新抽象和组织管理，在集成中台中可以进行节点管理，应用管理，数据管理，安全和权限管理，系统和应用的监控运维。

在此基础上，将实现交通应用的场景编排，比如常见的信控应用场景，v2x云控场景进行抽象，规范，标准化后进行统一编排，部署，监控运维。

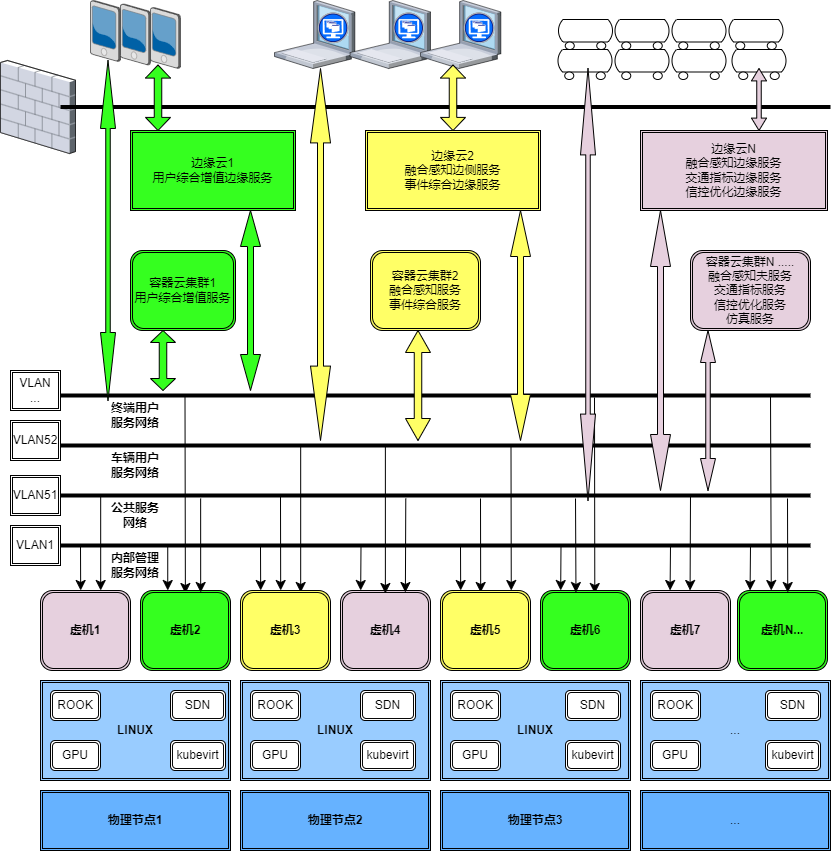
同时，我们将对不同任务进行分类识别，并应用不同的调度策略。比如支持：定时任务，突发任务，周期任务等。并逐步收集应用数据，识别应用和资源间的关系，逐步由常规的人工识别调度，转为由AI识别并调度，从而构建自主式的资源调度平台。

## 四、技术框架和实现路径

集成中台整体设计思路为通过分布式实时总线，在数据层面重构整个应用体系，充分吸收云原生技术，整合虚机，容器，分布式存储，分布式数据库，大数据技术（数仓和数据湖）， AI数据处理，训练和推理等技术，从而在应用和数据层面对交通应用场景提供系统，配套，完整的支持，同时又能不断的融合新技术。

实验室集成中台的构建模式为（k8s on k8s） + （边缘节点 on 容器）+ 集成中台 = 云边一体化自主式集成中台；

4.1 其整体部署架构如图：



### 4.1.1 不可变基础设施层

首先通过在云端构建容器云，一般在物理机上构建，同一层中构建SDN（虚拟化网络），分布式存储，并被容器云统一纳管；加上在边缘测的边缘设备，此为第一层，不可变基础设施层

### 4.1.2 按需组织的资源层

然后在容器云中开设虚机，分配网段资源，分配存储资源；在物理机中（主要为边缘MEC）通过容器化方式，划分计算和存储资源，这要可以按照业务和客户种类，划分各自的资源池，比如我们划分了六个大的资源池：（AI）算法中台资源池；服务中台资源池；数据中台资源池（又分为离线和在线资源池）；运营中台资源池；业务中台资源池；

在这六大资源池中，再各自部署其服务软件，由于内容较多，在此就不在一一列举。其主要为在虚机中，以容器云方式部署，这样会有更好的隔离和安全性。

同时，这些一资源池可通过集成中台进行统一的监控和运维，保证全局的可观测性。

数据中台可以为用户提供一个界面，给最终用户使用，所以数据中台又在集成中台上有一层。

### 4.1.3 专业化的服务层

此层对应传统BaaS层，主要提供容器化的各种公共服务，比如用户中心，消息中心，证书中心等等；

同时总线服务和集成中台也落地于此层；

### 4.1.4 按需定制的业务层

此层包括用户定制的BaaS层和SaaS层，主要各种定制开发服务，私有业务服务层以及用户业务应用层，比如各类交通场景应用，v2x应用，交通信控应用等应用。

用户可以通过集成中台进行统一的应用管理，也叫做业务编排。同时集成中台提供标准的对应业务的编排模板，比如常见的v2x应用，信控优化应用等。通过统一的编排，用户很容易的实现在云边端的应用部署。

结合全局感知（可观测性）和AI的预测，可以实现动态编排和应用调整。集成中台提供定时策略，周期策略，突发策略等特征编排模板给用户，依靠模板可以大量降低用户工作量，并实现交通应用的自主化。

## 4.2 通过五种方式来实现五个层面统一

### 4.2.1 功能丰富和快速实现的统一

由于把各种常用接口和底层资源都抽象为总线数据，应用只需要关注如何获得自己需要的数据和把结果送到总线便可，从而实现业务开发的统一便捷；绝大多数据应用无需再关心数据库，大数据，系统层，AI等负载繁复的调用，只需要实现自己的业务功能便可，从而实现简化和加速应用开发。

同时集成中台逐步沉淀和标准化各种通用的公共服务和专用的业务服务，从而避免多次造车轮资源和时间浪费；

### 4.2.2 灵活开放和系统稳健的统一

通过将各种开源软件集成到总线上，从而保证平台的灵活开放而有可以避免各自引入其它系统带来的不稳定和补一致性。

比如系统集成中台就是使用总线技术而集成容器和容器云到总线上，从而实现云边一体化的统一纳管。

### 4.2.3 专业集成和方便使用的统一

通过数据总线，很容易打通或者构建自己的数据中台，由于天然的分布式特性，可以按需构建各种总线，并且吞吐基本不受限，低延迟等特点，和容易抽象和屏蔽大数据平台，在总线上进行大规模数据处理；

在数据总线和集成中台的基础上，很容易构建自己独特的业务中台，而且也是统一的总线接口，从而减低使用难度。

### 4.2.4 专家系统和人工智能的统一

可以方便的挂接各类型的算法模块到总线上，这样可以很容易的实现算法和系统平台解耦，算法可以很容易的开发，测试和运行。

同时支持专业算法和AI类算法，并按需分配资源和运行，使用者基本无感算法类型差异。

集成平台可以对接AI平台，同时可以直接部署和运行各类算法型应用；

### 4.2.5 方便快捷和安全可控的统一

由于通过在底层数据总线的证书认证和加密技术，实现分布式的ACL控制，再通过绑定用户和证书，节点和证书的方式，从而实现全平台的通讯和基于用户角色的认证授权，这样保证用户可以很方便的访问有权数据，同时禁止非法访问。同时保证用户和数据隐私安全。

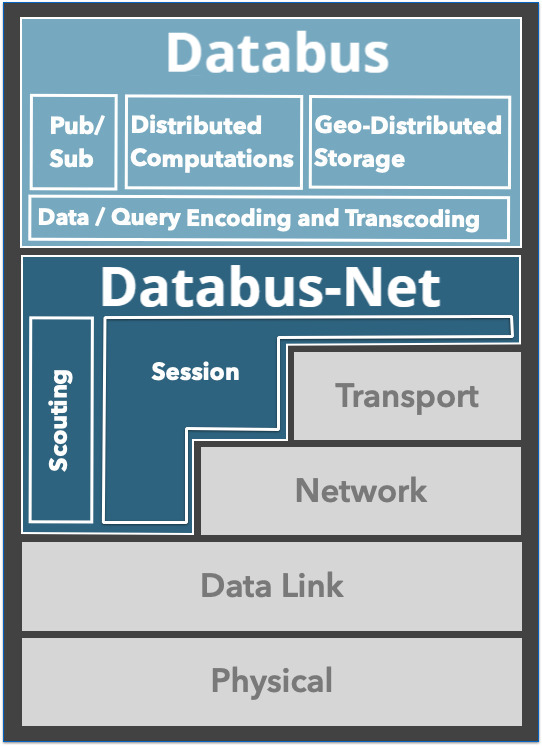
## **4.3 各层实现概要设计**

### 4.3.1. 数据总线架构概述

Databus是一个创新性的分布式数据传输系统，采用先进的微服务架构和事件驱动设计模式。它的核心是提供一个统一的数据空间抽象层，实现高效的跨平台、跨语言数据交换。系统采用零拷贝技术和发布/订阅模式，显著提升了数据传输效率。在性能方面，通过异步I/O和内存映射等技术，实现了微秒级的传输延迟和GB级的吞吐量。特别适用于对实时性要求高、数据量大的场景，如工业物联网、智能制造、金融交易等领域。同时，系统提供完整的安全机制，包括身份认证、访问控制、数据加密等多层次安全保护，确保数据传输的安全性和可靠性。

Databus 是一个创新的数据分发中间件，专为IoT和边缘计算场景设计。它采用了完全去中心化的架构，通过统一的抽象层来处理异构系统间的数据交换。在性能方面，其零拷贝设计实现了微秒级的端到端延迟，在基准测试中，在千兆网络环境下可实现35μs的往返延迟，单节点吞吐量可达到10Gbps。系统采用Rust语言开发核心组件，确保了内存安全性和高性能。支持包括TCP、UDP和共享内存在内的多种传输协议，并可无缝切换。其独特的位置透明寻址机制允许应用程序无需关心数据的物理位置，显著简化了分布式系统的开发复杂度。为了支持不同的应用场景，系统提供了C、C++、Python、Java、Rust等多语言绑定。

总线提供两级API：



**Databus-Net**

实现能够在数据链路，网络或传输层之上运行的网络层。

Databus-Net支持点对点和路由通信，提供了允许Pub/Sub(Push)通信以及Query/Reply (Pull) 通信的关键原语。它支持分段和有序可靠交付，并为 Discovery(ROS2 Discovery Serivice)提供了可插入的侦查抽象。

