云原生数据湖服务平台的设计与实现

魏文定[1],鄂海红[1],王　曦[2],宋美娜[1],宿兴辉[3]
(1. 北京邮电大学 计算机学院(国家示范性软件学院),北京 100876;
2. 中国科学院信息工程研究所,北京 100085;
3. 联洋国融(北京)科技有限公司,北京 100088)

摘　要:云原生数据湖已经成为数据管理和分析领域的研究热点,相关技术和应用也得到了广泛的关注和探索。数据湖部署存在着成本高、组件之间兼容性差等问题,存算不分离制约着数据湖平台延展性,缺乏完备的数据入湖体系容易引起数据湖沼泽的形成,导致用户无法从中提取数据价值。该文设计并实现了云原生数据湖服务平台,平台以 Kubernetes 为底层构建云原生环境,结合容器技术将数据湖组件镜像化,同时设计数据湖存算分离方案来提高数据湖平台的可扩展性和可移植性,并配合监控、组装生产线将镜像容器化,实现数据湖上云操作。并建立用户入湖作业与云原生计算引擎之间的桥梁,对入湖信息进行预处理,提供多类型作业以满足多元化入湖场景,以统一 catalog 的方式将数据写入数据湖中。实际运行结果表明,该平台既提高了数据湖平台的灵活性和可靠性,又确保了元数据和数据资产的规范性存储。

关键词:云原生;数据湖;大数据;生产线;数据湖上云
中图分类号:TP311　　**文献标识码**:A　　**文章编号**:1673-629X(2024)02-0017-06
doi:10.3969/j.issn.1673-629X.2024.02.003

Design and Implementation of a Cloud Native Data Lake Platform

WEI Wen-ding[1],E Hai-hong[1],WANG Xi[2],SONG Mei-na[1],SU Xing-hui[3]
(1. School of Computer Science (National Pilot Software Engineering School),Beijing University of Posts and Telecommunications,Beijing 100876,China;
2. Institute of Information Engineering,Chinese Academy of Sciences,Beijing 100085,China;
3. Lianyang Guorong (Beijing) Technology Co.,Ltd.,Beijing 100088,China)

Abstract:Cloud native data lakes have become a research hotspot in the field of data management and analytics,and related technologies and applications have received widespread attention and exploration. Data lake deployment suffers from high cost and poor compatibility between components,the lack of separation of storage and computation restricts the extensibility of the data lake platform,and the lack of a complete data entry system easily causes the formation of a data lake swamp,resulting in users being unable to extract data value from it. We design and implement a cloud native data lake service platform,which uses Kubernetes as the underlying layer to build a cloud native environment,and combines container technology to mirror data lake components. Meanwhile,the storage and computing separation scheme of the data lake is designed to improve the scalability and portability of the data lake platform,and the image is containerized with the monitoring and assembly production line to realize cloud operations on the data lake. The platform also establishes a bridge between the user's entry operations and the cloud native computing engine,pre-processes the entry information,provides multiple types of operations to meet diverse entry scenarios,and writes data to the data lake in a unified catalog manner. The actual operation results show that the platform not only improves the flexibility and reliability of the data lake platform,but also ensures the normative storage of metadata and data assets.

Key words:cloud native;data lake;big data;production line;data lake on cloud

0　引　言

随着全球数据产业[1]的蓬勃发展,数据系统正扮演着关键的支撑和促进角色。随着大数据行业的发展,开发人员所需要参考和分析的海量数据日益庞大,

收稿日期:2023-05-04　　**修回日期**:2023-09-06
基金项目:国家自然科学基金(62176026);北京自然科学基金(M22009)
作者简介:魏文定(1997-),男,硕士研究生,研究方向为数据湖与云原生;通信作者:鄂海红(1982-),女,博士,教授,CCF 会员(C1959M),研究方向为云原生与大数据中台、人工智能及应用。

对于大数据组件的依赖要求也越来越高,对数据计算和存储提出了新的需求。

数据仓库和数据湖[2-5]是两种比较有代表性的大数据产品,数据仓库其核心特征是面向主题、集成性、稳定性,帮助企业完成数据价值提取,缺点就是计算和存储高度耦合,而且无法存储和查询非结构化文件[6]。随之数据湖技术诞生,数据湖的核心是提供一个容纳所有形式的存储集[7-16]。