Databus-Net定义并建立在会话协议的基础上，该会话协议为有序尽力而为(best effort)和具有不受限制的MTU的可靠信道提供了抽象，而MTU与底层无关。

Databus-Net层仅关心数据传输，不关心数据内容或存储数据。

提供的原语：

* **Write**：将实时数据推送给相对应的订阅者。
* **Subscribe**：订阅实时数据。
* **Query**：从匹配的查询对象中查询数据。
* **Queryable**：能够响应查询的实体。

**Databus**

Databus层为Pub/Sub和分布式查询提供了高级API，以更简单和更面向数据的方式提供与Databus-Net API相同的抽象，并提供所有构建块来创建分布式存储。Databus层知道数据内容，并且可以应用基于内容的过滤和代码转换。

它处理数据表示编码和代码转换，并提供地理分布式存储和分布式计算值的实现。 Databus支持一系列数据编码，例如JSON，Protobuf，Idl，Asn等，以及跨支持格式的转码。 Databus层还提供了存储后端插件API，以简化第三方存储技术的集成。 当前支持的存储后端是Memory，MySQL，MariaDB，PostgreSQL，SQLite和InfluxDB。默认情况下，地理分布式存储在最终一致性下工作。

提供的原语：

* **Put**：将实时数据推送到匹配的订阅者和存储。 （相当于Databus-Net write）
* **Subscribe**：实时数据的订阅者。 （相当于Databus-Net订阅）
* **Get**：从匹配的存储和演算中获取数据。 （相当于Databus-Net查询）
* **Storage**：一个Databus-Net 订阅者侦听要存储的实时数据，以及一个Databus-Net可查询以响应匹配的get请求的组合。
* **eval**：能够响应请求的实体。 通常用于按需提供数据或构建RPC系统。 （相当于Databus-Net可查询）

#### 4.3.1.1 核心设计理念

##### 4.3.1.1.1 统一数据空间

统一数据空间是Databus的核心抽象概念，它提供了一个全局一致的数据视图和访问机制。系统使用分层的键值对结构组织数据，支持复杂的数据模型和查询操作。键的设计采用层次化的命名空间管理，类似于文件系统的路径结构，便于数据的组织和访问控制。值可以是任意类型的数据，支持基础数据类型、复杂对象、二进制流等。此外，统一数据空间还提供了强大的元数据管理功能，支持数据版本控制、生命周期管理、数据血缘追踪等高级特性，为上层应用提供丰富的数据管理能力。

在Databus中，统一数据空间采用了类似URI的层次化键值结构，例如："/company/department/device1/temperature"。键空间支持通配符匹配（如"/company/\*/temperature"）和表达式过滤。数据值支持任意类型，包括原始数据类型、结构化数据（JSON、Protocol Buffers等）和二进制流。系统实现了一个分布式键值存储引擎，使用内存映射文件技术来优化大数据集的处理性能。权限控制基于细粒度的访问控制列表（ACL），支持按路径和操作类型进行访问控制。在实际部署中，该设计在处理每秒数百万个数据点的场景下表现出色，存储引擎的写入延迟通常保持在微秒级别。

##### 4.3.1.1.2 去中心化架构

去中心化架构是Databus的另一个核心特性，采用分布式设计确保系统的高可用性和可扩展性。节点间采用P2P（点对点）通信模型，每个节点既可以作为数据提供者，也可以作为数据消费者。系统使用基于DHT（分布式哈希表）的路由算法，确保数据能够快速准确地到达目标节点。动态发现机制支持节点的自动发现和健康检查，提供故障自愈能力。负载均衡机制确保系统资源的合理利用，防止单点过载。数据分片和复制策略提供了数据的高可用性和容灾能力。

Databus采用基于Chord算法的改进版DHT（分布式哈希表）实现路由寻址，每个节点维护一个最多包含O(log N)个路由表项的路由表。节点发现使用组播和基于gossip协议的混合机制，支持跨网段自动发现。在广域网部署中，系统使用动态隧道技术处理NAT穿透问题。路由决策考虑网络延迟、带宽和节点负载等多个因素，通过动态权重算法选择最优路径。实测表明，在包含1000个节点的网络中，节点发现和路由表收敛时间通常在3秒内完成。系统还实现了分布式一致性协议，确保在网络分区情况下的数据一致性。

##### 4.3.1.1.3 性能优化

性能优化贯穿Databus的整个设计过程，从底层通信到上层应用都进行了全方位的优化。采用零拷贝技术减少数据传输过程中的内存拷贝次数，显著提升传输效率。使用内存池和对象池技术减少内存分配和回收的开销。实现了智能的批处理机制，可以自适应地合并小数据包，优化网络传输效率。采用多级缓存策略，包括进程内缓存、分布式缓存等，提高热点数据的访问速度。异步处理机制确保高并发场景下的系统吞吐量。

零拷贝实现基于操作系统的sendfile/splice机制和用户态协议栈（如DPDK），在数据传输过程中完全避免了用户空间和内核空间的数据拷贝。异步通信框架采用基于tokio的事件驱动模型，支持高并发连接处理。数据批处理机制会智能合并小数据包，当累积数据大小达到网络MTU大小或等待时间超过配置阈值（默认100微秒）时触发发送。QoS策略支持可靠传输和尽力而为两种模式，可靠传输模式使用NACK机制进行丢包恢复，同时实现了基于接收方窗口的流量控制。在实际测试中，在千兆网络环境下，单连接可实现接近线速（约900Mbps）的持续传输性能，同时CPU使用率保持在较低水平（通常小于10%）。

#### 4.3.1.3. 核心组件

##### 4.3.1.3.1 Router（路由器）

Router是Databus的核心组件，负责数据的智能路由和转发。它维护完整的网络拓扑信息，实时更新节点状态和连接关系。路由算法支持多种策略，包括最短路径、最低延迟、带宽优先等，可根据实际需求动态选择。支持跨网络段的数据传输，自动处理网络地址转换和协议转换。实现了高效的订阅关系管理，使用布隆过滤器等技术优化订阅匹配性能。提供流量控制和拥塞管理机制，确保网络的稳定运行。

##### 4.3.1.3.2 Peer（对等节点）

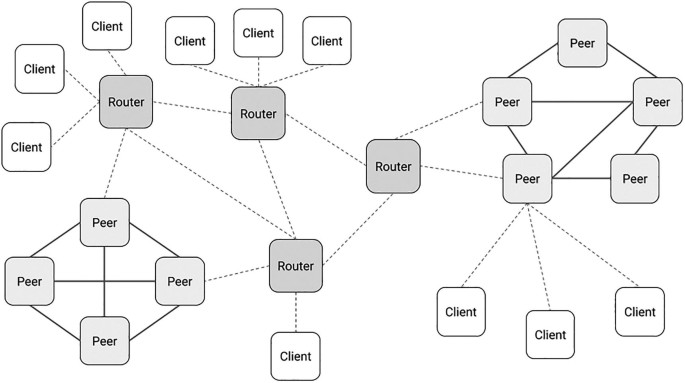
Peer作为系统的基础节点，提供核心的数据交换和存储功能。每个Peer都维护本地缓存，支持数据的快速访问和临时存储。实现了分布式查询引擎，支持复杂的数据检索操作。提供数据压缩和解压缩功能，优化存储空间和传输效率。支持数据的版本控制和冲突解决，确保数据一致性。实现了本地的事件处理引擎，支持数据的实时处理和转换。提供完整的监控接口，便于系统运维和问题诊断。

##### 4.3.1.3.3 Client（客户端）

Client提供简洁易用的API接口，支持多种编程语言和开发环境。实现了智能的连接池管理，优化网络连接资源的使用。支持数据的异步操作和批量处理，提供灵活的错误处理和重试机制。实现了本地的数据缓存和预取功能，优化应用性能。提供丰富的配置选项，支持客户端的个性化定制。包含完整的诊断工具，便于问题排查和性能优化。支持插件机制，方便功能扩展。

##### 4.3.1.3.4三种部署模式

* 对等方（peers）。用于在 对等网络（peer-to-peer）、网状（mesh）拓扑网络和其他节点通信，也可以通过路由器与广泛的系统通信。
* 客户（clients）。用于连接单个路由器（或点）来和系统其他部分通信。
* 路由器（routers）。用于在 client和peers之间、集团拓扑和网状拓扑网络之间路由数据。



#### 4.3.1.4. 关键特性

##### 4.3.1.4.1 通信模式

Databus支持多种灵活的通信模式，适应不同的应用场景需求。发布/订阅模式支持异步的消息通信，实现了消息的定时发送、延迟发送、条件触发等高级特性。查询/响应模式提供同步的数据访问能力，支持复杂的查询条件和结果过滤。推/拉模式结合了主动推送和按需获取的优点，优化数据传输效率。点对点通信模式支持节点间的直接数据交换，减少网络延迟。所有通信模式都支持QoS策略，确保服务质量。

##### 4.3.1.4.2 数据管理

数据管理功能覆盖数据的全生命周期，提供完整的数据治理能力。实时数据流处理支持流式计算和实时分析。历史数据存储实现了数据的持久化和归档。数据索引机制支持高效的检索和查询。数据压缩和解压缩优化存储空间和传输效率。数据加密确保安全性，支持细粒度的访问控制。数据备份和恢复机制保证数据的可靠性。提供完整的数据监控和审计功能。

##### 4.3.1.4.3 网络优化

网络优化是系统性能的关键因素，Databus在这方面进行了深入的优化。智能路由选择基于网络状况动态调整传输路径。负载均衡确保网络资源的合理利用。链路聚合提高带宽利用率。网络分片减少广播风暴。QoS策略保证关键业务的服务质量。网络监控实时掌握系统状态。自适应流控防止网络拥塞。错误重传确保数据可靠传输。

#### 4.3.1.5. 技术优势

##### 4.3.1.5.1 高性能

高性能是Databus的核心竞争力之一。通过零拷贝技术显著减少CPU和内存开销。异步处理机制提高系统并发能力。内存池技术优化资源利用。多级缓存加速数据访问。智能批处理提升传输效率。硬件加速支持提高处理速度。负载均衡确保资源充分利用。性能监控和调优工具助力系统优化。

##### 4.3.1.5.2 可靠性

可靠性设计贯穿系统各个层面，确保数据传输和存储的可靠性。故障检测和自动恢复机制处理各类异常情况。数据一致性协议保证分布式环境下的数据准确性。多副本策略提供数据冗余保护。事务机制确保操作的原子性。日志机制便于问题追踪和恢复。监控告警及时发现并处理问题。灾难恢复方案保障业务连续性。

##### 4.3.1.5.3 扩展性

系统的扩展性体现在多个维度。水平扩展支持动态增加节点，提升系统容量。垂直扩展通过升级硬件提高单节点性能。功能扩展通过插件机制增加新特性。协议扩展支持新的通信协议。存储扩展支持多种存储引擎。安全扩展支持新的安全机制。监控扩展支持自定义监控指标。接口扩展支持新的编程语言和框架。

##### 4.3.1.5.4 支持协议集

协议类型

a. 传输协议

- tcp：基础传输控制协议

- udp：用户数据报协议，适用于快速传输场景

- tls：传输层安全协议

- 共享内存：本地进程间通信协议

b. 服务协议

- DDS：数据分发服务协议，支持实时数据通信

- REST：表述性状态传输协议，基于HTTP的API设计规范

- zenoh：分布式消息队列协议，支持发布订阅模式

- mqtt/mqtts：轻量级物联网消息(及加密)传输协议

- quic：快速UDP互联网连接协议

c. 扩展协议

- WS：WebSocket协议，支持全双工通信

- WSS：安全WebSocket协议，基于TLS加密

##### 4.3.1.5.5. 支持编解码集

a. 数据格式

- Protobuf：Google的开源序列化框架

- JSON：轻量级数据交换格式

- ASN：抽象语法标记

- FastCDR：快速序列化协议

- CANP：CAN协议编解码格式

b. 特性说明

1. Protobuf特性：

- 二进制格式，高效压缩

- 跨语言支持

- 向前向后兼容

- 自动代码生成

2. JSON特性：

- 文本格式，可读性强

- 语言无关性

- 灵活的数据结构

- 广泛的工具支持

3. ASN特性：

- 标准化的编码规则

- 支持复杂数据类型

- 严格的类型定义

- 适用于通信协议

4. FastCDR特性：

- 高性能序列化

- 低延迟处理

- 内存效率高

- 适合实时系统

5. CANP特性：

- 专用于CAN总线

- 紧凑的数据格式

- 实时性能好

- 支持标准帧和扩展帧

c 使用场景

1. 实时数据传输：FastCDR、CANP

2. Web应用交互：JSON、WebSocket

3. 跨平台服务：Protobuf、REST

4. 工业现场总线：CANP、DDS

5. 物联网通信：MQTT、UDP

6. 安全通信：TLS、WSS

d 性能考虑

1. 编码效率：

- 二进制协议：Protobuf、FastCDR

- 文本协议：JSON、REST

2. 传输效率：

- 高效传输：UDP、QUIC

- 可靠传输：TCP、TLS

3. 资源占用：

- 轻量级：MQTT、UDP

- 重量级：DDS、WebSocket

4. 实时性能：

- 高实时：CANP、FastCDR

- 普通实时：其他协议

这些协议和编解码格式的选择需要根据具体应用场景、性能需求和系统架构来确定，通常会组合使用多种协议来满足不同层面的需求。

#### 4.3.1.6. 应用场景

##### 4.3.1.6.1 边缘计算

在边缘计算场景中，Databus发挥着关键作用。实现设备数据的高效采集和预处理，支持本地的实时分析和决策。边缘节点间的协同计算提高处理效率。数据过滤和压缩优化传输效率。本地缓存减少云端依赖。安全机制保护边缘数据。智能路由优化网络资源。故障自愈确保服务可用性。监控管理实现边缘设备的集中管理。

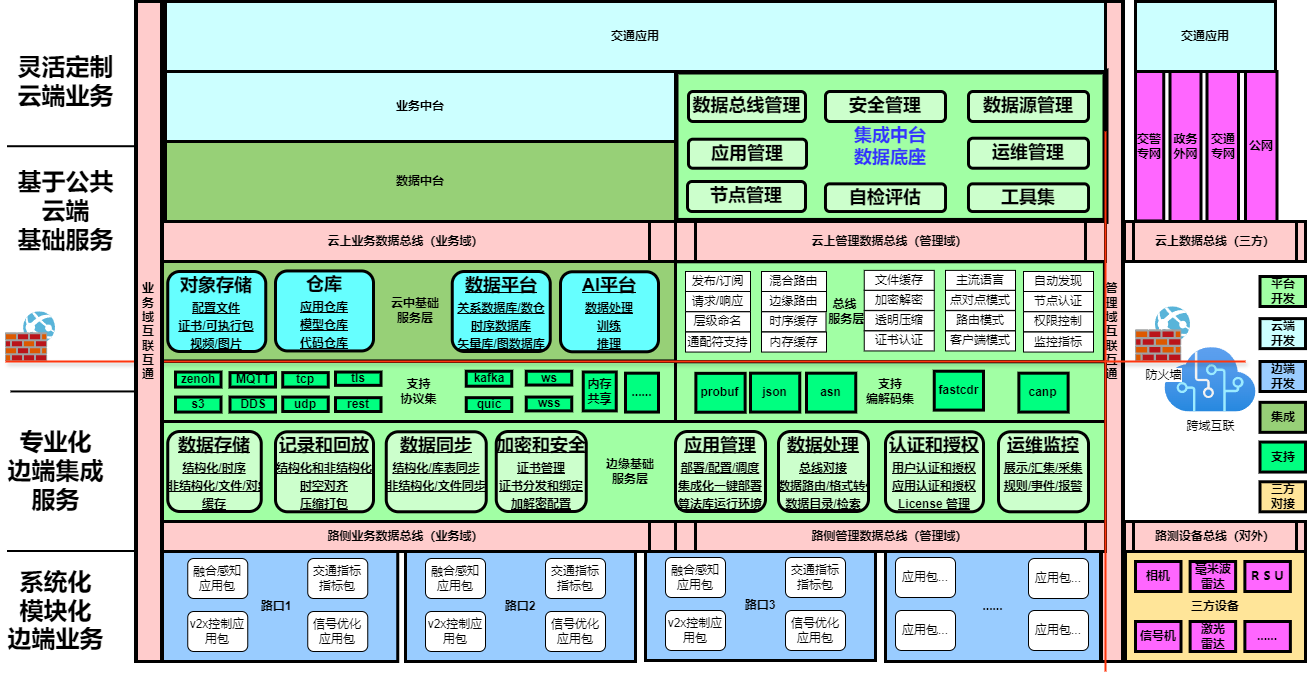
##### 4.3.1.6.2 物联网系统

在物联网领域，Databus提供了完整的数据采集和处理解决方案。支持海量设备的接入和管理，实现数据的实时采集和处理。设备状态监控确保系统稳定运行。控制指令可靠下发保证设备可控。数据分析提供决策支持。协议适配支持多种物联网协议。边缘计算减少云端压力。安全机制保护物联网安全。运维工具简化系统管理。

##### 4.3.1.6.3 分布式系统

在分布式系统中，Databus作为关键的数据基础设施，支持复杂的业务场景。微服务通信提供可靠的服务间数据交换。数据同步确保分布式环境下的数据一致性。事件驱动架构支持松耦合的系统集成。分布式缓存提高数据访问效率。事务处理确保操作的原子性。服务发现简化系统配置。监控告警保障系统可用性。运维工具辅助系统管理。

### 4.3.2 集成中台概要设计



集成中台架构示意图

#### 4.3.2.1. 节点管理模块

节点管理模块作为集成中台的基础设施层，承担着分布式系统中所有节点的生命周期管理责任。该模块通过建立统一的管理接口和标准化的通信协议，实现了节点的自动发现、注册、配置管理、状态监控和资源调度等核心功能。在架构设计上采用分布式设计理念，支持节点的动态扩展和弹性伸缩。通过引入心跳机制和健康检查，实时监控节点状态，并具备故障自动转移能力。模块还实现了跨地域节点管理，通过建立安全通道确保节点间通信的可靠性和数据传输的安全性。

##### 4.3.2.1.1 节点基础管理

节点基础管理子模块提供节点生命周期的完整管理能力，包括节点注册、配置、启停等基础操作。通过标准化的节点注册流程，实现新节点的自动发现和接入。配置管理采用中心化设计，支持配置的统一管理和动态下发。节点状态管理实现了实时监控和故障检测，通过心跳机制和健康检查确保节点的稳定运行。资源管理功能对节点的计算、存储、网络等资源进行统一管理，支持资源的动态分配和回收。子模块还提供了节点级别的性能优化和故障诊断能力。

##### 4.3.2.1.2 节点文件管理

节点文件管理子模块负责处理节点间的文件传输、存储和同步需求。该子模块实现了高效的文件传输机制，支持大文件传输和断点续传。文件同步功能确保配置文件、日志文件等关键数据在节点间保持一致。存储管理采用分布式设计，支持多种存储介质和存储策略。文件版本控制功能实现了文件的版本管理和历史追溯。子模块还提供了文件压缩、加密等增值功能，确保文件传输和存储的效率与安全性。

##### 4.3.2.1.3 节点监控调试

节点监控调试子模块提供全方位的监控和调试能力。通过采集系统层面的CPU、内存、磁盘IO等基础指标，实现对节点性能的实时监控。进程监控功能关注节点上运行的关键进程状态，支持进程级别的操作和调试。网络监控实时采集网络连接状态和性能指标。调试功能支持远程调试和日志实时查看，方便运维人员进行问题诊断和处理。子模块还集成了性能分析工具，支持系统性能瓶颈分析和优化建议。

##### 4.3.2.2. 应用管理模块

应用管理模块是实现应用全生命周期管理的核心组件，提供了从应用开发、测试到部署、运维的完整解决方案。该模块通过标准化的应用封装和管理流程，支持多种类型应用的统一管理，包括传统应用、微服务应用和容器化应用。采用中心化的配置管理和分布式的部署架构，确保应用配置的一致性和部署的灵活性。模块集成了完整的监控体系，实时采集应用运行状态和性能指标，支持异常检测和告警。

##### 4.3.2.2.1 应用配置管理

应用配置管理子模块负责应用的参数配置和环境管理。采用配置中心化设计，实现配置的统一管理、版本控制和动态下发。支持多环境配置管理，方便开发、测试、生产环境的配置隔离。配置变更管理实现了变更审核、灰度发布和回滚机制。配置加密功能确保敏感配置的安全性。子模块还提供配置依赖分析和影响评估能力，降低配置变更风险。

##### 4.3.2.2.2 应用部署监控

应用部署监控子模块提供应用部署的全流程管理和监控能力。支持应用包的版本管理和部署策略配置，实现自动化部署和回滚。部署过程监控实时展示部署进度和状态，支持部署过程的暂停、继续和终止操作。部署验证功能自动检查部署结果，确保应用正常运行。子模块还包含部署历史记录和对比分析功能，方便追踪部署变更。