最初,要使用数据湖平台,企业需要购买至少十几台服务器,然后寻找专业人员安装每个大数据组件。安装完成后,还需要建立开发平台、运维平台,并购买各种工具。这种建设成本、使用门槛以及决策风险都比较高。

随后,传统的数据湖平台开始逐渐暴露出了一些短板和不利因素:

(1)多个部门共享同一个集群,缺乏资源隔离和限制,导致相互之间产生影响。

(2)系统部署依赖复杂的手动操作,且运维成本较高。

(3)服务器在不定因素下出现故障,使得某个数据湖组件下线以致整个数据湖资源不可正常使用。

(4)缺乏标准的大数据组件发布流程,无法培养客户的自主数据能力。

(5)计算资源和存储资源之间存在着高度耦合关系,导致整个数据湖平台难以延展。

目前,有些厂商开发的数据湖平台是基于其自有的调度系统和分布式计算体系构建的。尽管它们正在进行 Kubernetes 改造,将部分调度工作迁移到 Kubernetes 上,但是绝大部分组件仍然基于原有的大数据平台架构运行,因此它们还没有真正实现云原生架构下的数据湖平台。

另外,对于数据入湖,缺乏完备的管理体系,数据湖会逐步形成沼泽湖,导致用户无法从中提取数据价值。同时,在数据处理层面上,入湖任务和云原生计算引擎之间缺乏完整的桥梁,容易导致数据无法写入数据湖或写入正确的位置。

云原生是一种新兴的应用程序开发和部署方法,它基于云计算、容器化、微服务和自动化等技术,旨在帮助开发人员和企业更加高效地构建、交付和运维应用程序。针对以上问题,结合云原生技术,对于数据湖组件进行改造并且完全实现云原生化,建立入湖任务和计算引擎之间的关系,就变得更加有必要了。

1 研究内容

整体架构如图 1 所示,从上往下依次为业务层、业务支撑层、容器层、运行环境层和存储层。

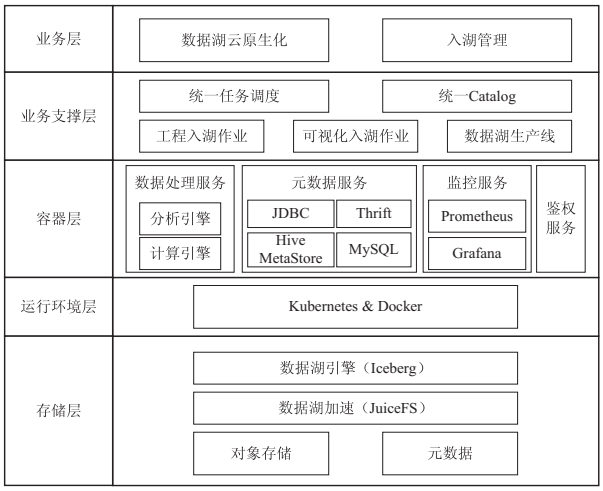

图 1 整体架构

业务层由数据湖云原生化和入湖管理组成。数据湖云原生化服务提供可视化界面供用户完成数据湖组件容器资源信息定义以及数据湖构建任务,在云原生环境中搭建数据湖平台。入湖管理负责外部数据源与数据湖信息对接,形成入湖预处理信息,并向后端提交任务。

业务支撑层对上层任务调度进行统一管理,支撑数据湖云原生服务发布,实现入湖作业对计算引擎的调度。提供统一 Catalog 接口,让不同计算引擎共享元数据,保证数据访问方式统一。

在容器层,数据湖云原生化之后以容器形式存在,主要分为数据处理服务、元数据服务、监控服务和鉴权服务。数据处理服务包括计算引擎(例如 Spark)和分析引擎(例如 Trino),支撑数据湖计算工作。元数据服务维护数据湖元数据信息。监控服务由 Prometheus 和 Grafana 组件组成,用于监控 Pod 健康状态。鉴权服务基于 RBAC(Role-Based Access Control)完成对容器授权。