##### 4.3.2.2.3 应用运行监控

应用运行监控子模块构建了全方位的应用监控体系。通过轻量级的监控探针，实时采集应用的运行状态、性能指标和业务指标。性能监控覆盖CPU使用率、内存占用、响应时间等关键指标。异常监控支持自定义异常规则，实现异常的实时告警。业务监控通过埋点机制采集业务指标，支持业务趋势分析。子模块还提供了丰富的可视化展示和报表功能，方便运维人员掌握应用运行状况。

##### 4.3.2.3. 数据总线管理模块

数据总线管理模块是整个平台的核心数据流转中枢，负责实现数据的统一接入、传输、转换和分发。采用分布式消息队列架构，支持高并发数据处理和实时数据流转。模块实现了数据格式标准化和协议适配，确保异构系统间的数据互通。通过主题管理和数据域划分，实现了数据的逻辑隔离和精细化管理。模块还集成了数据质量控制和监控功能，保证数据传输的准确性和可靠性。

##### 4.3.2.3.1 数据域管理

数据域管理子模块实现了数据的逻辑分区和管理。通过定义数据域，将数据按业务属性、安全级别等维度进行分类管理。数据域间的访问控制确保数据隔离和安全共享。支持数据域的动态创建、配置和扩展，满足业务发展需求。数据域监控功能实时展示数据流转状况和资源使用情况。子模块还提供了数据域间的数据映射和转换能力，支持跨域数据交换。

##### 4.3.2.3.2 数据集成管理

数据集成管理子模块提供了统一的数据接入和处理框架。支持多种数据源的接入适配，包括数据库、消息队列、文件系统等。实现了数据清洗、转换、加工的标准化处理流程。数据质量控制贯穿整个集成过程，通过规则引擎实现数据校验和处理。提供了数据集成任务的调度管理和监控功能。子模块还支持实时数据同步和批量数据处理，满足不同场景的数据集成需求。

##### 4.3.2.3.3 协议及序列化管理

协议及序列化管理子模块负责数据传输协议的适配和转换。支持主流的数据交换协议，如HTTP、WebSocket、MQTT等。实现了多种序列化格式的转换，包括JSON、XML、Protobuf等。协议适配层提供统一的接口规范，简化协议转换开发。性能优化机制确保协议转换的高效性。子模块还包含协议安全性验证和性能监控功能，保证数据传输的安全和效率。

#### 4.3.2.4. 数据源管理模块

数据源管理模块是实现数据接入和处理的基础组件，提供了统一的数据源接入框架和标准化的数据处理流程。模块支持多种类型数据源的管理，包括关系型数据库、非关系型数据库、文件系统等。通过统一的接口规范和数据模型，实现了异构数据源的统一管理。模块还提供了数据质量控制、数据安全防护和数据生命周期管理等核心功能，确保数据的可靠性和安全性。

##### 4.3.2.4.1 结构化数据管理

结构化数据管理子模块专注于关系型数据的处理和管理。实现了主流关系型数据库的接入适配，支持Oracle、MySQL、PostgreSQL等数据源的统一管理。提供数据模型管理功能，实现表结构的自动发现和同步。数据同步功能支持实时同步和批量同步两种模式，并提供断点续传能力。查询优化功能通过智能分析提供最优查询方案。子模块还包含数据版本管理和变更追踪功能，确保数据的一致性和可追溯性。

##### 4.3.2.4.2 非结构化数据管理

非结构化数据管理子模块处理文档、图片、视频等多媒体数据。实现了文件存储系统的统一管理，支持本地存储和云存储的混合部署。文件处理功能支持格式转换、压缩、提取等操作。元数据管理功能为非结构化数据提供结构化描述和检索能力。存储优化机制实现了数据的分级存储和冷热分离。子模块还提供了文件分发和同步功能，确保大规模文件的高效传输。

#### 4.3.2.5. 安全管理模块

安全管理模块构建了全面的安全防护体系，从身份认证、访问控制、数据加密等多个维度确保平台的安全性。采用PKI体系实现身份认证和传输加密，RBAC模型实现细粒度的权限控制。模块支持多种安全策略的配置和管理，包括密码策略、访问策略、审计策略等。通过安全审计和监控，实现了安全事件的实时检测和响应。

##### 4.3.2.5.1 身份认证与授权

身份认证与授权子模块实现统一的认证和授权管理。支持多种认证方式，包括用户名密码、证书、令牌等。集成了单点登录(SSO)功能，提供统一的用户认证入口。授权管理基于RBAC模型，支持角色的继承和组合。权限管理支持功能权限和数据权限的精细化控制。子模块还提供了认证授权过程的审计日志，记录关键操作痕迹。

##### 4.3.2.5.2 数据安全防护

数据安全防护子模块负责数据全生命周期的安全防护。实现了传输加密、存储加密和字段级加密等多层次加密机制。数据脱敏功能支持静态脱敏和动态脱敏，保护敏感信息。数据防泄漏系统(DLP)实现了敏感数据的识别和防护。访问控制功能确保数据访问的合规性。子模块还包含数据备份和恢复机制，保障数据安全。

#### 4.3.2.6. 运维管理模块

运维管理模块提供全面的运维能力，确保平台的稳定运行。模块实现了多维度的监控体系，覆盖系统资源、应用性能、业务指标等方面。告警管理系统支持多级别告警策略配置和智能告警分析。通过自动化运维工具，简化日常运维操作。资源管理功能实现了IT资源的可视化管理和优化配置。

运维管理模块同时支持采集各种应用的业务指标，这些要求应用开发和暴露其业业主指标，通过运维采集软件统一采集和监控。

##### 4.3.2.6.1 系统监控

系统监控子模块建立了全面的监控体系。实时采集系统层面的性能指标，包括CPU、内存、磁盘、网络等。应用监控功能关注应用的运行状态和性能表现。业务监控通过自定义指标实现业务层面的监控。告警系统支持多渠道告警通知和告警升级。子模块还提供了丰富的可视化展示，直观展现系统运行状况。

##### 4.3.2.6.2 运维自动化

运维自动化子模块提供了一系列自动化工具和脚本。作业调度系统支持复杂运维任务的自动化执行。变更管理功能实现了变更流程的规范化和自动化。配置管理实现了配置的自动化下发和验证。自动化测试工具支持系统功能和性能的自动化测试。子模块还包含问题自动诊断和修复能力，提高运维效率。

#### 4.3.2.7. 工具集模块

工具集模块提供了全面的开发和运维工具支持。包括开发工具、测试工具、部署工具和运维工具等多个类别。工具采用插件式架构，支持功能的灵活扩展。提供统一的工具管理界面，简化工具的使用和维护。通过标准化的接口规范，确保工具间的互操作性。模块还包含完整的使用文档和示例，方便用户快速掌握工具的使用方法。

以上设计方案通过模块化的架构设计，实现了系统各个功能模块的解耦和独立演进，同时通过统一的数据总线确保了模块间的有效协同。每个模块都具备完整的功能体系和清晰的职责边界，为整个平台的可扩展性和可维护性提供了保障。

#### 4.3.2.8集成中台边缘侧功能概要设计



##### 4.3.2.8.1. 设计总览

基于图中八大核心功能模块，边缘侧设计需要重点考虑数据采集、本地存储、安全防护和轻量级处理能力。设计将遵循"边缘智能、本地闭环、按需上云"的原则。

这些基础能力基本以数据总线的形式向边缘和云端提供服务。

##### 4.3.2.8.2. 功能模块设计

###### 4.3.2.8.2.1. 数据存储

结构化数据

功能职责：

- 支持总线上数据缓存

- 支持存到内存，文件和时序数据库

关键特性：

- 透明特性

- 支持缓存和查询

- 自带加密解密

非结构化文件存储

功能职责：

- 处理文档、图片等大文件存储

- 支持对象存储和本地文件存储

关键特性：

- 可以通过通过绑定topic直接上传文件到对象存储

- 可以通过动态监控目录的方式，实现文件同步到对象存储

缓存

功能职责：

- 提供高速数据缓存服务（轻量级redis）

###### 4.3.2.8.2.2. 记录和回放

结构化和非结构化数据存储

功能职责：

- 统一记录多类型数据

- 数据通过总线后进行统一的时空对齐，并整体保存

- 生成文件索引和关键点信息

- 支持文件播放，点播，快进

关键特性：

- 支持多种数据格式的统一存储

- 实现数据实时记录和批量导入

- 提供数据标签化管理能力

- 支持数据索引自动建立

- 实现数据关联分析功能

历史回放

功能职责：

- 提供历史数据查询

- 支持时间序列回放

- 管理回放速度控制

- 实现回放场景还原

关键特性：

- 支持多维度时间轴回放

- 实现数据回放速率动态调节

- 提供回放断点续播功能

- 支持多线程并发回放

- 内置回放性能优化机制

- 支持孪生平台和总线上统一回放

压缩打包

功能职责：

- 为总线上数据提供透明的压缩和解压缩

关键特性：

- 支持配置方式打开和关闭

###### 4.3.2.8.2.3. 数据同步

结构化库表同步

功能职责：

- 管理同步数据源

- 管理数据库表级同步

- 控制同步任务调度

- 监控同步状态管理

关键特性：

- 支持全量和增量同步策略

- 实现表结构映射

- 提供事务一致性保证机制

- 内置同步冲突检测和解决

- 支持自定义同步规则配置

非结构化文件同步

功能职责：

- 主要为本地文件同步到对象存储

- 处理大文件同步传输

- 实现目录结构同步

关键特性：

- 支持断点续传和并行传输

- 提供文件完整性校验机制

- 支持带宽限流和优先级控制

- 内置文件同步状态监控

###### 4.3.2.8.2.4. 加密和安全

证书管理

功能职责：

- 管理节点数字证书生命周期

- 处理证书申请和更新

- 控制证书存储安全

- 实现证书验证服务

- 实现证书和总线节点的绑定解绑，从而保证通讯安全

- 实现基于证书的ACL控制

关键特性：

- 实现证书自动续期机制

- 提供证书吊销列表管理

- 支持证书密钥保护

- 内置证书有效性监控

证书分发和授权

功能职责：

- 处理证书分发流程

- 管理授权策略控制

- 实现证书部署自动化

- 监控证书使用状态

关键特性：

- 支持批量证书分发机制

- 实现基于角色的授权控制

- 提供证书使用审计功能

- 支持证书分发加密传输

- 内置分发状态追踪能力

加密配置

功能职责：

- 支持基于证书解密解密（x509）

- 控制密钥生命周期

###### 4.3.2.8.2.5. 应用管理

边缘基础服务层

功能职责：

- 提供基础运行环境

- 管理应用版本，支持应用的配置，部署，运行监控

- 控制应用生命周期

- 实现应用监控告警

关键特性：

- 支持容器化部署管理

- 提供应用健康检查能力

- 支持应用资源管理，扩缩容（容器和容器云应用）

配置管理调度

功能职责：

- 管理应用配置信息

- 处理配置动态更新

- 控制配置版本管理

- 实现配置同步分发

关键特性：

- 支持配置热更新机制（需应用支持）

- 实现配置版本回滚能力

- 提供配置加密存储选项

- 关键配置自检和自测

版本化管理部署

功能职责：

- 管理应用版本发布

- 控制部署流程自动化

- 处理版本回滚操作

- 实现多环境部署管理

关键特性：

- 支持蓝绿部署和灰度发布

- 实现自动化部署流水线

- 提供部署状态实时监控

###### 4.3.2.8.2.6. 数据处理

功能职责：

- 提供数据的协议转换

- 提供数据的编解码转化

- 对数据质量进行评估

- 处理任务依赖关系

关键特性：

- 提供动态配置，数据在总线上动态转换

- 提供资源限流控制

- 支持多任务并行处理

- 内置任务监控和告警

###### 4.3.2.8.2.7. 认证和授权

用户认证和授权

功能职责：

- 管理用户身份和数字认证之间绑定关系

- 控制访问权限分配

应用认证和授权

功能职责：

- 管理节点身份和数字认证之间绑定关系

- 管理应用接入认证

- 控制API访问授权

- 处理应用间通信安全

- 实现总线上数据访问鉴别授权

关键特性：

- 支持x509证书认证

- 实现基于数据总线的网关集成

- 提供服务间认证机制

- 支持权限动态调整

- 内置接口调用监控

License管理

功能职责：

- 管理应用软件许可证（应用数管理）

- 管理总线接入许可证（接入数管理）

- 处理授权期限管理

- 实现授权验证服务

关键特性：

- 支持离线/在线激活

- 实现授权自动续期

- 提供授权使用统计

- 支持批量授权管理

- 内置授权合规检查

###### 4.3.2.8.2.8. 运维监控

展示汇集统计

功能职责：

- 管理监控数据展示

- 处理统计分析报表

- 控制数据可视化

- 实现多维度分析

关键特性：

- 支持实时数据展示

- 实现多样化图表展示

- 提供自定义报表能力

- 支持数据钻取分析

- 内置报表导出功能

监控事件展示

功能职责：

- 管理系统事件采集

- 处理告警事件展示

- 控制事件分级分类

- 实现事件追踪分析

关键特性：

- 支持事件实时推送

- 实现告警级别管理

- 提供事件关联分析

- 支持事件处理流程

- 内置事件统计报表

运维工具管理

功能职责：

- 管理运维工具集成，支持环境检测和节点，应用，总线自检

- 控制工具权限分配

- 处理工具调用日志

- 实现工具版本管理

关键特性：

- 支持工具在线升级

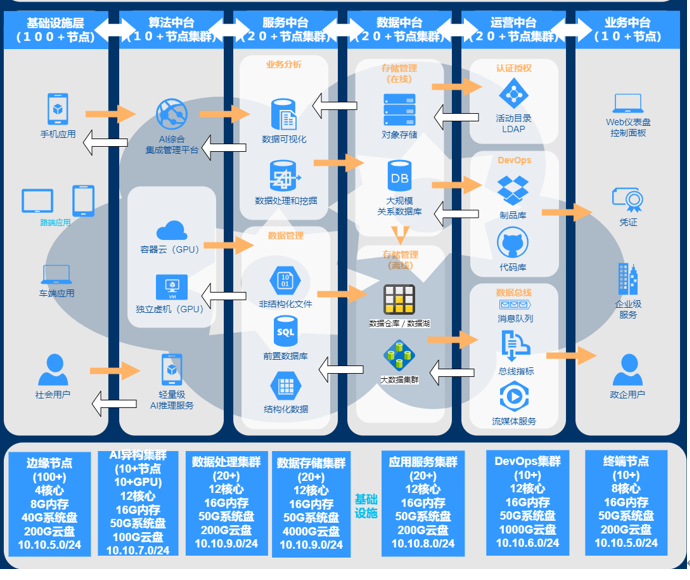
- 实现工具参数配置

- 提供工具使用审计

- 支持自定义工具接入

- 内置工具效能分析

### 4.3.3. 五大中台设计

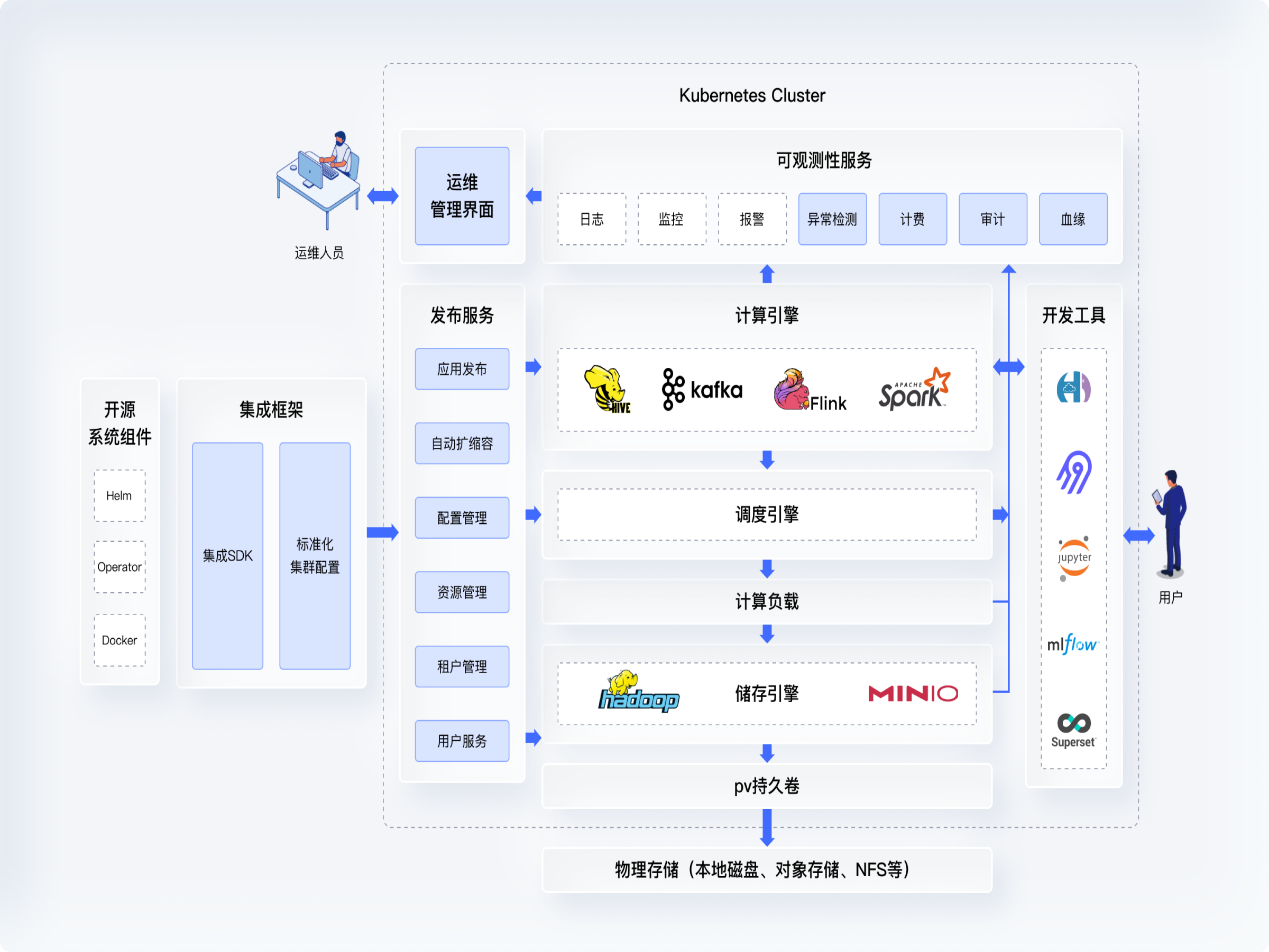


五大中台从基础云上划分各自的虚机，网络和存储资源，互相直接可以按需联通。

#### 4.3.3.1. 数据中台设计

KDP(Kubernetes Data Platform) 提供了一个基于 Kubernetes 的现代化混合云原生数据平台。它能够利用 Kubernetes 的云原生能力来有效地管理数据平台。

##### 4.3.3.1.1. 总体架构

[](https://camo.githubusercontent.com/06405978d2d471592b6a15b388ca7f8815d727d530f2f0ae54979a4fcee21709/68747470733a2f2f6c696e6b74696d652d7075626c69632e6f73732d636e2d71696e6764616f2e616c6979756e63732e636f6d2f6c696e6b74696d652d686f6d65706167652f6b64702f6b64702d61726368692e706e67)

##### 4.3.3.1.2. 技术优势

* 更高效的大数据集群运管：KDP通过标准化流程简化了大数据集群的运维，并提供UI界面进一步提升了部署、升级等操作的效率
* 更高效的大数据组件集成：KDP提供标准化自动化的大数据组件部署和运维，极大地缩短了大数据项目开发和上线时间
* 更高的集群资源利用率：对比传统大数据平台约30%左右的资源利用率，KDP可大幅提升至60%以上

##### 4.3.3.1.3. 技术亮点

* 基于 OAM(Open Application Model) 标准统一应用发布和管理流程，打通各组件之间的配置管理，实现 IaC(Infra as Code)
* 在大数据组件的 Operator 和 Helm Chart 之上创建了统一抽象层，实现发布和运维的标准化和自动化
* 对大数据核心组件进行代码级别的改造以支持 K8s 资源调度，网络及存储体系，并将这些组件的最新版本进行统一集成
* 利用 K8s 的命名空间实现多租户管理，资源隔离，实现按需动态资源配置，并实现了资源使用统计计费组件（企业版）
* 扩展并强化了多租户环境下的安全认证及鉴权机制，采用统一的 Kerberos 安全认证和基于 Ranger 的授权机制（企业版）
* 对计算引擎在云原生形态下的性能进行优化，例如：批流作业统一的 Volcano 调度，解决了 Spark on HDFS 的 Data Locality 问题（企业版）

##### 4.3.3.1.4. 集成框架

KDP基础设施层提供了一套基于 OAM(Open Application Model) 的标准集成流程，将开源大数据组件与统一系统服务对接，形成标准化配置文件。在K8s配置的基础上提供封装，简化大数据组件的配置流程，标准化组件与系统服务及其它组件之间的对接机制。主要包括：