在运行环境层,Kubernetes 与 Docker 共同构成云原生环境,为容器提供可运行环境。采取 Docker 对数据湖组件进行镜像化,目前 Docker 与 Kubernetes 具有很好的兼容性,促使数据湖组件能够在 Kubernetes 环境下容器化,基于云原生的特性在云原生环境上运行更多的容器,有助于发挥数据湖的可扩充性。

在存储层,维护数据湖数据内容,以 Iceberg 表格式将数据写入文件系统,构建 JuiceFS 分布式文件系统为数据存取提速,并提供统一存储访问接口。

2 数据湖云原生化

2.1 流程设计

为实现数据湖部署流程化,需要用户完成数据湖组件容器信息定义以及组件之间的关联,提交 Json 请求之后,由后端完成业务逻辑组织,最终实现数据湖云

原生化,实现细节如图 2 所示。数据湖变成服务资源之前,需要经过一系列化模块组织。在容器信息定义时,系统接收资源构建 Json 请求,并解析 Json 提取容器定义信息,再将信息持久化到数据库,同时对不存在镜像由镜像定义器完成构建。启动过程由数据湖启动器进行接管,实现数据湖的自动化部署。而模板引擎生成 Yaml 文件流,最后调用 Kubernetes 将 Yaml 配置跟相关镜像组合完成数据湖部署工作。

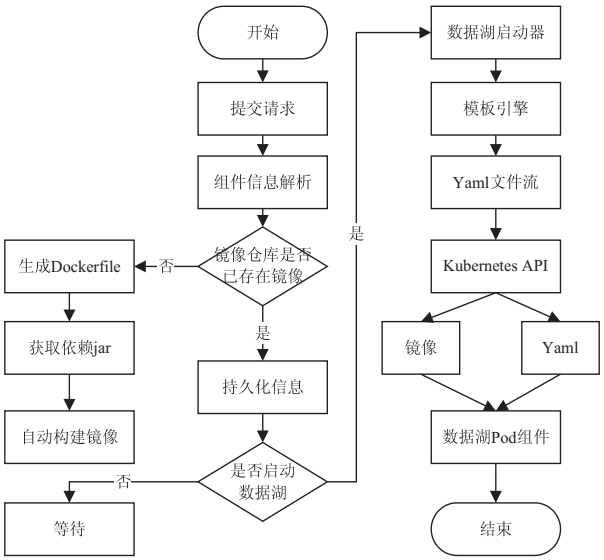

图 2　数据湖云原生化流程设计

2.2　数据湖存算分离方案

计算引擎在数据湖功能系统中起着承上启下的作用,而对象存储承担着数据存储重要角色。但计算引擎与对象存储之间存在着兼容性差问题,并且对象存储元数据操作性能低。S3fs,Goofys 与 JuiceFS 相比,不支持本地缓存,性能远低于 JuiceFS。因而在计算引擎与对象存储之间引入中间件 JuiceFS,JuiceFS 是一款面向云原生设计的高性能分布式文件系统。同时 JuiceFS 支持多协议兼容,如 POSIX,HDFS 和 S3 协议。在一定程度上能够提升系统的读写性能和可扩充能力。

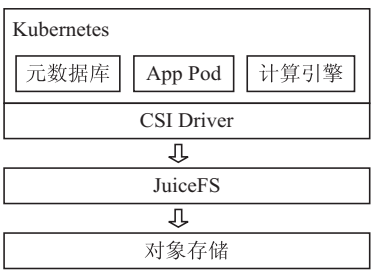

图 3　数据湖存算分离方案

由于该文采用云原生技术构建数据湖,能够轻量化数据湖服务调度,更好地发挥数据湖的效力。为使容器编排技术、计算引擎、对象存储更好地结合,需要通过存算分离降低技术连接复杂性。借助 Kubernetes 抽象存储接口(CSI)以及存储卷挂载技术(PV 和 PVC),将 JuiceFS 以 PV 的方式挂载到容器持久卷,如图 3 所示,以达到存算分离效果,进一步实现容器编排技术、计算引擎、对象存储之间的高效协作。

如图 4 所示,在容器编排系统安装 juicefs csi driver,完成对存储服务的绑定,在容器环境内形成高性能、简单易用的共享文件系统。在环境内初始化 StorageClass 和 PVC 后,只需在 Pod 声明 PVC,即可完成 Pod 对存储卷的挂载。