* 提供灵活的发布配置管理，允许用户指定大数据集群（底层载体为命名空间）发布，以及依赖组件的指定
* 如有依赖外部系统，可从系统变量中读取依赖系统访问地址及配置，无需硬编码
* 标准化 ConfigMap/Secret 等系统配置组件的发布和配置方式
* 提供日志，监控报警等运维插件配置，隐藏底层系统细节，自动化，简化配置
* 系统配置的版本管理，对比，回滚等功能

##### 4.3.3.1.5. 发布服务

KDP基础设施层提供了一套应用发布服务，负责大数据组件从配置文件到K8s集群的发布，更新，运维，升级操作。和通用PaaS平台最大的区别在于对大数据负载的支持，租户体系，用户管理以及其资源管理（后三点为企业版特性）的集成。主要包括：

* 实现 IaC(Infra as Code) 的发布运维方式，所有操作以修改配置文件的方式并以控制循环（Control Loop）的方式实现
* 负责有依赖关系的组件之间的发布流程，无需手动处理
* 系统组件和租户体系的集成，确保授权，鉴权，权证的发布以云原生的方式完成（企业版）
* 租户的管理，机构/用户以及其相关资源的生命周期管理（企业版）
* 根据运行负载情况实现动态扩容降容（企业版）

##### 4.3.3.1.6. 可观测性

KDP提供大数据组件以及其执行的工作负载日志，性能/稳定性的指标监控和报警，计费以及审计功能（企业版特性）。和通用PaaS平台最大的区别在于其支持：run-to-finish任务、二级调度任务和数据层面的可观测性洞察。主要包括：

* 日志：标准化所有组件包括批/流任务的日志输出
* 监控：配置组件的核心运维指标以及采集方式，对于批/流任务，需要采取push的方式采集指标
* 报警：根据指标设置合理的报警条件及优先级
* 异常检测：根据运维指标自动发现异常（企业版）
* 计费：需要专门的计费系统同时支持长跑服务 和 批/流任务（企业版）
* 审计：统一调度和运维操作接口，并接入审计系统（企业版）

##### 4.3.3.1.7. 调度服务

KDP为计算引擎组件提供云原生的调度机制支持，以提升资源使用率与运行效率。在通用K8s调度机制基础上实现二级调度，能更好的支持大数据类型负载的效率和SLA要求。主要包括：

* 支持大量批、流任务的调度
* 支持租户隔离与弹性资源分配（企业版）
* 在云原生环境下支持 Data Locality（企业版）
* 支持智能化、自动化的资源参数设置（企业版）
* 支持不同的抢占、优先级策略（企业版）
* 与现有的各种工具进行对接，满足SLA和资源使用的要求（企业版）

##### 4.3.3.1.8. 计算及存储引擎

| **组件** | **说明** |
| --- | --- |
| HDFS | \* 扩展了开源社区的 Helm Chart \* 支持了动态PV、容器网络及组件上下文配置管理 |
| Hive | \* 扩展开源代码支持 Hive SQL 以 Spark 作业方式运行 \* 支持在 Hue 或者 Beeline 客户端运行 Hive SQL \* Hive Table 可以存储在 HDFS 或者对象存储中 |
| Spark | \* 扩展了开源社区的 Spark Operator \* 通过自研 API 或者 JupyterLab 运行 Spark 作业 \* 扩展开源代码进行性能优化：Data Locality in HDFS、Sticky Sessions |
| Kafka | \* 扩展了开源社区的 Strimzi Kafka Operator \* 引入了 Kafka 集群管理界面 |
| Flink | \* 扩展了开源社区的 Flink Operator \* Flink 作业与 Spark 作业使用统一的调度 |

#### 4.3.3.2. 算法中台设计

##### 4.3.3.2.1. 概述

Kubeflow 是一种开源的 Kubernetes 原生框架，可用于开发、管理和运行机器学习（ML）工作负载。Kubeflow 是一个 AI/ML 平台，它汇集了多种工具，涵盖了主要的 AI/ML 用例：数据探索、数据管道、模型训练和模型服务。

Kubeflow 支持数据科学家通过一个门户访问这些功能，该门户提供了与这些工具进行交互的概要抽象信息。这意味着数据科学家无需了解 Kubernetes 插入这些工具所使用方法的具体细节。Kubeflow 本身专为在 Kubernetes 上运行而设计，并充分吸纳了许多关键概念，包括 Operator 模型。

##### 4.3.3.2.2. Kubeflow 有什么用途？

Kubeflow 通过提供一组工具和 API 来简化大规模训练和部署 ML 模型的过程，可以解决机器学习管道编排过程中涉及的诸多挑战。“管道”表示 ML 工作流，包括工作流的组件以及这些组件之间的交互方式。Kubeflow 能够满足一个项目中多个团队的需求，并允许这些团队从任何基础架构工作。这意味着，数据科学家可以从自己选择的云（包括 IBM Cloud、Google Cloud、Amazon 的 AWS 或 Azure）进行训练和提供 ML 模型。

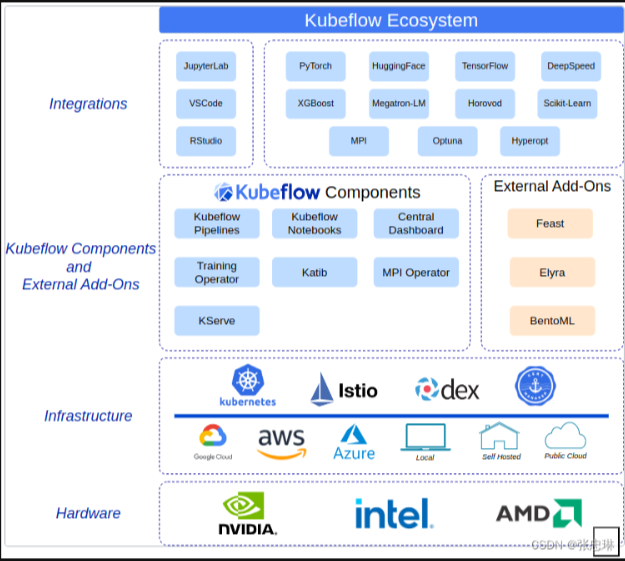
总的来说，Kubeflow 通过组织项目，同时利用云计算的强大功能，实现了机器学习运维（MLOp）的标准化。Kubeflow 的一些关键用例包括数据准备、模型训练、评估、优化和部署。

##### 4.3.3.2.3. 架构

Kubeflow是一个可以为想要构建和实验ML流水线的数据科学家提供的平台。Kubeflow 也为机器学习工程师和运营团队提供服务，将机器学习系统部署到不同的环境中进行开发、测试和生产级服务。

Kubeflow是Kubernetes的ML工具包。

下图显示了Kubeflow作为一个平台，用于在Kubernetes上部署ML系统的组件：

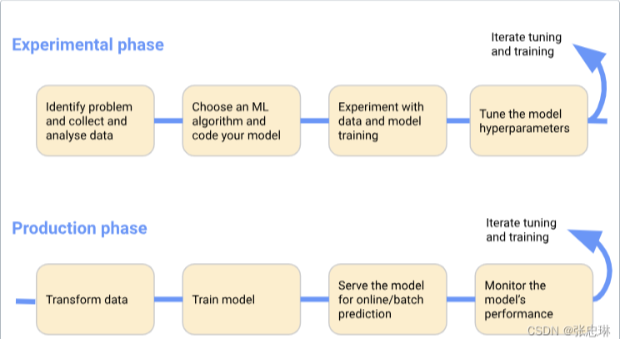


Kubeflow 构建于 Kubernetes 之上，作为部署、扩展和管理复杂系统的系统。使用Kubeflow配置接口，您可以指定工作流所需的ML工具。然后，您可以将工作流部署到各种云、本地和内部平台，以便进行试验和生产使用。

###### 4.3.3.2.3.1 介绍ML工作流

在开发和部署机器学习系统时，机器学习工作流通常由几个阶段组成。开发机器学习系统是一个迭代的过程。您需要评估机器学习工作流程各个阶段的输出，并在必要时对模型和参数应用更改，以确保模型持续产生您需要的结果。

为了简单起见，以下图表显示了工作流阶段的顺序。工作流末尾的箭头指向流，表示流程的迭代性质:



在机器学习工作流程中包含实验阶段和生产阶段。

A. 在实验阶段，基于初始假设开发模型，并反复测试和更新模型以产生你想要的结果:

确定想要机器学习系统解决的问题。

收集和分析训练机器学习模型所需的数据。

选择一个机器学习框架和算法，并对模型的初始版本进行编码。

对数据进行实验，并训练模型。

调整模型的超参数，以确保处理最有效率并获得最准确的结果。

B. 在生产阶段，您部署了一个执行以下过程的系统:

将数据转换成训练系统所需的格式。为了确保模型在训练和预测过程中行为一致，转换过程在实验阶段和生产阶段必须相同。

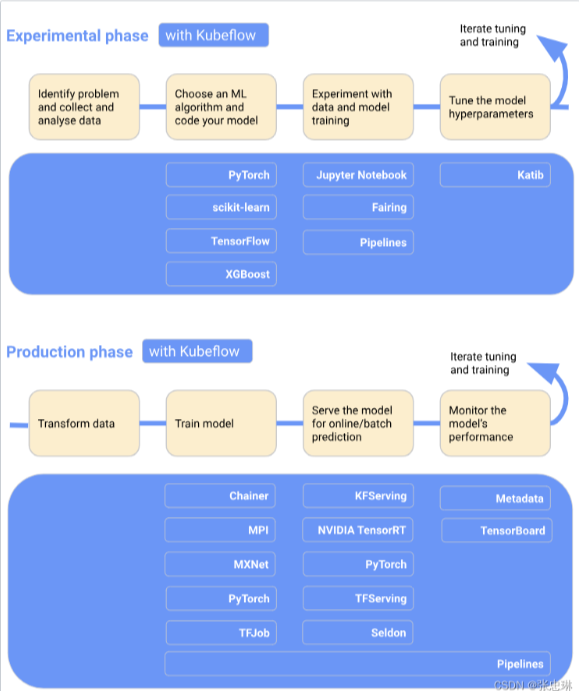
训练机器学习模型。

将模型提供给在线预测或批处理运行。

监控模型的性能，并将结果反馈到调整或重新训练模型的过程中。

###### 4.3.3.2.3.2 ML工作流的Kubeflow组件

下一张图将Kubeflow添加到工作流中，显示了Kubeflow在每个阶段哪些组件是有用的：



要了解更多信息，请阅读以下 Kubeflow 组件指南:

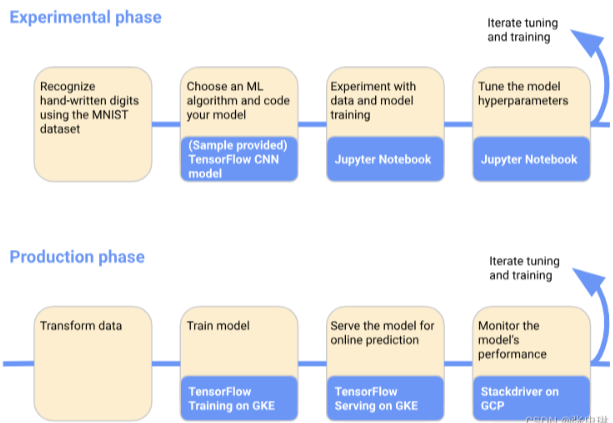
Kubeflow 包括用于生成和管理 Jupyter notebooks 的服务。使用 noteboks 进行交互式数据科学和机器学习工作流的实验。

Kubeflow Pipelines 是一个基于 Docker 容器构建、部署和管理多步骤 ML 工作流的平台。

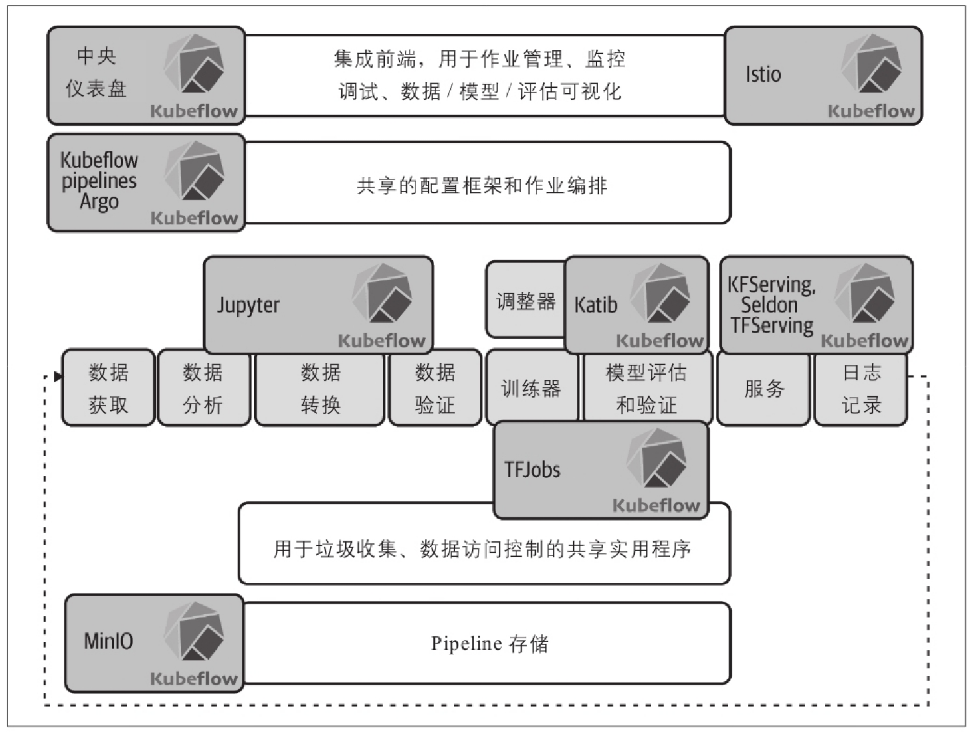
Kubeflow 提供了几个组件，您可以使用它们来构建机器学习培训、超参数调优和跨多个平台服务工作负载。

###### 4.3.3.2.3.3 ML工作流的例子

以下图表展示了一个简单的特定机器学习工作流程示例，您可以使用它来训练并提供一个在MNIST数据集上训练的模型：



##### 4.3.3.2.1. Kubeflow核心组件



###### 4.3.3.2.1.1 notebook（JupyterHub）

- 大多数项目的第一步是某种形式的原型设计和实验。Kubeflow用于原型设计和实验的工具是JupyterHub（https://jupyter.org/hub），这是一个多用户中心，可以生成、管理和代理单用户Jupyter notebook的多个实例。Jupyter notebook支持整个计算过程：开发、记录和执行代码，以及交流结果。

- 要创建一个新的服务器，你需要指定服务器名称和命名空间，选择一个镜像（从CPU优化、GPU优化或你可以创建的自定义镜像），并指定资源要求：CPU/内存、工作空间、数据卷、自定义配置等。一旦服务器被创建，你就可以连接到它并开始创建和编辑notebook。

- 为了让数据科学家在不离开notebook环境的情况下进行集群操作，Kubeflow在提供的notebook镜像中增加了kubectl，以允许开发人员使用notebook来创建和管理Kubernetes资源。Jupyter notebook pod运行在一个特殊的服务账户default-editor下，它对以下Kubernetes资源有命名空间级别的权限：

    - ·Pods

    - ·Deployments

    - ·Services

    - ·Jobs

    - ·TFJobs

    - ·PyTorchJobs

- 你可以将这个账户绑定到一个自定义角色，用于限制/扩展notebook服务器的权限。这使得notebook开发人员可以在不离开notebook环境的情况下执行角色中许可的Kubernetes命令。例如，创建一个新的Kubernetes资源可以直接在Jupyter notebook中运行以下命令：

- yaml文件的内容将决定创建什么资源。如果你不习惯编写Kubernetes资源，不用担心，Kubeflow Pipeline中包含创建资源的工具。

- 为了进一步增加Jupyter的功能，Kubeflow还在notebook中提供了对Pipeline和元数据管理等重要的Kubeflow组件的支持。Jupyter notebook还可以直接加载分布式训练作业。

###### 4.3.3.2.1.2 训练operator

- JupyterHub是用于数据初始实验和ML作业原型的良好工具。然而，当在生产中进行训练时，Kubeflow提供了一些训练组件来自动执行机器学习算法，包括：

    - ·Chainer训练

    - ·MPI训练

    - ·Apache MXNet训练

    - ·PyTorch训练

    - ·TensorFlow训练

- 在Kubeflow中，分布式训练作业由特定于应用程序的控制器（称为operator）管理。这些operator扩展了Kubernetes API，用于创建、管理和修改资源的状态。例如，要运行一个分布式的TensorFlow训练作业，用户只需要提供一个描述所需状态的规范（工作节点和参数服务器的数量等），TensorFlow operator组件将负责其余的工作，并负责管理训练作业的生命周期。

    - 这些operator允许自动化重要的部署概念，例如，可扩展性、可观测性和故障转移。它们也可以被Pipeline使用，与系统中的其他组件进行链式执行。

###### 4.3.3.2.1.3 Kubeflow Pipeline

- 除了提供实现特定功能的专门参数外，Kubeflow还包括Pipeline组件，用于编排机器学习应用程序的执行。Pipeline实现基于Argo Workflow，这是一个开源的、容器原生的Kubernetes工作流引擎。Kubeflow安装了所有的Argo组件。

- 在高层次上，Pipeline的执行包含以下组件：

    - Python SDK

        - 你可以使用Kubeflow Pipeline领域专用语言（DSL）创建组件或指定Pipeline。

    - DSL编译器

        - DSL编译器将Pipeline的Python代码转换为静态配置（YAML）。

    - Pipeline Service

        - Pipeline Service从静态配置中创建一个Pipeline运行。

    - Kubernetes资源

        - Pipeline Service调用Kubernetes API服务器来创建必要的Kubernetes自定义资源定义（CRD）来运行Pipeline。

    - 编排控制器

        - 一组编排控制器执行Kubernetes资源指定Pipeline所需的容器。容器在虚拟机上的Kubernetes pod中执行。一个示例控制器是Argo Workflow控制器，它负责编排任务驱动的工作流。

    - 制品存储

        - Kubernetes pod存储以下两类数据。

        - 元数据

            - 实验、作业、运行、单标量指标（一般为排序和过滤而汇总）等。Kubeflow Pipeline将元数据存储在MySQL数据库中。

        - 制品

            - Pipeline包、视图、大规模指标（如时间序列）通常用于调试和检测单个运行的性能。Kubeflow Pipeline将制品存储在MinIO服务器（https://docs.minio.io）、Google Cloud Storage（GCS）或Amazon S3（https://aws.amazon.com/s3）等存储库中。

- Kubeflow Pipeline让你拥有了让机器学习作业重复持续进行并处理新数据的能力。它在Python中提供了一个直观的DSL来编写Pipeline。随后，Pipeline会被编译成现有的Kubernetes工作流引擎（目前是Argo工作流）。Kubeflow的Pipeline组件使得用于构建端到端机器学习项目的所需工具更加容易使用和协调。除此之外，Kubeflow可以同时跟踪数据和元数据，提高我们对作业的理解能力。Pipeline还可以公开内置的机器学习算法的参数，方便Kubeflow进行调优。

###### 4.3.3.2.1.4 超参调优

- 为训练模型寻找合适的超参集可能是一项很有挑战性的任务。传统的方法（如网格搜索）不仅耗时且相当乏味。现有的超参系统大多被绑定在一个机器学习框架上，只有少数几个参数空间的搜索选项。

- Kubeflow提供了Katib组件，方便用户在Kubernetes集群上轻松执行超参优化。Katib的灵感来自Google的黑盒优化框架Vizier。它利用贝叶斯优化等先进的搜索算法来寻找最优的超参配置。

- Katib支持超参调优，并可以与任何深度学习框架集成工作，这些框架包括TensorFlow、MXNet和PyTorch。

- 与Google Vizier一样，Katib基于4个主要概念：

    - 实验（experiment）

        - 在可行空间上的单一优化运行。每个实验都包含描述可行空间的配置以及一组尝试。假设目标函数f（x）在实验过程中不发生变化。

    - 尝试（trial）

        - 一个参数值列表x，将导致f（x）的单一评估。一次尝试可以是“完成”，即已经对它进行了评估，并给它分配了目标值f（x）；否则就是“待定”。一次尝试对应一个作业。

    - 作业（job）

        - 负责评估“待定”尝试并计算其目标值的过程。

    - 建议（suggestion）

        - 一种构建参数集的算法。目前，Katib支持以下探索算法：

            - ·随机（Random）

            - ·网格（Grid）

            - ·超频（Hyperband）

            - ·贝叶斯优化

    - 通过使用这些核心概念，你可以提高模型的性能。由于Katib不绑定在一个机器学习库上，因此你可以用最少的修改探索新的算法和工具。

###### 4.3.3.2.1.5 模型推理

- Kubeflow使在大规模生产环境中轻松部署机器学习模型变得容易。它提供了几种模型服务选项，包括TFServing、Seldon服务、PyTorch服务和TensorRT。它还提供了一个总括性的实施方案KFServing（https://oreil.ly/qEvqq），包括自动缩放、联网、健康检查和服务器配置等模型推理的关注点。