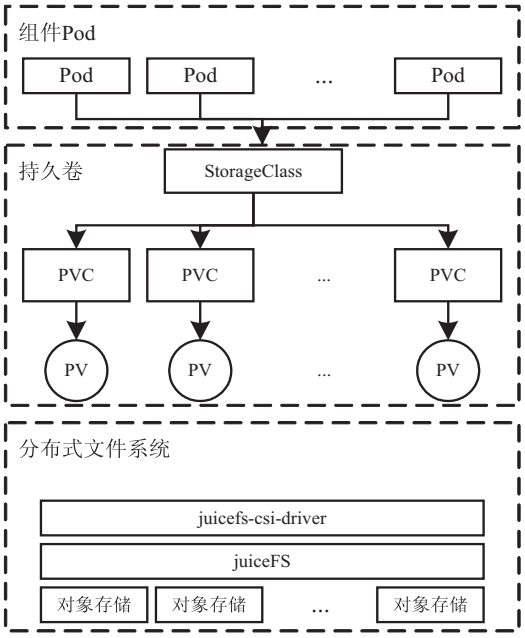

图 4　数据湖组件 Pod 与文件系统连接流程

借助中间件 JuiceFS 以及容器编排系统动态挂载持久卷技术,实现了计算和存储的分离。对于不同组件 Pod 只需声明已初始化 PVC,即可完成在文件系统的存储空间分配,同时容器编排系统自身的容错机制,即使 Pod 宕机之后再恢复健康状态,Pod 仍能恢复对原有持久卷的挂载,进一步保证数据湖系统运作的可靠性。

2.3　数据湖镜像构建

数据湖整体由关系数据库、对象存储、元数据库和计算引擎组成。而组件之间、组件与依赖包之间存在着版本依赖关系,例如 Hive MetaStore 及 Spark 与 Iceberg 之间存在着版本上关系。为减少组件版本适配的试错成本,在数据湖资源上云之前,对于部分镜像可以提前进行创建。构建流程如下:①准备数据湖组件依赖文件和脚本文件。②编写 Dockerfile 文件。③由 docker build 完成镜像生成。④对镜像进行标签化并上传至镜像仓库以备调用。

借助提前定制化镜像模型虽能很好地满足系统功能需要,但在数据湖组件升级过程中,需要人为进行干预,导致镜像生成依然过度依赖于人工。为解决这项

功能的不足,采取数据湖镜像定义新思路,其过程如图 5 所示。

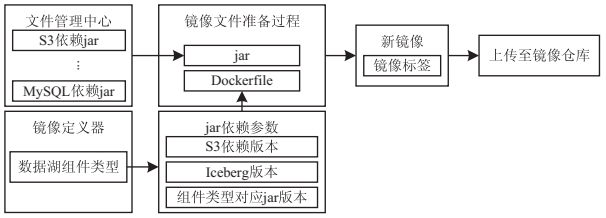

图 5 数据湖镜像生成过程

在数据湖云原生化时,由镜像定义器对镜像资源进行检查,若检查镜像仓库中不存在适配镜像资源时,发起创建镜像过程。根据当前组件类型,检索关系数据库中当前组件所依赖的 jar 包以及版本信息,从文件管理中心获取所需的 jar 包依赖,并基于 jar 版本信息构建临时 Dockerfile 文件。后由镜像定义器主动发起构建镜像请求,从私有仓库中获取各类基础镜像,并将所依赖的 jar 包 COPY 至镜像容器内,镜像构建完成后将镜像进行标签化,由镜像定义器远程控制 Docker 将镜像上传至镜像仓库以备后续调用。

2.4 模板引擎

在使用 Kubernetes 对资源进行部署过程中,对每类资源都需要重复编写大量配置文件,在系统移植和资源重新构建过程中,对于开发用户不太友好。因而,将每类部署文件进行定制模板,借助模板引擎屏蔽底层技术的实现细节,简化用户对数据湖部署的操作,实现简单易用原则。

模板引擎需要解决两个问题,分别是模板文件的定义和模板属性的赋值。

(1)模板文件的定义:在 Kubernetes 环境中包含着 Deployment, Service, ConfigMap 等多种资源对象,然而这些资源对象的属性存在一些不同的定义,同时,数据湖每类组件的部署方式也不同,所以无法通过一种通用模板来完成数据湖的部署工作。

(2)模板属性值转换:由于交互层采用界面化操作模式,除了需要完成对用户输入数据校验性工作,还需完成输入参数与可变属性值的映射关系。

为对模板文件进行合理化组织,采用模板文件分层设计,如图 6 所示。将不同组件按照类型进行区分,在每类组件下定义通用的 Kubernetes 资源对象模板。

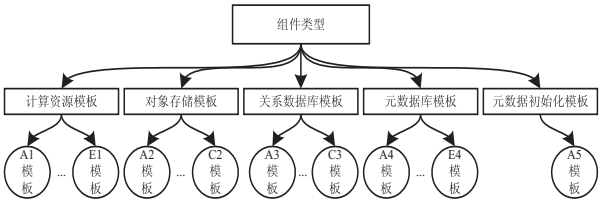

图 6 模板分层设计

在用户提交数据湖组件创建请求之后,后端获取已被持久化的组件资源定义信息,进一步获取组件类型,同时将持久化信息转化为 Set(组件资源定义参数集合),再由模板引擎根据组件类型调用相应类型模板,然后将 Set 反转形成模板可变属性值,最终构建 Yaml 文件流,如图 7 所示。

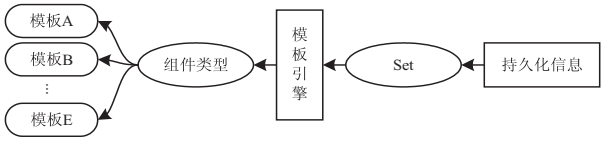

图 7 Yaml 文件流形成过程

对于用户而言,用户无需关注配置文件的解析过程,只需要在交互界面输入组件定义相关参数,即可形成在 Kubernetes 可被运行的 Yaml 文件,系统屏蔽了模板引擎的实现过程。

2.5 数据湖发布生产线

数据湖启动器以守护进程的形式在 Kubernetes 环境中运行,以 NodePort 形式对外暴露服务。此外提供一种 Pod 监控守护进程,用于辅助数据湖组件容器部署和维护,如图 8 所示。

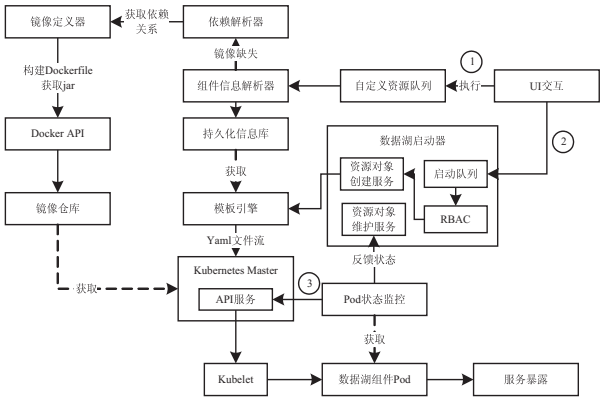

图 8 数据湖发布生产线

数据湖的发布需要完成三个步骤:

(1)自定义组件容器信息,对于缺失镜像完成自动化创建。

(2)根据服务参数,创建 Kubernetes 名称空间实现资源隔离,并完成 RBAC 授权服务。在容器启动之前,完成相应类型模板的创建工作,并向 Kubernetes API 传送 Yaml 文件流,之后由 Kubernetes 创建数据湖组件 Pod。

(3)在 Pod 启动过程中,不停监控 Pod 状态。当检测 Pod 处于不健康状态,且重启次数超过阈值仍然失败,停止重启操作,并告知用户创建数据湖失败,同时清理相关容器,避免资源占用。

3 统一 Catalog

为解决湖中数据存放问题,需要提供元数据统一语义,核心问题在于:把目录结构和表结构进行统一,

然后暴露给用户统一的语义层。因而需要完成 Catalog 的统一,Catalog 是数据资产的有序清单,使用元数据来帮助组织管理其数据,并帮助数据专业人员收集、组织、访问和丰富元数据以支持数据发现和治理。