- KFServing的整体实现是基于利用Istio（https://istio.io）（后面会讲到）和Knative服务（https://knative.dev）——Kubernetes上的Serverless容器。正如Knative文档中所定义的那样，Knative服务项目提供了中间件原语：

    - ·快速部署Serverless容器

    - ·自动扩容和缩容至零

    - ·Istio组件的路由和网络编程

- 由于模型服务本身就很棘手，所以快速的扩缩容很重要。通过自动将请求路由到较新的模型部署，Knative服务简化了对模型持续更新的支持。这就需要将未使用的模型缩容到零（最小化资源利用率），同时保持功能回滚的能力。由于Knative是云原生的，它受益于基础架构技术栈，因此提供了Kubernetes集群中的监控能力，如日志记录、跟踪和监控。KFServing还通过Knative事件为可插拔的事件源提供了可选的支持。

- 与Seldon类似，每个KFServing部署都是一个编排器，将以下组件连接在一起：

    - 前置处理器

        - 可选的组件，负责将输入数据转换为模型服务所需的内容或格式。

    - 预测器

        - 必选组件，负责实际的模型服务。

    - 后置处理器

        - 可选的组件，负责将模型服务结果转换/加强为输出所需的内容或格式。

- 其他组件可以增强整体模型服务的实施，但不在执行Pipeline之内。异常检测和模型解释器等工具可以在此环境中运行，且不会降低整个系统的速度。

- 虽然所有这些单独的组件和技术已经存在了很长时间，但将它们集成到Kubeflow的服务系统中，可以降低将新模型引入生产带来的复杂性。

- 除了直接支持ML操作的组件外，Kubeflow还提供了一些支持组件。

###### 4.3.3.2.1.6 元数据

- Kubeflow的重要组成部分是元数据管理，它提供了捕获和跟踪有关模型创建信息的功能。许多组织每天都建立数百个模型，但是很难管理所有模型的相关信息。ML元数据既是一个基础设施，也是一个库，用于记录和检索与ML开发人员和数据科学家的工作流相关的元数据。可在元数据组件中注册的信息包括：

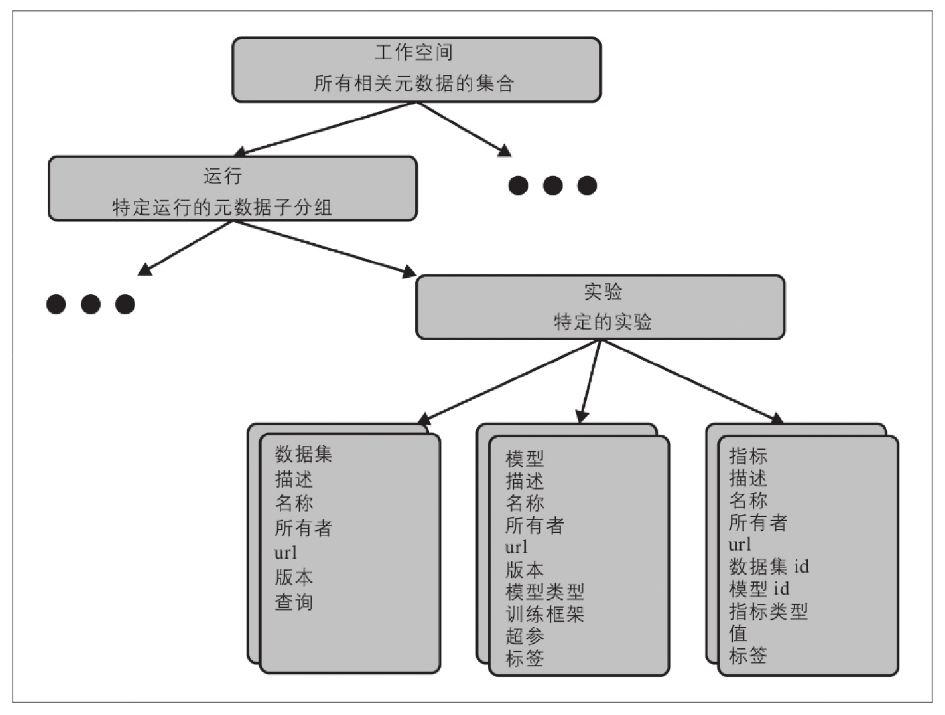
    - ·用于创建模型的数据来源。

    - ·通过Pipeline中的组件/步骤产生的制品。

    - ·组件/步骤的执行情况。

    - ·Pipeline和关联的沿袭信息。

- ML元数据跟踪ML工作流中所有组件和步骤的输入输出以及沿袭信息。



#### 4.3.3.3. 服务中台设计

#### 4.3.3.4. 运营中台设计

#### 4.3.3.5. 业务中台设计

### 4.3.4 基础设施层

#### 4.3.4.1. 中心基础设施层

#### 4.3.4.2. 边缘基础设施层

## 五、关键技术

### 5.1基于云原生的自动化调度

采用Kubernetes(K8S)作为底层资源编排和调度引擎,实现对异构计算、存储、网络资源的云原生管理。同时引入云边协同调度、智能数据调度、智能应用调度等增强组件,构建面向交通场景的自动化调度框架。具体包括：

#### 5.1.1异构资源管理

纳管高性能计算集群HPC、AI计算集群、边缘微数据中心等多种异构算力资源，对接FPGA、GPU、昇腾、RISC-V等异构硬件,实现芯片即服务和加速即服务，统一编排管理块存储、对象存储、分布式存储、时序存储等多元数据资源。

#### 5.1.2云边协同调度

采用KubeEdge等边缘计算框架,实现云边一体化调度和应用下发，设计跨域联邦调度策略,统筹公有云、私有云、边缘云的算力资源分配，探索基于区块链和联邦学习的可信调度机制,支持跨组织、跨区域协同。

#### 5.1.3智能数据调度

感知数据全生命周期状态,优化数据采集、数据分发、数据缓存策略，针对热点数据、流数据、批数据等制定差异化数据分层存储和生命周期管理策略，构建数据虚拟化层,打通数据孤岛,实现元数据驱动的自动化数据调度。

#### 5.1.4智能应用调度

建立应用模型与基础设施的映射关系,实现应用SLA、成本、性能等多目标优化，针对飞桨、深度学习框架、分布式训练等构建针对性的调度策略,提升AI任务性能发挥大语言模型的上下文学习能力,实现任务之间的关联理解和协同调度。

### 5.2基于凤凰架构的全域可观测体系

随着系统规模的膨胀和业务复杂度的上升,运维管理和故障定位面临巨大挑战。立足云原生可观测性最佳实践,以分布式跟踪、指标监控、日志分析为支柱,构建端到端的全域可观测体系，同时引入凤凰架构理念,在可观测基础上进一步实现故障自愈、架构自优化。

分布式跟踪：借鉴OpenTelemetry标准,在分布式数据总线、微服务网关、第三方API等关键节点设置调用链埋点,实现端到端数据流可视化溯源和性能瓶颈分析,降低问题定界复杂度。

指标监控：采用Prometheus等云原生监控方案,建立覆盖IaaS、PaaS、SaaS的立体化指标体系。同时充分利用LLM的多模态特征提取能力,实现基于时间序列、日志、事件等数据的关联分析和根因定位。

日志分析：采用ELK等云原生日志分析平台,对分散在车、路、云等不同位置的结构化/非结构化日志进行统一采集、解析和关联分析。利用LLM从海量日志中自动生成故障模式、智能答疑和处置流程。

领域安全：将区块链不可篡改、可追溯特性与零信任安全模型相结合,对车路协同、数据共享、应用调用等实现全流程可信。针对指标、日志、调用链等可观测数据,提供隐私保护和脱敏处理。

故障自愈：基于凤凰架构原则,从应用和数据两个维度构建自我修复和优化机制。针对常见故障场景,通过checkpoint、应用快照、数据副本等,实现秒级故障检测、边缘容错和就地恢复。

架构自优化：持续采集系统全生命周期的可观测数据,利用机器学习实现故障预测、容量预估、瓶颈识别,并结合LLM进行架构体检、技术债识别,动态调整技术栈和服务粒度,实现系统的持续进化。

### 5.3 LLM赋能的新一代智能交通感知、规划、决策能力

大语言模型(LLM)作为人工智能领域的重大突破,其在知识学习、多模态理解、逻辑推理等方面展现出卓越能力。将LLM引入云原生智能交通系统,可显著提升系统的感知、决策、进化、交互等核心能力。主要体现在:

多模态融合感知：利用LLM强大的语义理解能力,对路况图像、事件文本、语音指令等多模态异构交通数据进行统一建模,挖掘背后的关联性和因果性,形成更全面、连贯的交通运行状态感知。

知识增强决策优化：将海量结构化的交通专业知识与LLM结合,实现从数据到知识再到决策的跨越。通过迁移学习、上下文学习等机制,不断吸收新知识,对路网优化、交通预测、应急管理等决策任务实现持续优化。

人车路云自然交互：融合自然语言处理、语音识别、情感分析等多模态交互技术，让LLM作为人、车、路、云之间的交互枢纽,实现用户需求的精准理解和人性化响应,提供主动、顺滑、个性化的交通出行服务。

场景构建与策略迁移：利用LLM从海量驾驶日志、路网监控数据、路侧感知数据中自动学习提炼多种交通场景。通过few-shot learning等,实现不同城市、不同路况间调度策略的快速迁移和适配。

数字孪生与策略评估：将现实交通场景映射到数字空间,基于LLM进行交通流演绎推演、施策效果评估,提供沙盒环境进行各类极端工况、边界场景的策略验证,从而持续打磨和进化系统的领域智能。

## 六、实施路径

顶层设计：系统性梳理各地交通发展的痛点、难点,制定符合实际的建设方案。

短期目标：聚焦重点城市和典型场景,开展试点示范,探索可落地、可推广的实施路径。

中期目标：固化试点经验形成中台能力,跨区域复用,不断完善统一技术架构和标准规范。

长期目标：培育开放生态,持续进化形成分布式智慧交通操作系统,支撑行业数字化转型。

新一代智慧交通操作系统的建设是一个复杂的系统工程,需要交通、信息、电子、人工智能等多学科协同创新,打通数据壁垒,形成合力。立足自主创新,以开源开放的心态学习借鉴国内外先进经验,充分考虑多方主体的诉求,共建共享、合作共赢的生态圈，推动交通基础设施从数字化到网联化再到智能化的跨越,为交通强国建设提供有力支撑。