Spark,Iceberg 和 Trino 支持多种 Catalog 后端存储实现,不同 Catalog 存储方式和写入方式有所差异,给元数据管理和治理带来诸多影响,因而在 Catalog 层需要暴露一个统一的接口。

Hive Metastore(HMS)作为比较成熟的元数据管理系统,且 Spark,Iceberg 和 Trino 都兼容 HMS,统一使用 HMS 作为元数据基础存储组件。在创建 Catalog 时,统一调用 HMS 接口对 Catalog 持久化,最终形成统一的 Catalog,如图 9 所示。

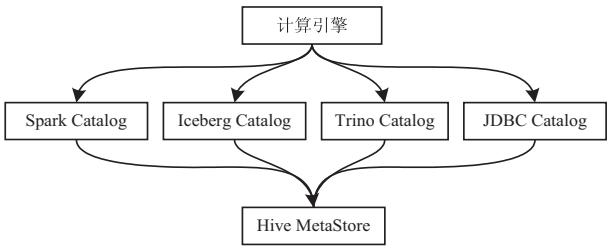

图 9　Catalog 统一设计

统一 Catalog 可以将元数据存储在一个中心化的位置,使数据湖中的所有数据都可以被发现、描述和理解。可以为数据湖中的所有数据提供一个单一的视图,从而更容易地找到需要的数据。

4　入湖作业管理

而数据湖要变成有机资源,需要数据的融入。由入湖作业、服务应用中心、云原生计算资源、云原生存储资源四个模块构成,如图 10 所示。

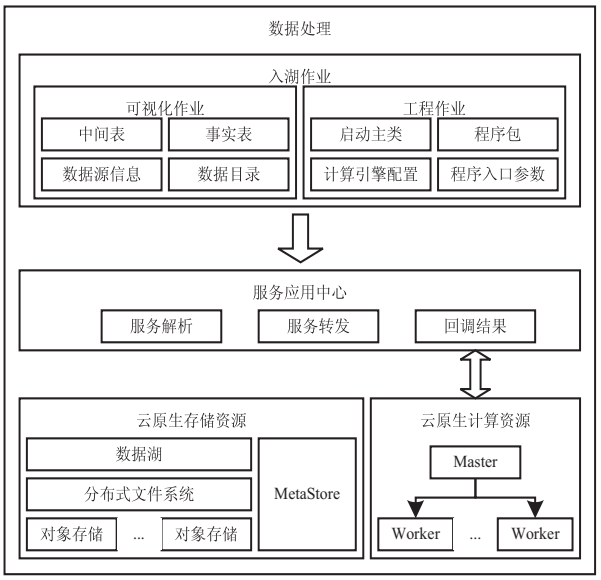

图 10　数据处理模块

入湖作业由可视化作业和工程作业两种作业构成,业务层识别入湖预处理信息,调动计算引擎将外源数据输入到存储空间。

(1)可视化作业。

获取数据源信息和来源表之后,在可视化界面生成事实表,事实表为数据湖表(即中间表)建立映射参考对象,并提供创建中间表 UI 操作,然后选择数据文件存放位置,保存之后,入湖信息生成。

在提交入湖任务后,服务应用中心接受请求并解析入湖需求,获取数据源信息以及湖表 Schema,同时提取数据文件输出位置信息,借助模板引擎更新计算引擎数据文件输出目录。成功获取计算资源之后,将入湖任务提交至计算引擎,由计算引擎完成数据源到数据湖的数据写入。任务完成之后,服务应用中心将处理结果返回至前端。

(2)工程作业。

将数据工程作业上传至文件系统,并完成主类、计算引擎配置参数和程序入口参数编写,保存之后,入湖信息生成。

在提交入湖任务后,服务应用中心接受请求并解析入湖需求,根据入湖信息生成 Dockerfile,如图 11 所示,由镜像定义器将数据工程作业生成镜像并上传至镜像仓库。由模板引擎根据计算引擎配置参数以及作业参数生成 Yaml,并向 Kubernetes API 传输 Yaml 文件流,完成临时计算作业集群的生成,由计算作业集群完成数据入湖工作。任务完成之后,服务应用中心将处理结果返回至前端,并触发临时计算作业集群清理工作,避免资源占用。提供多类型作业调动云原生计算资源来满足数据从数据源到数据湖的转变,以适应多元化业务场景。

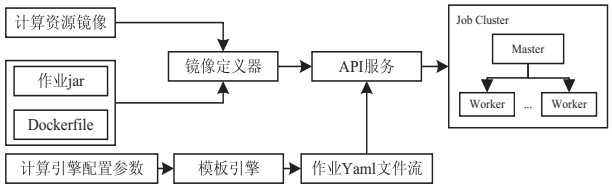

图 11　作业集群构建过程

5　结束语

在实际运行过程中,数据湖各项组件能够在 Kubernetes 以自动化形式进行部署,并完成数据源数据以 Iceberg 表格形式写入对象存储中。

在实践过程中,存在着数据湖组件部署过程复杂且难以维护、组件部署过度依赖人工化处理等问题。其次,存算不分离制约着数据湖平台的灵活性。将数据湖组件容器化,并借助容器编排系统对数据湖组件以生产线的方式进行组织解决部署和维护问题,设计

数据湖存算分离方案来提高数据湖平台的可扩展性和可移植性。通过统一 Catalog 的方式可以为数据湖中的所有数据提供一个单一的视图,更容易进行数据发现。对入湖信息进行预处理,提供多类型作业以满足多元化入湖场景,最终将数据写入数据湖正确的位置。

参考文献:

[1] 王 磊. “十三五”大数据产业发展回顾及“十四五”展望[J]. 中国经贸导刊,2020(21):51-54.

[2] 张桂刚,李 超,毛湘科,等. 区块链数据湖架构研究[J]. 计算机与数字工程,2023,51(1):86-92.

[3] 杨文哲,郝渊科,赵常胜,等. 基于对象代理的大数据共享可信数据湖平台[J]. 小型微型计算机系统,2023,44(6):1324-1328.

[4] 陈 氢,张 治. 融合多源异构数据治理的数据湖架构研究[J]. 情报杂志,2022,41(5):139-145.

[5] 马妍娇. 2022 年中国云原生数据湖应用洞察白皮书[J]. 数字经济,2022(9):18-27.

[6] OREŠČANIN D,HLUPIĆ T. Data lakehouse-a novel step in analytics architecture[C]//2021 44th international convention on information,communication and electronic technology (MIPRO). Opatija:IEEE,2021:1242-1246.

[7] CHA B R,PARK S,KIM J W. Design and interface testing of connected data architecture of DataLake[C]//2018 international conference on information and communication technology convergence (ICTC). Jeju:IEEE,2018:780-782.

[8] PARK S,CHA B R,KIM J. Design and implementation of connected DataLake system for reliable data transmission [C]//2019 23rd international computer science and engineering conference (ICSEC). Phuket:IEEE,2019:141-144.

[9] DIXON J. Pentaho,Hadoop,and data lakes[EB/OL]. 2010-10-14. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

[10] FANG H. Managing data lakes in big data era:what's a data lake and why has it became popular in data management ecosystem[C]//2015 IEEE international conference on cyber technology in automation,control,and intelligent systems (CYBER). Shenyang:IEEE,2015:820-824.

[11] WALKER C,ALREHAMY H. Personal data lake with data gravity pull[C]//2015 IEEE fifth international conference on big data and cloud computing. Dalian:IEEE,2015:160-167.

[12] MILOSLAVSKAYA N,TOLSTOY A. Big data,fast data and data lake concepts[J]. Procedia Computer Science,2016,88:300-305.

[13] LUO Z,NIU L,KORUKANTI V,et al. From batch processing to real time analytics:running presto® at scale[C]//2022 IEEE 38th international conference on data engineering (ICDE). Kuala Lumpur:IEEE,2022:1598-1609.

[14] SETHI R,TRAVERSO M,SUNDSTROM D,et al. Presto:SQL on everything[C]//2019 IEEE 35th international conference on data engineering (ICDE). Macao:IEEE,2019:1802-1813.

[15] CEREZO F,CUESTA C E,MORENO-HERRANZ J C,et al. Deconstructing the Lambda architecture:an experience report[C]//2019 IEEE international conference on software architecture companion (ICSA-C). Hamburg:IEEE,2019:196-201.

[16] LIN J. The lambda and the kappa[J]. IEEE Internet Computing,2017,21(5):60-66.